J. Math. Crypt. **2** (2008), 181–207
DOI 10.1515 / JMC.2008.009

Sieve algorithms for the shortest vector problem are practical

Phong Q. Nguyen and Thomas Vidick

Communicated by Tran van Trung

Abstract. The most famous lattice problem is the Shortest Vector Problem (SVP), which has many applications in cryptology. The best approximation algorithms known for SVP in high dimension rely on a subroutine for exact SVP in low dimension. In this paper, we assess the practicality of the best (theoretical) algorithm known for exact SVP in low dimension: the sieve algorithm proposed by Ajtai, Kumar and Sivakumar (AKS) in 2001. AKS is a randomized algorithm of time and space complexity $2^{O(n)}$, which is theoretically much lower than the super-exponential complexity of all alternative SVP algorithms. Surprisingly, no implementation and no practical analysis of AKS has ever been reported. It was in fact widely believed that AKS was impractical: for instance, Schnorr claimed in 2003 that the constant hidden in the $2^{O(n)}$ complexity was at least 30. In this paper, we show that AKS can actually be made practical: we present a heuristic variant of AKS whose running time is $(4/3+\varepsilon)^n$ polynomial-time operations, and whose space requirement is $(4/3+\varepsilon)^{n/2}$ polynomially many bits. Our implementation can experimentally find shortest lattice vectors up to dimension 50, but is slower than classical alternative SVP algorithms in these dimensions.

Keywords. Lattices, AKS Algorithm, sieve, LLL, enumeration.

AMS classification. 11Y16, 11H06.

1 Introduction

Lattices are discrete subgroups of $\mathbb{R}^m$. A lattice L can be represented by a basis, that is, a set of $n \leq m$ linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^m$ such that L is equal to the set $L(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^n x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$ of all integer linear combinations of the $\mathbf{b}_i$'s. The integer n is called the dimension of the lattice L, and helps to measure the hardness of lattice problems. Every lattice has a shortest vector, that is, a non-zero vector whose Euclidean norm is minimal among all other non-zero lattice vectors. The *shortest vector problem* (SVP) asks for such a vector: it is the most famous lattice problem, and is one of the very few potentially hard problems currently in use in public-key cryptography (see [26, 24] for surveys on lattice-based cryptosystems, and [15, 27] for recent developments). SVP is also well known for its applications in public-key cryptanalysis (see [26]): knapsack cryptosystems, RSA in special settings, DSA signatures in special settings, *etc*.

SVP algorithms can be classified in two categories: exact algorithms [21, 20, 4] (which provably output a shortest vector), and approximation algorithms [23, 31, 12, 13] (which output a non-zero lattice vector whose norm is provably not much bigger than that of a shortest vector). In high dimension (higher than 100), only approximation algorithms are practical, but both categories are in fact complementary: all exact

algorithms known first apply an approximation algorithm (such as LLL [23]) as a preprocessing, while all approximation algorithms known make intensive use of an exact algorithm in low dimension. More precisely, the celebrated LLL approximation algorithm [23] relies essentially on finding shortest vectors in dimension two, while the best approximation algorithm known (as well as its predecessors by Schnorr [31] and Gama *et al.* [12]), that of Gama and Nguyen [13], call (polynomially many times) an exact algorithm in dimension k, where the blocksize k is chosen in such a way that the cost of the overall algorithm remains polynomial. The heuristic BKZ algorithm [33] (implemented in NTL [34] and often used by cryptanalysts, see [14] for an experimental assessment) also crucially relies on an exact SVP algorithm in small dimension (typically chosen around 20): note however that recent experiments [14] suggest that the running-time bottleneck for BKZ on high-dimensional lattices is caused by the large number of calls, rather than the cost of the exact algorithm, which is why the BKZ blocksize is usually much smaller than the highest possible dimension for exact SVP. Thus, the best practical and/or theoretical SVP approximation algorithms all require an efficient exact SVP algorithm in low dimension.

It is therefore very important to know what is the best exact SVP algorithm in low dimension (say, less than 60), and to determine what is the highest dimension for which one solve exact SVP in the worst case. Because SVP is known to be NP-hard under randomized reductions [3], exact algorithms are not expected to be polynomial time. Surprisingly, there are so far essentially only two different algorithms for exact SVP:

- The deterministic enumeration algorithm discovered by Kannan and Pohst [28, 21] and its many variants [21, 20, 11, 33], which are essentially all surveyed in [1]. This algorithm enumerates a super-exponential number of potential shortest vectors, given a reduced basis. If the basis is only LLL-reduced, the running time is $2^{O(n^2)}$ polynomial-time operations, but Kannan [21] showed that one can perform suitable preprocessing in such a way that the overall running time (including preprocessing) is $2^{O(n \log n)}$ polynomial-time operations (see [18, 20] for a better constant than [21], and see [19] for a worst-case lattice basis). The algorithm used in practice is the Schnorr–Euchner variant [33] of the enumeration strategy, where the basis is either LLL reduced or BKZ reduced: here, the running time is therefore $2^{O(n^2)}$ polynomial-time operations.

- The randomized sieve algorithm [4] proposed in 2001 by Ajtai, Kumar and Sivakumar (AKS), whose running time is $2^{O(n)}$ polynomial-time operations. One drawback of AKS is that it has space complexity $2^{O(n)}$, whereas enumeration algorithms only require polynomial space. Even though this is a big drawback, one might still hope that this could be preferable to the $2^{O(n^2)}$ time complexity of currently used practical SVP enumeration algorithms.

Although the exponential complexity of AKS seems *a priori* much better than the super-exponential complexity of enumeration algorithms, no implementation and no practical analysis of AKS have ever been reported. This can perhaps be explained as follows:

- AKS is a fairly technical algorithm, which is very different from all other lattice algorithms. Ajtai, Kumar and Sivakumar use many parameters in their description [4], and their analysis does not explain what could be the optimal choice for these parameters. In particular, no explicit value of the $O()$ constant in the $2^{O(n)}$ complexity of AKS is given in the original paper [4].
- It was widely believed that AKS was impractical, because AKS uses exponential space and the complexity constants were thought to be large. Schnorr claimed in [32] that the $O()$ constant was at least 30, but Regev's alternative analysis [29] showed that it was at most 16.

OUR RESULTS. We show that sieve algorithms for the shortest vector problem are in fact practical, by developing an efficient heuristic variant of AKS which experimentally finds shortest vectors in dimension ≤ 50. Our variant runs in $(4/3+\epsilon)^n$ polynomial-time operations and uses $(4/3+\epsilon)^{n/2}$ polynomially many bits, where the $4/3$ constant is derived from a sphere covering problem. Interestingly, the $4/3$ constant is intuitively tight on the average, and seems to be supported by our experiments. To understand the principles of sieve algorithms, we first present a concrete analysis of the original AKS algorithm [4]. By choosing the AKS parameters carefully, we obtain a probabilistic algorithm which outputs a shortest vector with probability exponentially close to 1 within $2^{5.9n}$ polynomial-time operations. Though this shows that the original AKS algorithm is much more efficient than previously thought, this does not guarantee the practicality of the algorithm. Still, this concrete analysis is useful for the following reasons:

- The analysis is a worst-case analysis: the $2^{5.9n}$ running time does not reflect the true potential of sieve algorithms. For instance, the analysis also shows that the same algorithm approximates the shortest vector to within a constant factor 5 using only 2^{3n} polynomial-time operations. More generally, the analysis suggests that the real-life constants may be much smaller than the constants of the worst-case analysis. Note that many lattice algorithms typically perform better in practice than what their worst-case analysis suggests: see for instance [25] for the case of the LLL algorithm and [14] for the BKZ algorithm, where the experimental constants differ from the worst-case theoretical constants.
- The analysis explains what are the essential ingredients of sieve algorithms, which is crucial to develop faster variants.

However, our heuristic sieve algorithm turns out to be slower (up to dimension 50) than the $2^{O(n^2)}$ Schnorr–Euchner enumeration algorithm (with LLL preprocessing). In practice, the running time is very close to the $2^{O(n \log n)}$ Kannan–Helfrich enumeration algorithm [21, 20]. This shows that $O()$ constants and the exact cost of polynomial-time operations matter a lot in assessing the actual performance of lattice algorithms. We hope our results make it clear why sieve algorithms have an exponential running time, what is the expected value of the exponentiation constant in practice, and why they do not beat super-exponential enumeration techniques in practice.

ROAD MAP. The paper is organized as follows. In Section 2, we provide necessary background on lattices. In Section 3, we recall the AKS algorithm [4], and provide a concrete analysis of its complexity. In Section 4, we present and analyze a faster heuristic variant of AKS, and provide experimental results.

2 Background

Let $\|.\|$ and $\langle .,. \rangle$ be the Euclidean norm and inner product of $\mathbb{R}^n$. Vectors will be written in bold. We denote the n-dimensional ball of center $\mathbf{v} \in \mathbb{R}^n$ and radius R by $B_n(\mathbf{v}, R) = \{\mathbf{x} \in \mathbb{R}^n,\ \|\mathbf{x} - \mathbf{v}\| \leq R\}$, and we let $B_n(R) = B_n(\mathbf{O}, R)$. For a matrix M whose name is a capital letter, we will usually denote its coefficients by $m_{i,j}$. For any finite set S, let $|S|$ denote its number of elements. For any $X \subset \mathbb{R}^n$, we denote by $\mathrm{vol}\,(X)$ the volume of X. We refer to the survey [26] for a bibliography on lattices.

LATTICES. In this paper, by the term lattice, we mean a discrete subgroup of some $\mathbb{R}^m$. The simplest lattice is $\mathbb{Z}^n$, and for any linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$, the set $L(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^n m_i \mathbf{b}_i\ m_i \in \mathbb{Z}\}$ is a lattice. It turns out that in any lattice L, not just $\mathbb{Z}^n$, there must exist linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in L$ such that $L = L(\mathbf{b}_1, \ldots, \mathbf{b}_n)$. Any such n-tuple of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is called a *basis* of L: a lattice can be represented by a basis, that is, a row matrix. The *dimension* of a lattice L is the dimension n of the linear span of L. The lattice is full-rank if n is the dimension of the space. The first minimum $\lambda_1(L)$ is the norm of a shortest non-zero vector of L.

ORTHOGONALIZATION. Given a basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$, there exists a unique lower-triangular $n \times n$ matrix μ with ones on the diagonal and an orthogonal family $B^* = [\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*]$ such that $B = \mu B^*$. They can be computed using Gram–Schmidt orthogonalization, and will be called the *GSO* of B.

SIZE REDUCTION. A basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ is *size reduced* with factor $\eta \geq 1/2$ if its GSO family satisfies $|\mu_{i,j}| \leq \eta$ for all $1 \leq j < i$. An individual vector $\mathbf{b}_i$ is size reduced if $|\mu_{i,j}| \leq \eta$ for all $1 \leq j < i$. Size reduction usually refers to $\eta = 1/2$, and is typically achieved by successively size-reducing individual vectors. Size reduction was introduced by Hermite.

LLL REDUCTION. A basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of a lattice L is LLL reduced [23] with factor δ for $1/4 < \delta \leq 1$ if its GSO satisfies $|\mu_{i,j}| \leq 1/2$ for all $i > j$, as well as the $(n-1)$ Lovász conditions $(\delta - \mu_{i+1,i}^2)\|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2$. The first vector of such bases has the following properties: $\|\mathbf{b}_1\| \leq \alpha^{(n-1)/4}\mathrm{vol}\,(L)^{1/n}$ and $\|\mathbf{b}_1\| \leq \alpha^{(n-1)/2}\lambda_1(L)$, where $\alpha = 1/(\delta - 1/4)$. If no δ is given, it will mean the original choice $\delta = 3/4$ of [23], in which case $\alpha = 2$. It is well known that LLL-reduced bases can be computed in polynomial time if $1/4 < \delta < 1$.

HKZ REDUCTION. A basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of a lattice L is Hermite–Korkine–Zolotarev (HKZ) reduced if it is size reduced and if $\mathbf{b}_i^*$ is a shortest vector of the projected lattice $\pi_i(L)$ for all $1 \leq i \leq n$, where π_i is the orthogonal projection on the subspace orthogonal to $\mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$. In particular, the first basis vector is a shortest vector of the lattice.

BABAI'S ROUNDING. Given a basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of a full-rank lattice L, and any target vector $\mathbf{t} \in \mathbb{Q}^n$, linear algebra gives in polynomial time a lattice vector $\mathbf{v} \in L$, such that $\mathbf{v} - \mathbf{t} = \sum_{i=1}^{n} x_i \mathbf{b}_i$ where $|x_i| \leq 1/2$, and therefore $\|\mathbf{v} - \mathbf{t}\| \leq \sum_{i=1}^{n} \|\mathbf{b}_i\|/2$. This is Babai's rounding algorithm [6], which approximates the closest vector problem within an exponential factor, if the input basis is LLL reduced.

RANDOM LATTICES. There is a beautiful (albeit mathematically involved) theory of random lattices, which was initiated by Siegel to provide an alternative proof of the Minkowski–Hlawka theorem (see [17]). Recently, efficient methods [16, 2] have been developed to generate provably random lattices. Random lattices are very interesting for experiments, because they do not seem to have any exceptional property which could *a priori* be exploited by algorithms. We therefore used random lattices in our experiments, as was done in [25, 14] using [16].

3 Revisiting the AKS sieve algorithm

In this section we recall and analyze in detail the sieve algorithm by Ajtai, Kumar and Sivakumar [4] for the shortest vector problem. The goal is to understand AKS and to see how small the complexity constants of AKS can be, since no practical analysis of AKS has been reported. All the main ideas of this section have previously appeared either in the original paper [4] or in Regev's analysis [29], but the analysis will be useful to understand our heuristic variant of AKS, described in Section 4. We will prove the following concrete version of the AKS algorithm:

Theorem 3.1. *There is a randomized algorithm which, given as input any polynomial-size basis of an integer lattice $L \subseteq \mathbb{Z}^n$, outputs a shortest vector of L with probability exponentially close to 1, using at most $2^{5.90n}$ polynomial-time operations on numbers of polynomial size, and with space requirement at most $2^{2.95n}$ polynomial-size registers. Furthermore, the same algorithm can be used to output a non-zero lattice vector of norm less than $5\lambda_1(L)$ with probability exponentially close to 1, in less than 2^{3n} polynomial-time operations on numbers of polynomial size, using at most $2^{1.5n}$ polynomial-size registers.*

The analysis of [4] does not give any concrete value of the involved constants, while the analysis [29] uses fixed values of constants (for ease of exposition) which are not claimed to be optimal: more precisely, [29] shows a less efficient version of Theorem 3.1 where 5.9 is replaced by 16, because the most expensive step of [29] is a quadratic-time stage on roughly 2^{8n} points. We hope the presentation clarifies the main ideas of sieve algorithms, and provides intuition on how the various constants are related to each other. Like [4, 29], we will use real numbers for ease of exposition, but in practice all numbers will actually be represented using polynomial precision.

3.1 Overview

Let $L \subseteq \mathbb{Z}^n$ be a full-rank integer lattice. Let $S \subseteq \mathbb{R}^n$ be a subset containing $\lambda_1(L)$. Enumeration algorithms [21, 20] enumerate all vectors in $L \cap S$ for a choice of S well

suited to enumeration, and output the shortest one, which must be of norm $\lambda_1(L)$. With the best choice of S in [21, 20], we have $|L \cap S| = 2^{O(n \log n)}$, which leads to $2^{O(n \log n)}$ polynomial operations.

The main idea of sieve algorithms is to perform some kind of randomized enumeration of a smaller set. Namely, the algorithm will sample $N = 2^{\Omega(n)}$ vectors in $L \cap S$ for an S such that $|L \cap S| = 2^{O(n)}$. In particular, $S = B_n(R)$ where $R = O(\lambda_1(L))$ is a good candidate, as the following elementary lemma shows:

Lemma 3.2. *Let L be a lattice in $\mathbb{R}^n$, and $R > 0$. Then $|L \cap B_n(R)| \leq 2^{c_R n}$ where $c_R = \log_2\left(1 + \frac{2R}{\lambda_1(L)}\right)$.*

Proof. Let $r = (1-\varepsilon)\lambda_1(L)/2$ for some $0 < \varepsilon < 1$. By definition of $\lambda_1(L)$, the balls $B_n(\mathbf{v}, r)$ where $\mathbf{v} \in L \cap B_n(R)$ do not overlap and are all included in $B_n(R + \lambda_1(L)/2)$. It follows that $|L \cap B_n(R)|$ cannot exceed $\text{vol}\,(B_n(R+\lambda_1(L)/2))/\text{vol}\,(B_n(r)) = ((R+\lambda_1(L)/2)/r)^n$. The bound follows by letting ε decrease to 0. □

If the sampling is such that each point of $L \cap S$ has a probability of being output of order $|L \cap S|^{-1}$, and if $N \gg |L \cap S|$, one of the samples would be a shortest lattice vector with probability close to 1. Unfortunately, it is not known whether such a sampling procedure exists. Instead, we will prove that it is possible to sample vectors such that there exists $\mathbf{w} \in L \cap S$ such that $\mathbf{w}$ and $\mathbf{w} + \mathbf{s}$, where $\mathbf{s}$ is a shortest lattice vector, both have a non-zero probability of being output. Thus, by computing the shortest difference between N sampled vectors in $L \cap S$ where $N \gg |L \cap S|$, we will obtain a shortest lattice vector with probability close to 1.

Yet, sampling vectors in $L \cap B_n(R)$ for some $R = O(\lambda_1(L))$ looks very difficult. AKS first applies LLL reduction to sample $2^{O(n)}$ vectors in $L \cap B_n(R_0)$ for some $R_0 = 2^{O(n)}\lambda_1(L)$. This is not difficult, and can for instance be done by combining Babai's rounding (see Section 2) with the following folklore result:

Lemma 3.3. *There exists a constant $c_r > 0$ such that the following holds:*
Let $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ be an LLL-reduced basis of a lattice L. If $\mathbf{s}$ is a shortest vector of L, then there exists an index $i \in \{1, \ldots, n\}$ such that $\mathbf{s}$ belongs to the lattice spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_i$, and for all $j \leq i$: $\|\mathbf{b}_j\| \leq c_r^n \lambda_1(L)$.

Proof. It is a basic property of LLL reduction that, if $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ is LLL reduced, then $[\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_n)]$ is also an LLL-reduced basis of the projected lattice $\pi_i(L)$. Since $\pi(\mathbf{b}_i) = \mathbf{b}_i^*$ is the corresponding vector of the Gram–Schmidt orthogonalisation of the basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$, we have that, for all $1 \leq i \leq n$: $\|\mathbf{b}_i^*\| \leq 2^{(n-i)/2}\lambda_1(\pi_i(L))$.

Let $1 \leq i \leq n-1$ be such that $\mathbf{s}$ does not belong to the lattice spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_i$. Then the projection $\pi_{i+1}(\mathbf{s})$ is not zero, which implies that

$$\|\mathbf{b}_{i+1}^\star\| \leq 2^{(n-i-1)/2}\|\pi_{i+1}(\mathbf{s})\| \leq 2^{n/2}\|\mathbf{s}\| = 2^{n/2}\lambda_1(L)$$

Since $\mathbf{s}$ does not belong to the lattice spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_j$ for all $j \leq i$ either, we have that, for all $j \leq i+1$:

$$\|\mathbf{b}_j^\star\| \leq 2^{n/2}\lambda_1(L)$$

Therefore, since the basis is LLL reduced,

$$\|\mathbf{b}_j\|^2 \leq \|\mathbf{b}_j^\star\|^2 + \frac{1}{4}\sum_{k<j}\|\mathbf{b}_k^\star\|^2 \leq j2^{2n/2}\lambda_1(L)^2 \leq c_r^{2n}\lambda_1(L)^2,$$

for a certain constant c_r. Thus, for all $j \leq i+1$:

$$\|\mathbf{b}_j\| \leq c_r^n \lambda_1(L).$$

To conclude the proof of the lemma, take i to be the smallest index such that $\mathbf{s}$ belongs to the lattice spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_i$. □

By sampling random vectors of size $O(1)$ in $\mathbb{R}^n$ and rounding them to lattice vectors using Babai's algorithm, we can assume that we have a set S of many samples in $L \cap B_n(R_0)$. The main and most expensive part of the algorithm is to derive samples in $L \cap B_n(O(\lambda_1(L)))$ from S. This will be achieved by a very simple process: sieving, which will make the vectors of S shorter and shorter, by at least some geometric factor γ such that $0 < \gamma < 1$. Under suitable conditions, the sieve takes as input $2^{O(n)}$ vectors in $L \cap B_n(R)$ and outputs almost as many vectors in $L \cap B_n(\gamma R)$: each output vector will be a subtraction of two well-chosen input vectors. More precisely, the sieve will select a (not too big) subset C of the input set S, and the output set will be obtained by subtracting each vector of $S \setminus C$ with a well-chosen vector in C. Vectors in C are then discarded. Intuitively, the problem of finding the right subset C is closely related to a sphere covering problem, and this connection will be further discussed in Section 4. By iterating the sieve linearly many times, AKS finally obtains vectors in $L \cap B_n(O(\lambda_1(L)))$.

What we have just sketched is an idealized AKS algorithm, but the actual algorithm is more complicated, due to bottlenecks in the analysis caused by distributions over $L \cap B_n(R)$. More precisely, instead of sieving vectors in $L \cap B_n(R)$, we will sieve vectors in $B_n(R)$, while keeping lattice approximations of such vectors. We will store pairs $(\mathbf{v}, \mathbf{y}) \in L \times B_n(R)$ such that $\mathbf{v} \in L$ is an approximation of $\mathbf{y}$, namely $\|\mathbf{y}-\mathbf{v}\| \leq \xi$ where ξ is fixed, and therefore $\|\mathbf{v}\| \leq R + \xi$. The $\mathbf{v}$-part is important to remember the operations on lattice vectors, but the decisions made by the sieve will only be based on the $\mathbf{y}$-part. We will start with couples $(\mathbf{v}, \mathbf{y})$ such that $\|\mathbf{y}\| \leq R = R_0 = 2^{O(n)}\lambda_1(L)$ and the perturbation $\mathbf{y} - \mathbf{v}$ is uniformly distributed in $B_n(\xi)$ (this property will be essential to the complexity analysis). Given many such pairs $(\mathbf{v}, \mathbf{y})$, the new sieve will output pairs $(\mathbf{v}', \mathbf{y}') \in L \times B_n(\gamma R + \xi)$ such that $\|\mathbf{y}' - \mathbf{v}'\| \leq \xi$ and therefore $\|\mathbf{v}'\| \leq \gamma R + 2\xi$. Thus, by iterating the sieve a linear number of times, we will obtain lattice vectors in $B_n(O(\xi/(1-\gamma)))$. In order to sample vectors in $L \cap B_n(O(\lambda_1(L)))$, the perturbation must not be too big, namely $\xi = O(\lambda_1(L))$.

Algorithm 1 gives the overall structure of the AKS algorithm, which takes as input a lattice L and a triplet (γ, ξ, c_0): γ is the shrinking factor of the sieve, ξ is the size of the perturbations and $N = 2^{c_0 n}$ is the number of points which are initially sampled. Theorem 3.1 will be proved with a suitable choice of (γ, ξ, c_0). Let us give a few comments on the structure of Algorithm 1. In Step 5, which is the end of the initialization, the set S consists of N pairs $(\mathbf{v}, \mathbf{y}) \in L \times B_n(R)$ such that $\|\mathbf{y}-\mathbf{v}\| \leq \xi$ and $R = n \max_i \|\mathbf{b}_i\|$.

Algorithm 1 The AKS algorithm for the Shortest Vector Problem

Input: An LLL-reduced basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of a lattice L satisfying Lemma 3.3, and parameters $0 < \gamma < 1$, $\xi > 0$ such that $\xi = O(\lambda_1(L))$ and $c_0 > 0$.
Output: A shortest vector of L under suitable conditions on (γ, ξ, c_0).
1: $S \leftarrow \emptyset$
2: **for** $j = 1$ to $N = 2^{c_0 n}$ **do**
3: $S \leftarrow S \cup \mathsf{sampling}(B, \xi)$ using Algorithm 2.
4: **end for**
5: $R \leftarrow n \max_i \|\mathbf{b}_i\| + \xi$
6: **for** $j = 1$ to $k = \lceil \log_\gamma \left(\frac{0.01\xi}{R(1-\gamma)} \right) \rceil$ **do**
7: $S \leftarrow \mathsf{sieve}(S, \gamma, R, \xi)$ using Algorithm 3.
8: $R \leftarrow \gamma R + \xi$
9: **end for**
10: Compute $\mathbf{v}_0 \in L$ such that $\|\mathbf{v}_0\| = \min\{\|\mathbf{v} - \mathbf{v}'\|$ where $(\mathbf{v}, \mathbf{y}) \in S, (\mathbf{v}', \mathbf{y}') \in S, \mathbf{v} \neq \mathbf{v}'\}$
11: **return** $\mathbf{v}_0$.

At each iteration of the sieve loop (steps 6 to 9), the set S will still consist of pairs $(\mathbf{v}, \mathbf{y}) \in L \times B_n(R)$ such that $\|\mathbf{y} - \mathbf{v}\| \leq \xi$, but both $|S|$ and R will decrease. We now describe the two subroutines used by Algorithm 1: sampling and sieving.

3.2 Initial sampling

Algorithm 2 generates a pair $(\mathbf{v}, \mathbf{y}) \in L \times \mathbb{R}^n$ such that $\|\mathbf{y}\| \leq n \max_i \|\mathbf{b}_i\|$ and $\mathbf{y} - \mathbf{v}$ is uniformly distributed in $B_n(\xi)$. The sampling algorithm follows [29], rather than [4]:

Algorithm 2 Initial sampling

Input: A basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of a lattice L, and a real $\xi > 0$.
Output: A pair $(\mathbf{v}, \mathbf{y}) \in L \times \mathbb{R}^n$ such that $\|\mathbf{y}\| \leq n \max_i \|\mathbf{b}_i\|$ and $\mathbf{y} - \mathbf{v}$ is uniformly distributed in $B_n(\xi)$.
1: $\mathbf{x} \leftarrow_{\text{random}} B_n(\xi)$
2: $\mathbf{v} \leftarrow \mathsf{ApproxCVP}(-\mathbf{x}, B)$ where $\mathsf{ApproxCVP}$ is Babai's rounding algorithm [6].
3: $\mathbf{y} \leftarrow \mathbf{v} + \mathbf{x}$
4: **return** $(\mathbf{v}, \mathbf{y})$

we first generate the perturbation $\mathbf{x} = \mathbf{y} - \mathbf{v}$, then deduce a close lattice vector $\mathbf{v}$ using Babai's rounding algorithm [6], rather than first select a lattice vector and then add a suitable perturbation. The requirement $\|\mathbf{y}\| \leq n \max_i \|\mathbf{b}_i\|$ is guaranteed by Babai's rounding algorithm.

Algorithm 2 has two important properties which will be used in the analysis. The first property is that the approximate CVP algorithm used in Step 2 is additive with respect to the lattice: $\mathsf{ApproxCVP}(\mathbf{x} + \mathbf{z}, B) = \mathbf{z} + \mathsf{ApproxCVP}(\mathbf{x}, B)$ for every $\mathbf{x} \in \mathbb{R}^n$

and every lattice vector $\mathbf{z} \in L$. This means that if $\mathbf{x}$ is replaced at the end of Step 1 by $\mathbf{x} + \mathbf{z}$ where $\mathbf{z} \in L$, then $\mathbf{v}$ in Step 2 is replaced by $\mathbf{v} - \mathbf{z}$, and $\mathbf{y}$ remains the same.

The second property is the uniform distribution given by Step 1. If $\xi > \lambda_1(L)/2$, for any shortest lattice vector $\mathbf{s}$, the intersection $B_n(\xi) \cap B_n(\mathbf{s}, \xi)$ is significant:

Lemma 3.4. *Let $\mathbf{s}$ be a shortest vector of an n-dimensional lattice L and $\xi > \lambda_1(L)/2$. If we select $\mathbf{x}$ uniformly at random in $B_n(\xi)$, then $\mathbf{x} + \mathbf{s} \in B_n(\xi)$ with probability at least $2^{-c_{\mathcal{U}} n}$ where:*

$$c_{\mathcal{U}} = \frac{1}{2} \log \left(\frac{\xi^2}{\xi^2 - \lambda_1^2/4} \right).$$

Proof. Assume that $\lambda_1/2 < \xi < \lambda_1$, and let $X = B_n(\mathbf{O}, \xi) \cap B_n(-\mathbf{s}, \xi)$. Then all $\mathbf{x} \in X$ are such that both $\mathbf{x}$ and $\mathbf{x} + \mathbf{s}$ are in $B_n(\mathbf{O}, \xi)$. To estimate the volume of X, note that for any $\epsilon < \xi - \lambda_1/2$ it contains the cylinder of height $h = 2(\xi - \epsilon) - \lambda_1$ and basis the $(n-1)$-dimensional ball of radius $r = \sqrt{\xi^2 - (\xi - \epsilon)^2}$. The ratio of the volume of this cylinder to the volume of $B_n(\xi)$ is asymptotically of order $(r/\xi)^n$. The optimal bound is obtained by letting $\epsilon \to \xi - \lambda_1/2$, which yields the desired expression for $c_{\mathcal{U}}$. □

If $\mathbf{x} + \mathbf{s} \in B_n(\xi)$, then the additivity property shows that if $\mathbf{x}$ had been replaced by $\mathbf{x} + \mathbf{s}$ in Step 1, the $\mathbf{y}$-part of Algorithm 2's output would not have changed.

3.3 Sieving

The heart of the AKS algorithm is the sieve described in Algorithm 3, which is the L_2-sieve of [29], rather than the L_∞-sieve of [4]. Given as input a set $S = \{(\mathbf{v}_i, \mathbf{y}_i),\ i \in I\} \subseteq L \times B_n(R)$ such that $\forall i \in I,\ \|\mathbf{y}_i - \mathbf{v}_i\| \leq \xi$, the sieve outputs a new set $S' = \{(\mathbf{v}'_i, \mathbf{y}'_i),\ i \in I'\} \subseteq L \times B_n(\gamma R + \xi)$ such that for all $i \in I'$, $\|\mathbf{y}'_i - \mathbf{v}'_i\| \leq \xi$. This is achieved by selecting a special subset C of couples from S, which we call *centers*,

Algorithm 3 The sieve with perturbations

Input: A set $S = \{(\mathbf{v}_i, \mathbf{y}_i),\ i \in I\} \subseteq L \times B_n(R)$ and a triplet (γ, R, ξ) such that $\forall i \in I,\ \|\mathbf{y}_i - \mathbf{v}_i\| \leq \xi$.

Output: A set $S' = \{(\mathbf{v}'_i, \mathbf{y}'_i),\ i \in I'\} \subseteq L \times B_n(\gamma R + \xi)$ such that $\forall i \in I',\ \|\mathbf{y}'_i - \mathbf{v}'_i\| \leq \xi$.

```
C ← ∅
for i ∈ I do
    if ∃c ∈ C ‖y_i − y_c‖ ≤ γR then
        S' ← S' ∪ {(v_i − v_c, y_i − v_c)}
    else
        C ← C ∪ {i}
    end if
end for
return S'
```

and taking the difference of every vector in $S\backslash C$ with its closest point in C. Clearly, Algorithm 3 uses at most $|C| \times |S|$ polynomial-time operations, and has the following properties:

- The sieve preserves the perturbations: for any output pair $(\mathbf{v}'_i, \mathbf{y}'_i) \in S'$, there exists an input pair $(\mathbf{v}_i, \mathbf{y}_i) \in S$ such that $\mathbf{v}'_i - \mathbf{y}'_i = \mathbf{v}_i - \mathbf{y}_i$.
- The elements in C are chosen based only on the $\mathbf{y}$-part of the pairs $(\mathbf{v}_i, \mathbf{y}_i) \in S\backslash C$. In other words, the sieve bases its decisions solely on the $\mathbf{y}$- part, but it does update the $\mathbf{v}$-part.

The most important property of the sieve is that the number $|C|$ of centers is exponentially bounded, where the bound only depends on the shrinking factor γ: the following lemma shows that there exists a constant $c_{\mathcal{S}}$ depending only on γ such that, if $|S| \gg 2^{c_{\mathcal{S}} n}$, then the output set S' will be almost as big as the input set S.

Lemma 3.5. *Let* $\gamma_0 = \frac{1}{2}\left(\sqrt{5}-1\right) \approx 0.618$ *and* $\frac{1}{2} < \gamma < 1$. *If* $\gamma \geq \gamma_0$, *let* $c_{\mathcal{S}} = \log_2\left(\frac{2}{\gamma}\right)$, *otherwise,* $c_{\mathcal{S}} = \frac{1}{2}\log_2\left(2 + \frac{2}{2\gamma - 1}\right)$. *Then the number of centers used during Algorithm 3 always satisfies:* $|C| \leq 2^{c_{\mathcal{S}} n}$.

Proof. We assume without loss of generality that the set S contains the zero vector, and that this vector is the first to be added to C. All vectors $\mathbf{y}$ of norm less than γR are therefore directly associated to this vector, and we only add to C indices corresponding to vectors $\mathbf{y}$ such that $\|\mathbf{y}\| > \gamma R$. Moreover, a point is added to C only if it is at distance at least γR from all points already in C. In Lemma 4.2 of Section 4 we show that the ball of center $\mathbf{y} \in C_n(\gamma, R) = \{\mathbf{x} \in \mathbb{R}^n,\ \gamma R \leq \|\mathbf{x}\| \leq R\}$ and radius γR contains the angular sector $S_n(\mathbf{y}, \phi)$ of all points of $C_n(\gamma, R)$ making an angle less than ϕ with $\mathbf{y}$, where $\cos\phi = 1 - \gamma^2/2$ if $\gamma > \gamma_0$ and $\cos\phi = 1/(2\gamma)$ otherwise. The angle formed between the new point and any old points is therefore greater than ϕ, so the angular sectors $S_n(\mathbf{y}_i, \phi/2)$, for $i \in C$, are disjoint. The sum of their volumes over all vectors in C must then be less than the volume of the corona. Lemma 4.2 then provides the required bound. □

3.4 Complexity analysis

We now complete the analysis of the AKS algorithm (Algorithm 1). First of all, we note that by Lemma 3.3, one can compute polynomially many $\lambda \in \mathbb{Q}$ such that one of the λ's satisfies $\lambda_1(L) \leq \lambda \leq 1.01\lambda_1(L)$. Such an estimate on $\lambda_1(L)$ will be sufficient to select the parameters of the algorithm.

At step 9 of the algorithm, we are left with a set S which by Lemma 3.5 contains a large number of vectors in L, provided that the initial number of sampled vectors was big enough. Moreover, the total number k of iterations of the sieving procedure was chosen so that all these vectors have small norm $O(\lambda_1(L))$, but they could still all be the zero vector. The difficult part in the analysis is to show that we can extract a shortest vector from this set. This is done in the following lemma, which uses the properties of the sampling and sieving procedures that were put forward in Sections 3.2 and 3.3 respectively.

Lemma 3.6. *Let L be an n-dimensional lattice, (γ, ξ, c_0) a choice of parameters for Algorithm 1, k the number of iterations of the sieve loop, as defined in step 6 of Algorithm 1, and $R_\infty = (1 + 1.01/(1-\gamma))\xi = O(\lambda_1(L))$. Let c_{R_∞}, $c_{\mathcal{U}}$ and $c_{\mathcal{S}}$ be the constants defined respectively in Lemmas 3.2, 3.4 and 3.5. Finally, if $N_\infty = 2^{(c_0-c_{\mathcal{U}})n}/2 - k2^{c_{\mathcal{S}}n}$, then*

- *If $N_\infty \geq 8$, Algorithm 1 outputs with probability at least $1/2$ a non-zero vector of L with norm less than R_∞.*
- *If $N_\infty \geq 2^{c_{R_\infty}n+3}$, Algorithm 1 outputs with probability at least $1/2$ a shortest vector of L.*

Proof. We follow Regev's approach [29], by introducing a conceptual modification of the sampling part of the algorithm. Let $\mathbf{s}$ be a (fixed) shortest vector of the lattice, and $X = \{\mathbf{x} \in B_n(\xi),\ \mathbf{x}+\mathbf{s} \in B_n(\xi)\}$. By Lemma 3.4, $\mathrm{vol}\,(X)/\mathrm{vol}\,(B_n(\xi)) \geq 2^{-c_{\mathcal{U}}n}$. Let $\tau : B_n(\xi) \to B_n(\xi)$ be such that $\tau(\mathbf{x}) = \mathbf{x}+\mathbf{s}$ if $\mathbf{x} \in X$, $\tau(\mathbf{x}) = \mathbf{x}-\mathbf{s}$ if $\mathbf{x} \in X+\mathbf{s}$ (note that X and $(X+\mathbf{s})$ have no overlap if $\xi \leq \|\mathbf{s}\| = \lambda_1(L)$, which will be the case) and τ is the identity elsewhere. τ is such that for all $\mathbf{x} \in X$, $\mathsf{ApproxCVP}(\tau(\mathbf{x}), B) = \mathsf{ApproxCVP}(\mathbf{x}, B) \pm \mathbf{s}$.

We modify Algorithm 1 by applying τ with probability $1/2$ on every perturbation $\mathbf{x}$ just after it is chosen by the sampling procedure in step 1 of Algorithm 2. Note that, since applying τ preserves the uniform distribution on $B_n(\xi)$, and every other step of the algorithm is deterministic, the output distribution on vectors of L of the modified algorithm must be exactly the same as that of the original algorithm. Since the sieve only bases its decisions on the $\mathbf{y}$ part of all output couples $(\mathbf{v}, \mathbf{y})$, we can imagine for all output points that we only apply τ at the last step in the algorithm (step 10) (for a point that is used as a center τ must be applied when it is used, because $\mathbf{v}$ is used in determining the new value of the points that are associated to this center).

Of the $N = 2^{c_0 n}$ couples $(\mathbf{v}, \mathbf{y})$ that were sampled, we only consider those that are such that $\mathbf{y} - \mathbf{v} \in X$. By Lemma 3.4, with probability exponentially close to 1, there are at least $2^{(c_0-c_{\mathcal{U}})n}/2$ such vectors. Lemma 3.5 then tells us that, at step 10 in the algorithm, at least N_∞ couples in S are such that $\mathbf{y} - \mathbf{v} \in X$ (recall that the perturbations $\mathbf{x}$ are left unchanged by the sieve's operations). We call these couples *good*. Now consider the situation just before applying τ.

If $N_\infty \geq 8$, then there are at least 8 good vectors. There are two cases. Either at least $1/2$ of these are zero. Then, after applying τ, with probability at least $1/2$, one of the at least 4 good zero vectors is sent to $\mathbf{s}$, and another is left unchanged (remember we applied τ with probability $1/2$), so that, when taking differences at the last step of Algorithm 1, we recover $\mathbf{s}$. Or, at least $1/2$ of all good vectors are non- zero. Then, after applying τ, at least one of them will remain unchanged with probability greater then $1/2$, and at the final step, we recover a vector of norm less than R_∞. So, in all cases, with probability greater than $1/2$, the modified algorithm outputs a vector of norm less than R_∞.

Now assume that $N_\infty \geq 2^{c_{R_\infty}n+3}$. Since, by Lemma 3.2, $|L \cap B_n(R_\infty)| \leq 2^{c_{R_\infty}n}$, at least $N_\infty 2^{-c_{R_\infty}n}$ of the output points must correspond to the same vector $\mathbf{v}$. The same argument than that of the first case above then guarantees that the modified algorithm outputs a shortest vector of L with probability at least $1/2$.

To conclude the argument, recall that both the original and the modified algorithms produce the same output distributions on vectors of L, so the properties proved above for the modified algorithm carry on to the original algorithm. □

From Lemma 3.6, we conclude that Algorithm 1 outputs a shortest vector of its input lattice L with probability exponentially close to 1 provided that the following constraints on the parameters (γ, ξ, c_0) are met:

- $c_0 - c_{\mathcal{U}} > c_{\mathcal{S}}$: the number of good points sampled must be big enough so that only a small fraction is used as centers in steps 2–8 of Algorithm 3.
- $c_0 - c_{\mathcal{U}} > c_{R_\infty}$: the number of good points output in step 10 must be big enough to ensure that at that step the set S contains at least two couples $(\mathbf{v}, \mathbf{y}_1)$ and $(\mathbf{v} + \mathbf{s}, \mathbf{y}_2)$, so that a shortest vector $\mathbf{s}$ is recovered when taking the differences.

Furthermore, the second constraint can be dropped if we are only interested in obtaining a non-zero vector of norm less than R_∞. Using the values of the constants given in Lemmas 3.2, 3.4 and 3.5 and optimizing over γ and ξ, we find that, in the case where we want a shortest vector, the best choice of parameters is $\gamma = 0.518$ and $\xi = 0.7\lambda_1(L)$, which yields a value $c_0 < 2.95$. Since the running time of Algorithm 1 is dominated by the sieve procedure, whose complexity is of $poly(n) \cdot 2^{(c_0+c_{\mathcal{S}})n}$ operations, and the final step computing differences, of complexity $poly(n)2^{2c_0 n}$, the overall complexity of the algorithm is of $2^{5.9n}$ polynomial-time operations. In the case where we only want a short vector, taking $\gamma = 0.765$ and $\xi = 0.95\lambda_1(L)$ we get $c_0 < 1.5$ and $R_\infty < 5\lambda_1(L)$, so that with probability exponentially close to 1 we find a non- zero vector of norm less than $5\,\lambda_1(L)$ in less than 2^{3n} polynomial-time operations. This proves Theorem 3.1.

4 A fast heuristic sieve algorithm

The complexity analysis made in the previous section makes it clear that the AKS algorithm could in fact behave much better than what is proved in Theorem 3.1. For instance, the proofs of Lemmas 3.2 and 3.5 are both based on packing arguments, which are unlikely to be tight. This means that the constants c_R and $c_{\mathcal{S}}$ could probably be chosen smaller in practice: if so, we could sample initially much less points, thereby decreasing the running time of the sieve, which is $O(N^2)$ polynomial-time operations where N is the number of sampled points. Furthermore, it is rather pessimistic to require more vectors than lattice points inside a given ball to ensure that there is a collision: by the birthday paradox, the c_R constant could be replaced by $c_R/2$ if the vectors were uniformly distributed in the ball. Finally, while the perturbations $\mathbf{y}$ are necessary for the proof of correctness to go through, their practical interest is unclear. These observations suggest that we might be able to achieve a much better running time by stripping off all constraints on the original AKS algorithm which seem unnecessary in practice.

In this section, we present a heuristic variant of the AKS algorithm whose running time (resp. space complexity) will be essentially $(4/3)^n$ polynomial-time operations

(resp. $\sqrt{4/3}^n$ polynomially- sized registers): we will explain the origin of the $4/3$ constant. Furthermore, we argue that under a natural heuristic assumption on the distribution of lattice vectors used by the algorithm, the result will be a very short – if not shortest – lattice vector. Our heuristic assumption is supported by experiments in low dimension. Our experiments show that sieve algorithms are practical up to dimension at least 50. Unfortunately, we will see in Section 4.3 that our experiments also show that sieve algorithms are slower than the usual super-exponential enumeration techniques for such dimensions, and we will explain that phenomenon.

4.1 Description of the heuristic algorithm

The AKS algorithm of Section 3 generates N pairs at random, which are sieved many times. The total running time is of N^2 polynomial-time operations, and we showed that for some $N = 2^{O(n)}$ and an appropriate choice of the parameters γ and ξ the algorithm output a shortest lattice vector with overwhelming probability. Since the heart of the AKS algorithm is its sieve, which reduces the norm of its input vectors, it is natural to consider the heuristic variant described in Algorithm 4. This heuristic algorithm takes

Algorithm 4 Finding short lattice vectors based on sieving

Input: An LLL-reduced basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of a lattice L, a sieve factor γ such that $2/3 < \gamma < 1$, and a number N.
Output: A short non-zero vector of L.

1: $S \leftarrow \emptyset$
2: **for** $j = 1$ to N **do**
3: $S \leftarrow S \cup \mathsf{sampling}(B)$ using algorithm $\mathcal{K}$ described in Section 4.2.1.
4: **end for**
5: Remove all zero vectors from S.
6: $S_0 \leftarrow S$
7: **repeat**
8: $S_0 \leftarrow S$
9: $S \leftarrow \mathsf{latticesieve}(S, \gamma)$ using Algorithm 5.
10: Remove all zero vectors from S.
11: **until** $S = \emptyset$
12: Compute $\mathbf{v}_0 \in S_0$ such that $\|\mathbf{v}_0\| = \min\{\|\mathbf{v}\|, \mathbf{v} \in S_0\}$
13: **return** $\mathbf{v}_0$

as input two parameters: N and γ. In steps 1–5, it samples N non-zero lattice vectors of reasonable length, say $\leq R_0$. Then steps 7–11 apply a lattice sieve (Algorithm 5) which reduces the maximal norm of the vectors in S by a geometric factor γ. The lattice sieve is a simpler version of the sieve with perturbations (Algorithm 3). At the end of the algorithm, we obtain a non-zero lattice vector, but it is unclear how short that lattice vector is. This depends on the number of iterations of the lattice sieve: the main loop of Algorithm 4 repeats until the set S of all lattice vectors currently under consideration is empty. The size of S can decrease in two ways: first through Algorithm 5, which

Algorithm 5 The lattice sieve

Input: A subset $S \subseteq B_n(R)$ of vectors in a lattice L and a sieve factor $2/3 < \gamma < 1$.
Output: A subset $S' \subseteq B_n(\gamma R) \cap L$.

1: $R \leftarrow \max_{\mathbf{v} \in S} \|\mathbf{v}\|$
2: $C \leftarrow \emptyset$, $S' \leftarrow \emptyset$
3: **for** $\mathbf{v} \in S$ **do**
4: **if** $\|\mathbf{v}\| \leq \gamma R$ **then**
5: $S' \leftarrow S' \cup \{\mathbf{v}\}$
6: **else**
7: **if** $\exists \mathbf{c} \in C \ \|\mathbf{v} - \mathbf{c}\| \leq \gamma R$ **then**
8: $S' \leftarrow S' \cup \{\mathbf{v} - \mathbf{c}\}$
9: **else**
10: $C \leftarrow C \cup \{\mathbf{v}\}$
11: **end if**
12: **end if**
13: **end for**
14: **return** S'

removes the vectors in C from S, second by elimination of zero vectors in step 10 of Algorithm 4. To estimate the quality of the output vector we therefore need to evaluate the size of each of these two reductions.

For this, since Algorithm 5 only sieves the vectors which have norm between γR and R, we make the following natural assumption:

Heuristic: We assume that at any stage in Algorithm 4, the vectors in $S \cap C_n(\gamma, R)$ are uniformly distributed in $C_n(\gamma, R) = \{\mathbf{x} \in \mathbb{R}^n, \ \gamma R \leq \|\mathbf{x}\| \leq R\}$.

This assumption immediately shows that the loss of points in S through collisions to 0 in step 5 of Alg. 4 will remain negligible until the radius R is such that $|B_n(R) \cap L| \geq |S|^2$, meaning that it only happens when we already have a good approximation of λ_1: see Section 4.3.1 for further discussion of this and experimental results.

The critical point in assessing the complexity of Algorithm 5 is therefore to estimate the number of points in C, which we call *centers*, under the hypothesis above. This is done in the following lemma.

Lemma 4.1. *Let $n \in \mathbb{N}$ and $2/3 < \gamma < 1$. Define $c_{\mathcal{H}} = 1/(\gamma\sqrt{1 - \gamma^2/4})$ and $N_C = c_{\mathcal{H}}^n \lceil 3\sqrt{2\pi}(n+1)^{3/2} \rceil$. Let N be an integer, and S a subset of $C_n(\gamma, R)$ of cardinality N whose points are picked independently at random with uniform distribution.*

(1) If $N_C < N < 2^n$, then for any subset $C \subseteq S$ of size at least N_C whose points are picked independently at random with uniform distribution, with overwhelming probability, for all $\mathbf{v} \in S$, there exists a $\mathbf{c} \in C$ such that $\|\mathbf{v} - \mathbf{c}\| \leq \gamma$.

(2) If $N < 4\sqrt{\pi/2n}\sqrt{4/3}^n$, the expected number of points in S that are at distance at least γ from all the other points in S is at least $(1 - 1/n)N$.

Before proceeding to the proof of Lemma 4.1, les us show how it allows us to estimate the complexity of the heuristic sieve algorithm. The lemma shows that, at steps 7–11 in Algorithm 1, we expect the size of S to decrease by rougly $c_{\mathcal{H}}^n$, provided that the vectors in S are well distributed in $C_n(\gamma, R)$. As long as this distribution property holds, the norms of the vectors in S are reduced by a factor γ at each iteration. Since the initial norm of the vectors is $2^{O(n)}\lambda_1$, if the number of sampled vectors is $poly(n) \cdot c_{\mathcal{H}}^n$, then after a linear number of iterations we expect to be left with a large number of very small vectors. Since the running time of the sieve is quadratic, the total running time of the algorithm is expected to be of order $(4/3 + \epsilon)^n$, because at the limit $\gamma \to 1$ we have $c_{\mathcal{H}} = \sqrt{4/3}$. Note that Lemma 4.1 also shows that this estimate is tight. Indeed, if the number of points sampled is less than $\Omega(n^{-1/2}\sqrt{4/3}^n)$, we expect each iteration of the sieve to use at least a fraction $1 - 1/n$ of the available points as centers: this means that the set of lattice vectors will become depleted after a sub-linear number of iterations, which will be insufficient to give a constant approximation to the shortest vector.

In Section 4.3, we will present experiments which support those estimates up to dimension 48.

The $\sqrt{4/3}$ constant should be interpreted in the following way. Let an observer be situated at the origin in n-dimensional space. The sky is modeled as the sphere of radius 1 centered at the origin. A giant star, modeled by a ball of radius 1, is centered at some point in the sky. Then the fraction of the sky that is occupied by the star is $\sin^n \pi/3 = \sqrt{3/4}^n$, and about $\sqrt{4/3}^n$ stars are therefore required to cover the whole sky.

The rest of this section is devoted to the proof of Lemma 4.1. Since it is clearly independent of the scaling factor R, to simplify notations we restrict ourselves to $R = 1$ and let $C_n(\gamma) = C_n(\gamma, 1) = \{\mathbf{x} \in \mathbb{R}^n,\ \gamma \leq \|\mathbf{x}\| \leq 1\}$. Our problem is the following: let S be a set of N uniformly distributed points in $C_n(\gamma)$. What is the expected fraction of $C_n(\gamma)$ covered by $\bigcup_{\mathbf{x}\in S} B_n(\mathbf{x}, \gamma)$? This is related to problems of sphere coverings of convex sets, which have been well studied in the literature (for an overview see [8, Chap. 2]). However, our problem is a bit different from the usual *thinnest* covering problem: what is the minimal number of n-dimensional balls of radius γ required to cover the n-dimensional unit ball? In fact, very little is known about this thinnest covering when the dimension is large, and the only bounds known come from the study of random lattices (see for example [30]).

Since the expected size of C clearly decreases as γ goes to 1, we will take γ rather close to 1, so that $C_n(\gamma)$ is geometrically close to a sphere; using this intuition we can rigorously prove a lower bound on the expected fraction of $C_n(\gamma)$ covered by N uniformly distributed balls of radius γ.

Let $\Omega_n(\gamma)$ be the minimum fraction of $C_n(\gamma)$ that is covered by a ball of radius γ centered in a point of $C_n(\gamma)$. Lemma 4.2 below estimates $\Omega_n(\gamma)$ by following the approach taken in [7], approximating $B_n(\mathbf{x}, \gamma) \cap C_n(\gamma)$ by the spherical cap of angle φ: $S_n(\mathbf{x}, \varphi) = \{\mathbf{y} \in C_n(\gamma),\ |\langle \mathbf{y}, \mathbf{x}\rangle| \leq \|\mathbf{x}\|\|\mathbf{y}\| \cos\varphi\}$. This lemma is then used to give a proof of Lemma 4.1.

Lemma 4.2. *If* $1/2 < \gamma < 1$,

$$\frac{1}{3\sqrt{2\pi(n+1)}}\frac{1}{\cos\varphi}\sin^n\varphi < \Omega_n(\gamma) < \sqrt{\frac{2}{\pi n}}\sqrt{\frac{3}{4}}^n$$

where $\varphi = \cos^{-1}\left(\frac{1}{2\gamma}\right)$ *if* $\gamma \leq \gamma_0 = \frac{1}{2}\left(\sqrt{5}-1\right)$ *and* $\varphi = \cos^{-1}\left(1-\frac{\gamma^2}{2}\right)$ *if* $\gamma \geq \gamma_0$.

Proof. Let $\mathbf{x} \in C_n(\gamma)$, and $\alpha = \|\mathbf{x}\|$, so $\gamma \leq \alpha \leq 1$. Then $S_n(\mathbf{x}, \varphi_\alpha) \subset B_n(\mathbf{x}, \gamma) \cap C_n(\gamma) \subset S_n(\mathbf{x}, 2\pi/3)$, where the angle φ_α is such that $\cos\varphi_\alpha = \frac{\alpha}{2\gamma}$ if $\gamma \leq \gamma_0$ and $\cos\varphi_\alpha = \frac{1+\alpha^2-\gamma^2}{2\alpha}$ if $\gamma > \gamma_0$. To see this, consider the triangle of vertices $\mathbf{O}$, $\mathbf{x}$, and $\mathbf{y}$ such that $\mathbf{y}$ is at distance γ from $\mathbf{x}$ and $\beta \in [\gamma, 1]$ from $\mathbf{O}$. φ_α is the minimal angle in $\mathbf{O}$ over all such triangles for $\beta \in [\gamma, 1]$, whereas the maximum of this same angle is easily seen to be less then $2\pi/3$. The minimal angle is obtained for $\alpha = 1$, and by the estimates of [7], Corollary 3.2, we have that

$$\frac{1}{3\sqrt{2\pi(n+1)}}\frac{1}{\cos\varphi_\alpha}\sin^n\varphi_\alpha < \frac{\mathrm{vol}\left(S(\mathbf{x},\varphi_\alpha)\right)}{\mathrm{vol}\left(C_n(\gamma)\right)} < \frac{1}{\sqrt{2\pi n}}\frac{1}{\cos\varphi_\alpha}\sin^n\varphi_\alpha$$

which concludes the proof of the lemma. □

Proof of Lemma 4.1

If $c_{\mathcal{H}}$ is as described, then by Lemma 4.2 we have $\Omega_n(\gamma) > \frac{1}{3\sqrt{2\pi(n+1)}}c_{\mathcal{H}}^{-n}$. The expected fraction of $C_n(\gamma)$ that is not covered by N_C balls of radius γ centered at randomly chosen points of $C_n(\gamma)$ is $(1-\Omega_n(\gamma))^{N_C}$. We have

$$\begin{aligned} N_C\log(1-\Omega(\gamma)) &\leq -N_C\Omega(\gamma) \\ &\leq -(n+1)3\sqrt{2\pi(n+1)}\frac{1}{3\sqrt{2\pi(n+1)}} \\ &\leq -\log N. \end{aligned}$$

The expected fraction of the corona covered by N_C points chosen at random is then at least $1-2^{-N}$, so the expected number of uncovered points is smaller than 1. Since this number is an integer, it must be 0 with probability at least $1/2$.

For the second part of the lemma, let S be a set of N points picked uniformly at random in the corona, and, for $i \in S$, X_i the random variable that is 1 if there is no point in S (other than i) that is at distance less than γ of i, and 0 otherwise. If $\alpha = \sqrt{\frac{2}{\pi n}}\sqrt{\frac{3}{4}}^n$, then by Lemma 4.2 we have $\mathrm{Prob}[X_i = 1] \geq (1-\alpha)^{N-1}$. $X = \sum_{i\in S} X_i$ is the number of isolated points in S, by linearity of expectation $\mathrm{E}[X] \geq N(1-\alpha)^{N-1} \geq (1-1/n)N$ as long as $\alpha N < n/4$. □

4.2 Optimizations

We discuss practical aspects for the sampling and sieve steps.

4.2.1 Sampling

Using Algorithm 2 with a reasonable choice of $\xi = O(\lambda_1(L))$, we can obtain many lattice vectors of norm $2^{O(n)}\lambda_1(L)$. However, the distribution of the lattice vectors that are output is not clear, whereas our heuristic assumption assumes that they are uniformly distributed in the corona $C_n(\gamma)$. Intuitively it is clear that the sampled points should not be biased towards any single direction. Moreover, the number of sieve executions is linear in n, where the linear coefficient is directly related to the $O()$ constant in $2^{O(n)}\lambda_1(L)$, so it is important to have a reasonable constant. Algorithm 2 used Babai's rounding algorithm [6], but Babai's nearest plane algorithm [6] would yield a better constant. To minimize theoretical issues with the heuristic assumption, we used Klein's randomized variant of the nearest plane algorithm [22], because it has good theoretical guarantees and is very efficient in practice. Klein's algorithm samples lattice vectors according to a distribution that is statistically close to a gaussian with arbitrary (although not too small) variance. The gaussian ensures that no direction in space is privileged, and we can choose the variance to be sufficiently small so that with high probability the sampled vectors are reasonably small. More precisely, a detailed analysis of Klein's algorithm appeared in [15], where it was used to construct various cryptographic tools with trapdoors. Gentry *et al.* [15] proved the following theorem:

Theorem 4.3. *[22, 15] Let $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ be any basis of a lattice L, and $s \geq \max_i \|\mathbf{b}_i\| \cdot w(\sqrt{\log n})$. There exists a randomized polynomial-time algorithm $\mathcal{K}$ whose output distribution is within negligible statistical distance of the restriction to L of a gaussian centered at $\mathbf{0}$ and with variance $\sqrt{2\pi}s$, i.e. with density proportional to $p(\mathbf{v}) = \exp(-\pi \|\mathbf{v}\|^2 / s^2)$.*

4.2.2 Alternative sieves

The lattice sieve (Algorithm 5) reduces the maximal norm of its input set of vectors S by selecting a subset C of S of centers which will ultimately be discarded, so that $|C|$ vectors are lost at each iteration of the sieve. Algorithm 5 naively solves the following problem: given a vector $\mathbf{v} \in C_n(\gamma)$ and a set $C \subset C_n(\gamma)$, find a vector $\mathbf{c} \in C$ such that $\|\mathbf{v} - \mathbf{c}\| \leq \gamma$ if such a vector exists. This is a nearest-neighbour problem, which has been well studied in the literature. It is known that in practice, when the dimension is high, unless one accepts to be faced with huge space requirements it is impossible to do much better than the trivial $O(n|C|)$ algorithm which computes each of the $|C|$ norms $\{\|\mathbf{v} - \mathbf{c}\|,\ c \in C\}$. To get better results, researchers have recently been considering the following *approximate nearest-neighbour* problem: Let $c > 1$ and $R > 0$. If there exists $\mathbf{c} \in C$ such that $\|\mathbf{v} - \mathbf{c}\| \leq R$, return any point $\mathbf{c}'$ such that $\|\mathbf{v} - \mathbf{c}'\| \leq cR$.

Currently the best algorithms for solving these problems in practice are the hashing-based algorithms of [9] and [5], which complexities of $O(n|C|^{1/c})$ and $O(n|C|^{1/c^2})$ respectively. In our case, a parameter set would be $R = \gamma \simeq 0.97$ and $c \in (1, 1/0.97)$. In theory, any fixed c only influences the number of iterations that we will have to make, but not the number of centers used at each iteration, so we could take $c \simeq 1/0.97 = 1.031$. Unfortunately, such an improvement is too small to outweigh the increase in the size of the constants implied by the $O(\cdot)$ for dimensions of practical

interest. We implemented the first algorithm of [9], and found that it performed only marginally better than the naive algorithm for dimensions higher than 50. The main advantage of the naive version is that it only uses very simple operations: additions and multiplications of integers.

4.3 Experimental results

Since our analysis of Algorithm 4 is based on heuristic assumptions, it is very important to perform experiments to see if they are justified or not. The crucial point in assessing the complexity of the algorithm is to estimate the number of vectors that are lost at each iteration of the sieve, either as centers or through a collision.

Experiments were performed on a 3.2GHz-P4 computer with 1GB of RAM. The heuristic AKS algorithm was implemented in C, using Victor Shoup's NTL library [34]. To give an idea of how practical AKS is, let us first give figures for the Leech lattice, which is the densest 24-dimensional lattice, and which has 196,560 minimal vectors. We started with an LLL-reduced basis with bad reduction factor $\delta = 0.5$, which had basis vectors roughly twice as long as shortest vectors. After sampling $8,000$ vectors (at best 1.9 times longer than the first minimum), 50 iterations of the sieve with factor $\gamma = 0.97$ produced a shortest lattice vector. The total running time was half a second, and the maximal number of centers used per sieve iteration was about 400.

To give a broader picture, we performed twenty experiments on random lattices for each dimension n such that $30 \leq n \leq 48$:

- Pick a random lattice using the simple method described in [16] (see [25] for details), and compute an LLL-reduced basis.
- Repeatedly run Algorithm 4 with $\gamma = 0.97$ with an increasing number of points N, until the output vector $\mathbf{v}_0$ is a shortest vector of the lattice (this is checked using the Schnorr–Euchner enumeration procedure). Once Algorithm 4 is successful, we store:
  (1) The maximal number of centers which were used at any iteration of the sieve. See Figure 1 for the average value of this maximal number.
  (2) The evolution of the number of points sieved during consecutive iterations. See Figures 2 and 3.
  (3) The total running time of the algorithm. See Figure 4 for average values, where it is compared with the running time of other SVP algorithms.

4.3.1 Evolution of the number of sieved points

The heuristic version of the AKS sieve algorithm keeps iterating until its current set S of lattice vectors becomes empty; it then returns the last non-zero lattice vector that it has manipulated. At each iteration, the maximal norm of a vector in S is multiplied by a constant factor $\gamma < 1$, at the cost of a reduction in the size of S. As we saw in

Section 4.1, the crucial point in assessing the quality of this algorithm is therefore to have good estimates on the evolution of the size of S. The size can decrease in two ways: first, a vector may be lost when it is used as a center, and second, a vector is discarded if it is reduced to zero by the sieve; this only happens when there were two identical vectors in S, and one was used as a center for the other.

The crucial assumption that we made in order to bound the number of vectors lost in this fashion was that the vectors of S were uniformly distributed. We saw in Section 4.2.1 that there existed efficient sampling procedures which provided minimal guarantees on the distribution of the initially sampled points. However, it is not clear whether each iteration of the sieve preserves this uniformity, or if it introduces a small bias that could accumulate over a large number of iterations.

In this section, we give experimental results which support our theoretical estimates.

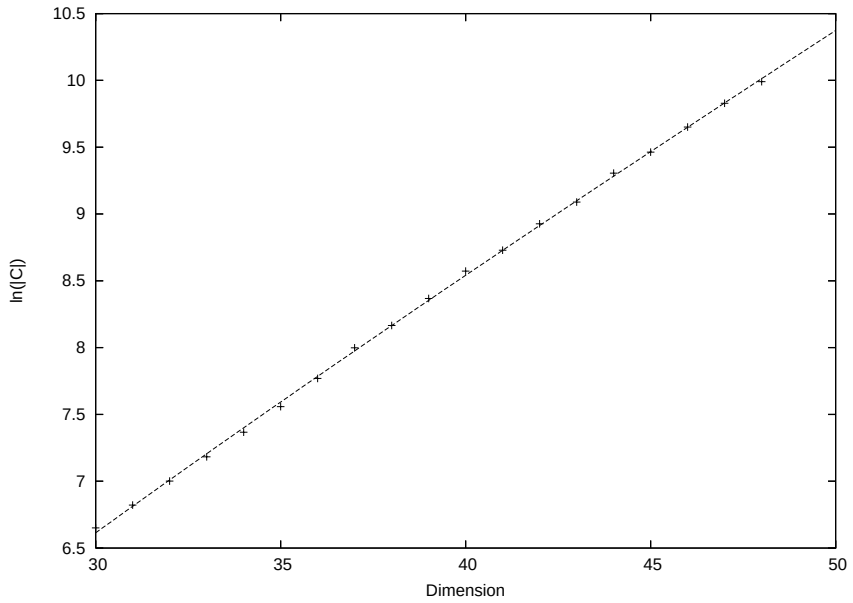

Figure 1. Logarithm of the maximal number of centers used in any iteration of the sieve algorithm, experimentally.

Number of centers used at each sieve iteration.

Lemma 4.1 upper bounded the number of centers which were expected to be used by the sieve by $N_C = c_{\mathcal{H}}^n \cdot \lceil 3\sqrt{2\pi}(n+1)^{3/2} \rceil$ where $\ln c_{\mathcal{H}} \approx 0.165$ if $\gamma = 0.97$. Figure 1 gives the natural logarithm of the maximal number of centers used in any iteration of the sieve. A numerical interpolation of the graph with a curve of equation $f(x) = ax + b\ln(x) + c$ yields the following values: $a = 0.163(\pm 0.017)$, $b = 0.102(\pm 0.65)$, $c = -1.73(\pm 1.72)$. Note that, for our range of n, $b \ln n$ is not negligible at all compared

to an. Since 0.163 is very close to 0.165, the experimental value of a is rather close to the theoretical prediction. But the exponent of n in the estimate of N_C is better than expected: it is $e^{0.102} \approx 1.1$ rather than 1.5. This is perhaps due to the fact that the covering of $B_n(R)$ used by the lattice sieve is more efficient than the random covering used by Lemma 4.1, since a point is added only if it is outside the union of the balls $B_n(\mathbf{c}, R)$ for all $\mathbf{c} \in C$.

Number of collisions.

If we start the sieve with a large number of vectors in S, then there is bound to be a non-negligible number of collisions after a number of iterations: since a definite number of vectors are used as centers at each iteration, as $R \to \lambda_1$ the size of S decreases in an arithmetic fashion, but should be no more than 1 (or, say, $2n$) when $R = \lambda_1$.

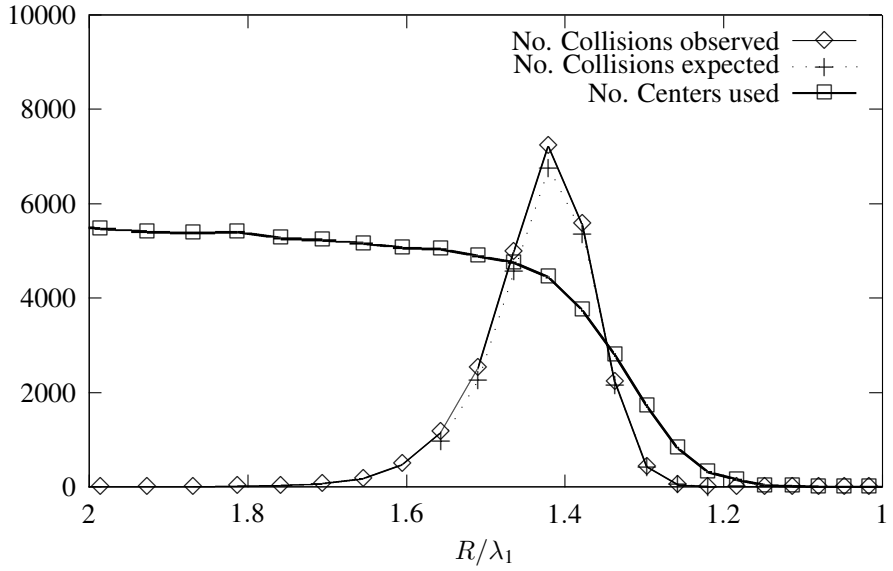

Figure 2. Cardinality of S, the set of vectors that are being sieved.

If we assume, as previously, that at every step of the sieving procedure, the set of vectors currently being sieved is uniformly distributed among the lattice vectors in the corona, then it is possible to give a precise estimate of the expected number of collisions:

Lemma 4.4. *Assume p vectors are randomly chosen with replacement from a set of cardinality N. Then the expected number of different vectors picked is $N - N(1 - 1/N)^p$, so the expected number of vectors lost through collisions is $p - N + N(1 - 1/N)^p$.*

This number is negligible for $p \ll \sqrt{N}$. Since we expect the number of lattice vectors inside the ball of radius R/λ_1 to be of order R^n, the effect of collisions will only become noticeable when R/λ_1 is less than $|S|^{2/n}$. As we saw in Section 4, it is sufficient to take an initial S of size $|S| \approx \sqrt{4/3}^n$, which gives $R/\lambda_1 \approx 4/3$. So we expect collisions to play a role only when we already have a very good approximation to λ_1 - and even then, collisions mean that we have a good proportion of all lattice vectors, meaning we will probably be able to recover the shortest one.

To confront these estimates with experimental results, we picked a random lattice of dimension 35, sampled a large number of lattice vectors and then sieved them. We first plotted the total number of vectors that are manipulated by the sieve at each iteration, as a function of the quality of these vectors as measured by their maximum norm R divided by λ_1 (Figure 2). The first portion of the curve decreases regularly as a constant number of points are used as centers at each iteration. When the size of the manipulated vectors becomes approximately $1.5\lambda_1$, we see a sharp decline in the number of vectors, which is due to a large number of collisions, as can be verified on Figure 3.

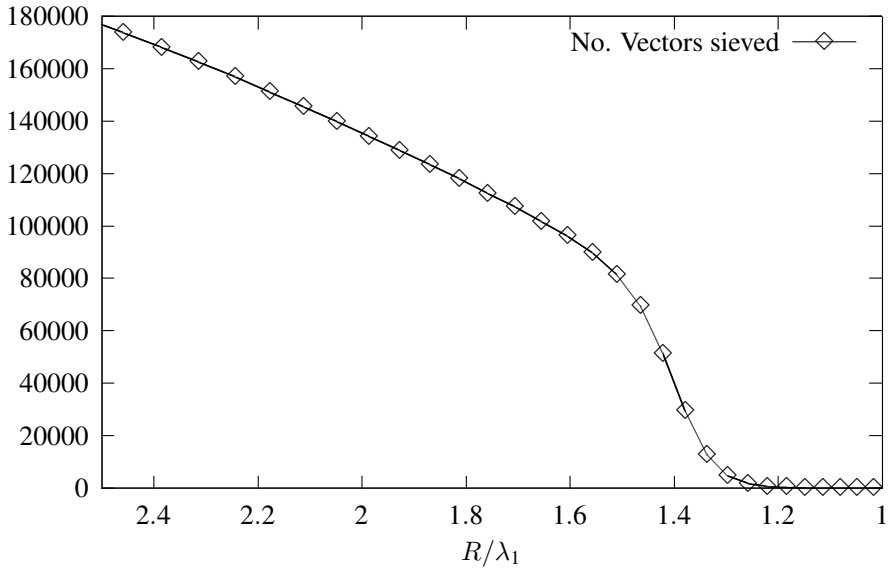

Figure 3. Number of collisions observed.

In Figure 3 we plot the number of centers used at each iteration and the number of collisions that are observed. We also plotted the expected number of collisions in case these points were uniformly distributed, by computing the exact number of lattice points inside a given ball, using a simple enumeration algorithm, and, given the number of vectors in S, computing the expected number of collisions using Lemma 4.4. We see that the two curves match rather well: as far as collisions are concerned, it seems that the set of vectors manipulated by the sieve behaves as if it were sampled uniformly

from all the vectors in $B_n(R)$. We also see on this figure that the number of centers used in any iteration is remarkably steady, collapsing only when the radius becomes very close to λ_1. These observations suggest that the distribution of the vectors that are manipulated by the sieve remains very well behaved throughout its numerous iterations.

4.3.2 Comparison with other SVP algorithms

In this section we compare the performance of the heuristic AKS Algorithm with the other two main enumeration algorithms, which we first describe briefly. The basic idea for these algorithms is to find a bounded set such that a shortest vector is guaranteed to be in this set, and then enumerate all vectors in the set. There is a tradeoff between the time needed for enumeration, governed by the size of the set, and the preprocessing time, used to reduce the size of the set that is to be enumerated.

Let L be a lattice with basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$. Let $\mathbf{b}_i^*$ be the Gram–Schmidt vectors, and $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}$ the Gram– Schmidt coefficients.

ENUMERATION TECHNIQUES. If $\mathbf{x} = x_1\mathbf{b}_1 + \ldots + x_n\mathbf{b_n}$ is any lattice vector, then

$$\|\mathbf{x}\|^2 = \sum_{j=1}^{n} \left(x_j + \sum_{i=j+1}^{n} \mu_{i,j} x_i \right) \|\mathbf{b}_j^*\|^2.$$

If $\mathbf{x}$ is further a shortest vector, then $\|x\| \leq \lambda_1 \leq \|\mathbf{b}_1\|$, hence $x_n \leq \|\mathbf{b}_1\|/\|\mathbf{b}_n^*\|$, and we can take $x_n \geq 0$. It is then possible to recursively find intervals containing the other coordinates $x_{n-1}, \ldots, x_1$, and it is easy to see that the search space can be circumscribed to a set of cardinality no more than

$$\frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_n^*\|} \prod_{j=2}^{n-1} \left(\left\lfloor \frac{2\|\mathbf{b}_1\|}{\|\mathbf{b}_j^*\|} \right\rfloor + 1 \right).$$

If the basis is LLL reduced with optimal reduction factor, then the enumeration procedure permforms less than $\sqrt{4/3}^{n^2/2+O(n)}$ polynomial-time operations. Here, the $\sqrt{4/3}$ constant is Hermite's constant in dimension two and corresponds to the worst-case of LLL reduction: extensive experiments [25] suggest that in practice, $\sqrt{4/3}$ can be replaced by ≈ 1.04, so that the asymptotic complexity is expected to be $1.02^{n^2+O(n)}$ polynomial-time operations. In other words, we have a super-exponential complexity with a constant extremely close to 1.

We have just described the principle of enumeration techniques. However, it must be stressed that there are several ways to enumerate all the candidates $\mathbf{x}$, and this has a great impact on the running time in practice. The Schnorr–Euchner enumeration [33] is the most efficient way known of enumerating all the candidates: it is implemented by all main lattice software. It is based on the following two ideas (see the survey [1] for more details):

- each time a shorter vector is found, its norm, rather than $\|\mathbf{b}_1\|$, is used to bound the coordinates x_i.

- each time we find an interval containing the coordinate x_i for $i < n$, we search through this interval starting from the middle, rather than from the boundaries.

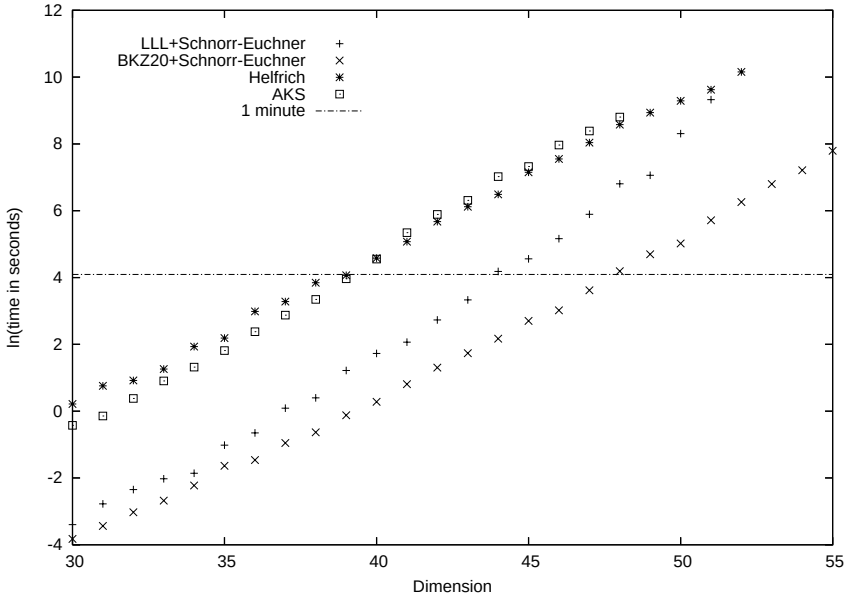

Figure 4. Comparison of the logarithms of the running times of the main SVP algorithms

KANNAN–HELFRICH ENUMERATION [21, 20]. The idea behind Kannan's algorithm is to spend much more time finding a good basis so as to decrease the cost of enumeration. Let π_2 be the orthogonal projection in the direction of $\mathbf{b}_1$. We first find an HKZ-reduced basis of the lattice $\pi_2(L)$, which is lifted to a weakly-reduced basis $[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$ of L such that $[\pi_2(\mathbf{b}_2), \ldots, \pi_2(\mathbf{b}_n)]$ is an HKZ-reduced basis of $\pi_2(L)$ and $\|\mathbf{b}_1^*\| \leq 2\|\mathbf{b}_2^*\|$. This is achieved through recursive calls to the algorithm. One finally finds the shortest vector by basic enumeration, like in the Schnorr–Euchner algorithm [33]. It can be shown that the overall asymptotic cost of the algorithm is dominated by the last enumeration: since $\mathbf{b}_2^*$ is a shortest vector of $\pi_2(L)$, we have that $\|\mathbf{b}_2^*\|^{n-1} \leq O(\sqrt{n-1})\|\mathbf{b}_2^*\| \cdots \|\mathbf{b}_n^*\|$, which implies that the cost of the enumeration is at most $n^{n/2+o(n)}$ polynomial-time operations. Hanrot and Stehlé [18] recently presented an improved analysis, which shows that the cost of the enumeration is at most $n^{n/(2e)+o(n)}$ polynomial-time operations; and they later showed in [19] that this upper bound is also a lower bound for certain bases of certain lattices.

The experimental running times of all three algorithms are compared in Figure 4. To emphasize the importance of the quality of the input basis, we used the Schnorr–Euchner enumeration technique [33] in two different cases:

- The input basis is LLL reduced. Figure 4 shows that the logarithm of the running time is convex and grows like $O(n^2)$.
- The input basis is BKZ-20 reduced: the running time is still $2^{O(n^2)}$ but with an even smaller $O()$ constant. This is because experimentally, the only major difference between a BKZ-20 basis and an LLL basis is the slope of the $\log ||\mathbf{b}_i^*||$.

Since $2^{O(n)} \ll 2^{O(n \log n)} \ll 2^{O(n^2)}$, the asymptotic analysis suggests that the AKS algorithm should be faster than Kannan's algorithm, which itself should be faster than the basic Schnorr–Euchner enumeration algorithm. However, our experiments show that this is not the case in the dimension range 20–50: Figure 4 shows that in this range the heuristic AKS has running time comparable to the Kannan–Helfrich algorithm, but both are significantly slower than Schnorr–Euchner enumeration with LLL preprocessing, and the gap with BKZ-20 is of course bigger.

This can be explained as follows:

- It is in fact no surprise that the basic Schnorr–Euchner enumeration is the fastest. Indeed, its complexity is $2^{O(n^2)}$ polynomial-time operations, but here, the complexity $2^{O(n^2)}$ should in practice be replaced by $1.02^{n^2+O(n)}$ for reasons explained above. Moreover, the polynomial-time unit is extremely small in practice, because it consists only of a few floating-point arithmetic operations.
- On the other hand, the complexity of the Kannan–Helfrich algorithm is $2^{O(n \log n)}$ polynomial-time operations, which could be bigger than $1.02^{n^2+O(n)}$ for small n, and where the polynomial-time unit is not as small, due to large preprocessing. Indeed, in order to make the basis quasi-HKZ-reduced, the algorithm must perform a great number of projections and lifting of basis vectors: hence, the polynomial-time unit is much larger than a few floating- point arithmetic operations like in the basic Schnorr–Euchner enumeration.
- Finally, the heuristic AKS only involves very basic floating- point operations, but the $(4/3 + \varepsilon)^n$ complexity actually hides polynomial factors. First of all, the estimate N_C given by Lemma 4.1 is roughly $n^{3/2}(\sqrt{4/3} + \varepsilon)^n$, which already implies a polynomial factor n^3 in the running time (since the cost of the sieve is quadratic). Besides, we have to run the sieve a linear number of times, which hides another polynomial factor. And polynomial factors may not be negligible at all compared to exponential factors in low dimensions: $n^{3/2} \leq (\sqrt{4/3} + \varepsilon)^n$ for $n \leq 38$.

These results show that, from a practical point of view, the exponential factors are surprisingly less important than the polynomial factors, because the dimensions of interest are rather small.

Let us finally comment briefly on space requirements: both the Schnorr–Euchner and the Kannan–Helfrich enumeration techniques have essentially no memory requirement except that of storing $O(1)$ n-dimensional lattice vectors. In contrast, the fast sieve algorithm requires about $O(poly \cdot \sqrt{4/3}^n)$ space and, like for the running time,

it is the polynomial factors which dominate. In practice, memory usage becomes significant when n is around 50: in our experiments, the 48-dimensional examples used about 700 megabytes of RAM, where vectors were stored using the C++ long type.

5 Conclusion

We have shown that sieve algorithms for the shortest vector problem are in fact practical. We have also provided an analysis which is supported by experiments: our heuristic sieve algorithm behaves essentially as predicted, which is rather unusual among lattice algorithms. More precisely, at each application of the sieve, we know approximately how many centers will be used and how much shorter the vectors will become. Our results show that while sieve algorithms can be made practical, they are unlikely to beat enumeration algorithms for dimensions of practical interest. Furthermore, beyond dimension 50, the space requirement of sieve algorithms becomes problematic.

References

[1] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, *Closest point search in lattices*, IEEE Trans. Inform. Theory 48 (2002), pp. 2201–2214.

[2] M. Ajtai, *Generating Random Lattices according to the Invariant Distribution*, Draft of March 2006.

[3] M. Ajtai, *The Shortest Vector Problem in L_2 is NP-hard for Randomized Reductions*. Proc. of 30th STOC. ACM, 1998, Available at [10] as TR97-047.

[4] M. Ajtai, R. Kumar, and D. Sivakumar, *A sieve algorithm for the shortest lattice vector problem*. Proc. 33rd STOC, pp. 601–610, ACM, 2001.

[5] A. Andoni and P. Indyk, *Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions*. FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), pp. 459–468. IEEE Computer Society, Washington, DC, USA, 2006.

[6] L. Babai, *On Lovász lattice reduction and the nearest lattice point problem*, Combinatorica 6 (1986), pp. 1–13.

[7] K. Böröczky and G. Wintsche, *Covering the sphere by equal spherical balls*. Discrete and Computational Geometry, The Goodman-Pollack Festschrift, pp. 237–253, 2003.

[8] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*. Springer-Verlag, 1998, Third edition.

[9] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, *Locality-sensitive hashing scheme based on p-stable distributions*. SCG '04: Proceedings of the twentieth annual symposium on Computational geometry, pp. 253–262. ACM Press, New York, NY, USA, 2004.

[10] ECCC, http://www.eccc.uni-trier.de/eccc/, The Electronic Colloquium on Computational Complexity.

[11] U. Fincke and M. Pohst, *Improved methods for calculating vectors of short length in a lattice, including a complexity analysis*, Math. Comp. 44 (1985), pp. 463–471.

[12] N. Gama, N. Howgrave-Graham, H. Koy, and P. Q. Nguyen, *Rankin's Constant and Blockwise Lattice Reduction*. Proc. CRYPTO '06, Lecture Notes in Computer Science 4117, pp. 112–130. Springer, 2006.

[13] N. Gama and P. Q. Nguyen, *Finding Short Lattice Vectors within Mordell's Inequality*. STOC '08 – Proc. 40th ACM Symposium on the Theory of Computing. ACM, 2008.

[14] ———, *Predicting Lattice Reduction*. Advances in Cryptology – Proc. EUROCRYPT '08, Lecture Notes in Computer Science. Springer, 2008.

[15] C. Gentry, C. Peikert, and V. Vaikuntanathan, *Trapdoors for Hard Lattices and New Cryptographic Constructions*, Cryptology ePrint Archive, Report 2007/432, 2007, To appear in STOC '08.

[16] D. Goldstein and A. Mayer, *On the equidistribution of Hecke points*, Forum Math. 15 (2003), pp. 165–189.

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

[18] G. Hanrot and D. Stehlé, *Improved Analysis of Kannan's Shortest Lattice Vector Algorithm*. Advances in Cryptology - Proc. CRYPTO 2007, Lecture Notes in Computer Science 4622, pp. 170–186. Springer, 2007.

[19] ———, *Worst-Case Hermite-Korkine-Zolotarev Reduced Lattice Bases*, CoRR abs/0801.3331 (2008).

[20] B. Helfrich, *Algorithms to construct Minkowski reduced and Hermite reduced bases*, Theoretical Computer Science 41 (1985), pp. 125–139.

[21] R. Kannan, *Improved algorithms for integer programming and related lattice problems*. Proc. of 15th STOC, pp. 193–206. ACM, 1983.

[22] P. Klein, *Finding the Closest Lattice Vector when It's Unusually Close*. Proc. of SODA '00, ACM–SIAM, 2000.

[23] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Mathematische Ann. 261 (1982), pp. 513–534.

[24] D. Micciancio and S. Goldwasser, *Complexity of lattice problems*, The Kluwer International Series in Engineering and Computer Science, 671. Kluwer Academic Publishers, Boston, MA, 2002, A cryptographic perspective. MR MR2042139 (2004m:94067)

[25] P. Q. Nguyen and D. Stehlé, *LLL on the Average*. Proc. of the 7th International Algorithmic Number Theory Symposium, (ANTS-VII), LNCS 4076, pp. 238–256. Springer-Verlag, 2006.

[26] P. Q. Nguyen and J. Stern, *The Two Faces of Lattices in Cryptology*. Proc. of CALC '01, LNCS 2146. Springer-Verlag, 2001.

[27] C. Peikert and B. Waters, *Lossy Trapdoor Functions and Their Applications*, Cryptology ePrint Archive, Report, 2007. Report 2007/279. To appear in STOC '08.

[28] M. Pohst, *On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications*, ACM SIGSAM Bulletin 15 (1981), pp. 37–44.

[29] O. Regev, *Lecture notes on lattices in computer science*, 2004.

[30] C. A. Rogers, *Lattice coverings of space*, Mathematika 6 (1959), pp. 33–39.

[31] C.-P. Schnorr, *A hierarchy of polynomial lattice basis reduction algorithms*, Theoretical Computer Science 53 (1987), pp. 201–224.

[32] ———, *Lattice reduction by random sampling and birthday methods*, Proc. STACS 2003, Lecture Notes in Comput. Sci. 2607, Springer, Berlin, 2003, pp. 145–156.

[33] C.-P. Schnorr and M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Math. Programming 66 (1994), pp. 181–199.

[34] V. Shoup, *Number Theory C++ Library (NTL) version 5.4*, Available at http://www.shoup.net/ntl/.

Received 4 October, 2007

Author information

Phong Q. Nguyen, ENS, France.
Email: pnguyen@di.ens.fr

Thomas Vidick, Berkeley, USA.
Email: vidick@berkeley.edu