Electronic Colloquium on Computational Complexity, Report No. 65 (2009)

Faster exponential time algorithms for the shortest vector problem*

Daniele Micciancio Panagiotis Voulgaris

University of California, San Diego
{daniele,pvoulgar}@cs.ucsd.edu

Abstract

We present new faster algorithms for the exact solution of the shortest vector problem in arbitrary lattices. Our main result shows that the shortest vector in any n-dimensional lattice can be found in time $2^{3.199n}$ and space $2^{1.325n}$. This improves the best previously known algorithm by Ajtai, Kumar and Sivakumar [Proceedings of STOC 2001] which was shown by Nguyen and Vidick [J. Math. Crypto. 2(2):181–207] to run in time $2^{5.9n}$ and space $2^{2.95n}$. We also present a practical variant of our algorithm which provably uses an amount of space proportional to τ_n, the "kissing" constant in dimension n. Based on the best currently known upper and lower bounds on the kissing constant, the space complexity of our second algorithm is provably bounded by $2^{0.41n}$, and it is likely to be at most $2^{0.21n}$ in practice. No upper bound on the running time of our second algorithm is currently known, but experimentally the algorithm seems to perform fairly well in practice, with running time $2^{0.48n}$, and space complexity $2^{0.18n}$.

Keywords: Algorithm Analysis, Cryptography, Shortest Vector Problem, Sieving algorithms, Software implementations

1 Introduction

The shortest vector problem (SVP) is the most famous and widely studied computational problem on point lattices. It is the core of many algorithmic applications (see survey papers [14, 6, 21]), and the problem underlying many cryptographic functions (e.g., [2, 3, 26, 27, 23, 9]). Still, our understanding of the complexity of this problem, and the best known algorithms to solve it, is quite poor. The asymptotically fastest known algorithm for SVP (namely, the AKS Sieve introduced by Ajtai, Kumar and Sivakumar in [4]) runs in probabilistic exponential time $2^{O(n)}$, where n is the dimension of the lattice. However, even the fastest known practical variant of this algorithm [22] is outperformed by the asymptotically inferior $2^{O(n^2)}$-time Schnorr-Euchner enumeration algorithm [30] at least up to dimension $n \approx 50$, at which point it becomes impractical. A similar situation exists in the context of approximation algorithms for SVP, where the BKZ (block Korkine-Zolotarev) algorithm of [30] (which is not even known to run in polynomial time) is preferred in practice to provable polynomial time approximation algorithms like [29, 7].

This discrepancy between asymptotically faster algorithms and algorithms that perform well in practice is especially unsatisfactory in the context of lattice based cryptography, where one needs

*Supported in part by NSF grant CCF-0634909. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

ISSN 1433-8092

to extrapolate the running time of the best known algorithms to ranges of parameters that are practically infeasible in order to determine appropriate key sizes for the cryptographic function.

In this paper we present and analyze new algorithms to find the shortest vector in arbitrary lattices that both improve the best previously known worst-case asymptotic complexity and also have the advantage of performing pretty well in practice, thereby reducing the gap between theoretical and practical algorithms. More specifically, we present

- a new probabilistic algorithm that provably finds the shortest vector in any n dimensional lattice (in the worst case, and with high probability) in time $\tilde{O}(2^{3.199n})$ and space $\tilde{O}(2^{1.325n})$, improving the $\tilde{O}(2^{5.9n})$-time and $\tilde{O}(2^{2.95n})$-space complexity bounds of the asymptotically best previously known algorithm [4, 22], and
- a practical variant of our theoretical algorithm that admits much better space bounds, and outperforms the best previous practical implementation [22] of the AKS sieve [4].

The space complexity of our second algorithm can be bounded by $\tilde{O}(\tau_n)$, where τ_n is the so called "kissing" constant in n-dimensional space, i.e., the maximum number of equal n-dimensional spheres that can be made to touch another sphere, without touching each other. The best currently known lower and upper bounds on the kissing constant are $2^{(0.2075+o(1))n} < \tau_n < 2^{(0.401+o(1))n}$ [5]. Based on these bounds we can conclude that the worst-case space complexity of our second algorithm is certainly bounded by $2^{0.402n}$. Moreover, in practice we should expect the space complexity to be at most $2^{0.21n}$, because finding family of lattices for which the algorithm uses more than $2^{0.21n}$ space would imply denser arrangements of hyperspheres than currently known, a long standing open problem in the study of spherical codes. So, input lattices for which our algorithm uses more than $2^{0.21n}$ space either do not exists (i.e., the worst-case space complexity is $2^{0.21n}$), or do not occur in practice because they are very hard to find. The practical experiments reported in Section B are consistent with our analysis, and suggest that the space complexity of our second algorithm is indeed bounded by $2^{0.21n}$. Unfortunately, we are unable to prove any upper bound on the running time of our second algorithm, but our experiments suggest that the algorithm runs in time $2^{0.48n}$.

The rest of the paper is organized as follows. In the following subsections we provide a more detailed description of previous work (Section 1.1) and an overview of our new algorithms (Section 1.2). In Section 2 we give some background about point lattices and the shortest vector problem. In Section 3 we describe our new algorithms and state theorems for their complexity. Finally, in Section 4 we discuss open problems. In the appendix we give the proof for the asymptotic bounds on time and space complexity in Section A, and we report experimental results in Section B.

1.1 Prior work

Algorithms for the exact solution of the shortest vector problem can be classified in two broad categories: enumeration algorithms, and sieving algorithms. *Enumeration algorithms*, given a lattice basis $\mathbf{B}$, systematically explore a region of space (centered around the origin) that is guaranteed to contain the shortest lattice vector. The running time of these algorithms is roughly proportional to the number of lattice points in that region, which, in turn depends on the quality of the input basis. Using LLL reduced basis [16], the Fincke-Pohst enumeration algorithm [24] finds the shortest lattice vector in time $\tilde{O}(2^{O(n^2)})$. Several variants of this algorithm have been proposed (see [1] for a survey,) including the Schnorr-Euchner enumeration method [30], currently used in state of the art practical lattice reduction implementations [31, 25]. Using a clever preprocessing method, Kannan [13] has given an improved enumeration algorithm that finds the shortest lattice vector in time $2^{O(n \log n)}$. This is the asymptotically best deterministic algorithm known to date, but does not perform well in practice due to the substantial overhead incurred during preprocessing (see [10] for further information about the theoretical and practical performance of Kannan's algorithm).

The AKS Sieve, introduced by Ajtai, Kumar and Sivakumar in [4], lowers the running time complexity of SVP to a simple exponential function $2^{O(n)}$ using randomization. We refer collectively to the algorithm of [4] and its variants as proposed in [22] and in this paper, as *sieve algorithms*. A major practical drawback of sieve algorithms (compared to the polynomial space deterministic enumeration methods) is that they require exponential space. A careful analysis of the AKS Sieve is given by Nguyen and Vidick in [22], building on ideas from [28]. Their analysis shows that the AKS Sieve runs in $\tilde{O}(2^{5.9n})$-time using $\tilde{O}(2^{2.95n})$ space. Nguyen and Vidick [22] also propose a practical variant of the AKS sieve, and demonstrate experimentally that the algorithm can be run in practice using reasonable computational resources, but it is not competitive with enumeration methods at least up to dimension 50, at which point the algorithm is already impractical.

1.2 Overview

In this paper we improve on [4, 22] both in theory and in practice, proposing new variants of the sieve algorithm with better exponential worst-case running time and space complexity bounds, and better practical performance. In order to describe the main idea behind our algorithms, we first recall how the sieve algorithm of [4] works. The algorithm starts by generating a large (exponential) number of random lattice points P within a large (but bounded) region of space C. Informally, the points P are passed through a sequence of finer and finer "sieves", that produce shorter and shorter vectors, while "wasting" some of the sieved vectors along the way. (The reader is referred to the original article [4] as well as the recent analysis [22] for a more detailed and technical description of the AKS sieve.)

While using many technical ideas from [4, 22], our algorithms depart from the general strategy of starting from a large pool P of (initially long) lattice vectors, and obtaining smaller and smaller sets of shorter vectors. Instead, our algorithms start from an initially empty list L of points, and increase the length of the list by appending new lattice points to it. In our first algorithm, the points in the list never change: we only keep adding new vectors to the list. Before a new point $\mathbf{v}$ is added to the list, we attempt to reduce the length of $\mathbf{v}$ as much as possible by subtracting the vectors already in the list from it. Reducing new lattice vectors against the vectors already in the list allows to prove a lower bound on the angle between any two list points of similar norm. This lower bound on the angle between list points allows us to apply the linear programming bound for spherical codes of Kabatiansky and Levenshtein [12] to prove that the list L cannot be too long. The upper bound on the list size then easily translates to corresponding upper bounds on the time and space complexity of the algorithm.

Similarly to previous work [4, 22], in order to prove that the algorithm produces non-zero vectors, we employ a now standard perturbation technique. Specifically, instead of generating a random lattice point $\mathbf{v}$ and reducing it against the vectors already in the list, we generate a perturbed lattice point $\mathbf{v} + \mathbf{e}$ (where $\mathbf{e}$ is a small error vector), and reduce $\mathbf{v} + \mathbf{e}$ instead. The norm of the error $\mathbf{e}$ is large enough, so that the lattice point $\mathbf{v}$ is not uniquely determined by $\mathbf{v} + \mathbf{e}$. This uncertainty about $\mathbf{v}$ allows to easily prove that after reduction against the list, the vector $\mathbf{v}$ is not too likely to be zero.

Our practical variant of the algorithm does the following. Beside reducing new lattice points $\mathbf{v}$ against the points already in the list L, the algorithm also reduces the points in L against $\mathbf{v}$, and against each other. As a result, the list L has the property that any pair of vectors in L forms a Gauss reduced basis. It follows from the properties of Gauss reduced bases that the angle between any two list points is at least $\pi/3$, and the list forms a good spherical code. In particular, the list length never exceeds the kissing constant τ_n, which is defined as the highest number of points that can be placed on a sphere, while keeping the minimal angle between any two points at least $\pi/3$.[1] As already discussed, this allows to bound the space complexity of our second algorithm by $2^{0.402n}$ in theory, or $2^{0.21n}$ in practice. Unfortunately, we are unable to bound the running time of this modified algorithm, as we don't know how to prove that it produces nonzero vectors. However, the algorithm seems to work very well in practice, and outperforms the best previously known variants/implementations of the AKS Sieve [22] both in theory (in terms of provable space bounds) and in practice (in terms of experimentally observed space and time complexity.)

2 Background

In this section we review standard definitions and notation used in the algorithmic study of lattices, mostly following [20].

General: For any real x, $\lfloor x \rfloor$ denotes the largest integer not greater than x. For a vector $\mathbf{x} = (x_1, \ldots, x_n)$ we define $\lfloor \mathbf{x} \rfloor$ as $(\lfloor x_1 \rfloor, \ldots, \lfloor x_n \rfloor)$. We write log for the logarithm to the base 2, and $\log_q$ when the base q is any number possibly different from 2. We use $\omega(f(n))$ to denote the set of functions growing faster than $c \cdot f(n)$ for any $c > 0$. A function $e(n)$ is *negligible* if $e(n) < 1/n^c$ for any $c > 0$ and all sufficiently large n. We write $f = \tilde{O}(g)$ when $f(n)$ is bounded by $g(n)$ up to polylogarithmic factors, i.e., $f(n) \leq \log^c g(n) \cdot g(n)$ for some constant c and all sufficiently large n.

The n-dimensional Euclidean space is denoted $\mathbb{R}^n$. We use bold lower case letters (e.g., $\mathbf{x}$) to denote vectors, and bold upper case letters (e.g., $\mathbf{M}$) to denote matrices. The ith coordinate of $\mathbf{x}$ is denoted x_i. For a set $S \subseteq \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^n$ and $a \in \mathbb{R}$, we let $S + \mathbf{x} = \{\mathbf{y} + \mathbf{x} \colon \mathbf{y} \in S\}$ denote the translate of S by $\mathbf{x}$, and $aS = \{a\mathbf{y} \colon \mathbf{y} \in S\}$ denote the scaling of S by a. The Euclidean norm (also known as the ℓ_2 norm) of a vector $\mathbf{x} \in \mathbb{R}^n$ is $\|\mathbf{x}\| = (\sum_i x_i^2)^{1/2}$, and the associated distance is $\mathrm{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$. We will use $\phi_{\mathbf{x},\mathbf{y}}$ to refer to the angle between the vectors $\mathbf{x}, \mathbf{y}$.

The distance function is extended to sets in the customary way: $\mathrm{dist}(\mathbf{x}, S) = \mathrm{dist}(S, \mathbf{x}) = \min_{\mathbf{y} \in S} \mathrm{dist}(\mathbf{x}, \mathbf{y})$. We often use matrix notation to denote sets of vectors. For example, matrix $\mathbf{S} \in \mathbb{R}^{n \times m}$ represents the set of n-dimensional vectors $\{\mathbf{s}_1, \ldots, \mathbf{s}_m\}$, where $\mathbf{s}_1, \ldots, \mathbf{s}_m$ are the columns of $\mathbf{S}$. We denote by $\|\mathbf{S}\|$ the maximum length of a vector in $\mathbf{S}$. The linear space spanned by a set of m vectors $\mathbf{S}$ is denoted $\mathrm{span}(\mathbf{S}) = \{\sum_i x_i \mathbf{s}_i \;:\; x_i \in \mathbb{R} \text{ for } 1 \leq i \leq m\}$. For any set of n linearly independent vectors $\mathbf{S}$, we define the half-open parallelepiped $\mathcal{P}(\mathbf{S}) = \{\sum_i x_i \mathbf{s}_i : 0 \leq x_i < 1 \text{ for } 1 \leq i \leq n\}$. Finally, we denote by $\mathcal{B}_n(\mathbf{x}, r)$ the closed Euclidean n-dimensional ball of radius r and center $\mathbf{x}$, $\mathcal{B}_n(\mathbf{x}, r) = \{\mathbf{w} \in \mathbb{R}^n : \|\mathbf{w} - \mathbf{x}\| \leq r\}$. If no center is specified, then the center is zero $\mathcal{B}_n(r) = \mathcal{B}_n(\mathbf{0}, r)$.

[1] The name "kissing" constant originates from the fact that $\pi/3$ is precisely the minimal angle between the centers of two nonintersecting equal spheres that touch (kiss) a third sphere of the same radius.

Lattices: We now describe some basic definitions related to lattices. For a more in-depth discussion, see [19]. An n-dimensional *lattice* is the set of all integer combinations

$$\Big\{\sum_{i=1}^{n} x_i\mathbf{b}_i \colon x_i \in \mathbb{Z} \text{ for } 1 \leq i \leq n\Big\}$$

of n linearly independent vectors $\mathbf{b}_1, \dots, \mathbf{b}_n$ in $\mathbb{R}^n$.[2] The set of vectors $\mathbf{b}_1, \dots, \mathbf{b}_n$ is called a *basis* for the lattice. A basis can be represented by the matrix $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_n] \in \mathbb{R}^{n\times n}$ having the basis vectors as columns. The lattice generated by $\mathbf{B}$ is denoted $\mathcal{L}(\mathbf{B})$. Notice that $\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} \colon \mathbf{x} \in \mathbb{Z}^n\}$, where $\mathbf{Bx}$ is the usual matrix-vector multiplication.

For any lattice basis $\mathbf{B}$ and point $\mathbf{x}$, there exists a unique vector $\mathbf{y} \in \mathcal{P}(\mathbf{B})$ such that $\mathbf{y} - \mathbf{x} \in \mathcal{L}(\mathbf{B})$. This vector is denoted $\mathbf{y} = \mathbf{x} \bmod \mathbf{B}$, and it can be computed in polynomial time given $\mathbf{B}$ and $\mathbf{x}$. A sub-lattice of $\mathcal{L}(\mathbf{B})$ is a lattice $\mathcal{L}(\mathbf{S})$ such that $\mathcal{L}(\mathbf{S}) \subseteq \mathcal{L}(\mathbf{B})$. The *determinant* of a lattice $\det(\mathcal{L}(\mathbf{B}))$ is the (n-dimensional) volume of the fundamental parallelepiped $\mathcal{P}(\mathbf{B})$ and is given by $|\det(\mathbf{B})|$.

The *minimum distance* of a lattice Λ, denoted $\lambda(\Lambda)$, is the minimum distance between any two distinct lattice points, and equals the length of the shortest nonzero lattice vector:

$$\lambda(\Lambda) = \min\{\text{dist}(\mathbf{x}, \mathbf{y}) : \mathbf{x} \neq \mathbf{y} \in \Lambda\} = \min\{\|\mathbf{x}\| : \mathbf{x} \in \Lambda \setminus \{\mathbf{0}\}\} \ . \tag{1}$$

We often abuse notation and write $\lambda(\mathbf{B})$ instead of $\lambda(\mathcal{L}(\mathbf{B}))$.

Definition 2.1 (Shortest Vector Problem) *An input to* SVP *is a lattice* $\mathbf{B}$*, and the goal is to find a lattice vector of length precisely* $\lambda(\mathbf{B})$.

Theorem 2.2 (Kabatiansky and Levenshtein [12]) *Let* $A(n, \phi_0)$ *be the maximal size of any set* C *of points in* $\mathbb{R}^n$ *such that the angle between any two distinct vectors* $\mathbf{v}_i, \mathbf{v}_j \in C$ *(denoted* $\phi_{\mathbf{v}_i,\mathbf{v}_j}$*) is at least* ϕ_0*. If* $0 < \phi_0 < 63^\circ$*, then for all sufficiently large* n*,* $A(n, \phi_0) = 2^{cn}$ *for some*

$$c \leq -\frac{1}{2}\log(1 - \cos(\phi_0)) - 0.099.$$

3 Algorithms

In this section we describe our two algorithms for the shortest vector problem. In Section 3.1, we describe *List Sieve*. *List Sieve* takes as input a lattice basis and a target norm μ. If there exist lattice vectors with norm less then or equal to μ, it finds one with high probability, else it gives $\perp$. Notice that using *List Sieve* we can easily find the shortest vector with high probability by doing binary search on the value of μ. In Section 3.2 we describe the *Gauss Sieve*, a practical variant of the List Sieve with much better *provable* worst-case space complexity bound $\tilde{O}(\tau_n)$, where τ_n is the kissing constant in dimension n.

3.1 The List Sieve

The List Sieve algorithm works by iteratively building a list L of lattice points. At every iteration, the algorithm attempts to add a new point to the list. Lattice points already in the list are never

[2] Strictly speaking, this is the definition of a *full-rank* lattice. Since only full-rank lattices are used in this paper, all definitions are restricted to the full-rank case.

Algorithm 1 ListSieve($\mathbf{B}, \mu$) *Output:* $\mathbf{v} : \mathbf{v} \in \mathbf{B} \wedge \|\mathbf{v}\| \leq \mu$ OR: $\perp$

```
function ListSieve($\mathbf{B}$, $\mu$)
    $L \leftarrow \{\mathbf{0}\}$, $\delta \leftarrow 1 - 1/n$, $i \leftarrow 0$
    $\xi \leftarrow 0.685$ ▷ The choice of $\xi$ is explained in the analysis
    $K \leftarrow 2^{cn}$ ▷ $c$ is going to be defined in the analysis
    while $i < K$ do
        $i \leftarrow i + 1$
        $(\mathbf{p}_i, \mathbf{e}_i) \leftarrow$ Sample$(\mathbf{B}, \xi\mu)$
        $\mathbf{p}_i \leftarrow$ ListReduce$(\mathbf{p}_i, L, \delta)$
        $\mathbf{v}_i \leftarrow \mathbf{p}_i - \mathbf{e}_i$
        if $(\mathbf{v}_i \notin L)$ then
            if $\exists \mathbf{v}_j \in L : \|\mathbf{v}_i - \mathbf{v}_j\| < \mu$ then
                return $\mathbf{v}_i - \mathbf{v}_j$
            end if
            $L \leftarrow L \cup \{\mathbf{v}_i\}$
        end if
    end while
    return $\perp$
end function

function Sample($\mathbf{B}$, $d$)
    $\mathbf{e} \xleftarrow{\$} \mathcal{B}_n(d)$
    $\mathbf{p} \leftarrow \mathbf{e} \bmod \mathbf{B}$
    return $(\mathbf{p}, \mathbf{e})$
end function

function ListReduce($\mathbf{p}$, $L$, $\delta$)
    while $(\exists \mathbf{v}_i \in L : \|\mathbf{p} - \mathbf{v}_i\| \leq \delta\|\mathbf{p}\|)$ do
        $\mathbf{p} \leftarrow \mathbf{p} - \mathbf{v}_i$
    end while
    return $\mathbf{p}$
end function
```

modified or removed. The goal of the algorithm is to produce shorter and shorter lattice vectors, until two lattice vectors within distance μ from each other are found, and a lattice vector achieving the target norm can be computed as the difference between these two vectors. At every iteration, a new lattice point is generated by first picking a (somehow random, in a sense to be specified) lattice point $\mathbf{v}$, and reducing the length of $\mathbf{v}$ as much as possible by repeatedly subtracting from it the lattice vectors already in the list L when appropriate. Finally, once the length of $\mathbf{v}$ cannot be further reduced, the vector $\mathbf{v}$ is included in the list.

The main idea behind our algorithm design and analysis is that reducing $\mathbf{v}$ with the list vectors L ensures that no two points in the list are close to each other. This is because if $\mathbf{v}$ is close to a list vector $\mathbf{u} \in L$, then $\mathbf{u}$ is subtracted from $\mathbf{v}$ before $\mathbf{v}$ is considered for inclusion in the list. In turn, using linear programming bounds from the study of sphere packings, this allows to prove an upper bound on the size of the list L. This immediately gives upper bounds on the space complexity of the algorithm. Moreover, if at every iteration we were to add a new lattice point to the list, we could immediately bound the running time of the algorithm as roughly quadratic in the list size, because the size of L would also be an upper bound on the number of iterations, and each iteration takes time proportional[3] to the list size $|L|$. The problem is that some iterations might give collisions, lattice vectors $\mathbf{v}$ that already belong to the list. These iterations leave the list L unchanged, and as a result they just waste time. So the main hurdle in the time complexity analysis is bounding the probability of getting collisions.

This is done using the same method as in the original sieve algorithm [4]: instead of directly working with a lattice point $\mathbf{v}$, we use a perturbed version of it $\mathbf{p} = \mathbf{v} + \mathbf{e}$, where $\mathbf{e}$ is a small

[3]Each iteration involves a small (polynomial) number of scans of the current list L.

random error vector of length $\|\mathbf{e}\| \leq \xi\mu$ for an appropriate value of $\xi > 0.5$. Since $\mathbf{e}$ is small, the vectors $\mathbf{p}$ and $\mathbf{v}$ are close to each other, and reducing the length of $\mathbf{p}$ (by subtracting list vectors from it) results in a short lattice vector $\mathbf{v} = \mathbf{p} - \mathbf{e}$. Using standard techniques [17], one can ensure that the conditional distribution of the lattice vector $\mathbf{v} = \mathbf{p} - \mathbf{e}$, given $\mathbf{p}$, is uniform within the sphere of radius $\xi\mu$ centered around $\mathbf{p}$. So, if this sphere contains two lattice points at distance at most $\mu < 2\xi\mu$ one from the other, the probability of sampling the same of these two vectors twice is the same as sampling two vectors withing distance at most μ. This directly translates to an upper bound on the probability of getting a collision. Using this bound we can bound from above the number of samples needed to find a solution vector with high probability. The complete pseudo-code of the List Sieve is given as Algorithm 1. Here we explain the main operations performed by the algorithm.

Sampling. The pair $(\mathbf{p}, \mathbf{e})$ is chosen picking $\mathbf{e}$ uniformly at random within a sphere of radius $\xi\mu$, and setting $\mathbf{p} = \mathbf{e} \bmod \mathbf{B}$. This ensures that, by construction, the sphere $\mathcal{B}(\mathbf{p}, \xi\mu)$ contains at least one lattice point $\mathbf{v} = \mathbf{p} - \mathbf{e}$. Moreover, the conditional distribution of $\mathbf{v}$ (given $\mathbf{p}$) is uniform over all lattice points in this sphere. Notice also that for any $\xi > 0.5$, the probability that $\mathcal{B}(\mathbf{p}, \xi\mu)$ contains more than one lattice point is strictly positive: if $\mathbf{s}$ is a lattice vector achieving the target norm, then the intersection of $\mathcal{B}(\mathbf{0}, \xi\mu)$ and $\mathcal{B}(\mathbf{s}, \xi\mu)$ is not empty, and if $\mathbf{e}$ falls within this intersection, then $\mathbf{v}$ and $\mathbf{v} + \mathbf{s}$ are within distance $\xi\mu$ from $\mathbf{p}$. As we will see, larger values of ξ result in larger bounds on the size of the list L, while smaller values of ξ result in the algorithm producing collisions with higher probability. The running time of the algorithm depends both on the list size and the collision probability. Therefore, in order to minimize the running time we need to choose ξ as an appropriate trade-off between keeping both the list size and collision probability small. In Section A we show how to choose ξ.

List reduction. The vector $\mathbf{p}$ is reduced by subtracting (if appropriate) lattice vectors in L from it. The vectors from L can be subtracted in any order. Our analysis applies independently from the strategy used to choose vectors from L. For each $\mathbf{v} \in L$, we subtract $\mathbf{v}$ from $\mathbf{p}$ only if $\|\mathbf{p} - \mathbf{v}\| < \|\mathbf{p}\|$. Notice that reducing $\mathbf{p}$ with respect to $\mathbf{v}$ may make $\mathbf{p}$ no longer reduced with respect to some other $\mathbf{v}' \in L$. So, all list vectors are repeatedly considered until the length of $\mathbf{p}$ can no longer be reduced. Since the length of $\mathbf{p}$ decreases each time it gets modified, and $\mathbf{p}$ belongs to a discrete set $\mathcal{L}(\mathbf{B}) - \mathbf{e}$, this process necessarily terminates after a finite number of operations. In order to ensure *fast* termination, as in the LLL algorithm, we introduce a slackness parameter $\delta < 1$, and subtract $\mathbf{v}$ from $\mathbf{p}$ only if this reduces the length of $\mathbf{p}$ by at least a factor δ. As a result, the running time of each invocation of the list reduction operation is bounded by the list size $|L|$ times the logarithm (to the base $1/\delta$) of the length of $\mathbf{p}$. For simplicity, we take $\delta(n) = 1 - 1/n$, so that the number of iterations is bounded by a polynomial $\log(n\|\mathbf{B}\|)/\log(1 - 1/n)^{-1} = n^{O(1)}$, and $\delta(n) = 1 - o(1)$ is equal to 1 in the limit.

Termination. When the algorithm starts it computes the maximum number K of samples it is going to use. If a lattice vector achieving norm at most μ is not found after reducing K samples, the algorithm decides that $\lambda_1(\mathbf{B}) > \mu$ and outputs $\perp$. This behavior results a one-sided error algorithm. If $\lambda_1(\mathbf{B}) > \mu$ the algorithm cannot find a lattice vector with norm at most μ and as a result it will always decide correctly $\perp$. However if $\lambda_1(\mathbf{B}) \leq \mu$, the algorithm might terminate before finding a vector with norm $\leq \mu$. In Section A we will show how to choose K so that this error probability is exponentially small.

Now we are ready to state our main theorem for *List Sieve*:

Theorem 3.1 *Let ξ be a real number greater than 0.5 and $c_1(\xi) = log(\xi + \sqrt{\xi^2 + 1}) + 0.401$, $c_2(\xi) = 0.5log(\xi^2/(\xi^2 - 0.25))$. Given a basis $\mathbf{B}$ and a target norm μ, such that $\lambda_1(\mathbf{B}) \leq \mu$,* List Sieve *outputs a lattice vector with norm less or equal to μ with high probability, using $K = \tilde{O}(2^{(c_1+c_2)n})$ samples. The total space required is $N = \tilde{O}(2^{c_1 n})$, and total time $= \tilde{O}(2^{(2\cdot c_1+c_2)n})$.*

The proof is given in Section A.

Corollary 3.2 *For $\xi \simeq 0.685$ we achieve optimal time complexity $< \tilde{O}(2^{3.199n})$ with space complexity $\tilde{O}(2^{1.325n})$.*

3.2 The Gauss Sieve

Algorithm 2 GaussSieve($\mathbf{B}$) *Output:* $\mathbf{v} : \mathbf{v} \in \mathbf{B} \wedge \|\mathbf{v}\| \leq \lambda_1(\mathbf{B})$

```
function GaussSieve($\mathbf{B}$, $\mu$)
    $L \leftarrow \{\mathbf{0}\}$, $S \leftarrow \{ \}$, $K \leftarrow 0$
    while $K < c$ do
        if $S$ is not empty then
            $\mathbf{v}_{new} \leftarrow$S.pop()
        else
            $\mathbf{v}_{new} \leftarrow$SampleGaussian($\mathbf{B}$)
        end if
        $\mathbf{v}_{new} \leftarrow$GaussReduce($\mathbf{v}_{new}, L, S$)
        if ($\mathbf{v}_{new} = \mathbf{0}$) then
            $K \leftarrow K + 1$
        else
            $L \leftarrow L \cup \{\mathbf{v}_{new}\}$
        end if
    end while
end function

function GaussReduce($\mathbf{p}$, $L$, $S$)
    while ($\exists \mathbf{v}_i \in L : \|\mathbf{v}_i\| \leq \|\mathbf{p}\|$
                $\wedge \|\mathbf{p} - \mathbf{v}_i\| \leq \|\mathbf{p}\|$) do
        $\mathbf{p} \leftarrow \mathbf{p} - \mathbf{v}_i$
    end while
    while ($\exists \mathbf{v}_i \in L : \|\mathbf{v}_i\| > \|\mathbf{p}\|$
                $\wedge \|\mathbf{v}_i - \mathbf{p}\| \leq \|\mathbf{v}_i\|$) do
        $L \leftarrow L \setminus \{\mathbf{v}_i\}$
        $S.push(\mathbf{v}_i - \mathbf{p})$
    end while
    return $\mathbf{p}$
end function
```

We now present a practical variant of the List Sieve with much better space complexity. The Gauss Sieve follows the same general approach of building a list of shorter and shorter lattice vectors, but when a new vector $\mathbf{v}$ is added to the list, not only we reduce the length of $\mathbf{v}$ using the list vectors, but we also attempt to reduce the length of the vectors already in the list using $\mathbf{v}$. In other words, if $\min(\|\mathbf{v} \pm \mathbf{u}\|) < \max(\|\mathbf{v}\|, \|\mathbf{u}\|)$, then we replace the longer of $\mathbf{v}, \mathbf{u}$ with the shorter of $\mathbf{v} \pm \mathbf{u}$. As a result, the list L always consists of vectors that are pairwise reduced, i.e., they satisfy the condition $\min(\|\mathbf{u} \pm \mathbf{v}\|) \geq \max(\|\mathbf{u}\|, \|\mathbf{v}\|)$. This is precisely the defining condition of reduced basis achieved by the Gauss/Lagrange basis reduction algorithm for two dimensional lattices, hence the name of our algorithm.

It is well known that if $\mathbf{u}, \mathbf{v}$ is a Gauss reduced basis, then the angle between $\mathbf{u}$ and $\mathbf{v}$ is at least $\pi/3$ (or 60 degree). As a result, the maximum size of the list (and space complexity of the algorithm) can be immediately bounded by the kissing number τ_n. Unfortunately, we are unable to prove interesting bounds on the running time of the algorithm. As for the List Sieve, the main problem encountered in the analysis is to prove that collisions are not likely to occur, unless we have already found a shortest lattice vector. In our first algorithm we solved the problem by adding an error term $\mathbf{e}$ to the new lattice vector $\mathbf{v}$ being added to the list. We could still do this here in order to guarantee that, with reasonably high probability, after reducing $\mathbf{v}$ with the list, $\mathbf{v}$ is not too likely to be one of the vectors already present in the list. The problems arise when we try to reduce the vectors already in the list using the newly added vector $\mathbf{v}$. We recall that all vectors in the list belong to the lattice: there is no error term or residual randomness in the list vectors, and we cannot use the same information theoretic argument used for $\mathbf{v}$ to conclude that the list vectors do not collide with each other. Notice that collisions are problematic because they reduce the list size, possibly leading to nonterminating executions that keep adding and removing vectors from the list. In practice (see experimental results in Section B) this does not occur, and the running time of the algorithm seems to be slightly more than quadratic in the list size, but we do not know how to prove any worst-case upper bound.

Since we cannot prove any upper bound on the running time of the Gauss Sieve algorithm, we apply some additional simplifications and practical improvements to the *List Sieve*. For example, in the Gauss Sieve, we remove the use of error vectors $\mathbf{e}$ from the initial sampling algorithm, and choose $\mathbf{p} = \mathbf{v}$ at random. This has the practical advantage that the Gauss Sieve only works with lattice points, allowing an integer only implementation of the algorithm (except possibly for the sampling procedure which may still internally use floating point numbers). Following [22], we sample the new lattice vectors $\mathbf{v}$ using Klein's randomized rounding algorithm [15], but since we cannot prove anything about running time, this choice is largely arbitrary.

The Gauss Sieve pseudo-code is shown as Algorithm 2. The algorithm uses a stack or queue data structure S to temporarily remove vectors from the list L. When a new point $\mathbf{v}$ is reduced with L, the algorithm checks if any point in L can be reduced with $\mathbf{v}$. All such points are temporarily removed from L, and inserted in S for further reduction. The Gauss Sieve algorithm reduces the points in S with the current list before inserting them in L. When the stack S is empty, all list points are pairwise reduced, and the Gauss Sieve can sample a new lattice point $\mathbf{v}$ for insertion in the list L. Unfortunately, we cannot bound the number of samples required to find the shortest vector with high probability. As a result we have to use a heuristic termination condition. Based on experiments a good heuristic is to terminate after a certain number $c(n)$ of collisions has occurred (see section B).

4 Extensions and open problems

An interesting feature common to all sieve algorithms is that they can be slightly optimized to take advantage of the structure of certain lattices used in practical cryptographic constructions, like the NTRU lattices [11], or the cyclic lattices of [18]. The idea is the following. The structured lattices used in this constructions have a non-trivial automorphism group, i.e., they are invariant under certain (linear) transformations. For example, cyclic lattices have the property that whenever a vector $\mathbf{v}$ is in the lattice, then all n cyclic rotations of $\mathbf{v}$ are in the lattice. When reducing a new point $\mathbf{p}$ against a list vector $\mathbf{v}$, we can use all rotations of $\mathbf{v}$ to decrease the length of $\mathbf{p}$. Effectively, this allows us to consider each list point as the implicit representation of n list points. This approximately translates to a reduction of the list size by a factor n. While this reduction may not be much from a theoretical point of view because the list size is exponential, it may have a substantial impact on the practical performance of the algorithm.

There are a number of open problems concerning algorithms for the shortest vector problem. It is still unclear if we can get a $2^{O(n)}$ algorithm that is using only polynomial space, or even how to get a deterministic $2^{O(n)}$ on time and space algorithm. Concerning sieve based algorithms we identify two possible lines of research. Firstly, improving the current algorithms. Bounding the running time of Gauss Sieve, or getting a faster heuristic would be very interesting. Another interesting question is whether the bound of Kabatiansky and Levenshtein [12] can be improved when the lattice is known to be cyclic, or has other interesting structure. The second line of research is to use sieving as a subroutine for other algorithms that currently use enumeration techniques. Our early experimental results hint that sieving could solve SVPs in higher dimensions than we previously thought possible. It is especially interesting for example, to examine if such a tool can give better cryptanalysis algorithms.

5 Acknowledgments

We thank Phong Nguyen and Thomas Vidick for providing the implementation of the Sieve algorithm described in [22] that we used for our experimental comparison.

References

[1] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, Aug. 2002.

[2] M. Ajtai. Generating hard instances of lattice problems. *Complexity of Computations and Proofs, Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.

[3] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of STOC '97*, pages 284–293. ACM, May 1997.

[4] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of STOC '01*, pages 266–275. ACM, July 2001.

[5] J. Conway and N. Sloane. *Sphere Packings, Lattices and Groups.* Springer, 1999.

[6] C. Dwork. Positive applications of lattices to cryptography. In I. Prívara and P. Ruzicka, editors, *Mathematical Foundations of Computer Science 1997*, volume 1295 of *LNCS*, pages 44–51. Springer, Aug. 1997.

[7] N. Gama and P. Q. Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of STOC '08*, pages 207–216. ACM, May 2008.

[8] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *Proceedings of EUROCRYPT '08*, volume 4965 of *LNCS*, pages 31–51. Springer, 2008.

[9] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of STOC '08*, pages 197–206. ACM, May 2008.

[10] G. Hanrot and D. Stehlé. Improved Analysis of Kannan's Shortest Lattice Vector Algorithm. *Arxiv preprint arXiv:0705.0965*, 2007.

[11] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: a ring based public key cryptosystem. In *Proceedings of ANTS-III*, volume 1423 of *LNCS*, pages 267–288. Springer, June 1998.

[12] G. Kabatiansky and V. Levenshtein. Bounds for packings on a sphere and in space. *Problemy Peredachi Informatsii*, 14(1):3–25, 1978.

[13] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the fifteenth annual ACM symposium on theory of computing - STOC '83*, pages 193–206. ACM, Apr. 1983.

[14] R. Kannan. *Annual Review of Computer Science*, volume 2, chapter Algorithmic Geometry of numbers, pages 231–267. Annual Review Inc., Palo Alto, California, 1987.

[15] P. Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the 11th symposium on discrete algorithms*, San Francisco, California, Jan. 2000. SIAM.

[16] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982.

[17] D. Micciancio. Improving lattice based cryptosystems using the Hermite normal form. In *Proceedings of CaLC '01*, volume 2146 of *LNCS*, pages 126–145. Springer, Mar. 2001.

[18] D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, Dec. 2007. Preliminary version in FOCS 2002.

[19] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, Mar. 2002.

[20] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measure. *SIAM Journal on Computing*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[21] P. Nguyen and J. Stern. The two faces of lattices in cryptology. In *Proceedings of CaLC '01*, volume 2146 of *LNCS*, pages 146–180. Springer, Mar. 2001.

[22] P. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology*, 2(2):181–207, jul 2008.

[23] C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In *Proceedings of STOC '08*, pages 187–196. ACM, May 2008.

[24] M. Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *ACM SIGSAM Bulletin*, 15(1):37–44, 1981.

[25] X. Pujol and D. Stehlé. Rigorous and efficient short lattice vectors enumeration, 2008. In *À paraître dans les actes de la conférence Asiacrypt*, 2008.

[26] O. Regev. New lattice based cryptographic constructions. *Journal of the ACM*, 51(6):899–942, 2004. Preliminary version in STOC 2003.

[27] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of STOC '05*, pages 84–93. ACM, June 2005.

[28] O. Regev. Lecture notes on lattices in computer science, 2004. *Availab le at http://www. cs. tau. ac. il/ odedr/teaching/lattices_fall_2004/index. html, last accessed*, 28, 2008.

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

[30] C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, Aug. 1994. Preliminary version in FCT 1991.

[31] V. Shoup. NTL: A library for doing number theory, 2003.

APPENDIX

A Analysis of *List Sieve*

In this section prove time and space upper bounds for the *List Sieve* algorithm. In subsection A.1 we use the fact that the list points are far apart to get an upper bound N on the number of list points. Then in subsection A.2 we prove that collisions cannot happen too often. We use this fact to prove that after a certain number of samples we get a vector with norm less then or equal to μ with high probability, provided such a vector exists.

A.1 Space complexity

We first establish a simple lemma that will be useful to bound the distance between lattice points.

Lemma A.1 *For any two vectors* $\mathbf{x}, \mathbf{y}$ *and real* $0 < \delta < 1$, $\|\mathbf{x}-\mathbf{y}\| > \delta \cdot \|\mathbf{x}\|$ *if and only if* $\|(1-\delta^2)\mathbf{x}-\mathbf{y}\| > \delta\|\mathbf{y}\|$.

Proof. Squaring both sides in the first inequality, we get $\|\mathbf{x}-\mathbf{y}\|^2 > \delta^2\|\mathbf{x}\|^2$. Expanding $\|\mathbf{x}-\mathbf{y}\|^2$ into $\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\langle\mathbf{x},\mathbf{y}\rangle$ and rearranging the terms gives

$$(1-\delta^2)\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\langle\mathbf{x},\mathbf{y}\rangle > 0.$$

Next, multiply by $1-\delta^2 > 0$ and rearrange, to get

$$(1-\delta^2)^2\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2(1-\delta^2)\langle\mathbf{x},\mathbf{y}\rangle > \delta^2\|\mathbf{y}\|^2.$$

Finally, notice that the left hand side is the expansion of $\|(1-\delta^2)\mathbf{x}-\mathbf{y}\|^2$. ■

We can now prove a bound on the size of the list generated by the List Sieve algorithm.

Theorem A.2 *The number of points in* L *is bounded from above by* $N = poly(n) \cdot 2^{c_1 n}$ *where*

$$c_1 = \log(\xi + \sqrt{\xi^2+1}) + 0.401.$$

Proof: Let $\mathbf{B}$ be the input basis, and μ be the target length of the List Sieve algorithm. Notice that as soon as the algorithm finds two lattice vectors within distance μ from each other, the algorithm terminates. So, the distance between any two points in the list L must be greater than μ. In order to bound the size of L, we divide the list points into groups, according to their length, and bound the size of each group separately. Consider all list points belonging to a sphere of radius $\mu/2$

$$S_0 = L \cap \mathcal{B}(\mu/2).$$

Clearly S_0 has size at most 1, because the distance between any two points in $\mathcal{B}(\mu/2)$ is bounded by μ. Next, divide the rest of the space into a sequence of coronas

$$S_i = \{\mathbf{x} : \gamma^{i-1}\mu/2 \le \|\mathbf{x}\| < \gamma^i\mu/2\}$$

for $i = 1, 2, \ldots$ and $\gamma = 1 + 1/n$. Notice that we only need to consider a polynomial number of coronas $\log_\gamma(2n\|\mathbf{B}\|/\mu) = O(n^c)$, because all list points have length at most $n\|\mathbf{B}\|$. We will prove an exponential bound on the number of points in each corona. The same bound will hold (up to polynomial factors) for the total number of points in the list L.

So, fix a corona $S_i = \{\mathbf{v} \in L \colon R \leq \|\mathbf{v}\| \leq \gamma R\}$ for some $R = \gamma^{i-1}\mu/2$, and consider two arbitrary points $\mathbf{v}_a, \mathbf{v}_b \in S_i$. We will show that

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) \leq \frac{1}{2} + \xi^2(\sqrt{1+1/\xi^2} - 1) + o(1) = 1 - \frac{1}{2(\xi + \sqrt{\xi^2+1})^2} + o(1). \tag{2}$$

Notice that this upper bound on $\cos(\phi)$ is greater than 0.5, and as a result $\phi_{\mathbf{v}_a,\mathbf{v}_b} < 60°$. Therefore, we can safely use Theorem 2.2 with the bound (2), and conclude that the number of points in S_i is at most $2^{c_1 n}$ where

$$\begin{aligned} c_1 &= -\frac{1}{2}\log(1-\cos(\phi)) - 0.099 \\ &\leq \log\left(\sqrt{2}(\xi + \sqrt{\xi^2+1})\right) - 0.099 \\ &= \log(\xi + \sqrt{\xi^2+1}) + 0.401 \end{aligned}$$

as stated in the theorem. It remains to prove (2). First note that

$$\|\mathbf{v}_a - \mathbf{v}_b\|^2 = \|\mathbf{v}_a\|^2 + \|\mathbf{v}_b\|^2 - 2\|\mathbf{v}_a\| \cdot \|\mathbf{v}_b\| \cdot cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}).$$

Solving for $cos(\phi_{\mathbf{v}_a,\mathbf{v}_b})$ we obtain

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) = \frac{1}{2}\left(\frac{\|\mathbf{v}_a\|}{\|\mathbf{v}_b\|} + \frac{\|\mathbf{v}_b\|}{\|\mathbf{v}_a\|} - \frac{\|\mathbf{v}_a - \mathbf{v}_b\|^2}{\|\mathbf{v}_a\| \cdot \|\mathbf{v}_b\|}\right). \tag{3}$$

We bound (3) in two different ways. Using the fact that the distance between any two list points is at least μ we immediately get

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) \leq \frac{1}{2}\left(\frac{\|\mathbf{v}_a\|}{\|\mathbf{v}_b\|} + \frac{\|\mathbf{v}_b\|}{\|\mathbf{v}_a\|} - \frac{\mu^2}{\|\mathbf{v}_a\| \cdot \|\mathbf{v}_b\|}\right) \leq 1 - \frac{\mu^2}{2R^2} + o(1) \tag{4}$$

Notice that this bound is very poor when R is large. So, for large R, we bound (3) differently. Without loss of generality, assume that $\mathbf{v}_b$ was added after $\mathbf{v}_a$. As a result the perturbed point $\mathbf{p}_b = \mathbf{v}_b + \mathbf{e}_b$ is reduced with $\mathbf{v}_a$, i.e., $\|\mathbf{p}_b - \mathbf{v}_a\| > \delta\|\mathbf{p}_b\|$. Using Lemma A.1, we get

$$\|(1-\delta^2)\mathbf{p}_b - \mathbf{v}_a\| > \delta\|\mathbf{v}_a\|.$$

We use triangular inequality to replace $\mathbf{p}_b$, with $\mathbf{v}_b$:

$$\|(1-\delta^2)\mathbf{v}_b - \mathbf{v}_a\| \geq \|(1-\delta^2)\mathbf{p}_b - \mathbf{v}_a\| - (1-\delta^2)\|\mathbf{e}_b\| > \delta\|\mathbf{v}_a\| - (1-\delta^2)\xi\mu.$$

Substituting in (3) we get

$$\begin{aligned} \cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) &= \cos(\phi_{\mathbf{v}_a,(1-\delta^2)\mathbf{v}_b}) \\ &= \frac{1}{2}\left(\frac{\|\mathbf{v}_a\|}{(1-\delta^2)\|\mathbf{v}_b\|} + \frac{(1-\delta^2)\|\mathbf{v}_b\|}{\|\mathbf{v}_a\|} - \frac{\|\mathbf{v}_a - (1-\delta^2)\mathbf{v}_b\|^2}{(1-\delta^2)\|\mathbf{v}_a\| \cdot \|\mathbf{v}_b\|}\right) \\ &\leq \frac{\|\mathbf{v}_a\|^2 + (1-\delta^2)^2\|\mathbf{v}_b\|^2 - (\delta\|\mathbf{v}_a\| - (1-\delta^2)\xi\mu)^2}{2(1-\delta^2)\|\mathbf{v}_a\| \cdot \|\mathbf{v}_b\|} \\ &= \frac{\|\mathbf{v}_a\|}{2\|\mathbf{v}_b\|} + \frac{\xi\mu\delta}{\|\mathbf{v}_b\|} + (1-\delta^2)\frac{\|\mathbf{v}_b\|^2 - (\xi\mu)^2}{2\|\mathbf{v}_a\|\|\mathbf{v}_b\|} \\ &= \frac{1}{2} + \frac{\xi\mu}{R} + o(1). \end{aligned}$$

Combining this bound with (3), we see that

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) \leq \min\left\{1 - \frac{\mu^2}{2R^2}, \frac{1}{2} + \frac{\xi\mu}{R}\right\} + o(1). \tag{5}$$

Notice that as R increases, the first bound gets worse and the second better. So, the minimum (5) is maximized when

$$1 - \frac{\mu^2}{2R^2} = \frac{1}{2} - \frac{\xi\mu}{R}.$$

This is a quadratic equation in $x = \mu/R$, with only one positive solution

$$\frac{\mu}{R} = \sqrt{1+\xi^2} - \xi = \xi(\sqrt{1+1/\xi^2} - 1),$$

which, substituted in (5), gives the bound (2). ■

A.2 Time Complexity

Let us revisit *List Sieve.* It samples random perturbations $\mathbf{e}_i$ from $\mathcal{B}_n(\xi\mu)$, sets $\mathbf{p}_i = \mathbf{e}_i \bmod \mathbf{B}$ and reduces it with the list L. Then it considers the lattice vector $\mathbf{v}_i = \mathbf{p}_i - \mathbf{e}_i$. This vector might be one of the following:

- Event **C:** $\mathbf{v}_i$ is a collision ($dist(L, \mathbf{v}_i) = 0$)
- Event **L:** $\mathbf{v}_i$ is a new list point ($dist(L, \mathbf{v}_i) > \mu$)
- Event **S:** $\mathbf{v}_i$ is a solution ($0 < dist(L, \mathbf{v}_i) \leq \mu$).

We will prove that if there exist a lattice vector $\mathbf{s}$ with $\|\mathbf{s}\| \leq \mu$, event **S** will happen with high probability after a certain number of samples.

We first give a lower bound to the volume of a hypersphere cap. We will use this to upper bound the probability of getting collisions:

Lemma A.3 *Let* $\mathrm{Cap}_{R,h}$ *be the n-dimensional cap with height h of a hypersphere* $\mathcal{B}_n(R)$*. Then for* $R_b = \sqrt{2hR - h^2}$*:*

$$\frac{\mathrm{Vol}(\mathrm{Cap}_{h,R})}{\mathrm{Vol}(\mathcal{B}_n(R))} > \left(\frac{R_b}{R}\right)^n \cdot \frac{h}{2R_b n}$$

Proof: The basis of $\mathrm{Cap}_{R,h}$ is an $n-1$ dimensional hypersphere with radius $R_b = \sqrt{2hR - h^2}$. Therefore $\mathrm{Cap}_{R,h}$ includes a cone C_1 with basis $\mathcal{B}_{n-1}(R_b)$ and height h. Also notice that a cylinder C_2 with basis $\mathcal{B}_{n-1}(R_b)$ and height $2 \cdot R_b$ includes $\mathcal{B}_n(R_b)$. Using the facts above we have:

$$\mathrm{Vol}(\mathrm{Cap}_{h,R}) > \mathrm{Vol}(C_1) = \mathrm{Vol}(\mathcal{B}_{n-1}(R_b))\frac{h}{n} = \mathrm{Vol}(C_2)\frac{h}{2R_b n} > \mathrm{Vol}(\mathcal{B}_n(R_b))\frac{h}{2R_b n}$$

Therefore:

$$\frac{\mathrm{Vol}(\mathrm{Cap}_{h,R})}{\mathrm{Vol}(\mathcal{B}_n(R))} > \frac{\mathrm{Vol}(\mathcal{B}_n(R_b))}{\mathrm{Vol}(\mathcal{B}_n(R))} \cdot \frac{h}{2R_b n} = \left(\frac{R_b}{R}\right)^n \cdot \frac{h}{2R_b n}$$

■

Theorem A.4 *If there exists a lattice point* $\mathbf{s}$ *with* $\|\mathbf{s}\| \leq \mu$, *then* List Sieve *will output a lattice point with norm* $\leq \mu$ *with high probability after using* $K = \tilde{O}(2^{(c_1+c_2)n})$ *samples, with*

$$c_1 = \log(\xi + \sqrt{\xi^2+1}) + 0.401 \qquad c_2 = \log\left(\frac{\xi}{\sqrt{\xi^2 - 0.25}}\right).$$

Proof: Consider the intersection of two hyperspheres with radius $\xi\mu$ and centers $\mathbf{0}$ and $-\mathbf{s}$, $I = \mathcal{B}_n(\mathbf{0}, \xi\mu) \cap \mathcal{B}_n(-\mathbf{s}, \xi\mu)$. Also we will use $I' = I + \mathbf{s} = \mathcal{B}_n(\mathbf{0}, \xi\mu) \cap \mathcal{B}_n(\mathbf{s}, \xi\mu)$. Notice that $\mathbf{e}_i \in I$ if and only if $\mathbf{e}'_i = \mathbf{e}_i + \mathbf{s} \in I'$, therefore both perturbations $\mathbf{e}_i, \mathbf{e}'_i$ are equally probable. Now notice that *Sample* will give exactly the same point $\mathbf{p}_i$ for both $\mathbf{e}_i, \mathbf{e}'_i$.

$$\mathbf{p}'_i = \mathbf{e}'_i \bmod \mathbf{B} = \mathbf{e}_i \bmod \mathbf{B} + \mathbf{s} \bmod \mathbf{B} = \mathbf{e}_i \bmod \mathbf{B} = \mathbf{p}_i$$

As a result *ListReduce* given $\mathbf{p}_i$ is oblivious to which perturbation of $\mathbf{e}_i, \mathbf{e}'_i$ was chosen. After the reduction, $\mathbf{p}_i$ can correspond to $\mathbf{v}_i = \mathbf{p}_i - \mathbf{e}_i$ and $\mathbf{v}'_i = \mathbf{p}_i - \mathbf{e}'_i = \mathbf{v}_i - \mathbf{s}$ with equal probability. The distance between $\mathbf{v}_i, \mathbf{v}'_i$ is at most $\|\mathbf{s}\| \leq \mu$ therefore they cannot be both in L. So at least one of $\mathbf{v}_i, \mathbf{v}'_i$ is not a collision. As a result we can organize the perturbations of $I \cup I'$ in pairs. From each pair at least one perturbation does not give a collision. As a result the probability of not C given $\mathbf{e}_i \in I \cup I'$ is $Pr[\mathbf{C'}|\mathbf{e}_i \in I \cup I'] \geq 0.5$.

Now notice that $I \cup I'$ contains two disjoint caps with height $h = \xi\mu - \|\mathbf{s}\|/2 \geq \xi\mu - \mu/2$ on a n-dimensional hypersphere of radius $\xi\mu$. We use lemma A.3 to bound from below the probability of getting $\mathbf{e}_i$ in $I \cup I'$:

$$Pr[\mathbf{e}_i \in I \cup I'] = \frac{2\text{Vol}(\text{Cap}_{\xi\mu - 0.5\mu, \xi\mu})}{\text{Vol}(\mathcal{B}_n(\xi\mu))} > \left(\frac{\sqrt{\xi^2 - 0.25}}{\xi}\right)^n \cdot \frac{\xi - 0.5}{n\sqrt{\xi^2 - 0.25}} = 2^{-c_2 n} \cdot \frac{\xi - 0.5}{n\sqrt{\xi^2 - 0.25}}.$$

Therefore the probability of not getting a collision is bounded from below by

$$Pr[\mathbf{C'}] \geq Pr[\mathbf{C'}|\mathbf{e}_i \in I \cup I']Pr[\mathbf{e}_i \in I \cup I'] \geq 2^{-c_2 n} \cdot \frac{\xi - 0.5}{2n\sqrt{\xi^2 - 0.25}} = p.$$

Now given that the probability of event $\mathbf{C'}$ is at least p, the number of occurrences of $\mathbf{C'}$ after K samples is lower bounded by a random variable X following binomial distribution $Binomial(K, p)$. Let $F(N; K, p) = Pr[X \leq N]$ the probability of getting no more than N occurrences of $\mathbf{C'}$ after K samples. If we set $K = 2Np^{-1} = \tilde{O}(2^{(c_1+c_2)n})$ we can use Chernoff's inequality:

$$F(N; K, p) \leq exp\left(-\frac{1}{2p}\frac{(Kp - N)^2}{K}\right) = exp\left(-\frac{N}{4}\right) \leq \frac{1}{2^{O(n)}}$$

As a result after K samples we will get at least $N + 1$ occurrences of event $\mathbf{C'}$ with high probability. These events can either be $\mathbf{L}$ or $\mathbf{S}$ events. However the list size cannot grow more than N, and as a result the number of $\mathbf{L}$ events can be at most N. So event $\mathbf{S}$ will happen with high probability. ■

Theorem A.5 *The total running time of the algorithm is* $2^{(2 \cdot c_1 + c_2)n}$ *with*

$$c_1 = \log(\xi + \sqrt{\xi^2+1}) + 0.401 \qquad c_2 = \log\left(\frac{\xi}{\sqrt{\xi^2 - 0.25}}\right).$$

Proof: Let us consider the running time of *List Reduce.* After every pass of the list L the input vector p_i to *List Reduce* gets δ times shorter. Therefore the total running time is $log_\delta(\|\mathbf{B}\|n) = poly(n)$ passes of the list and each pass costs $O(N)$ operations. Now notice that our algorithm will run *List Reduce* for K samples, this gives us total running time of $\tilde{O}(K \cdot N) = \tilde{O}(2^{(2c_1+c_2)n})$. ■

Setting the parameter ξ: The space complexity gets better as ξ grows smaller. By choosing ξ arbitrarily close to 0.5, we can achieve space complexity 2^{sn} for any $s > (\log 0.401) + (1+\sqrt{5})/2 \approx 1.095$, and still keep $2^{O(n)}$ running time, but with a large constant in the exponent. At the cost of slightly increasing the space complexity, we can substantially reduce the running time. The value of ξ that yields the best running time is $\xi \simeq 0.685$ which yields space complexity $< 2^{1.325n}$ and time complexity $< 2^{3.199n}$.

B Practical performance of Gauss Sieve

In this section we describe some preliminary experimental results on the *Gauss Sieve.* We will describe the design of our experiments and then we will discuss the results on the space and time requirements of our algorithm.

Experiment setting: For our preliminary experiments we have generated square $n \times n$ bases corresponding to random knapsack problems modulo a prime of $\simeq 10 \cdot n$ bits. These bases are considered "hard" instances and are frequently used in bibliography [22, 8] to evaluate lattice algorithms. In our experiments we used the *Gauss Sieve* with $c = 500$, the *NV Sieve* implementation from [22] and the NTL library for standard enumeration techniques [30]. For every $n = 30, \ldots, 62$ we generated 6 random lattices, and measured the average space and time requirements of the algorithms. For sieving algorithms the logarithms of these measures grow almost linearly with the dimension n. We use a simple model of 2^{cn} to fit our results and we use least squares estimation to compute c. We run our experiments on a Q6600 Pentium, using only one core, and the algorithms were compiled with exactly the same parameters. The experiments are certainly not exhaustive, however we believe that they are enough to give a sense of the practical performance of our algorithm, at least in comparison to previous sieving techniques.

Size complexity: To evaluate the size complexity of the *Gauss Sieve* we measure the maximum list size. (We recall that in the Gauss Sieve the list can both grow and shrink, as list points collide with each other. So, we consider the maximum list size during each run, and then average over the input lattice.) Our experiments show that the list size grows approximately as $2^{0.18n}$. This is consistent with our theoretical *worst-case* analysis, which bounds the list size by the kissing constant τ_n. Recall that τ_n can be reasonably conjectured to be at most $2^{0.21n}$. The fact that the experimental results are even better than that should not be surprising, as τ_n is a bound on the worst-case list size, rather than average list size when the input lattice is chosen at random. The actual measured exponent 0.18 may depend on the input lattice distribution, and it would be interesting to run experiments on other distributions. However, in all cases, we can expect the space complexity to be bounded by $2^{0.21n}$. Somehow similar bounds on space complexity are given in Nguyen and Vidick's work [22]. *Gauss Sieve* improves *NV Sieve* results in two ways:

- *Theory:* Our τ_n bound is proven under no heuristic assumptions, and gives an interesting connection between sieving techniques and spherical coding.

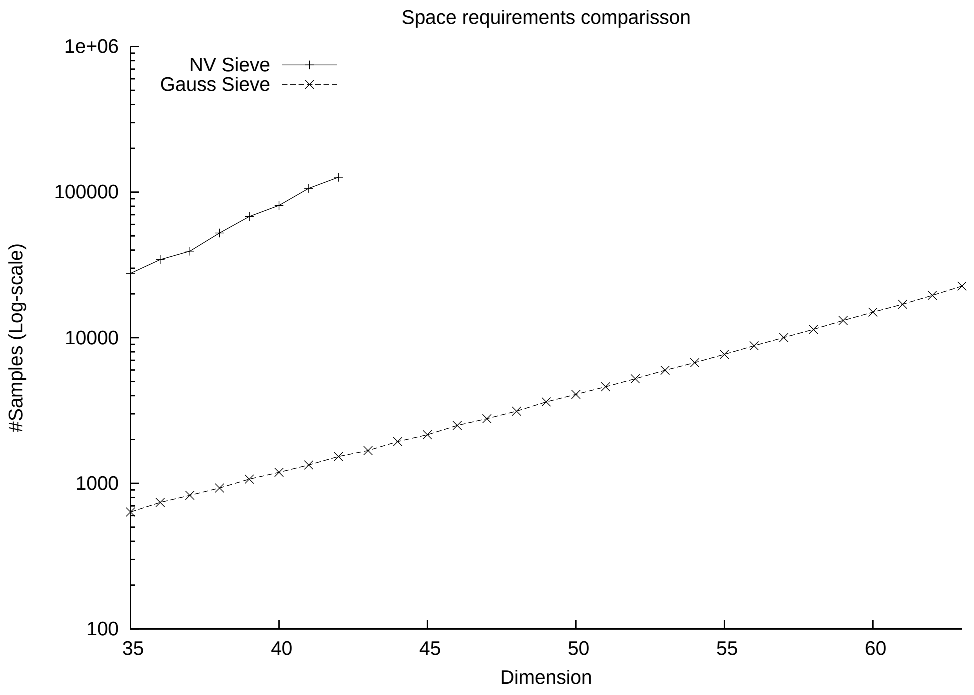

Figure 1: Space requirements of Sieving algorithms

- *Practice:* In practice *Gauss Sieve* uses far fewer points (e.g., in dimension $n \simeq 40$, the list size is smaller approximately by a factor $\simeq 70$) See figure 1.

Running time: We cannot bound the number of samples required by *Gauss Sieve* to find the shortest vector with high probability. However we found that a good termination heuristic is to stop after a certain number c of collisions. For our experiments we used $c = 500$, and *Gauss Sieve* was always able to find the solution. However better understanding for a good termination condition of the algorithm is required.

The running grows approximately as $2^{(0.48n)}$, which is similar to the experiments of *NV Sieve*. However once more *Gauss Sieve* is by far superior in practice. For example, in dimension $\simeq 40$, the 70-fold improvement on the list size and the (at least) quadratic dependency of the running time on the space bound, already suggested a $70^2 \simeq 0.5 \cdot 10^4$ speedup. In practice we measured an slightly bigger $\simeq 10^4$ improvement. In Figure 2 we also give the running time of the Schnorr-Euchner (SE) enumeration algorithm as implemented in NTL.[4] This preliminary comparison with SE is meant primarily to put the comparison between sieve algorithms in perspective. In [22], Nguyen and Vidick had compared their variant of the sieve algorithm with the same implementation of SE used here, and on the same class of random lattices. Their conclusion was that while sieve algorithms have better asymptotics, the SE algorithm still reigned in practice, as the cross-over point is way beyond dimension $n \simeq 50$, and their algorithm was too expensive to be run in such high dimension.

[4]The running time of enumeration algorithms, is greatly affected by the quality of the initial basis. To make a fair comparison we have reduced the basis using BKZ with window 20.

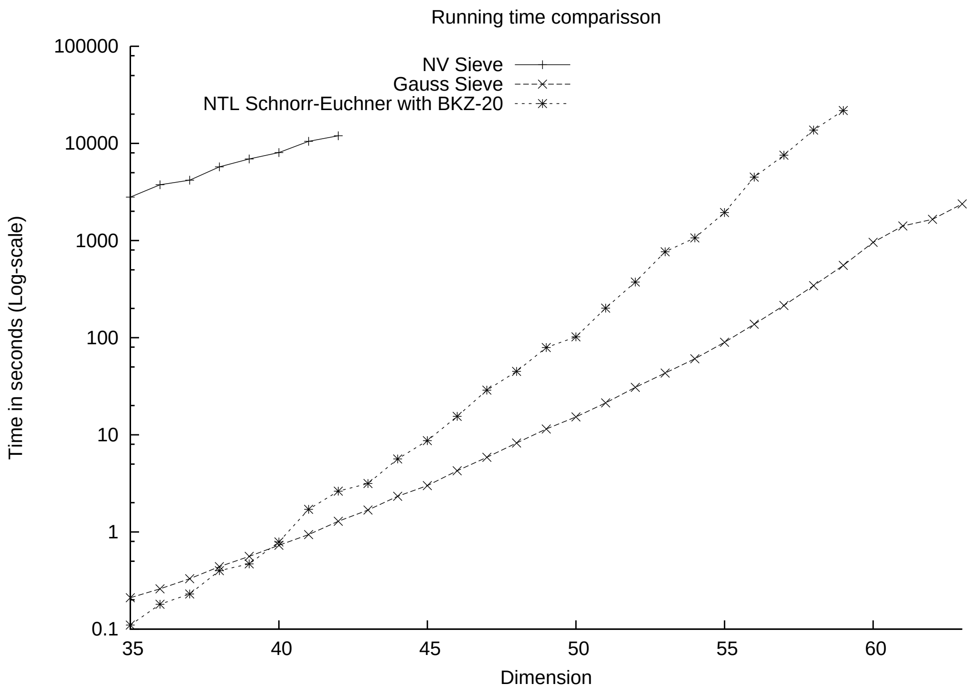

Figure 2: Running Times

Including our sieve algorithm in the comparison, changes the picture quite a bit: the crossover point between the Gauss Sieve and the Schnorr-Euchnerr reference implementation used in [22] occurs already in dimension $n \simeq 40$. This improved performance shows that sieve algorithms have the potential to be used in practice as an alternative to standard enumeration techniques. We remark that the enumeration algorithms have been the subject of intensive study and development over several decades, and many heuristics are known (e.g., pruning) to substantially speed up enumeration. The development of similar heuristics for sieve algorithms, and a through comparison of how heuristics affect the comparison between enumeration and sieving, both in terms of running time and quality of the solution, is left to future research.