Faster exponential time algorithms for the shortest vector problem[*]

Daniele Micciancio[†] Panagiotis Voulgaris[†]

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 $2^{1.325n}$), or in space $2^{1.095n}$ (and still time $2^{O(n)}$). 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. 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 $2^{0.52n}$, and space complexity $2^{0.2n}$.

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, 20]), and the problem underlying many cryptographic functions (e.g., [2, 3, 25, 26, 22, 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 [21] is outperformed by the asymptotically inferior $2^{O(n^2)}$-time Schnorr-Euchner enumeration algorithm [29] 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 heuristic algorithm of [29] (which is not even known to run in polynomial time) is preferred in practice to provable polynomial time approximation algorithms like [28, 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 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 for the shortest vector problem 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:

- *List Sieve:* 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 $\tilde{O}(2^{1.325n})$ (or space $2^{1.095n}$ and still $2^{O(n)}$ time), 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, 21], and
- *Gauss Sieve:* A practical variant of List Sieve that admits much better space bounds, and outperforms the best previous practical implementation [21] 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 near $2^{0.21n}$, because finding a family of lattices for which the algorithm uses more than $2^{0.21n}$ space would imply denser arrangements of hyperspheres than

[*] 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.

[†] University of California, San Diego, e-mail: {daniele, pvoulgar}@ cs.ucsd.edu

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 exist (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 5 are consistent with our analysis, and suggest that the space complexity of our second algorithm is indeed near $2^{0.21n}$. 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 $2^{0.52n}$.

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) and contribution (Section 1.3). 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 about their complexity. Section 4 gives the proof for the time and space complexity asymptotic bounds, while Section 5 contains our experimental results. In Section 6 we give a detailed comparison between our algorithms and the AKS sieve. Section 7 concludes with a discussion of open problems.

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 a nonzero 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 an LLL reduced basis [16], the Fincke-Pohst enumeration algorithm [23] 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 [29], currently used in state of the art practical lattice reduction implementations [30, 24]. 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 [21] 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 [21], building on ideas from [27]. 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 [21] 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, 21] 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. 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 [21] for a more detailed and technical description of the AKS sieve.)

While using many technical ideas from [4, 21], 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 *List Sieve*, 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 us 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, 21], 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. Unfortunately the introduction of errors reduces the effectiveness of sieving and increases the space complexity.

In practice, as shown in [21], variants of sieving algorithms without errors, perform much better, but lack theoretical time bounds. *Gauss Sieve* is a practical variant of *List Sieve* without errors which incorporates a new heuristic technique. 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 $\pi/3$, that is 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 [21] both in theory (in terms of provable space bounds,) and in practice (in terms of experimentally observed space and time requirements).

1.3 Contribution: Our contribution is both analytical and algorithmic. On the analytical side, we explicitly introduce the use of sphere packing bounds in the study of sieve algorithms for lattice problems. Such usage was already implicit in previous work, but somehow obfuscated by the complexity of previous algorithms and analyses. Our simpler algorithms and explicit connection to sphere packings, allows us to relate the performance of sieve algorithms to well studied quantities in the theory of spherical codes, and make use of the best bounds known to date [12]. We remark that these bound are broadly applicable, and can be used to improve the analysis of previous sieving algorithms [4, 21] as well. However, this is not the only source of improvement. In Section 6 we sketch how to apply sphere packing bounds to the analysis of the original sieve algorithm [4, 21], and show that, even using the powerful linear programming bound of [12], only yields provable space and time complexity bounds approximately equal to $2^{1.97n}$ and $2^{3.4n}$, which is still worse than the performance of our theoretical algorithm by an exponential factor.

On the algorithmic side, we introduce a new sieving strategy, as described in the previous section. While at first sight the new strategy may look like a simple reordering of the instructions executed by the original sieve of Ajtai, Kumar and Sivakumar [4], there are deeper algorithmic differences that lead to a noticeable reduction in both the provable space and time complexity. The main source of algorithmic improvement is the way our new sieving strategy deals with useless points, i.e., samples that potentially yield zero vectors in the original sieve. In our algorithms these vectors are immediately recognized and discarded. In the original sieve algorithm these vectors are generated at the outset, and remain undetected during the entire execution, until the algorithm reaches its final stage. Both in the analysis of [4, 21] and in this paper, such useless points are potentially an overwhelming fraction of all samples, leading to a noticeable difference in performance. For a more detailed description of the relevant algorithmic differences between the AKS sieve and our new proposal, the reader is referred to Section 6.

2 Background

In this section we review standard definitions and notation used in the algorithmic study of lattices, mostly following [19].

General: 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 $S \subseteq \mathbb{R}^n$,

[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.

$\mathbf{x} \in \mathbb{R}^n$ 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 $\operatorname{dist}(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|$. 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 usual way: $\operatorname{dist}(\mathbf{x},S) = \operatorname{dist}(S,\mathbf{x}) = \min_{\mathbf{y}\in S}\operatorname{dist}(\mathbf{x},\mathbf{y})$. 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 $\operatorname{span}(\mathbf{S}) = \{\sum_i x_i\mathbf{s}_i : x_i \in \mathbb{R} \text{ for } 1 \le i \le m\}$. For any set of n linearly independent vectors $\mathbf{S}$, we define the half-open parallelepiped $\mathcal{P}(\mathbf{S}) = \{\sum_i x_i\mathbf{s}_i : 0 \le x_i < 1 \text{ for } 1 \le i \le n\}$. Finally, we denote by $\mathcal{B}_n(\mathbf{x},r)$ the closed n-dimensional Euclidean ball of radius r and center $\mathbf{x}$, $\mathcal{B}_n(\mathbf{x},r) = \{\mathbf{w} \in \mathbb{R}^n : \|\mathbf{w}-\mathbf{x}\| \le r\}$. If no center is specified, then the center is zero $\mathcal{B}_n(r) = \mathcal{B}_n(\mathbf{0},r)$.

Lattices: We now describe some basic definitions related to lattices. For a more in-depth discussion, see [18]. An n-dimensional *lattice* is the set of all integer combinations

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

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

$$\begin{aligned}\lambda(\Lambda) &= \min\{\operatorname{dist}(\mathbf{x},\mathbf{y}) : \mathbf{x} \ne \mathbf{y} \in \Lambda\}\\ &= \min\{\|\mathbf{x}\| : \mathbf{x} \in \Lambda \setminus \{\mathbf{0}\}\}\ .\end{aligned}$$

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 basis $\mathbf{B} \in \mathbb{Z}^{n\times n}$, and the goal is to find a vector $\mathbf{x}$ in $\mathcal{L}(\mathbf{B})$ such that $\|\mathbf{x}\| = \lambda(\mathbf{B})$.*

For simplicity in this paper we consider only inputs to SVP where all the entries in $\mathbf{B}$ have bitsize polynomial in n, i.e., $\log(\|\mathbf{B}\|) = \operatorname{poly}(n)$. This allows to express the complexity of SVP simply as a function of a single parameter, the lattice dimension n. All the results in this paper can be easily adapted to the general case by introducing an explicit bound $\log\|\mathbf{B}\| \le M$ on the size of the entries, and letting the time and space complexity bound depend polynomially in M.

Theorem 2.1. *(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 in C is at least ϕ_0. If $0 < \phi_0 < 63°$, then for all sufficiently large n, $A(n,\phi_0) \le 2^{cn}$ for*

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

Notice that when $\phi_0 = 60°$ this is equivalent to the kissing constant:

$$\tau_n = A(n, 60°) \le 2^{0.401n}.$$

We remark that these upper bounds are probably not tight, as there is no known matching lower bound. For example, for the case of the kissing constant, the best currently known lower bounds only proves that $\tau_n > 2^{0.2075+o(1)}$ [5].

3 Algorithms

In this section we describe our two algorithms for the shortest vector problem. For simplicity we assume that these algorithms

- take as input, beside the input basis $\mathbf{B}$, also a parameter $\mu \in [\lambda_1(\mathbf{B}), 1.01\cdot\lambda_1(\mathbf{B})]$ which approximates the length of the shortest nonzero lattice vector within a constant factor 1.01, and
- are only required to produce a nonzero lattice vector of length bounded by μ, possibly larger than λ_1.

[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.

This is without loss of generality because any such algorithm can be turned, using standard techniques, into an algorithm that solves SVP exactly by trying only polynomially many possible values for μ.

In Section 3.1, we describe *List Sieve*. 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 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 vector list L ensures that no two points in the list are close to each other. [3] We use this fact to bound from below the angle between any two list points of similar norm and use Theorem 2.1 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[4] 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 random error vector of length $\|\mathbf{e}\| \leq \xi\mu$ for an appropriate value of $\xi > 0.5$. As before the length of $\mathbf{p}$ is reduced using list points, but instead of adding $\mathbf{p}$ to the list we add the corresponding lattice vector $\mathbf{v} = \mathbf{p} - \mathbf{e}$. We will see that some points $\mathbf{p} = \mathbf{v}_1 + \mathbf{e}_1 = \mathbf{v}_2 + \mathbf{e}_2$ correspond to two different lattice points $\mathbf{v}_1$, $\mathbf{v}_2$ at distance precisely $\|\mathbf{v}_1 - \mathbf{v}_2\| = \lambda_1(\mathbf{B})$ from each other. For example, if $\mathbf{s}$ is the shortest nonzero vector in the lattice, then setting $\mathbf{p} = -\mathbf{e}_1 = \mathbf{e}_2 = \mathbf{s}/2$ gives such a pair of points $\mathbf{v}_1 = \mathbf{0}, \mathbf{v}_2 = \mathbf{s}$. The distance between two points in L is greater than μ or else the algorithm terminates and as a result at most one of the possible lattice vectors $\mathbf{v}_1, \mathbf{v}_2$ is in the list. This property can be used to get an upper bound on the probability of getting a collision.

Unfortunately the introduction of perturbations comes at a cost. As we have discussed above, sieving produces points that are far from L and as a result we can prove a lower bound on the angles between points of similar norm. Indeed after sieving with L the point $\mathbf{p}$ will be far from any point in L. However the point that is actually added to the list is $\mathbf{v} = \mathbf{p} - \mathbf{e}$ which can be closer to L than $\mathbf{p}$ by as much as $\|\mathbf{e}\| \leq \xi\mu$. That makes the resulting bounds on the angles worse. This worsening gets more and more significant as the norm of the points gets smaller. Fortunately we can also bound the distance between points in L by μ, which gives a good lower bound on the angles between shorter points. The space complexity of the algorithm is determined by combining these two bounds to obtain a global bound on the angle between any two points of similar norm, for any possible norm.

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 ball of radius $\xi\mu$, and setting $\mathbf{p} = \mathbf{e} \bmod \mathbf{B}$. This ensures that, by construction, the ball $\mathcal{B}(\mathbf{p}, \xi\mu)$ 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 ball. 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 of length $\lambda_1(\mathbf{B})$, 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 both $\mathbf{v}$ and $\mathbf{v} + \mathbf{s}$ are within distance $\xi\mu$ from $\mathbf{p}$.

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

[3] 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.

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

Algorithm 1 ListSieve($\mathbf{B}, \mu$) *Output:* $\mathbf{v} : \mathbf{v} \in \mathcal{L}(\mathbf{B}) \wedge \|\mathbf{v}\| \le \mu$ 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 $\exists \mathbf{v}_j \in L : \|\mathbf{v}_i - \mathbf{v}_j\| \le \mu$ 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 $(\exists \mathbf{v}_i \in L : \|\mathbf{p} - \mathbf{v}_i\| \le \delta\|\mathbf{p}\|)$ do
        $\mathbf{p} \leftarrow \mathbf{p} - \mathbf{v}_i$
    end while
    return $\mathbf{p}$
end function
```

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 $\log(n\|\mathbf{B}\|)/\log(1 - 1/n)^{-1} = n^{O(1)}$.

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 outputs $\perp$. In Section 4 we will show how to choose K so that if $\lambda_1(\mathbf{B}) \le \mu \le 1.01\lambda_1(\mathbf{B})$ the algorithm finds a vector with norm bounded by μ with probability exponentially close to 1.

Now we are ready to state our main theorem.

THEOREM 3.1. *Let* ξ *be a real number such that* $0.5 < \xi < 0.7$ *and* $c_1(\xi) = \log(\xi + \sqrt{\xi^2 + 1}) + 0.401$, $c_2(\xi) = 0.5\log(\xi^2/(\xi^2 - 0.25))$. List Sieve *solves* SVP *in the worst case, with probability exponentially close to* 1, *in space* $\tilde{O}(2^{c_1 n})$, *and time* $\tilde{O}(2^{(2\cdot c_1 + c_2)n})$.

Proof. Immediately follows from Theorem 4.2 and Theorem 4.1, proved in Section 4. ■

Setting the parameter ξ: A smaller ξ gives smaller perturbations and reduces the space complexity. On the other hand we need $\xi > 0.5$ to bound the collision probability and consequently the running time. By choosing ξ arbitrarily close to 0.5, we can achieve space complexity 2^{cn} for any $c > \log((1 + \sqrt{5})/2) + 0.401 \approx 1.095$, and still keep exponential running time $2^{O(n)}$, 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 $< 2^{3.199n}$.

3.2 The Gauss Sieve

As we have discussed in the previous section the introduction of perturbations substantially increases the space requirements. In an attempt to make sieving algorithms practical Nguyen and Vidick in [21] have proposed a heuristic variant of AKS that does not use perturbations. Their experiments show that in practice collisions are not common and the algorithm performs quite well. We also introduce a practical variant of our algorithm without perturbations which we call the *Gauss Sieve*.

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

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

```
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
            $\mathbf{v}_{new} \leftarrow$SampleKlein($\mathbf{B}$)
        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
```

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 degrees). 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 any bounds on the probability of collisions or the running time. 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 5) this does not occur, and the running time of the algorithm seems to be between quadratic and cubic in the list size, but we do not know how to prove any worst-case upper bound.

As in [21], we do not use perturbations, and just choose $\mathbf{p} = \mathbf{v}$ at random using Klein's randomized rounding algorithm [15] (denoted *SampleKlein* in the description of the algorithm). However since we cannot prove anything about running time, the choice of sampling algorithm is largely arbitrary. In the context of the Gauss Sieve, not using perturbation has the main practical advantage of allowing to work with lattice points only. This allows an integer only implementation of the algorithm (except possibly for the sampling procedure which may still internally use floating point numbers).

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 a nonzero 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 5).

4 Analysis of *List Sieve*

In this section we prove time and space upper bounds for the *List Sieve* algorithm, assuming it is given as a hint a value μ such that $\lambda_1 \leq \mu \leq 1.01 \cdot \lambda_1$. In subsection 4.1 we use the fact that the list points are far apart to get an upper bound N on the list size. Then in subsection 4.2 we prove that collisions are not too common. We use this fact to prove that after a certain number of samples are processed, the algorithm finds a nonzero vector with norm less than or equal to μ with high probability.

4.1 Space complexity

Theorem 4.1. *The number of points in L is bounded* *from above by* $N = \mathrm{poly}(n) \cdot 2^{c_1 n}$ *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 ball 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 spherical shells

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

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

So, fix a spherical shell $S_i = \{\mathbf{v} \in L\colon R < \|\mathbf{v}\| \le \gamma R\}$ for some $R = \gamma^{i-1}\mu/2$. Consider two arbitrary points $\mathbf{v}_a, \mathbf{v}_b \in S_i$ and let $\phi_{\mathbf{v}_a,\mathbf{v}_b}$ be the angle between them. We will show that

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) \le 1 - \frac{1}{2(\xi+\sqrt{\xi^2+1})^2} + o(1). \tag{4.1}$$

The upper bound on $\cos(\phi)$ is greater than 0.5. (Equivalently the minimum angle is less than 60°.) Therefore, we can safely use Theorem 2.1 with the bound (4.1), and conclude that the number of points in S_i is at most $2^{c_1 n}$ where

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

as stated in the theorem. It remains to prove (4.1).

We use the fact that

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) = \frac{\mathbf{v}_a \cdot \mathbf{v}_b}{\|\mathbf{v}_a\|\|\mathbf{v}_b\|}, \tag{4.2}$$

to transform any upper bound on $\mathbf{v}_a \cdot \mathbf{v}_b$ into a corresponding upper bound on $\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b})$. We give two bounds on $\mathbf{v}_a \cdot \mathbf{v}_b$. First remember that $\|\mathbf{v}_a - \mathbf{v}_b\| > \mu$. Squaring the terms yields $\mathbf{v}_a \cdot \mathbf{v}_b < (\mathbf{v}_a^2 + \mathbf{v}_b^2 - \mu^2)/2$ and we can use (4.2) to get

$$\begin{aligned} \cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) &\le \frac{\mathbf{v}_a^2 + \mathbf{v}_b^2 - \mu^2}{2\|\mathbf{v}_a\|\|\mathbf{v}_b\|} \\ &\le \frac{\mathbf{v}_a^2 + \mathbf{v}_b^2}{2\|\mathbf{v}_a\|\|\mathbf{v}_b\|} - \frac{\mu^2}{2\|\mathbf{v}_a\|\|\mathbf{v}_b\|}. \end{aligned}$$

Since $R < \|\mathbf{v}_a\|, \|\mathbf{v}_a\| \le \gamma R = (1+o(1))R$, we get

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) \le 1 - \frac{\mu^2}{2R^2} + o(1). \tag{4.3}$$

Notice that this bound is very poor when R is large. So, for large R, we bound $\mathbf{v}_a \cdot \mathbf{v}_b$ differently. Without loss of generality, assume that $\mathbf{v}_b$ was added after $\mathbf{v}_a$. As a result the perturbed point $\mathbf{p}_b = \mathbf{v}_b + \mathbf{e}_b$ was reduced with $\mathbf{v}_a$, i.e., $\|\mathbf{p}_b - \mathbf{v}_a\| > \delta\|\mathbf{p}_b\|$. After squaring we get $\mathbf{p}_b \cdot \mathbf{v}_a < ((1-\delta^2)\mathbf{p}_b^2 + \mathbf{v}_a^2)/2$ and therefore

$$\begin{aligned} \mathbf{v}_a \cdot \mathbf{v}_b &= \mathbf{v}_a \cdot \mathbf{p}_b - \mathbf{v}_a \cdot \mathbf{e}_b \\ &< ((1-\delta^2)\mathbf{p}_b^2 + \mathbf{v}_a^2)/2 + \|\mathbf{v}_a\|\xi\mu. \end{aligned}$$

Using (4.2) gives

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) \le \frac{(1-\delta^2)\mathbf{p}_b^2 + \mathbf{v}_a^2}{2\|\mathbf{v}_a\|\|\mathbf{v}_b\|} + \frac{\|\mathbf{v}_a\|\xi\mu}{\|\mathbf{v}_a\|\|\mathbf{v}_b\|}.$$

Since $1 - \delta^2 = o(1)$, we get

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) \le \frac{1}{2} + \frac{\xi\mu}{R} + o(1). \tag{4.4}$$

Combining the two bounds on $\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b})$ we get

$$\cos(\phi_{\mathbf{v}_a,\mathbf{v}_b}) \le \min\left\{1 - \frac{\mu^2}{2R^2}, \frac{1}{2} + \frac{\xi\mu}{R}\right\} + o(1). \tag{4.5}$$

As R increases, the first bound gets worse and the second better. So, the minimum (4.5) 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

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

which, substituted in (4.5), gives the bound (4.1). ■

4.2 Time Complexity In this subsection we prove bounds on the running time and success probability of our algorithm. We recall that the *List Sieve* samples random perturbations $\mathbf{e}_i$ from $\mathcal{B}_n(\xi\mu)$, sets $\mathbf{p}_i = \mathbf{e}_i \bmod \mathbf{B}$ and reduces $\mathbf{p}_i$ with the list L. Then it considers the lattice vector $\mathbf{v}_i = \mathbf{p}_i - \mathbf{e}_i$. At this point, one of the following (mutually exclusive) events occurs:

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

We will prove that if $\lambda_1(\mathbf{B}) \leq \mu \leq 1.01\lambda_1(\mathbf{B})$, 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 (an alternative proof is given in [21]), which will be used to bound the probability of getting collisions.

Lemma 4.1. *Let* $\mathrm{Cap}_{R,h}$ *be the n-dimensional spherical cap with height h of a hypersphere* $\mathcal{B}_n(R)$. *Then for* $R_b = \sqrt{2hR - h^2}$,

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

Proof. The basis of $\mathrm{Cap}_{R,h}$ is an $n-1$ dimensional hypersphere of 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:

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

Therefore

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

■

In the following theorem we assume $\xi < 0.7$ as this yields a slightly simpler proof, and the values of ξ that optimize the space and time complexity of our algorithm satisfy this constraint anyway. The theorem remains valid for larger values of ξ.

Theorem 4.2. *If* $\lambda_1(\mathbf{B}) \leq \mu \leq 1.01\lambda_1(\mathbf{B})$ *and* $0.5 < \xi < 0.7$ *then* List Sieve *outputs a lattice point with norm* $\leq \mu$ *with probability exponentially close to 1 as long as the number of samples used is at least* $K = \tilde{O}(2^{(c_1+c_2)n})$, *where*

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

Proof. Let $\mathbf{s}$ be a shortest nonzero vector in $\mathcal{L}(\mathbf{B})$. Consider the intersection of two n-dimensional balls of radius $\xi\mu$ and centers $\mathbf{0}$ and $-\mathbf{s}$: $I_0 = \mathcal{B}_n(\mathbf{0}, \xi\mu) \cap \mathcal{B}_n(-\mathbf{s}, \xi\mu)$ and also $I_1 = \mathcal{B}_n(\mathbf{0}, \xi\mu) \cap \mathcal{B}_n(\mathbf{s}, \xi\mu) = I_0 + \mathbf{s}$. Notice that $\xi\mu \leq 0.707\lambda_1(\mathbf{B}) < \|\mathbf{s}\|$ and as a result I_0 and I_1 do not intersect.

Consider an error vector $\mathbf{e}_i$ in $I_0 \cup I_1$, and the corresponding perturbed point $\mathbf{p} = \mathbf{v} + \mathbf{e}_i$ generated by the list reduction procedure. The conditional distribution of $\mathbf{v}$ given $\mathbf{p}$, is uniform over the lattice points in $\mathbf{p} - I_0 \cup I_1$. We know that this set contains at least one lattice point $\mathbf{v} = \mathbf{p} - \mathbf{e}_i$. Moreover, if $\mathbf{v} \in \mathbf{p} - I_b$, then $\mathbf{v} - (-1)^b\mathbf{s} \in \mathbf{p} - I_{1-b}$. Finally, notice that the diameter of each $\mathbf{p} - I_b$ is bounded by

$$2\sqrt{(\xi\mu)^2 - \lambda_1^2/4} \leq 2\sqrt{0.707^2 - 0.25}\lambda_1 < \lambda_1.$$

It follows that $\mathbf{v}$ and $\mathbf{v}' = \mathbf{v} - (-1)^b\mathbf{s}$ are the only two lattice vectors in $\mathbf{p} - I_0 \cup I_1$, and at most one of them belongs of L because $\|\mathbf{v} - \mathbf{v}'\| = \lambda_1 \leq \mu$. This proves that, conditioned on $\mathbf{e}_i \in I_0 \cup I_1$, the probability that $\mathbf{v} \in L$ is at most 1/2:

$$Pr[\mathbf{C}|\mathbf{e}_i \in I_0 \cup I_1] \leq 1/2.$$

Now notice that $I \cup I'$ contains four disjoint caps with height $h = \xi\mu - \|\mathbf{s}\|/2 \geq \xi\mu - \mu/2$ on an n-dimensional ball of radius $\xi\mu$. We use Lemma 4.1 to bound from below the probability that $\mathbf{e}_i \in I_0 \cup I_1$:

$$\begin{aligned} Pr[\mathbf{e}_i \in I_0 \cup I_1] &= \frac{4\mathrm{Vol}(\mathrm{Cap}_{\xi\mu, \xi\mu - 0.5\mu})}{\mathrm{Vol}(\mathcal{B}_n(\xi\mu))} \\ &> \left(\frac{\sqrt{\xi^2 - 0.25}}{\xi}\right)^n \cdot \frac{\xi - 0.5}{n\sqrt{\xi^2 - 0.25}} \\ &= 2^{-c_2 n} \cdot \frac{\xi - 0.5}{n\sqrt{\xi^2 - 0.25}}. \end{aligned}$$

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

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

Now given that the probability of event **C'** is at least p, the number of occurrences of **C'** when K samples are processed is bounded from below 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 **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 if List Sieve uses K samples, then, with exponentially high probability, at least $N+1$ of them will not satisfy event $\mathbf{C}$. These events can either be $\mathbf{L}$ or $\mathbf{S}$ events. However the list size cannot grow beyond N, and as a result the number of $\mathbf{L}$ events can be at most N. So event $\mathbf{S}$ will happen at least once with high probability. ■

Theorem 4.2 is used to set the variable K in the *List Sieve* algorithm, in order to ensure success probability exponentially close to 1. A bound on the running time of the algorithm immediately follows.

Corollary 4.1. *The total running time of the algorithm is $2^{(2\cdot c_1+c_2)n}$ where c_1, c_2 are as in theorem 4.2.*

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

5 Practical performance of Gauss Sieve

In this section we describe some early experimental results on *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 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 the experimental evaluation of lattice algorithms [21, 8]. In our experiments we used *Gauss Sieve*, the *NV Sieve* implementation from [21] and the NTL library for standard enumeration techniques [29]. For every $n = 35, \ldots, 63$ we generated 6 random lattices, reduced them using BKZ with window 20, 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.

Termination of Gauss Sieve: As we have already discussed we cannot bound the number of samples required by *Gauss Sieve* to find a nonzero shortest vector with high probability. A natural termination condition is to stop after a certain number c of collisions. In order to get an estimate for a good value of c, we measured the number of collisions before a nonzero shortest vector is found. Although the number of collisions vary from lattice to lattice we have found that setting c to be let's say a $\simeq 10\%$ of the list size, is a conservative choice. The results that follow are based on this termination condition. *Gauss Sieve* found the nonzero shortest vector for 173 out of the 174 of the lattices we tested. Interestingly enough the failure was for dimension 38, and can be probably attributed to the very short list size used by the Gauss Sieve in small dimension.

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 $2^{0.2n}$. 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 near $2^{0.21n}$. The actual measured exponent 0.2 may depend on the input lattice distribution, and it would be interesting to run experiments on other distributions. However, in all cases, we expect the space complexity to be below $2^{0.21n}$. *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.
- *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: Fitting the running time with our simple model 2^{cn} gives $c = 0.52$, which is similar to the experiments of *NV Sieve*. However once more *Gauss Sieve* is better in practice. For example, in dimension $\simeq 40$, the 70-fold improvement on the list size, gives a 250× running time improvement. In Figure 2 we also give the running time of the Schnorr-Euchner (SE) enu-

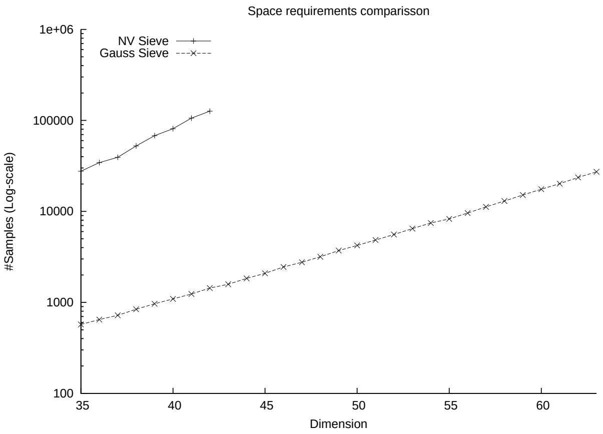

Figure 1: Space requirements of Sieving algorithms

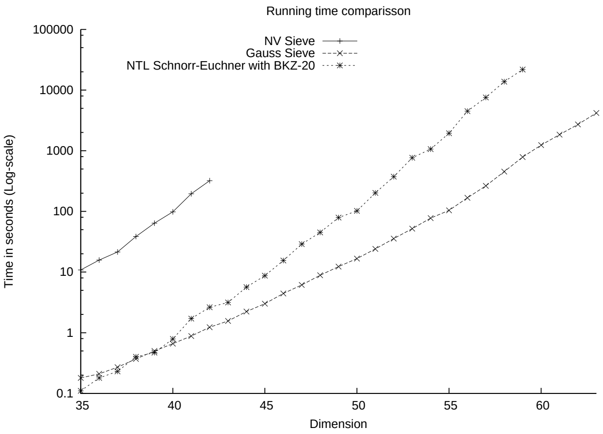

Figure 2: Running Times

meration algorithm as implemented in NTL.[5] This preliminary comparison with SE is meant primarily to put the comparison between sieve algorithms in perspective. In [21], 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. Including our sieve algorithm in the comparison, changes the picture quite a bit: the crossover point between the *Gauss Sieve* and the Schnorr-Euchner reference implementation used in [21] 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. Although currently sieving algorithms cannot compare with more advanced enumeration heuristics (e.g., pruning), we hope that it is possible to develop similar heuristics for sieve algorithms. The development of such heuristics, and a thorough study 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.

6 Comparison with AKS

In this section we give a detailed comparison between our new algorithms and previous work, highlighting the source of improvements both in the theoretical analysis and practical performance. We show how to improve the analysis of AKS using the packing bounds of Theorem 2.1 and a number of other observations. Our analysis shows that the time and space bounds of AKS can be improved to $2^{3.4n}$ and $2^{1.97n}$. However even after the improved analysis, AKS is outperformed by the conceptually simpler *List Sieve*. Intuitively there are two reasons for this:

1. AKS algorithm has to generate all the points from the beginning. As a result the points that will cause collisions (which in a worst case analysis are exponentially more) increase the space bounds. *List Sieve* on the other hand discards collisions as early as possible.

2. AKS sieving cannot provably continue to produce shorter vectors after a certain norm. As a result it generates enough "short" vectors so that the difference of two of them will give a nonzero shortest vector. The point arrangements we get out of this procedure are slightly worse than the sieving of *List Sieve.*

In the following, we assume the reader has some familiarity with [21] and only highlight the elements of the algorithm that are directly relevant to the comparison to our work. Apart from the parameters ξ, μ used in *List Sieve*, AKS requires a parameter γ the shrinking factor. The core procedure of AKS takes as input a set of perturbed points P_i (the norm of the perturbations is bounded by $\xi\mu$) with maximum norm R_i. It generates a maximal subset L_i of P_i with the property that the distance between any two points in L_i is greater than γR_i. Then it uses L_i to reduce the norms of the remaining

[5] 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.

points in $P_i \setminus L_i$ to at most $\gamma R_i + \xi\mu$. The algorithm starts with a large pool of points P_0 with maximum norm R_0. Applying the core procedure described above it generates a sequence of sets $P_0, P_1, P_2, \ldots$ with shorter and shorter maximum norms $R_0, R_1, R_2, \ldots$. Notice that the maximum norm of the sets P_i can be easily reduced near $R_k \simeq \xi\mu/(1-\gamma)$ but not further. The algorithm runs a polynomial number k of steps and acquires a set P_k of perturbed points with maximum norm $R_k \simeq \xi\mu/(1-\gamma)$. After the removal of the perturbations these points correspond to a set of lattice vectors V_k with maximum norm $R_\infty \simeq \xi\mu(1+1/(1-\gamma))$. Then AKS computes the pairwise differences of all the points in V_k to find a nonzero shortest vector.

We bypass most of the details (the reader is referred to [21] for the complete analysis), to show directly how to improve the space and time bounds of AKS. Let the total number of generated points $|P_0| = 2^{c_0 n}$, the probability of a point not being a collision $p = 2^{-c_u n}$, the maximum number of points used for sieving $|L_i| \leq 2^{c_s n}$, and the number of points required in V_k to find a vector with the required norm is $2^{c_R n}$. After k steps of the algorithm the points in P_k will be $2^{c_0 n} - k2^{c_s n}$ and the points in V_k at most $(2^{c_0 n} - k2^{c_s n})/2^{c_u n}$ because of collisions. Notice that we need enough points in P_0 to generate the lists L_i, $c_0 > c_s$ and acquire enough points in V_k, $c_0 - c_u > c_R$. Therefore the space complexity is $2^{c_0 n}$ with $c_0 = max\{c_s, c_R + c_u\}$. Every sieving step reduces $2^{c_0 n}$ points with the $2^{c_s n}$ points of L_i, as a result the sieving procedure needs $2^{(c_0+c_s)n}$ time. Finally we need to acquire the pairwise differences of at least $2^{c_R n}$ points in V_k, but for each point we might need to use $2^{c_u n}$ points from P_k. As a result the total running time for the last step is $2^{2c_R n + c_u n}$. Totally the time complexity is $2^{c_T n}$ with $c_T = max\{c_0 + c_s, 2c_R + c_u\}$.

Now we are ready to introduce the linear programming bounds. Let A a set of points inside an n-dimensional ball of radius R and pairwise distance $\geq r$. Using similar techniques with our main theorems we can show that the angle between any two points with similar norm is $\cos\phi < 1 - r^2/(2R^2)$ and using theorem 2.1 we get that the total number of points in A are bounded by 2^{cn} with $c = 0.401 + \log(R/r)$. Now we can give bounds to all the constants defined above. First $c_u = \log\left(\frac{\xi}{\sqrt{\xi^2 - 0.25}}\right)$ which is identical to c_2 in our analysis. Notice that the points in any set L_i are in a ball of radius R_i and have pairwise distance γR_i therefore $c_s = 0.401 + \log(1/\gamma)$. On the other hand the set V_k has minimum distance $> \mu$ while the maximum radius of the points is $R_\infty \simeq \xi\mu(1+1/(1-\gamma))$ therefore we need $c_R = 0.401 + \log(\xi(1+1/(1-\gamma)))$. To minimize the space constant $c_0 = max\{c_s, c_R + c_u\}$ we set $\xi = 0.71, \gamma = 0.382$ which gives space requirements $2^{1.79n}$ and time $2^{3.58n}$. On the other hand if we want to minimize the time complexity good values are $\xi = 0.69$ and $\gamma = 0.49$ which give space $2^{1.97n}$ and time $2^{3.4n}$.

7 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 [17]. 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 noticeable 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 uses only polynomial space, or even how to get a deterministic algorithm with $2^{O(n)}$ time and space complexity. 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.

Acknowledgments

We thank Phong Nguyen and Thomas Vidick for providing the implementation of the sieve algorithm of [21] which we used for the 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. In *Proceedings of Crypto '07.*

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

[18] 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.

[19] 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.

[20] 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.

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

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

[23] 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.

[24] X. Pujol and D. Stehlé. Rigorous and efficient short lattice vectors enumeration, 2008. In *Proceedings of Asiacrypt '08*, pages 390–405. Springer, 2008.

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

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

[27] O. Regev. Lecture notes on lattices in computer science, 2004. *Available at http://www.cs.tau.ac.il/˜odedr/teaching/lattices_fall_2004/index. html.*

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

[29] 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.

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