Journal of Information Processing Vol.23 No.1 67–80 (Jan. 2015)

[DOI: 10.2197/ipsjjip.23.67]

Regular Paper

An Accelerated Algorithm for Solving SVP Based on Statistical Analysis

Masaharu Fukase[1,a)] Kenji Kashiwabara[2,b)]

Received: December 23, 2013, Accepted: September 12, 2014

Abstract: In this paper, we propose an accelerated algorithm for solving the shortest vector problem (SVP). We construct our algorithm by using two novel ideas, i.e., *the choice of appropriate distributions of the natural number representation* and *the reduction of the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors*. These two ideas essentially depend on statistical analysis. The first technique is to generate lattice vectors expected to be short on a particular distribution of natural number representation. We determine the distribution so that more very short lattice vectors have a chance to be generated while lattice vectors that are unlikely to be very short are not generated. The second technique is to reduce the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors. For that, we restrict the insertion index of a new lattice vector. We confirmed by theoretical and experimental analysis that the smaller the sum is, the more frequently a short lattice vector tends to be found. We solved an SVP instance in a higher dimension than ever, i.e., dimension 132 using our algorithm.

Keywords: lattice, SVP, Gram-Schmidt orthogonalized vectors, RSR, normal distribution

1. Introduction

An integer lattice L is the set of all linear combinations with integer coefficients of a set of linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{Z}^m$. $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is called a basis of the lattice L. One of the most famous computational problems concerning lattices is the shortest vector problem (SVP). The SVP is the problem of finding a shortest nonzero lattice vector for a given lattice. The SVP was proved to be NP-hard under randomized reduction in Ref. [1].

The SVP is closely related to breaking lattice-based public key cryptosystems. We can estimate the security of lattice-based public key cryptosystems using SVP algorithms [3], [4]. The Lenstra-Lenstra-Lovász (LLL) algorithm [8] was a major breakthrough in SVP algorithms. It generates a reduced basis of proven quality in polynomial time. The Block Korkin-Zolotarev (BKZ) algorithm [14] combines the LLL algorithm with an enumeration technique in low dimensional sublattices. Although there is no guaranteed run time bound for the BKZ algorithm, it works better than the LLL algorithm in practice. The Random Sampling Reduction (RSR) algorithm [15] combines the BKZ algorithm with the Sampling Algorithm (SA) [15] that generates a lattice vector from a particular set of lattice vectors. Note that the BKZ algorithm was updated as BKZ 2.0 [3], in which the enumeration is much accelerated by Gama-Nguyen-Regev extreme pruning [5].

We developed an accelerated algorithm for solving SVP by extending Schnorr's sampling technique in RSR. We construct our algorithm by using the following two novel ideas, i.e., *appropriate distributions of the natural number representation* and *the reduction of the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors*. These two ideas depend on statistical analysis. We reduce the expected value of the squared lengths of generated vectors using these techniques.

Appropriate distributions of the natural number representation

We seek optimal linear combinations of basis vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ in order to obtain a very short lattice vector. We can represent a lattice vector by some sequence of natural numbers via the Gram-Schmidt orthogonalization. We call this representation the natural number representation. We determine appropriate distributions of the natural number representation so that the expected value of the squared length of the generated lattice vector gets smaller. Then, the density of nonzero numbers in the natural number representation tends to be low. By determining the appropriate distribution of the natural number representation, we can generate short lattice vectors.

In Ref. [15], a lattice vector whose natural number representation consists of only 0-1 sequence is considered. We consider not only 0-1 sequences but also sequences consisting of natural numbers, i.e., 0-1-2 sequence for example. A similar concept to the natural number representation was indicated in Ref. [9]. However, this concept did not consider the appropriate distribution of natural number representation.

The reduction of the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors

We increase the frequency with which a short lattice vector is found by reducing the sum of the squared lengths of the

[1] Dokkyo University, Souka, Saitama 340–0042, Japan
[2] Department of General Systems Studies, The University of Tokyo, Meguro, Tokyo 153–8902, Japan
[a)] fukase@dokkyo.ac.jp
[b)] kashiwa@idea.c.u-tokyo.ac.jp

Gram-Schmidt orthogonalized vectors. We call our method the Restricting Reduction (RR) Algorithm. We confirm by experimental analysis that the smaller the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors is, the more frequently a short lattice vector tends to be found. We show that the distribution of the squared lengths of short candidate vectors approximates the normal distribution because the Gram-Schmidt coefficients of a generated lattice vector can be supposed to be statistically independent with respect to indices. The mean value of the normal distribution can be represented by the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors. According to our statistical analysis, the mean value is $1/12$ times the sum under some assumptions. Then, the smaller the sum is, the smaller the expected value of the squared lengths of short candidate vectors is.

The RR algorithm has the following two steps.

(1) We restrict the indices of vectors in the basis where a short candidate vector can be inserted.

(2) We increase the restriction index.

We reduce the sum by the RR algorithm.

In the sampling technique in Ref. [15], a short candidate vector is immediately inserted to a basis and the generating system consisting of a basis and a short candidate vector is reduced by BKZ. This process might increase the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors in the short term while the very first vectors in the basis might be improved.

We also show a possible extension of the RR algorithm, which we call the Extended Restricting Reduction (ERR) algorithm. By this extension, we aim to to more efficiently generate short candidate vectors and more strongly reduce the sum.

We experimentally confirmed that rapid speedups have been achieved by using above two ideas. We applied our algorithm to the SVP challenge from Technical University of Darmstadt [13], which was opened in 2010 for the sake of assessing the performance of SVP algorithms and has prompted intense competition among many researchers concerning lattices. We solved an SVP in dimension 132, which is the highest dimension of the SVP instances solved ever [13].

The remainder of this paper is organized as follows. In Section 2, we explain some basic concepts of lattices and Schnorr's sampling technique. In Section 3, we define the set of lattice vectors from which we generate short candidate vectors based on natural number representation of lattice vectors. In Section 4, we theoretically and experimentally show the effectiveness of generating a lattice basis whose sum of the squared lengths of the Gram-Schmidt orthogonalized vectors is small. In Section 5, we show how to reduce the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors. In Section 6, we show the results of applying our techniques to the SVP challenge. In Section 7, we examine the application possibility of our techniques to other types of lattices. In Section 8, we extend the RR algorithm and show the complete algorithm that we used to solve an SVP in the highest dimension of the SVP instances solved ever. In Section 9, we compare our algorithm with the RSR algorithm and a variant of it and show an advantage of our algorithm in terms of efficiency.

2. Preliminaries

2.1 Lattice

Given a set of n linearly independent vectors $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{Z}^{m\times n}$, the *integer lattice* $L \subset \mathbb{Z}^m$ spanned by B is defined as the set $L(B) = \{B\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^n\}$ of all integral combinations of $\mathbf{b}_i$'s. The integer n is called the *dimension* of L. When $n = m$, we say that L is *full-dimensional*. The ordered set of vectors $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{Z}^{m\times n}$ is called a *basis* of L. In this paper we concentrate on full-dimensional integer lattices. A lattice has infinitely many bases that generate the lattice when $n \geq 2$. The *volume* of a lattice $L = L(B)$, denoted by $\mathrm{Vol}(L)$, is defined as the volume of the parallelepiped spanned by the columns of B, i.e., $\mathrm{Vol}(L) = \mathrm{Vol}(\{B\mathbf{x} \mid \mathbf{x} \in [0,\ 1)^n\})$. The volume is a lattice invariant, i.e., it does not depend on any particular basis.

For a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$, the corresponding *Gram-Schmidt orthogonalized vectors* $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^* \in \mathbb{R}^n$ are defined by $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$ with $\mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle$ where $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i$ is the inner product in $\mathbb{R}^n$. For every i, $\mathbf{b}_i^*$ is the component of $\mathbf{b}_i$ that is orthogonal to $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. Consequently, vectors $\mathbf{b}_i^*$ and $\mathbf{b}_j^* (j \neq i)$ are orthogonal. We can compute the volume of the lattice by the product of the lengths of the orthogonalized vectors $\mathrm{Vol}(L(B)) = \prod_{i=1}^n \|\mathbf{b}_i^*\|$ where $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^n x_i^2}$ is the *Euclidean norm*.

Let $\mathbf{v} = B\mathbf{x}$ with $\mathbf{x} \in \mathbb{Z}^n$ be a vector in the lattice generated by the basis B. From the definition of the Gram-Schmidt orthogonalized vectors, we can represent $\mathbf{v} \in L(B)$ with $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ and $\mu_{i,j}$ of B, i.e., $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^*$ with $\boldsymbol{\nu} \in \mathbb{R}^n$ such that $\nu_j = \sum_{i=1}^n x_i \mu_{i,j}$. Because $\mathbf{b}_j^*$ are orthgonally pairwise, $\|\mathbf{v}\|^2 = \sum_{j=1}^n \nu_j^2 \|\mathbf{b}_j^*\|^2$. This equation means that for a lattice vector $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^*$ to be short, $|\nu_j|$ need to be small. In the following, we call ν_j *the Gram-Schmidt coefficients* of $\mathbf{v}$.

Let $\pi_i : \mathbb{R}^n \to \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$ be the orthogonal projection. That is, $\pi_i(\mathbf{v}) = \mathbf{v} - \sum_{j=1}^{i-1} (\langle \mathbf{v}, \mathbf{b}_j^* \rangle / \|\mathbf{b}_j^*\|^2) \mathbf{b}_j^*$. The projection of a lattice L onto the i-th orthogonal complement $\mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$ is $\pi_i(L) = \{\pi_i(\mathbf{v}) | \mathbf{v} \in L\}$. In our algorithm, we repeatedly find a non-zero lattice vector $\mathbf{v}$ with smaller $\|\pi_i(\mathbf{v})\|^2 = \sum_{j=i}^n \nu_j^2 \|\mathbf{b}_j^*\|^2$ in each i-th orthogonal complement.

We denote the length of the shortest nonzero lattice vector in a lattice L by $\lambda_1(L)$. *The shortest vector problem (SVP)* is defined as follows:

Definition 1 **(SVP)** Given a lattice basis B, find a nonzero lattice vector $B\mathbf{x}$ such that $\|B\mathbf{x}\| = \lambda_1(L(B))$.

A known estimation of the length of the shortest vector is $(1/\sqrt{\pi})\Gamma(n/2+1)^{1/n} \cdot (\mathrm{Vol}(L))^{1/n}$ [12]. Here, $\Gamma(n/2+1)$ is the Gamma function for $n/2+1$. In the following, we call the amount $(1/\sqrt{\pi})\Gamma(n/2+1)^{1/n} \cdot (\mathrm{Vol}(L))^{1/n}$ Gaussian Heuristic and denote it as $GH(L)$ following Ref. [5] et al. $GH(L)$ is the radius of the n-ball whose volume is $\mathrm{Vol}(L)$. In the SVP challenge, given a lattice basis B, one is asked to find nonzero lattice vector $B\mathbf{x}$ such that $\|B\mathbf{x}\| < 1.05 \cdot GH(L)$.

2.2 Schnorr's Sampling Technique

In this paper, we extend Schnorr's Sampling Technique.

Schnorr proposed Random Sampling Reduction (RSR) [15]. In RSR, the sampling algorithm (SA) generates a single lattice vector $\mathbf{v}$ satisfying the following Eq. (1) for some $1 \le u \le n$.

$$|\nu_j| \le \begin{cases} \frac{1}{2} & \text{for } j < n-u \\ 1 & \text{for } n-u \le j < n \end{cases}, \quad \nu_n = 1. \tag{1}$$

Because the number of the candidates for ν_j with $|\nu_j| \le 1/2$ is 1 and the number of the candidates for ν_j with $|\nu_j| \le 1$ is 2, there are 2^u distinct lattice vectors satisfying Eq. (1). Let $S_{u,B}$ be the set of lattice vectors in $L(B)$ satisfying Eq. (1) for specified u. Here, B is assumed to be reduced by some lattice basis reduction algorithm such as LLL and BKZ. It is well known that the initial vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_k^*$ for some $1 \le k < n$ are longer than subsequent vectors $\mathbf{b}_j^*$ for $j > k$ if B is reduced [4], [9], [15]. So Gram-Schmidt coefficients $\nu_1, \ldots, \nu_k$ have a larger impact on the overall length of $\mathbf{v}$ than ν_j for $j > k$. Recall that for a lattice vector $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^*$ to be short, each $|\nu_j|$ needs to be small. Then, it is reasonable to assume that a lattice vector $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^*$ in $S_{u,B}$ for some $1 \le u \le n$ is likely to be short.

Algorithm 1 Sampling Algorithm (SA)

Input:
B : a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$
u : an integer such that $1 \le u < n$
Output:
$\mathbf{v}$: a lattice vector satisfying Eq. (1).

```
compute μ_{i,j} such that μ_{i,j} = ⟨b_i, b*_j⟩/⟨b*_j, b*_j⟩
v := b_n
for j = 1, ..., n − 1 do
    μ_j := μ_{n,j}
end for
for i = n − 1, ..., 1 do
    select y ∈ Z randomly such that |μ_i − y| ≤ { 1/2 if i < n − u ; 1 if i ≥ n − u }
    v := v − y b_i
    for j = 1, ..., i − 1 do
        μ_j := μ_j − y μ_{i,j}
    end for
end for
```

Given a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$, RSR generates by calling SA up to 2^u distinct lattice vectors $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^*$ satisfying Eq. (1) until a vector $\mathbf{v}$ with $\|\mathbf{v}\|^2 < 0.99\|\mathbf{b}_1\|^2$ is found. Subsequently RSR inserts the vector found by SA into the basis, and BKZ is used to reduce the new basis consisting of $\mathbf{v}, \mathbf{b}_1, \ldots, \mathbf{b}_n$. This random sampling by SA and BKZ processes are iterated several times.

3. Determining a Distribution of Natural Number Representation

In order to efficiently generate a very short lattice vector, we enumerate all candidates of optimal linear combinations of basis vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$. We provide the candidates in terms of the distribution of natural number representation of lattice vectors. In the following, we explain how to determine the appropriate distribution of natural number representation of lattice vectors as our first novel idea and the significance of it. First, we define *the natural number representation* of a lattice vector $\mathbf{v} \in L(B)$ as follows:

Definition 2 **(The Natural Number Representation)** Let B be a lattice basis. Given a lattice vector $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^* \in L(B)$, the natural number representation of $\mathbf{v}$ is $\mathbf{z}(\mathbf{v}) \in \mathbb{N}^n$ such that $-(z_j+1)/2 < \nu_j \le -z_j/2$ or $z_j/2 < \nu_j \le (z_j+1)/2$.
Here, $\mathbb{N}$ denotes the set $\{0, 1, 2, 3, \ldots\}$. z_j is well defined because the above intervals $(-(z_j+1)/2, -z_j/2]$ and $(z_j/2, (z_j+1)/2]$ cover all the real numbers exactly once over $z_j \in \mathbb{N}$. We have the following theorem.

Theorem 1 The map $f : L(B) \to \mathbb{N}^n$ given by $f(\mathbf{v}) = \mathbf{z} \in \mathbb{N}^n$ such that $-(z_j+1)/2 < \nu_j \le -z_j/2$ or $z_j/2 < \nu_j \le (z_j+1)/2$ is a bijection, where $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^* \in L(B)$.

Proof: Consider distinct two lattice vectors $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^*$ and $\mathbf{v}' = \sum_{j=1}^n \nu_j' \mathbf{b}_j^*$. Let i be the largest index for which ν_i and ν_i' differ. ν_i and ν_i' differ by an integer, otherwise it is impossible that both $\mathbf{v}$ and $\mathbf{v}'$ are lattice vectors. From the definition of $\mathbf{z}(\mathbf{v})$, $\mathbf{z}(\mathbf{v})$ and $\mathbf{z}(\mathbf{v}')$ differ at least at the index i. Therefore, the map is an injection.

Next, consider $\mathbf{z} \in \mathbb{N}^n$. Given $\mathbf{z}$, we have the lattice vector $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^*$ that corresponds to $\mathbf{z}$ as follows. For the last nonzero z_{j_0} with some j_0, we set u_{j_0} in the order $\{1, -1, 2, -2, 3, -3, \ldots\}$ when z_{j_0} is taken in ascending order $\{1, 2, 3, 4, 5, 6, \ldots\}$. For $j > j_0$, we set u_j to 0. For $j < j_0$, we set u_j to $-\lceil \sum_{i=j+1}^n u_i \mu_{i,j} \rfloor + (-1)^{z_j} \lceil z_j/2 \rceil$ if $u_i \ge 0$ and $-\lceil \sum_{i=j+1}^n u_i \mu_{i,j} \rfloor + (-1)^{z_j+1} \lceil z_j/2 \rceil$ if $u_i < 0$ where $\lceil x \rfloor$ is obtained by rounding x to the closest integer as defined by $\lceil x \rfloor = \lceil x - 1/2 \rceil$ and $\mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle$. Then, the natural number representation of a lattice vector $B\mathbf{u}$ is $\mathbf{z}$. Thus, the map is a surjection. These prove the theorem. □

We extend the distribution of $\mathbf{v} = \sum_{j=1}^n \nu_j \mathbf{b}_j^*$ generated by SA. While SA generates a lattice vector from $S_{u,B}$ defined in Section 2.2, we generate a lattice vector from the set $V_B(\mathbf{s}, \mathbf{t})$ defined as follows:

Definition 3 $V_B(\mathbf{s}, \mathbf{t}) = \{\mathbf{v} \in L(B) : \mathbf{d}(\mathbf{v}) \le \mathbf{s}, \mathbf{w}(\mathbf{v}) \le \mathbf{t}\}$ with $\mathbf{d} \in \mathbb{N}^c$ and $\mathbf{w} \in \mathbb{N}^c$ such that $d_i(\mathbf{v}) = \#\{z_j(\mathbf{v}) : z_j(\mathbf{v}) = i, 1 \le j \le n\}$ and $w_i(\mathbf{v}) = n - \min\{j : z_j(\mathbf{v}) = i\} + 1$ for some $c \in \mathbb{Z}_+$, all $i \in \mathbb{N}$ such that $i \le c$, and $\mathbf{s} \in \mathbb{N}^c$,$\mathbf{t} \in \mathbb{N}^c$ for c.
Here, $\#S$ denotes the number of elements of a set S, and $\mathbb{Z}_+$ denotes the set $\{1, 2, 3, 4, \ldots\}$. Note that $\sum_{i=0}^c d_i(\mathbf{v}) = n$ for any $\mathbf{v} \in L(B)$. In Appendix A.1, we explain how to sample a lattice vector in $\mathbf{v} \in L(B)$. Also, in Appendix A.2, we explain what the definition of $V_B(\mathbf{s}, \mathbf{t})$ means and the reason why we define $V_B(\mathbf{s}, \mathbf{t})$ in this way.

We observed that the natural number representation $\mathbf{z}(\mathbf{v})$ of a very short lattice vector $\mathbf{v}$ has the following properties for the basis B that is sufficiently reduced:

- z_j is always 0 for some indices j from the front.
- The density of z_j with $z_j \ge 1$ on $\mathbf{z}$ is relatively low.
- The index j with $z_j \ge 2$ is relatively large.

The $\mathbf{b}_j^*$ with a small index j is relatively long if the basis B is sufficiently reduced. Then, $\mathbf{v}$ cannot be short if some z_j for a small index j is large. Therefore, it is plausible that the natural number representation $\mathbf{z}$ of a very short vector has the above properties. We specifically discuss these properties later in terms of the expected values of the squared lengths of generated lattice vectors.

There are two main differences between the search in $V_B(\mathbf{s}, \mathbf{t})$ and the search in $S_{u,B}$. The first one is that a lattice vector $\mathbf{v}$ such

Table 1 Density for a very short vector.

Dim	Average of the density	Standard deviation of the density
120	0.158	0.017
126	0.162	0.023
128	0.165	0.0082
130	0.178	0.017

that $z_j \geq 2$ for some j can be included in V_B while such $\mathbf{v}$ cannot be included in $S_{u,B}$ because $z_j < 2$ for all j for any lattice vector in $S_{u,B}$. By setting s_j for $j \geq 2$ to a nonzero positive number for $V_B(\mathbf{s}, \mathbf{t})$, we can generate a lattice vector such that $z_j \geq 2$ for some j. The second one is that the density of z_j with $z_j \geq 1$ on $\mathbf{z}$ can be adjusted for a lattice vector in $V_B(\mathbf{s}, \mathbf{t})$ while it cannot be adjusted for a lattice vector in $S_{u,B}$, which means many lattice vectors for which the density is high are also included in $S_{u,B}$.

We experimentally confirmed that the density for a very short vector tends to be low. We calculated the density for a very short vector which satisfies the goal norm of the SVP challenge in each of dimensions 120, 126, 128, and 130. We calculated the density as $\#\{j : z_j \geq 1\}/n$. In each dimension, we calculated the density for 10 reduced bases and took the average. **Table 1** shows the average and the standard deviation of the density in each dimension. As we see in Section 9, it seems that we need to set u for $S_{u,B}$ to about 30 even in dimensions around 80. In higher dimensions, we might need to set u to ≥ 30. This means that vectors with the density $\geq 30/120 = 0.25$ are sampled in dimension 120, for example. From the tendency of the density for a very short vector, it seems to be inefficient to sample vectors with such density as ≥ 0.25.

By setting s_j to a relatively small number compared with t_j for any $j > 0$, we can generate only lattice vectors for which the density is relatively low. Thus, the two differences above make the possibility of the success of the search much higher.

Note that the density of z_j such that $z_j = r \in \mathbb{Z}_+$ becomes lower as r increases for a very short lattice vector, so that we should set s_r to a smaller number. We can explain that the natural number representation of a very short lattice vector tends to have the above property as follows.

In our analysis, we generalize the following Randomness Assumption (RA), on which the analysis of RSR [15] depends.

Randomness Assumption (RA) The Gram-Schmidt coefficients ν_j of the vectors $\mathbf{v} = \sum_{j=1}^{n} \nu_j \mathbf{b}_j^*$ generated by SA are uniformly distributed in $[-1/2, 1/2]$ for $j < n - u$, and in $[-1, 1]$ for $n - u \leq j < n$ and statistically independent with respect to j.

We consider the generalized range of ν_j compared with Ref. [15]. Therefore, we introduce an assumption by generalizing RA as follows:

Assumption 1 The Gram-Schmidt coefficients ν_j of the generated vectors $\mathbf{v} = \sum_{j=1}^{n} \nu_j \mathbf{b}_j^*$ are uniformly distributed in $(-(z_j + 1)/2, -z_j/2]$ and $(z_j/2, (z_j + 1)/2]$ and statistically independent with respect to j.

First, we have the following Proposition 1:

Proposition 1 Let $\mathbf{v}$ be a lattice vector and let $\mathbf{z}$ be a natural number representation of $\mathbf{v}$. Under Assumption 1, $E[\|\mathbf{v}\|^2] = \sum_{j=1}^{n} (3z_j^2 + 3z_j + 1)\|\mathbf{b}_j^*\|^2/12$.

Proof: From the definition of $\mathbf{z}$, $\nu_j \in (-(z_j + 1)/2, -z_j/2]$ or $\nu_j \in (z_j/2, (z_j + 1)/2]$ for all j. By Assumption 1, ν_j is uniformly distributed in $(-(z_j + 1)/2, -z_j/2]$ and $(z_j/2, (z_j + 1)/2]$. By symmetry, it is enough to consider only the range $(z_j/2, (z_j + 1)/2]$. The probability density function $f(x)$ of ν_j is $f(x) = 2$ for $z_j/2 \leq x \leq (z_j + 1)/2$, and the probability density function $g(x)$ of ν_j^2 is $g(x) = 1/\sqrt{x}$. Then, $E[\nu_j^2] = \int_{(z_j/2)^2}^{((z_j+1)/2)^2} x(1/\sqrt{x})dx = 2/3[x^{3/2}]_{(z_j/2)^2}^{((z_j+1)/2)^2} = ((z_j + 1)^3 - z_j^3)/12 = (3z_j^2 + 3z_j + 1)/12$. Then, $E[\|\mathbf{v}\|^2] = \sum_{j=1}^{n} (3z_j^2 + 3z_j + 1)\|\mathbf{b}_j^*\|^2/12$. □

While we need to generate a sufficient number of short candidate vectors, we need to reduce $E[\|\mathbf{v}\|^2]$ as possible. From the above, $E[\nu_j^2] = (3z_j^2 + 3z_j + 1)/12$ is in order $1/12, 7/12, 19/12, \ldots$ when z_j is in order $0, 1, 2, \ldots$. Schnorr's geometric series assumption (GSA) [15] states $\mathbf{b}_j^*$ gets shorter as j increases. Then, we should set z_j to 1 or 2 at a higher frequency at large indices where $\|\mathbf{b}_j^*\|^2$ is relatively small.

Regarding how to set $\mathbf{s}$ and $\mathbf{t}$ in $V_B(\mathbf{s}, \mathbf{t})$, we set $\mathbf{s}$ and $\mathbf{t}$ to $(n, 13, 1)$ and $(n, 55, 15)$, respectively, for an n-dimensional SVP challenge. This determination is based on empirical observations as in Appendix A.2. We should not set z_j to 1 or 2 at a high frequency at relatively small indices since $\|\mathbf{b}_j^*\|^2$ is relatively large. Therefore, we do not use $\mathbf{v}$ in $V_B(\mathbf{s}, \mathbf{t})$ such that $z_j \geq 1$ appears at a high frequency at relatively small indices. In fact, we need to ignore many such vectors in $V_B(\mathbf{s}, \mathbf{t})$ since $\#V_B(\mathbf{s}, \mathbf{t})$ with $\mathbf{s} = (n, 13, 1)$ and $\mathbf{t} = (n, 55, 15)$ is too large. Thus, we use heuristic methods to set $\mathbf{s}$ and $\mathbf{t}$ and select vectors which are likely to be short in $V_B(\mathbf{s}, \mathbf{t})$.

In an exceptional case, however, we used a more general approach in solving a 132 dimensional SVP challenge. In the following, we explain the method.

(1) We select a lattice basis B approximating GSA.
(2) We determine the number of short candidate vectors to be generated. This is a trade-off between the running time and the length of the shortest one among all short candidate vectors. In our implementation, we set the number to 5×10^7.
(3) We set $\mathbf{s}$ and $\mathbf{t}$ so that $V_B(\mathbf{s}, \mathbf{t})$ includes a sufficiently larger number of vectors than the above number. In dimension 132, we set $\mathbf{s}$ and $\mathbf{t}$ to $\mathbf{s} = (130, 19, 2)$ and $\mathbf{t} = (130, 35, 12)$, respectively.
(4) Among all vectors in $V_B(\mathbf{s}, \mathbf{t})$, we select those of which the expected values of the lengths are relatively small in advance of the search. Specifically, in the above example, we select the first 5×10^7 vectors in order starting with the smallest expected value. The expected values can be computed from Proposition 1 and G-S lengths of B.
(5) We store the natural number representations of those vectors in a list and utilize the list in the search.

We compute the list for a lattice basis B approximating GSA at the beginning of the execution of the program. Then, we apply the list to an arbitrary basis during execution of the program. However, it is impractical to generate a list with a huge number of elements, e.g., 5×10^7 elements. In order to avoid this, we generate two or three lists with smaller number of elements and use elements with a shorter length in each list. Then, we obtain all of the natural number representations needed by combining elements in these lists.

4. Relation Between the Success of Search for a Very Short Lattice Vector and the Sum of Squared G-S Lengths

In this section, we explain why we aim to decrease the sum of the squared lengths of the Gram-Schmidt orthogonalized vectors. In the next section, we provide the Restricting Reduction (RR) algorithm, by which a new lattice basis has the smaller sum of the squared lengths of the Gram-Schmidt orthogonalized vectors. We briefly call the lengths of the Gram-Schmidt orthogonalized vectors G-S lengths in the following.

We use the sum of squared G-S lengths, i.e., $\sum_{j=1}^{n}\|\mathbf{b}_j^*\|^2$, as a measurement of how good the basis is. We observed that the smaller $\sum_{j=1}^{n}\|\mathbf{b}_j^*\|^2$ is, the more frequently a short lattice vector tends to be found. We can explain this tendency as follows.

Let k be the minimum index j such that $\nu_j \geq 1$ for any lattice vector $\mathbf{v} = \sum_{j=1}^{n}\nu_j\mathbf{b}_j^*$ in $V_B(\mathbf{s},\mathbf{t})$. Under Assumption 1, $E[\nu_j^2] = \int_0^{(1/2)^2} x(1/\sqrt{x})dx = 1/12$ for $j < k$ in the same way as in Section 3. Then,

$$E[\|\mathbf{v}\|^2] = 1/12\sum_{j=1}^{k-1}\|\mathbf{b}_j^*\|^2 + E[\sum_{j=k}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2]. \quad (2)$$

In the form above, the value $\sum_{j=k}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2$ cannot be easily estimated because the value of this part depends on the natural number representation. Recall that $\mathbf{b}_j^*$ tends to get shorter as j increases if B is reduced. So, the contribution of the term $\sum_{j=k}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2$ to the whole value of $E[\|\mathbf{v}\|^2]$ is small enough to ignore as long as ν_j^2 for $j \geq k$ are not too large and k is not too small. Then, it is allowable to replace the range $(-(z_j+1)/2, -z_j/2]$ and $(z_j/2, (z_j+1)/2]$ of ν_j for $j \geq k$ with the range $(-1/2, 1/2]$. Therefore, it makes sense to introduce the following assumption by replacing $E[\sum_{j=k}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2]$ with $1/12\sum_{j=k}^{n}\|\mathbf{b}_j^*\|^2$.

Assumption 2

$$E[\|\mathbf{v}\|^2] \approx 1/12\sum_{j=1}^{n}\|\mathbf{b}_j^*\|^2. \quad (3)$$

Thus, the mean value of $\|\mathbf{v}\|^2$ approximates $\sum_{j=1}^{n}\|\mathbf{b}_j^*\|^2/12$.

Under Assumption 1, $\nu_j\|\mathbf{b}_j^*\|$ are statistically independent and uniformly distributed in $[-\|\mathbf{b}_j^*\|/2, \|\mathbf{b}_j^*\|/2]$ for $j < k$. We replace $k-1$ with n similarly to the above, i.e., we assume that $\nu_j\|\mathbf{b}_j^*\|$ are statistically independent and uniformly distributed in $[-\|\mathbf{b}_j^*\|/2, \|\mathbf{b}_j^*\|/2]$ for $j \leq n$. From the fact that the probability density function of $x = \nu_j^2$ is $g(x) = 1/\sqrt{x}$ for $0 < x \leq (1/2)^2$, $E[\nu_j^2] = \int_0^{(1/2)^2} x(1/\sqrt{x})dx = 1/12$ and $V[\nu_j^2] = \int_0^{(1/2)^2}(x^2 - 1/12)(1/\sqrt{x})dx = 1/180$. Therefore, the mean value $E[\nu_j^2\|\mathbf{b}_j^*\|^2]$ is $\|\mathbf{b}_j^*\|^2/12$ and the variance $V[\nu_j^2\|\mathbf{b}_j^*\|^2]$ is $\|\mathbf{b}_j^*\|^4/180$.

We can estimate the distribution of $\|\mathbf{v}\|^2$ by applying the generalized central limit theorem because n is large enough and $\nu_j\|\mathbf{b}_j^*\|$ are statistically independent with respect to j. By applying the generalized central limit theorem, we can introduce the following assumption.

Assumption 3 The distribution of $\sum_{j=1}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2$ follows the normal distribution $N(\mu,\sigma^2)$ with $\mu = \sum_{j=1}^{n}\|\mathbf{b}_j^*\|^2/12$ and $\sigma = (\sum_{j=1}^{n}\|\mathbf{b}_j^*\|^4/180)^{1/2}$.

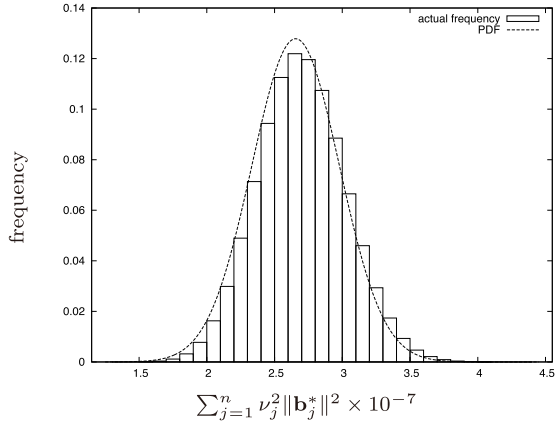

Fig. 1 The actual distribution of $\sum_{j=1}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2$.

Therefore, by considering the probability density function $\phi(x) = (1/\sqrt{2\pi}\sigma)\exp[-(x-\mu)^2/2\sigma^2]$ of the normal distribution function, the probability that the length of a generated lattice vector $\mathbf{v}$ is shorter than γ is

$$\begin{aligned}&(1/\sqrt{2\pi}\sigma)\int_{-\infty}^{\gamma^2}\exp[-(x-\mu)^2/2\sigma^2]dx\\ &= (1/2)(1+\mathrm{erf}((\gamma^2-\mu)/\sqrt{2}\sigma)).\end{aligned} \quad (4)$$

Here, erf is the error function such that $\mathrm{erf}(x) = (2/\sqrt{\pi})\int_0^x \exp[-t^2]dt$. From Eq. (4), the probability that a short lattice vector is found can be computed by the mean value μ and the variance σ^2. We emphasize that the mean value μ is determined by the sum of squared G-S lengths and the variance σ^2 is determined by the sum of the fourth power of G-S lengths. Therefore, the probability is determined by the sum of squared G-S lengths and the sum of the fourth power of G-S lengths.

Here, we experimentally verify Assumption 3. Although Assumption 3 does not hold strictly, it is close enough. For example, see **Fig. 1**. Figure 1 shows the actual distribution of $\sum_{j=1}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2$ for $\mathbf{v} \in V_B(\mathbf{s},\mathbf{t})$ with $\mathbf{s} = (120, 7)$, $\mathbf{t} = (120, 30)$. Here, B is a basis for a 120 dimensional SVP challenge problem. The actual mean value and variance of $\sum_{j=1}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2$ were 2.7×10^7 and 1.049×10^{13}, respectively. On the other hand, the theoretical values of μ and σ^2 of $\sum_{j=1}^{n}\nu_j^2\|\mathbf{b}_j^*\|^2$ were 2.656×10^7 and 1.047×10^{13}, respectively. In Fig. 1, the probability density function of $N(2.656 \times 10^7, 1.047 \times 10^{13})$ is also drawn. We can see that the distribution approximates $N(2.656 \times 10^7, 1.047 \times 10^{13})$.

Our claim is that the smaller $\sum_{j=1}^{n}\|\mathbf{b}_j^*\|^2$ is, the higher the probability is. In the following, we experimentally confirm the above theoretical analysis. **Figure 2** shows the relation between the expected probability and the sum of basis for the 120 dimensional SVP challenge problem. Here, the probability is computed from Eq. (4), and the target length is $\gamma = 1.05 \cdot GH(L)$. From Fig. 2, we can see that the probability gets higher for the smaller $\sum_{j=1}^{n}\|\mathbf{b}_j^*\|^2$. The graph of Fig. 2 is not a smooth curve but has some steps since the probability is determined by non-fixed parameters μ and σ^2.

Note that the probability calculated based on Eq. (4) itself does not provide an accurate approximation of the actual frequency with which a very short lattice vector is found. The gap between the calculated probability and the actual frequency seems to be caused by the following factors:

(1) treating the distribution of the squared lengths of lattice vec-

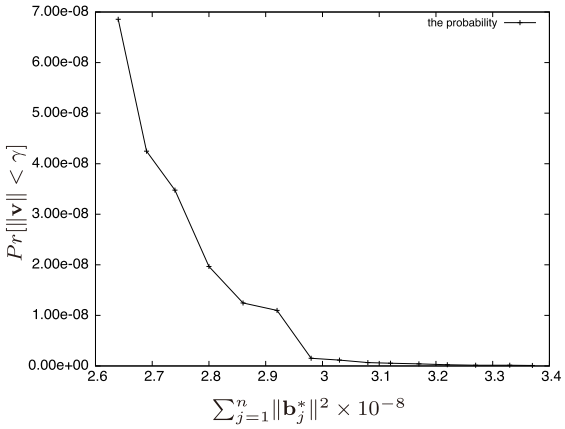

Fig. 2 The relation between the probability that $Pr[\|\mathbf{v}\| < \gamma = 1.05 \cdot GH(L)]$ and the sum of squared G-S lengths.

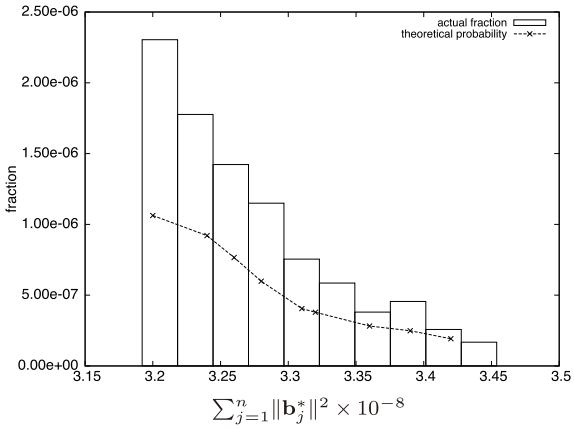

Fig. 3 The relation between the fraction and the sum of squared G-S lengths.

tors as the continuous probability distribution as above although it is in fact the discrete probability distribution.

(2) assuming that ν_j are statistically independent with respect to indices and uniformly distributed in some range.

(3) assuming n is large enough for the generalized central limit theorem.

(4) replacing the range $(-(z_j+1)/2, -z_j/2]$ and $(z_j/2, (z_j+1)/2]$ of ν_j for $j \geq k$ with the range $(-1/2, 1/2]$ in Eq. (2).

However, the calculated probability shows the tendency of the magnitude relation of the frequencies. We can see that the probability increases as the sum of squared G-S lengths decreases in Fig. 2, which derives the claim that the smaller $\sum_{j=1}^{n} \|\mathbf{b}_j^*\|^2$ is, the higher the probability is. We can also see that the probability gets larger for larger $\sigma = (\sum_{j=1}^{n} \|\mathbf{b}_j^*\|^4/180)^{1/2}$ from Eq. (4) if $\mu = \sum_{j=1}^{n} \|\mathbf{b}_j^*\|^2/12$ is the same.

We also experimentally calculated the fraction of the number of found short vectors and the number of all generated vectors. We ran our algorithm, which we will explain in Section 5, for a SVP challenge lattice of dimension 128 in about one day. We used one process on a 2.6 GHz laptop computer "Let'snote" (Intel Core i5). We note that a lattice basis is transformed many times in our algorithm, and consequently the sum of squared G-S lengths continues to be changed. We calculate the correlation between the fraction and the sum of squared G-S lengths. As a result, the calculated correlation coefficient is −0.892, i.e., there was a strong correlation between the fraction and the sum of squared G-S lengths. Here, by a short vector, we mean a lattice vector which is shorter or slightly longer than $\mathbf{b}_1$, of which length was always 3128.62 during program execution. **Figure 3** shows the relation between the fraction and the sum of squared G-S lengths. We divided the range of the sum of squared G-S lengths into ten parts with equal intervals in Fig. 3. We computed the fraction for each range of the sum. Figure 3 supports the claim that the smaller $\sum_{j=1}^{n} \|\mathbf{b}_j^*\|^2$ is, the higher the probability that a short lattice vector is found is. In Fig. 3, we also show the theoretical probabilities computed from Eq. (4) for actual bases, of which sums are in the range of the graph. We can see that the tendency of the magnitude relation is the same although the theoretical values do not provide accurate approximations of the experimental values.

5. How to Decrease the Sum of Squared G-S Lengths

We showed the effectiveness of generating a lattice basis of which the sum of squared G-S lengths is as small as possible in Section 4. In this section, we show how to decrease the sum of squared G-S lengths. We call the algorithm *the Restricting Reduction* (*RR*) *algorithm*. By the RR algorithm, a new lattice basis has the smaller sum of squared G-S lengths.

In Schnorr's sampling technique, when a short lattice vector is found by SA, it is immediately inserted to the basis and the generating system, which consists of a lattice basis and a generated vector, is reduced by BKZ. This process might increase the sum of squared G-S lengths, even if $\mathbf{b}_j$ for very small j, e.g., $j \in \{1, \ldots, 10\}$, might be improved. We aim to reduce $\sum_{j=1}^{n} \|\mathbf{b}_j^*\|^2$ as much as possible by a more efficient way of forming a generating system. The essential idea of our method is restricting the index of vectors in the basis where a short candidate vector can be inserted, so that the sum is not increased. We further reduce the sum by changing the index in increasing order.

We define *the insertion index* $h(\mathbf{v})$ of a generated vector $\mathbf{v}$ for some δ as follows:

Definition 4 Let B be a lattice basis, $\mathbf{v}$ be a lattice vector in $L(B)$, and $\delta \in \mathbb{R}$ with $\delta \leq 1$. The insertion index $h(\mathbf{v})$ of $\mathbf{v}$ is j such that $\min\{j : \|\pi_j(\mathbf{v})\|^2 < \delta\|\mathbf{b}_j^*\|^2, n+1\}$.

We do not insert $\mathbf{v}$ with $h(\mathbf{v}) < l$ for some integer $1 \leq l \leq n$. We call such an integer l *the restriction index*. Once the sum of squared G-S lengths of a local basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{l-1})$ is decreased, it is never influenced by the subsequent insertion. Therefore, by increasing l, the sum of the squared G-S lengths of a local basis $(\mathbf{b}_l, \ldots, \mathbf{b}_n)$ gets smaller from the front.

We show the outline of the RR algorithm in the following Algorithm 2. It is shown that the RR algorithm works very well in practice, although there is no theoretical guarantee that a lattice vector to satisfy the goal norm of the SVP instance is always obtained by using the RR algorithm. We generate lattice vectors in $V_B(\mathbf{s}, \mathbf{t})$ for each l. As explained in Section 3, we use some subset of $V_B(\mathbf{s}, \mathbf{t})$ in practice. We denote the subset by $\overline{V}_B(\mathbf{s}, \mathbf{t})$. We store all generated vectors $\mathbf{v}$ such that $h(\mathbf{v}) \leq n$ as a set P of vectors for all possible l. After generating lattice vectors, we take a lattice vector $\mathbf{v}$ from P with the minimum pair $(h(\mathbf{v}), \|\mathbf{v}\|^2)$ larger than

$(l, 0)$ in the lexicographic order. Then, we insert $\mathbf{v}$ into B at the index $h(\mathbf{v})$. After the insertion, we reduce a generating system, which consists of a lattice basis and a generated vector. We maintain the generated vectors $\mathbf{v}$ such that $h(\mathbf{v}) < l$, which cannot be inserted at this stage due to the restriction index l.

We increase the restriction index l when the number of times that $\mathbf{b}_l$ is not changed reaches a constant c. Basically, we set c to 1. The further l increases, the more likely a short lattice vector is found because $\sum_{j=1}^{n} \|\mathbf{b}_j^*\|^2$ continues to decrease during this step. The exception of the above restriction is when l reaches l_{max}, which is a parameter given as input. In that case, the generated vector $\mathbf{v}$ with the minimum pair $(h(\mathbf{v}), \|\mathbf{v}\|^2)$ is inserted into B at the index $h(\mathbf{v})$ without the restriction index l. That means that $\mathbf{b}_j$ for very small j, e.g., $j \in \{1, \ldots, 10\}$, is steadily improved, and has a chance to satisfy the goal norm of the SVP instance. On the other hand, that means $\sum_{j=1}^{n} \|\mathbf{b}_j^*\|^2$ considerably increases. Then, the whole step above is repeated from the beginning after LLL-reduction. Here, we use LLL-reduction of a linearly dependent generating system in Ref. [14], which removes a linearly dependent vector from the generating system.

Algorithm 2 The Restricting Reduction (RR) Algorithm

Input:

B : a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$

δ : the decreasing factor

$l_{\max}$: a parameter to determine the limit of the restriction index

$\mathbf{s}, \mathbf{t}$: parameters to determine the distribution of generated vectors

Output:

B : a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ such that $\mathbf{b}_1$ is a short candidate vector to satisfy the goal norm of the SVP instance

1: **for** $l = 1, \ldots, l_{\max}$ **do**
2: **repeat**
3: generate lattice vectors in $\overline{V}_B(\mathbf{s}, \mathbf{t})$ and store all the lattice vectors $\mathbf{w}$ with $h(\mathbf{w}) \leq n$ in a set P of vectors
4: take a lattice vector $\mathbf{v}$ from P with the minimum pair $(h(\mathbf{v}), \|\mathbf{v}\|^2)$ larger than $(l, 0)$ in the lexicographic order
5: insert $\mathbf{v}$ at the index $h(\mathbf{v})$
6: update P so that P includes only the lattice vectors $\mathbf{w}$ such that $h(\mathbf{w}) < l$
7: reduce a generating system consisting of $[\mathbf{b}_1, \ldots, \mathbf{b}_{h_0-1}, \mathbf{v}, \mathbf{b}_{h_0}, \ldots, \mathbf{b}_n]$ by LLL and form a new lattice basis B
8: **until** $\mathbf{b}_l$ is not changed c times /* Basically, we set c to 1. */
9: **end for**
10: take a lattice vector $\mathbf{v}$ from P with the minimum pair $(h(\mathbf{v}), \|\mathbf{v}\|^2)$
11: insert $\mathbf{v}$ at the index $h(\mathbf{v})$
12: reduce a generating system $[\mathbf{b}_1, \ldots, \mathbf{b}_{h_0-1}, \mathbf{v}, \mathbf{b}_{h_0}, \ldots, \mathbf{b}_n]$ by LLL and form a new lattice basis B

Figure 4 shows the transitions of the sum of squared G-S lengths in the for loop in the algorithm. From the transitions, we can see the effectiveness of the RR algorithm in terms of decreasing the sum of squared G-S lengths. Here, the lattice of dimension 128 with seed 0 in the SVP challenge is considered. The sum of squared G-S lengths is steadily decreased in the case of using the restriction index l while it is almost unchanged in the case of not using the restriction index l. We calculated the transition in about one day both in the cases of using the restriction index l and not using the restriction index l.

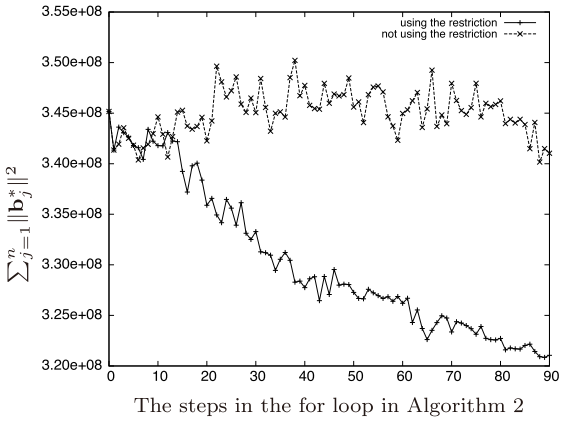

Fig. 4 The transitions of the sum of squared G-S lengths by the RR algorithm.

Table 2 The result of the experiment to solve some SVP challenges.

Dim	Achieved Norm	Previous Norm	Estimated Norm	Goal Norm
132	3012	unsolved (-)	2902 (1.038)	3047 (0.988)
130	3025	unsolved (-)	2887 (1.048)	3031 (0.998)
128	2984	unsolved (-)	2868 (1.040)	3012 (0.991)
126	2944	2969 (0.9913)	2839 (1.037)	2981 (0.988)
120	2756	2830 (0.9740)	2769 (0.995)	2907 (0.948)
118	2782	2868 (0.9700)	2760 (1.008)	2898 (0.960)
116	2786	2825 (0.9861)	2729 (1.021)	2866 (0.972)
114	2697	2735 (0.9859)	2707 (0.996)	2843 (0.949)
104	2594	2644 (0.9838)	2584 (1.004)	2713 (0.956)
102	2555	2597 (0.9837)	2566 (0.996)	2694 (0.948)

6. Solving the SVP Challenges

We applied our algorithm to the Darmstadt's SVP challenge. We wrote all of our code in JAVA. In an exceptional case, we wrote our code in C++ for a 132 dimensional SVP challenge problem. We used 3.4 GHz iMac (Intel Core i7) and 2.6 GHz Mac mini (Intel Core i7). In exceptional cases, we used a 2.6 GHz laptop computer "Let'snote" (Intel Core i5) for an SVP in dimension 120 and we used a 3.1 GHz iMac (Intel Core i7) besides the above computers in dimension 132. We solved an SVP in dimension 132, which is the highest dimension of the SVP solved ever, for 150 days. We also solved the several SVP for a few days or a few weeks. **Table 2** shows the results. The column of "Dim" shows the dimensions, the column of "Achieved norm" shows the norm that we achieved using our algorithm, the column of "Previous norm" shows the norm achieved by other algorithms, the column of "Estimated norm" shows the estimated norms of the shortest vector, i.e., $GH(L)$, and the column of "Goal norm" shows the goals of the SVP challenge, i.e., $GH(L)$. In Table 2, the numbers in parentheses show the ratios of the norm achieved by our algorithm to the corresponding norms.

Table 3 shows the information about the computers that we used for the experiment. **Table 4** shows the information about the computational costs. In all cases, we ran a single thread in each process. In Table 4, the running time for dimensions 102, 104, 114, and 120 are not provided since we used the information on the previous vectors found by other algorithms.

In the following, we provide what is the best Hermite factor we can achieve with our algorithm. The Hermite factor, which is

Table 3 The computers that we used.

Computer	CPU	Chip Number	Operating Frequency
iMac	Intel Core i7	4	3.4 GHz
iMac	Intel Core i7	4	3.1 GHz
Mac mini	Intel Core i7	4	2.6 GHz
Let'snote	Intel Core i5	4	2.6 GHz

Table 4 The information about the computational costs to solve SVP challenges.

Dim	Running Time	Parallel Processing Level
132	150 days	12
130	62 days	4
128	20 days	4
126	15 days	4
120	-	1
118	15 days	4
116	22 days	4
114	-	4
104	-	4
102	-	4

Table 5 Achieved Hermite factors.

Dim	Hermite factor
132	1.00823^{132}
130	1.00838^{130}
128	1.00839^{128}
126	1.00844^{126}
120	1.00833^{120}
118	1.00851^{118}
116	1.00869^{116}
114	1.00855^{114}
104	1.00875^{104}
102	1.00904^{102}

popularized by Gama et al. [4], is the most common measurement of the quality of SVP algorithms. The Hermite factor is defined as $\|\mathbf{b}_1\|/\mathrm{Vol}(L)^{1/n}$. According to Ref. [4], the stronger an SVP algorithm is, the smaller the Hermite factor it can achieve. **Table 5** shows Hermite factors achieved in our experiment. These Hermite factors correspond to the norms shown at the column of "Achieved norm" in Table 2. We can see from Table 5 that the Hermite factors are very small in all dimensions of the SVP we solved. The Hermite factors are considerably small especially in dimensions 120 and 132, where the Hermite factors 1.00833^{120} and 1.00823^{130} are achieved, respectively. These small Hermite factors indicate the strength of our algorithm.

Table 6 shows the ratio of the squared norms of the very short lattice vectors to the sum of squared G-S lengths. In Table 6, the column of "Dim" shows the dimensions, the column of "Sum of squared G-S lengths" shows the sum of squared G-S lengths of the bases when the very short lattice vectors were found, the column of "Squared norms" shows the squared norms of the very short lattice vectors, and the column of "Ratio" shows the ratios of the squared norms of the very short lattice vectors to the sum of squared G-S lengths. From Table 6, we can see that the ratios are almost constant. Thus, by reducing the sum of squared G-S lengths, we can find very short lattice vectors more frequently.

Table 6 The ratio of the squared norms of the very short lattice vectors to the sum of squared G-S lengths.

Dim	Sum of squared G-S lengths	Squared norms	Ratio
132	3.576×10^8	9.071×10^6	0.0254
130	3.132×10^8	9.149×10^6	0.0292
128	3.315×10^8	8.902×10^6	0.0268
126	3.089×10^8	8.664×10^6	0.0280
120	2.702×10^8	7.596×10^6	0.0281
118	2.746×10^8	7.740×10^6	0.0282
116	2.813×10^8	7.761×10^6	0.0276
114	2.603×10^8	7.272×10^6	0.0280
104	2.128×10^8	6.728×10^6	0.0316
102	2.178×10^8	6.526×10^6	0.0300

We used other techniques, which are shown in Section 8, besides the RR algorithm to solve SVP challenges. These techniques include some algorithmic and implementational techniques. However, the RR algorithm is the central part of our total techniques. In fact, we solved the SVP in dimensions 102, 104, and 114 by using only the RR algorithm without the techniques shown in Section 8.

7. Application Possibility of the RR Algorithm to Other Lattices

In this section, we experimentally confirm that the RR algorithm is applicable to other types of lattices than the SVP challenge lattices. As we saw in Section 3, we can explain by Schnorr's GSA the properties of the natural number representation of a very short lattice vector and how to determine a distribution of it. This indicates that if a lattice basis for some lattice approximates GSA, then the RR algorithm is applicable to the lattice. Here, we consider GGH lattices [6], Micciancio's GGH lattices [10], and NTRU lattices [7]. These types of lattices are related to lattice-based cryptography. Reduced bases for GGH lattices and Micciancio's GGH lattices approximate GSA [9], [15]. On the other hand, those for NTRU lattices somewhat violate GSA. However, most squared G-S lengths approximate a geometric sequence (see, e.g., Ref. [9]). In the following, we test if the RR algorithm is applicable to these types of lattices.

In Section 3, we analyzed the density $\#\{j : z_j \geq 1\}/n$ for a very short vector in SVP challenge lattices. Here, we experimentally confirm that the density for a very short vector tends to be low also in the above types of lattices. As mentioned in the above, these types of lattices are related to lattice-based cryptography and very short vectors in each type of lattice can be obtained in the key generation process. We calculated the density for very short vectors that correspond to the secret key in each type of lattice. Note that the number of such vectors is related to the dimension n of the lattice concerned and specifically, the number is n, n and $n/2$ for GGH lattices, Micciancio's GGH lattices and NTRU lattices, respectively. In each case, we calculated the density for a reduced basis in a lattice and took the average among all very short vectors that correspond to the secret key. **Table 7** shows the average and the standard deviation of the density. Table 7 indicates that the density tends to be low also in these types of lattices as in SVP challenge lattices. This shows the application

Table 7 Density for very short vectors.

Type of Lattices	Dim	Average of the density	Standard deviation of the density
GGH	180	0.0863	0.017
Micciancio's GGH	160	0.145	0.062
NTRU	214	0.169	0.029

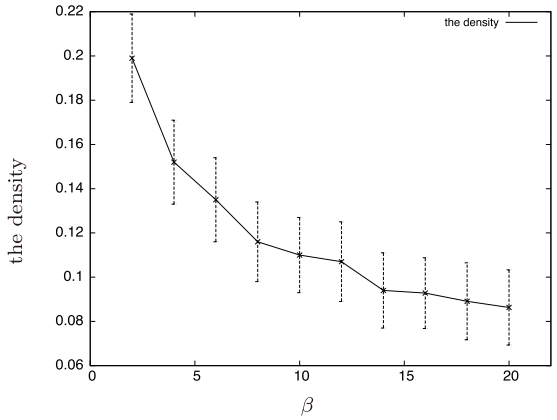

Fig. 5 Density for very short vectors in a GGH lattice.

possibility of our analysis in Section 3 to these types of lattices.

In the above example, we used a (δ,β)-BKZ reduced basis with $\delta = 0.99$ and $\beta = 20$ for each type of lattice. Here, a (δ,β)-BKZ reduced basis is the output computed by the BKZ algorithm and δ and β are parameters that adjust the quality of the output. Recall that $\mathbf{z}$ is determined by a lattice basis B. According to our empirical observation, the density $\#\{j : z_j \geq 1\}/n$ seems to be strongly dependent on the strength of reduction for B rather than the type of lattices. We calculated the average and the standard deviation of the density with increasing β. **Figure 5** shows the result of the calculation. We conducted the calculation for the same GGH lattice as in Table 7. We reduced a lattice basis by $(0.99,\beta)$-BKZ reduction with increasing β and obtained 10 reduced bases. We calculated the density for each of the 10 reduced bases. Note that the larger β is, the stronger $(0.99,\beta)$-BKZ reduction is. From Fig. 5, we can see that the more strongly a lattice basis is reduced, the lower the density tends to be. This can be explained by GSA as follows. Under GSA, the length of $\mathbf{b}_j^*$ at large indices gets larger for a more strongly reduced basis. Consequently, $\#\{j : z_j \geq 1\}$ needs to be smaller in order for a lattice vector $\mathbf{v}$ to be very short.

In order for the RR algorithm to be applicable to the above types of lattices, we need to show that Assumption 3 is applicable to them in addition to the analysis in Section 3. In the following, we experimentally confirm that Assumption 3 is close enough for those types of lattices.

Figure 6 shows the actual distribution of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ of $\mathbf{v} \in V_B(\mathbf{s},\mathbf{t})$ with $\mathbf{s} = (180, 7)$, $\mathbf{t} = (180, 30)$ for a basis B of a 180 dimensional GGH lattice. Note that here we use not some subset of $V_B(\mathbf{s},\mathbf{t})$ but $V_B(\mathbf{s},\mathbf{t})$ itself. The actual mean value and variance of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ were 6.751×10^5 and 9.448×10^9, respectively. On the other hand, the theoretical values of μ and σ^2 of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ were 6.748×10^5 and 9.447×10^9, respectively. Figure 6 also shows the probability density function of $N(6.748 \times 10^5, 9.447 \times 10^9)$.

Figure 7 shows the actual distribution of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ of $\mathbf{v} \in V_B(\mathbf{s},\mathbf{t})$ with $\mathbf{s} = (160, 7)$, $\mathbf{t} = (160, 30)$ for a basis B of a 160 dimensional Micciancio's GGH lattice. The actual mean value and variance of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ were 5.322×10^7 and 5.772×10^{13}, respectively. On the other hand, the theoretical values of μ and σ^2 of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ were 5.316×10^7 and 5.774×10^{13}, respectively. Figure 7 also shows the probability density function of $N(5.316 \times 10^7, 5.774 \times 10^{13})$.

Figure 8 shows the actual distribution of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ of $\mathbf{v} \in V_B(\mathbf{s},\mathbf{t})$ with $\mathbf{s} = (214, 7)$, $\mathbf{t} = (214, 30)$ for a basis B of a 214 dimensional NTRU lattice. The actual mean value and

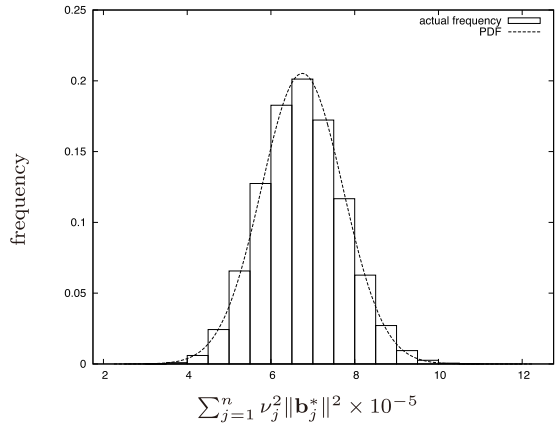

Fig. 6 The actual distribution of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ for a basis of a 180 dimensional GGH lattice.

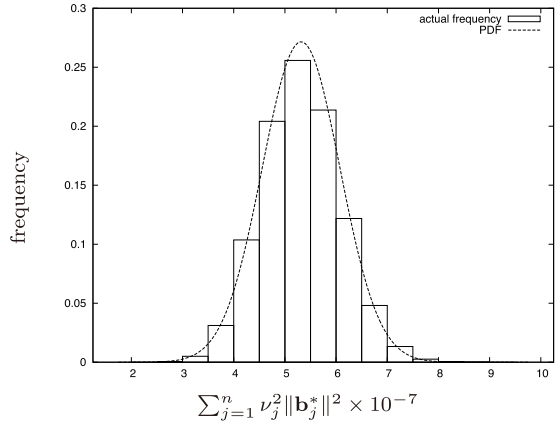

Fig. 7 The actual distribution of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ for a basis of a 160 dimensional Micciancio's GGH lattice.

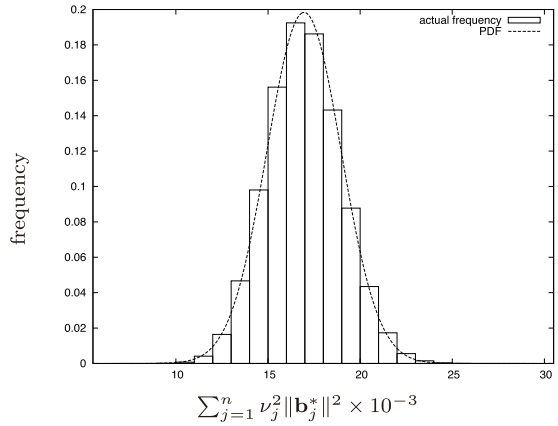

Fig. 8 The actual distribution of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ for a basis of a 214 dimensional NTRU lattice.

variance of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ were 1.696×10^3 and 4.045×10^6, respectively. On the other hand, the theoretical values of μ and σ^2 of $\sum_{j=1}^{n} \nu_j^2 \|\mathbf{b}_j^*\|^2$ were 1.696×10^3 and 4.044×10^6, respectively. Figure 8 also shows the probability density function of $N(1.696 \times 10^3, 4.044 \times 10^6)$.

From the above, we can see that Assumption 3 is close enough for each type of lattice. This shows the application possibility of the RR algorithm to these types of lattices.

8. Extension of the RR Algorithm

As mentioned in Section 6, we used other techniques besides the RR algorithm to solve SVP challenges. Since those techniques are basically ones to more efficiently generate short candidate vectors and more strongly reduce the sum of squared G-S lengths, we consider them to be a possible extension of the techniques shown in the preceding sections. In the following, we explain the extension. Then, we show the complete algorithm that we used to solve an SVP instance in a higher dimension than ever. We call the algorithm *the Extended Restricting Reduction* (*ERR*) *algorithm*. Regarding theoretical and experimental analysis of the techniques shown in this section, it is our future work to show to what extent each of those techniques contributes to the whole performance of our algorithm.

In order to explain the ERR algorithm, we define *the lexicographical ordering* on G-S lengths as follows:

Definition 5 Let B and B' be lattice bases, and let $[\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*]$ and $[\mathbf{b'}_1^*, \ldots, \mathbf{b'}_n^*]$ be the Gram-Schmidt orthogonalized vectors corresponding to B and B', respectively. Let j be the smallest index for which $\mathbf{b}_j^*$ and $\mathbf{b'}_j^*$ differ. B' is smaller in lexicographical ordering than B if and only if $\|\mathbf{b'}_j^*\| < \|\mathbf{b}_j^*\|$.

8.1 Stock Vectors

In the RR algorithm, we consider only lattice vectors shorter than i-th G-S length $\|\mathbf{b}_i^*\|$ in i-th orthogonal complement. In the extension of the RR algorithm, we consider not only lattice vectors shorter than i-th G-S length $\|\mathbf{b}_i^*\|$ but also lattice vectors slightly longer than $\|\mathbf{b}_i^*\|$. We call those lattice vectors *stock vectors*. Although lattice vectors which are slightly longer than the G-S length cannot be used to update the basis immediately, it is worth storing those lattice vectors in the memory since those lattice vectors might have chances to be used later. Those lattice vectors might be enough short to update the basis in some orthogonal complement as the lattice basis is iteratively transformed.

During generating short candidate lattice vectors, we store the lattice vectors each of which is relatively short on the orthogonal complement $\mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^{\perp}$ for $1 \le i \le n$. Specifically, we define *i-th stock vector* of B as follows:

Definition 6 Let B be a lattice basis, and let r be a real number with $r \ge 1$. For $1 \le i \le n$, an i-th stock vector is a lattice vector $\mathbf{v}$ with $\|\pi_i(\mathbf{v})\|^2 < r\|\mathbf{b}_i^*\|^2$.

We denote the set of i-th stock vectors of B by $W_{i,B}^r$. We call r the relaxation factor for stock vectors. Note that $W_{i,B}^r$ is actually determined by $[\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}]$ in addition to i and r.

In the RR algorithm, a set $\overline{V}_B(\mathbf{s}, \mathbf{t})$ of short candidate lattice vectors is used. We store the stock vectors belonging to $\overline{V}_B(\mathbf{s}, \mathbf{t})$ into memory. We set r to 1.4, for example.

8.2 Utilizing Two Types of Lattice Bases

In the following, we introduce a technique to reduce the sum of squared G-S lengths over a relatively long period. In the technique, we use two types of lattice bases. In order to generate short candidate vectors, we need a lattice basis of which the sum of squared G-S lengths is smaller. On the other hand, it is desirable to keep the smallest basis in lexicographical ordering. This is a dilemma, and the solution for it is to simultaneously maintain these two types of lattice bases, i.e., the lattice basis which is the current smallest in lexicographical ordering and the lattice basis of which sum of squared G-S lengths is the current smallest. We call the former basis and the latter basis the lexicographically optimal basis and the smallest sum basis, respectively. Also, we denote the former basis and the latter basis by G and S, respectively. The lexicographically optimal basis will be used as a smallest sum basis later since the sum of G-S lengths should be reduced from the front indices.

We summarize how to use the two type of lattice bases as follows.

Step 1 Preprocess S to reduce the sum of G-S lengths in the back indices.

Step 2 Generate short candidate lattice vectors in $\overline{V}_S(\mathbf{s}, \mathbf{t})$ and store stock vectors.

Step 3 Update S and G by using the stock vectors generated in Step 1 and Step 2.

Step 4 Increase the restriction index l.

The above processes from Step 1 to Step 4 repeat until no lattice vector that can be used to update S and G is found. In the case that such a lattice vector is not found, we substitute the lexicographically optimal basis G for S and reset the restriction index l. We reduce the sum of squared G-S lengths from the front indices again.

The role of the preprocessing in Step 1 is to reduce the sum of squared G-S lengths of S so that the expected values of the lengths of short candidate lattice vectors generated in Step 2 get smaller. Therefore, the framework of the preprocessing is basically the same with that of the RR algorithm. For the preprocessing, we generate short candidate lattice vectors in $\overline{V}_S(\mathbf{s}', \mathbf{t}')$ with $\mathbf{s}'$ and $\mathbf{t}'$, where $\mathbf{s}'$ and $\mathbf{t}'$ are determined so that $\#V_S(\mathbf{s}', \mathbf{t}')$ is much smaller than $\#V_S(\mathbf{s}, \mathbf{t})$. As with $V_S(\mathbf{s}, \mathbf{t})$, we use some subset $\overline{V}_S(\mathbf{s}', \mathbf{t}')$ of $V_S(\mathbf{s}', \mathbf{t}')$. In our implementation, in dimension 130, e.g., we set $\mathbf{s}$ and $\mathbf{t}$ to $(130, 13, 1)$ and $(130, 55, 15)$, respectively. Then, we set $\mathbf{s}'$ and $\mathbf{t}'$ to $(130, 4)$ and $(130, 22)$, respectively. We choose a subset $\overline{V}_S(\mathbf{s}', \mathbf{t}')$ so that $\#\overline{V}_S(\mathbf{s}', \mathbf{t}')$ is 2,000 or so. We repeatedly transform S. We also store stock vectors in this step for Step 3.

In Step 2, we generate short candidate lattice vectors by using the smallest sum basis S. We use S for generating short candidate lattice vectors since the sum of squared G-S lengths for S is small.

In Step 3, we update S under the restriction index l. As in the RR algorithm, we take a lattice vector $\mathbf{v}$ with the minimum pair $(h(\mathbf{v}), \|\mathbf{v}\|^2)$ larger than $(l, 0)$ in the lexicographic order. Then, we insert $\mathbf{v}$ into S at the index $h(\mathbf{v})$. While we reduce the sum of

squared G-S lengths of S, we update G by using stock vectors without any restriction.

In Step 4, we control the restriction index by using a list of variables, each of which we call the unchanged counter. The i-th unchanged counter is increased by 1 if i-th G-S length is unchanged for each i. On the other hand, the i-th unchanged counter is initialized to 0 when i-th G-S length is updated. Then, we increase the restriction index l if the l-th unchanged counter reaches the value of an upper limit function of l.

8.3 Double Loops for Reducing the Sum of Squared G-S Lengths

In Section 8.2, we explained a technique to reduce the sum of squared G-S lengths by using two lattice bases. Here, we show a revised version of the technique, in which we use three lattice bases. For that, we nest loops shown in Section 8.2. We call this algorithm the Extended Restriction Reduction (ERR) algorithm. In the following Algorithm 3, we show the outline of the ERR algorithm, which is an extension of the RR algorithm. The ERR algorithm works very well in practice, although there is no guaranteed quality for the output lattice basis. We assume that the input basis B is somewhat reduced by the LLL algorithm or the BKZ algorithm.

We use three types S, M, and G of lattice bases. G is the lexicographically optimal basis. S is the smallest sum basis S with restriction index l. M is a lattice basis between S and G lexicographically. We call the lattice basis M the medium basis. M has restriction index m with $m \leq l$. In the inner loop, the restriction index l increases. In the inner loop, M and S are in a similar relation between the lexicographically optimal basis and the smallest sum basis. In the outer loop, the restriction index m increases. In the outer loop, G and M are in a similar relation between the lexicographically optimal basis and the smallest sum basis. We reduce the sum of squared G-S lengths of M from the front indices gradually.

We increase these restriction indices in the following way. We count the number of times that i-th basis vector in S is not changed as $\text{unchanged}_{\text{inner}}(i)$ for any i. We count the number of times that i-th basis vector in M is not changed in the outer loop as $\text{unchanged}_{\text{outer}}(i)$ for any i. We use upper limit functions $c_{\text{inner}}(i)$ and $c_{\text{outer}}(i)$ that determine the upper values of $\text{unchanged}_{\text{inner}}(l)$ and $\text{unchanged}_{\text{outer}}(m)$, respectively. If the unchanged counter $\text{unchanged}_{\text{inner}}(l)$ reaches the upper value $c_{\text{inner}}(l)$, l is increased. If the unchanged counter $\text{unchanged}_{\text{outer}}(m)$ reaches the upper value $c_{\text{outer}}(m)$, m is increased. The upper values $c_{\text{inner}}(l)$ and $c_{\text{outer}}(m)$ indicate the maximum number of times the process stays at the indices l and m, respectively. As m increases, computational time in the inner repeat-until might be shorter since the value of l is m at the initial stage in the inner repeat-until. We set $c_{\text{inner}}(i)$ and $c_{\text{outer}}(i)$ so that computational time in the inner repeat-until might not be very different for each m. We set $c_{\text{inner}}(i)$ and $c_{\text{outer}}(i)$ to some small integers, e.g., 2 or 3, at small indices. On the other hand, we set $c_{\text{inner}}(i)$ and $c_{\text{outer}}(i)$ to a relatively large integers, e.g., 20 or so, at large indices.

We show below which part of Algorithm 3 corresponds to each of the Steps 1–4 in Section 8.2.

Step 1 line 9 in Algorithm 3.
Step 2 line 10 in Algorithm 3.
Step 3 lines 11 and 13 in Algorithm 3.
Step 4 line 14 in Algorithm 3.
Thus, the Steps 1–4 in Section 8.2 correspond to the inner repeat-until in Algorithm 3.

Algorithm 3 The Extended Restricting Reduction (ERR) Algorithm

Input:
B : a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$
δ : the decreasing factor
r : a relaxation factor for stock vectors
$\mathbf{s}, \mathbf{t}$: parameters to determine the distribution of generated vectors
$\mathbf{s}', \mathbf{t}'$: parameters to determine the distribution of generated vectors
Output:
G : a lattice basis G, which is earlier in lexicographical ordering than the input basis

1: $m := 0$
2: $G := B$
3: $M := G$
4: reset $unchanged_{\text{outer}}(i)$ and $unchanged_{\text{inner}}(i)$
5: **repeat**
6: $l := m$
7: $S := M$
8: **repeat**
9: S :=Preprocessing($S, \delta, r, n - t'_1, \mathbf{s}', \mathbf{t}'$)
10: generate lattice vectors in $\overline{V}_S(\mathbf{s}, \mathbf{t})$ and store the stock vectors in W.
11: S :=Updating(S, l, W, δ)
12: M :=Updating(M, m, W, δ)
13: G :=Updating($G, 0, W, \delta$)
14: $l := \min\{j : \text{unchanged}_{\text{inner}}(j) \leq c_{\text{inner}}(j)\}$
15: update $\text{unchanged}_{\text{inner}}$
16: **until** No vector is found for updating S
17: $m := \min\{j : \text{unchanged}_{\text{outer}}(j) \leq c_{\text{outer}}(j)\}$
18: update $\text{unchanged}_{\text{outer}}$
19: **until** No vector is found for updating M

8.4 Parallel Processing

In this section, we explain the parallel processing technique used in our program. We run multiple processes of JAVA programs on a computer which has multiple CPU-cores. We parallelize the whole steps of Algorithm 3, i.e., we naively execute Algorithm 3 on each CPU-core. We assign a different basis of the same lattice to each process. All processes interact with each other through stock vectors as follows. i-th stock vectors with small relaxation factor are stored in files on a disk and shared with all processes. Therefore, a short vector found by one process can be used by another process. We associate the file name with the sequence of G-S lengths from 1 to $i - 1$ since the i-th orthogonal complement is determined by Gram-Schmidt orthogonalized vectors from 1 to $i - 1$. The greatest part of the running time is spent is spent in the steps of generating short lattice candidate vectors. Therefore, the time of writing stock vectors to files and reading them from files is ignorable. We note that the only difference between a serial and parallel version of the ERR algorithm is that files of stock vectors are shared among all processes in the latter.

Algorithm 4 The Preprocessing Algorithm

Input:
B : a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$
δ : the decreasing factor
r : a relaxation factor for stock vectors
$l_{\max}$: a parameter to determine the limit of the restriction index
$\mathbf{s}, \mathbf{t}$: parameters to determine the distribution of generated vectors
Output:
B : a lattice basis B with the small sum of squared G-S lengths

1: **for** $l = 1, \ldots, l_{\max}$ **do**
2: **repeat**
3: generate lattice vectors in $\overline{V}_B(\mathbf{s}, \mathbf{t})$ and store all the lattice vectors $\mathbf{w}$ with $h(\mathbf{w}) \leq n$ in a set P of vectors
4: store the stock vectors in W
5: take a lattice vector $\mathbf{v}$ from P with the minimum pair $(h(\mathbf{v}), \|\mathbf{v}\|^2)$ larger than $(l, 0)$ in the lexicographic order
6: insert $\mathbf{v}$ at the index $h(\mathbf{v})$
7: update P so that P includes only the lattice vectors $\mathbf{w}$ such that $h(\mathbf{w}) < l$
8: reduce a generating system consisting of $[\mathbf{b}_1, \ldots, \mathbf{b}_{h_0-1}, \mathbf{v}, \mathbf{b}_{h_0}, \ldots, \mathbf{b}_n]$ by LLL and form a new lattice basis B
9: **until** $\mathbf{b}_l$ is not changed c times /* We set c to a large number, e.g., 500. */
10: **end for**

In order to work cooperatively, each process should solve the same problem, which has the same orthogonal complement. However, lattice bases used in processes should be different, for otherwise all processes provide the same vectors. Therefore, we use bases which are different only in the back indices. Then, they give different $V(\mathbf{s}, \mathbf{t})$.

Although the lattice bases in processes are similar at the initial stage, they might differ much from each other as the processes progress. In order to avoid this, the lattice bases need to be synchronized autonomously among all processes in some sense. For that, we use two tricks:

(1) We add the basis vectors $\mathbf{b}_i$ of the lattice basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ with small indices into stock vectors.
(2) We adjust the restriction indices to relax the restriction of the insertion for the above vectors.

By the first trick, basis vectors in the front indices are synchronized. By the second trick, the restriction indices in processes are synchronized.

Algorithm 5 The Updating Algorithm

Input:
B : a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$
l : the restriction index for B
W : a set of stock vectors
δ : the decreasing factor
Output:
B : an updated lattice basis B

1: **repeat**
2: take a lattice vector $\mathbf{v}$ from W with the minimum pair $(h(\mathbf{v}), \|\mathbf{v}\|^2)$ larger than $(l, 0)$ in the lexicographic order
3: insert $\mathbf{v}$ at the index $h(\mathbf{v})$
4: reduce a generating system consisting of $[\mathbf{b}_1, \ldots, \mathbf{b}_{h_0-1}, \mathbf{v}, \mathbf{b}_{h_0}, \ldots, \mathbf{b}_n]$ by LLL-reduction with δ and form a new lattice basis B
5: **until** no vector to update B is found in W

9. Efficiency of the ERR Algorithm

In this section, we experimentally confirm that the ERR algorithm has an advantage over the RSR algorithm and the Simple Sampling Reduction (SSR) algorithm [2], [9], which is an improved version of the RSR algorithm. Here, we experimentally compare the ERR algorithm with the RSR algorithm and the SSR algorithm. We conducted the experiment for the SVP of dimensions around 80 with seed 0. Our choice of the dimensions is based on the report in Ref. [12]. In Ref. [12], Schneider reported that it seems to be hard to solve SVP challenge instances using the SSR algorithm in a reasonable running time in dimension more than 85. In all cases of the experiment, we used one process with a single thread on a 1.7 GHz MacBook Air (Intel Core i5). We wrote the code for the RSR algorithm and the SSR algorithm in C++ with the version 6.0.0 of the NTL software package [16]. Regarding the ERR algorithm, we used the same program written in JAVA as we used to solve SVP challenges. **Table 8** shows the comparison of the running time required to find a lattice vector

Table 8 Comparison of the running time.

Dim	ERR	RSR	SSR
70	found 1 sec. $\mathbf{s} = (70, 13, 1)$, $\mathbf{t} = (70, 55, 15)$	found 97 sec. $k = 36$	found 38 sec. $u = 22$
72	found 4 sec. $\mathbf{s} = (72, 13, 1)$, $\mathbf{t} = (72, 55, 15)$	not found 17 min. $k = 36$	not found 52 min. $u = 28$
74	found 3 sec. $\mathbf{s} = (74, 13, 1)$, $\mathbf{t} = (74, 55, 15)$	found 61 min. $k = 38$	found 72 min. $u = 29$
76	found 7 sec. $\mathbf{s} = (76, 13, 1)$, $\mathbf{t} = (76, 55, 15)$	found 24 min. $k = 38$	not found 224 min. $u = 30$
78	found 39 sec. $\mathbf{s} = (78, 13, 1)$, $\mathbf{t} = (78, 55, 15)$	not found 101 min. $k = 38$	not found 237 min. $u = 30$
80	found 75 sec. $\mathbf{s} = (80, 13, 1)$, $\mathbf{t} = (80, 55, 15)$	not found 221 min. $k = 39$	found 8 sec. $u = 30$
82	found 3 min. $\mathbf{s} = (82, 13, 1)$, $\mathbf{t} = (82, 55, 15)$	not found 293 min. $k = 39$	not found 343 min. $u = 30$
84	found 9 min. $\mathbf{s} = (84, 13, 1)$, $\mathbf{t} = (84, 55, 15)$	not found 469 min. $k = 39$	found 42 min. $u = 30$
86	found 14 min. $\mathbf{s} = (86, 13, 1)$, $\mathbf{t} = (86, 55, 15)$	not found 355 min. $k = 39$	not found 290 min. $u = 30$
88	found 6 min. $\mathbf{s} = (88, 13, 1)$, $\mathbf{t} = (88, 55, 15)$	not found 335 min. $k = 39$	found 54 min. $u = 30$
90	found 8 min. $\mathbf{s} = (90, 13, 1)$, $\mathbf{t} = (90, 55, 15)$	not found 386 min. $k = 39$	not found 346 min. $u = 30$

Table 9 Comparison of the memory consumption.

Dim	ERR	RSR	SSR
70	160 MB	3 MB	3 MB
72	155 MB	3 MB	3 MB
74	161 MB	3 MB	3 MB
76	158 MB	4 MB	4 MB
78	155 MB	3 MB	4 MB
80	162 MB	4 MB	4 MB
82	155 MB	4 MB	4 MB
84	156 MB	4 MB	4 MB
86	169 MB	4 MB	4 MB
88	166 MB	4 MB	4 MB
90	168 MB	4 MB	5 MB

which satisfies the goal norm of the SVP challenge. In Table 8, the parameters that adjust the search space size for each algorithm are shown besides the running time. Here, the parameter k for the RSR algorithm is the one which, together with the constant related to GSA, determines the parameter u for $S_{u,B}$. For example, u took the values from 29 to 31 when $k = 38$ in dimension 80. Note that regarding the ERR algorithm we use some subset $\overline{V}_B(\mathbf{s}, \mathbf{t})$ of $V_B(\mathbf{s}, \mathbf{t})$ as explained in Section 3.

In all cases, we used a (δ, β)-BKZ reduced basis with $\delta = 0.99$ and $\beta = 20$ as input. The results in Table 8 do not include the the running time required for the initial reduction of a basis. For the initial reduction, we used BKZ_XD, which is one of the BKZ variants implemented in the NTL software package. The running time required for BKZ_XD with $\delta = 0.99$ and $\beta = 20$ was within 1 minute in all dimensions except for 90. In dimension 90, the running time was 76 seconds. In Table 8, "found 97 sec.", e.g., shows that a solution for the SVP challenge was found in 97 seconds. On the other hand, "not found 17 min.", e.g., shows that the algorithm concerned terminated in 17 minutes and a solution for the SVP challenge was not found. **Table 9** shows the memory consumption measured for each dimension.

From Table 8, we can see that the program for the ERR algorithm was faster than the ones for the other algorithms in all dimensions except for 80. Although the tools used to write the code for the three algorithms are different, the implementation in JAVA language is slower than that in C++ language in general and Table 8 shows that the JAVA implementation of ERR algorithm is much faster than the C++ implementations of other algorithms. This is a qualitative evidence of the efficiency of our algorithm. That is, we can say that our algorithm is more efficient in terms of the running time than the RSR algorithm and the SSR algorithm. On the other hand, our algorithm requires much more memory than those algorithms. However, the increase of the memory consumption is not so rapid as to make our algorithm impractical as in Table 9. In fact, even in dimension 130, the memory consumption, which was measured in the same environment as above, was 181 MB.

References

[1] Ajtai, M.: The Shortest Vector Problem in L_2 is NP-hard for Randomized Reductions (Extended Abstract), *Proc. 30th Annual ACM Symposium on Theory of Computing*, pp.10–19 (1998).

[2] Buchmann, J. and Ludwig, C.: Practical Lattice Basis Sampling Reduction, *Proc. ANTS 2006*, *LNCS*, Vol.4076, pp.222–237 (2005).

[3] Chen, Y. and Nguyen, P.Q.: BKZ2.0: Better Lattice Security Estimates, *ASIACRYPT 2011*, *LNCS*, Vol.7073, pp.1–20 (2011).

[4] Gama, N. and Nguyen, P.Q.: Predicting Lattice Reduction, *EUROCRYPT 2008*, *LNCS*, Vol.4965, pp.31–51 (2008).

[5] Gama, N., Nguyen, P.Q. and Regev, O.: Lattice Enumeration Using Extreme Pruning, *EUROCRYPT 2010*, *LNCS*, Vol.6110, pp.257–278 (2010).

[6] Goldreich, O., Goldwasser, S. and Halevi, S.: Public-Key Cryptosystems from Lattice Reduction Problems, *CRYPTO 1997*, *LNCS*, Vol.1294, pp.112–131 (1997).

[7] Hoffstein, J., Pipher, J. and Silverman, J.H.: NTRU: A Ring-Based Public Key Cryptosystem, *Proc. ANTS III*, *LNCS*, Vol.1423, pp.267–288 (1998).

[8] Lenstra, A.K., Lenstra, H.W. and Lovász, L.: Factoring Polynomials with Rational Coefficients, *Mathematische Ann*, Vol.261, pp.513–534 (1982).

[9] Ludwig, C.: Practical Lattice Basis Sampling Reduction, PhD thesis, TU Darmstadt (2005).

[10] Micciancio, D.: Improving Lattice Based Cryptosystems Using the Hermite Normal Form, *CaLC2001*, *LNCS*, Vol.2146, pp.126–145 (2001).

[11] Micciancio, D.: The Geometry of Lattice Cryptography, Foundations of Security Analysis and Design VI, *LNCS*, Vol.6858, pp.185–210 (2011).

[12] Schneider, M.: Computing Shortest Lattice Vectors on Special Hardware. PhD thesis, TU Darmstadt (2011).

[13] Schneider, M. and Gama, N.: SVP Challenge, available from ⟨http://www.latticechallenge.org/svp-challenge/⟩.

[14] Schnorr, C.P. and Euchner, M.: Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems, *Math. Programming*, Vol.66, pp.181–199 (1994).

[15] Schnorr, C.P.: Lattice Reduction by Random Sampling and Birthday Methods, *STACS 2003*, *LNCS*, Vol.2607, pp.145–156, Springer-Verlag (2003).

[16] Shoup, V.: NTL - A Library for Doing Number Theory, available from ⟨http://www.shoup.net/ntl/index.html⟩.

Appendix

A.1 Method to Generate a Lattice Vector in $V_B(\mathbf{s}, \mathbf{t})$

The method that we implemented to generate a lattice vector in $V_B(\mathbf{s}, \mathbf{t})$ is a slightly modified version of the enumeration algorithm ENUM [14]. ENUM determines the Gram-Schmidt coefficients of a lattice vector to be output by exhaustive search. We determine the Gram-Schmidt coefficients not by exhaustive search but by the natural number representation. In the following Algorithm 6, we show the outline of the method, which we call *the Generating Algorithm* (*GA*).

A.2 An Empirical Explanation of $V_B(\mathbf{s}, \mathbf{t})$

Here, we explain what the definition of $V_B(\mathbf{s}, \mathbf{t})$ means and the reason why we define $V_B(\mathbf{s}, \mathbf{t})$ in this way.

Example 1 We consider the SVP of dimension 128 with seed 0 in the SVP challenge. A very short vector $\mathbf{v}$ in the lattice L corresponding to this SVP is as follows:

$\mathbf{v}$ = (322 −301 −24 119 218 −342 119 444 −329 312 92 66 −64 −62 181 −204 138 −497 −259 297 −448 627 137 217 89 42 337 33 1 63 −303 −290 21 1 −412 370 −111 335 171 29 −77 −65 139 −432 38 312 −528 −339 −66 175 664 223 192 277 270 −175 60 98 −34 −424 −243 160 −41 −162 224 −61 −428 −331 270 230 −80 193 −345 −244 10 −438 −19 117 −315 259 424 −227 −213 −174 84 66 −506 −152 −253 233 256 −122 144 −256 363 102 204 −35 −127 456 247 −134 237 12 132 −317 109 191 469 −110 −381 −233 −605 −503 180

Algorithm 6 Generating Algorithm (GA)

Input:
B : a lattice basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$
$\mathbf{z} \in \mathbb{N}^n$: a natural number representation
Output:
$\mathbf{v}$: a lattice vector

compute $\mu_{i,j}$ such that $\mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle$
for $i = 1, \ldots, n$ **do**
  $u_i := 0$
end for
for $i = n, \ldots, 1$ **do**
  $y_i := 0$
  for $j = i + 1, \ldots, n$ **do**
    $y_i := y_i - u_j \mu_{j,i}$
  end for
  $u_i := \lfloor y_i + 0.5 \rfloor$
  if $u_i < y_i$ **then**
    $\delta_i := -1$
  else
    $\delta_i := 1$
  end if
  if $z_i \bmod 2 = 0$ **then**
    $u_i := u_i + \delta_i \lceil z_i / 2 \rceil$
  else
    $u_i := u_i - \delta_i \lceil z_i / 2 \rceil$
  end if
end for
$\mathbf{v} := B\mathbf{u}$

−196 277 −137 39 −423 −37 126 371 53 −396 121 176 −213).
The lattice vector $\mathbf{v} \in L$ is short enough to satisfy the criterion in the SVP challenge, i.e., $\|\mathbf{v}\| < 1.05 \cdot GH(L)$. We represent $\mathbf{v}$ in terms of $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ that are the Gram-Schmidt orthogonalized vectors of some reduced basis B in L, i.e., $\mathbf{v} = \sum_{j=1}^{n} \nu_j \mathbf{b}_j^*$. Then, ν is as follows:

ν = (0.057 0.02 0.001 0.114 −0.112 0.096 −0.13 −0.125 −0.095 −0.065 −0.031 −0.116 0.045 −0.032 −0.086 −0.167 −0.129 −0.17 −0.119 0.099 −0.047 −0.025 0.017 0.173 0.176 −0.061 −0.093 0.021 −0.143 −0.068 −0.107 0.137 0.094 0.168 0.049 −0.21 −0.003 −0.158 −0.146 0.025 0.104 0.073 −0.004 0.123 0.124 −0.083 0.149 −0.041 −0.088 0.235 −0.002 −0.224 −0.404 0.514 0.055 −0.093 0.367 −0.125 −0.193 −0.197 −0.258 −0.054 −0.03 0.009 −0.082 −0.15 −0.152 0.127 −0.204 0.096 −0.335 0.05 0.24 0.068 0.643 −0.091 −0.184 −0.176 0.699 0.287 −0.019 −0.373 −0.153 −0.705 −0.493 0.036 −0.146 0.283 −0.052 −0.268 −0.423 −0.957 0.408 −0.924 0.045 −0.524 0.328 −0.419 −0.207 −0.036 0.049 0.016 0.629 −0.552 0.235 0.546 −0.632 −0.252 0.328 −0.589 0.519 0.39 0.716 −0.127 −1.542 −0.042 −0.012 0.66 1.236 −1.339 −0.162 −1.782 −1.38 −0.27 0.987 0.388 2.15 1).

In this case, the last nonzero Gram-Schmidt coefficient is $\nu_{128} = 1$. Because $1/2 < \nu_{128} \leq (1+1)/2$, $z_{128} = 1$ by the definition of $\mathbf{z}(\mathbf{v})$. z_j for $j < 128$ is also determined in the same way. Thus, the natural number representation $\mathbf{z}(\mathbf{v})$ is as follows:

$\mathbf{z}(\mathbf{v})$ = (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 1 1 0 1 0 3 0 0 1 2 2 0 3 2 0 1 0 4 1).

From the definitions of $\mathbf{d}$ and $\mathbf{w}$, $d_0 = 105$, $d_1 = 17$, $d_2 = 3$, $d_3 = 2$, $d_4 = 1$ and $w_0 = 126$, $w_1 = 75$, $w_2 = 10$, $w_3 = 14$, $w_4 = 2$. This means that $V_B(\mathbf{s}, \mathbf{t})$ with $\mathbf{s} = (105, 17, 3, 2, 1)$, $\mathbf{t} = (126, 75, 10, 14, 2)$ or $\mathbf{s} = (127, 20, 5, 3, 2)$, $\mathbf{t} = (128, 75, 15, 15, 5)$ and so on, includes $\mathbf{v}$ while $V_B(\mathbf{s}, \mathbf{t})$ with $\mathbf{s} = (127, 15, 3, 2, 1)$, $\mathbf{t} = (128, 75, 15, 15, 5)$ or $\mathbf{s} = (127, 20, 5, 3, 2)$, $\mathbf{t} = (128, 70, 15, 15, 5)$ and so on, do not include $\mathbf{v}$.
Thus, if $\mathbf{s}$ and $\mathbf{t}$ are properly set, $V_B(\mathbf{s}, \mathbf{t})$ includes $\mathbf{v}$.

Masaharu Fukase received B.S., M.S., and Ph.D. degrees from the University of Tokyo, Japan, in 2006, 2008, and 2011, respectively. Since 2011, he has been a project research assistant of Dokkyo University, Japan. His research interest is in lattices.

Kenji Kashiwabara received B.S., M.S., and Ph.D. degrees from Tokyo Institute of Technology, Japan, in 1991, 1993, and 1998, respectively. He is currently a research associate of the University of Tokyo, Japan from 2007. His research interests are in Cryptography and Combinatorics.