Improved Progressive BKZ Algorithms and their Precise Cost Estimation by Sharp Simulator

Yoshinori Aono[1⋆], Yuntao Wang[2], Takuya Hayashi[1], and Tsuyoshi Takagi[3,4]

[1] National Institute of Information and Communications Technology, Tokyo, Japan.
[2] Graduate School of Mathematics, Kyushu University, Fukuoka, Japan.
[3] Institute of Mathematics for Industry, Kyushu University, Fukuoka, Japan,
[4] CREST, Japan Science and Technology Agency

Abstract. In this paper, we investigate a variant of the BKZ algorithm, called progressive BKZ, which performs BKZ reductions by starting with a small blocksize and gradually switching to larger blocks as the process continues. We discuss techniques to accelerate the speed of the progressive BKZ algorithm by optimizing the following parameters: blocksize, searching radius and probability for pruning of the local enumeration algorithm, and the constant in the geometric series assumption (GSA). We then propose a simulator for predicting the length of the Gram-Schmidt basis obtained from the BKZ reduction. We also present a model for estimating the computational cost of the proposed progressive BKZ by considering the efficient implementation of the local enumeration algorithm and the LLL algorithm. Finally, we compare the cost of the proposed progressive BKZ with that of other algorithms using instances from the Darmstadt SVP Challenge. The proposed algorithm is approximately 50 times faster than BKZ 2.0 (proposed by Chen-Nguyen) for solving the SVP Challenge up to 160 dimensions.

Keywords: Lattice basis reduction, progressive BKZ, Gram-Schmidt orthogonal basis, geometric series assumption

1 Introduction

Lattices in cryptography have been actively used as the foundation for constructing efficient or high-functional cryptosystems such as public-key encryptions [16,25,40], fully homomorphic encryptions [21,9], and multilinear maps [20]. The security of lattice-based cryptography is based on the hardness of solving the (approximate) shortest vector problems (SVP) in the underlying lattice [14,31,34,35]. In order to put lattice-based cryptography into practical use, we must precisely estimate the secure parameters in theory and practice by analyzing the previously known efficient algorithms for solving the SVP.

Currently the most efficient algorithms for solving the SVP are perhaps a series of BKZ algorithms [45,46,12,13]. Numerous efforts have been made to

⋆ E-mail: aono@nict.go.jp. This work was supported by JSPS KAKENHI Grant Number 26730069.

estimate the security of lattice-based cryptography by analyzing the BKZ algorithms. Lindner and Peikert [31] gave an estimation of secure key sizes by connecting the computational cost of BKZ algorithm with the root Hermite factor from their experiment using the NTL-BKZ [48]. Furthermore, van de Pol and Smart [50] estimated the key sizes of fully homomorphic encryptions using a simulator based on Chen-Nguyen's BKZ 2.0 [12]. Lepoint and Naehrig [30] gave a more precise estimation using the parameters of the full-version of BKZ 2.0 paper [13]. On the other hand, Liu and Nguyen [32] estimated the secure key sizes of some LWE-based cryptosystems by considering the BDD in the associated q-ary lattice. Aono et al. [7] gave another security estimation for LWE-based cryptosystems by considering the challenge data from the Darmstadt Lattice Challenge [49]. Recently, Albrecht et al. presented a comprehensive survey on the state-of-the-art of hardness estimation for the LWE problem [5].

The above analyzing algorithms are usually called "lattice-based attacks", which have a generic framework consisting of two parts:

(1)Lattice reduction: This step aims to decrease the norm of vectors in the basis by performing a lattice reduction algorithm such as the LLL or BKZ algorithm.

(2)Point search: This step finds a short vector in the lattice with the reduced basis by performing the enumeration algorithm.

In order to obtain concrete and practical security parameters for lattice-based cryptosystems, it is necessary to investigate the trade-offs between the computational cost of a lattice reduction and that of a lattice point search.

For our total cost estimation, we further limit the lattice-based attack model by (1) using our improved progressive BKZ algorithm for lattice reduction, and (2) using the standard (sometimes randomized) lattice vector enumeration algorithm with sound pruning [19]. To predict the computational cost under this model, we propose a simulation method to generate the computing time of lattice reduction and the lengths of the Gram-Schmidt vectors of the basis to be computed.

BKZ Algorithms: Let $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ be the basis of the lattice. The BKZ algorithms perform the following local point search and update process from index $i = 1$ to $n - 1$. The local point search algorithm, which is essentially the same as the algorithm used in the second part of the lattice-based attacks, finds a short vector in the local block $B_i = \pi_i(\mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta-1})$ of the fixed blocksize β (the blocksize shrinks to $n - i + 1$ for large $i \geq n - \beta + 1$). Here, the lengths of vectors are measured under the projection π_i which is defined in Section 2.1. Then, the update process applies lattice reduction for the degenerated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}, \mathbf{b}_i, \ldots, \mathbf{b}_n)$ after inserting vector $\mathbf{v}$ at i-th index.

The point search subroutine finds a short vector in some searching radius $\alpha \cdot \mathrm{GH}(B_i)$ with some probability which is defined over random local blocks of the fixed dimension. Here, $\mathrm{GH}(B_i)$ is an approximation of the length of the shortest vector in the sublattice generated by B_i.

In the classical BKZ algorithms [46,45], the local point search calls a single execution of a lattice vector enumeration algorithm with a reasonable pruning for

Table 1. Technical comparison from BKZ 2.0

Technique	BKZ 2.0 [12]	Our algorithm
Enumeration setting		
randomizing basis [19]	yes	no
optimal pruning [19]	yes	yes
blocksize β	fixed	iteratively increasing (Sec. 6.1)
search radius $\alpha \cdot \mathrm{GH}(B_i)$	$\sqrt{1.1} \cdot \mathrm{GH}(B_i)$	} optimized by GSA (Sec. 4)
probability p	optimized by simulator	} optimized by GSA (Sec. 4)
Preprocessing local block	optimal BKZ strategy	progressive BKZ
Terminating BKZ strategy	simulator based (fixed)	FEC based (adaptive, Sec. 5)
Predicting $\|\mathbf{b}_i^*\|$	simulator based	simulator based (Sec. 5.1)

searching tree. The BKZ 2.0 algorithm proposed by Chen and Nguyen [12] uses the extreme pruning technique [19], which performs the lattice enumeration with success probability p for $\lfloor 1/p \rceil$ different bases $G_1, \ldots, G_{\lfloor 1/p \rceil}$ obtained by randomizing the local basis B_i. They use the fixed searching radius as $\sqrt{1.1} \cdot \mathrm{GH}(B_i)$. We stress that BKZ 2.0 is practically the fastest algorithm for solving the approximate SVP of large dimensions. Indeed, many top-records in the Darmstadt Lattice Challenge [49] have been solved by BKZ 2.0.

Our Contributions: In this paper we revisit progressive BKZ algorithms, which have been mentioned in several studies; these include [18,12,44,47,24]. The main idea of progressive BKZ is that performing BKZ iteratively starting with a small blocksize is practically faster than the direct execution of BKZ with a larger blocksize. The method used to increase the blocksize β strongly affects the overall computational cost of progressive BKZ. The research goal here is to find an optimal method of increasing the blocksize β according to the other parameters in the BKZ algorithms.

One major difference between BKZ 2.0 and our algorithm is the usage of randomized enumeration in local blocks. To find a very short vector in each local block efficiently, BKZ 2.0 uses the randomizing technique in [19]. Then, it reduces each block to decrease the cost of lattice enumeration. Although it is significantly faster than the enumeration without pruning, it introduces overhead because the bases are not good in practice after they have been randomized. To avoid this overhead, we adopted the algorithm with a single enumeration with a low probability.

Moreover, BKZ of a large blocksize with large pruning (i.e., a low probability) is generally better in both speed and quality of basis than that of a small blocksize with few pruning (i.e., a high probability), as a rule of thumb. We pursue this idea and add the freedom to choose the radius $\alpha \cdot \mathrm{GH}(L)$ of the enumeration of the local block; this value is fixed in BKZ 2.0 as $\sqrt{1.1} \cdot \mathrm{GH}(L)$.

To optimize the algorithm, we first discuss techniques for optimizing the BKZ parameters of enumeration subroutine, including the blocksize β, success probability p of enumeration, and α to set the searching radius of enumeration as $\alpha \cdot \mathrm{GH}(B_i)$. We then show the parameter relationship that minimizes the computational cost for enumeration of a BKZ-β-reduced basis. Next, we introduce the

new usage of *full enumeration cost* (FEC), derived from Gama-Nguyen-Regev's cost estimation [19] with a Gaussian heuristic radius and without pruning, to define the quality of the basis and to predict the cost after BKZ-β is performed. Using this metric, we can determine the timing for increasing blocksize β that provides an optimized strategy; in previous works, the timing was often heuristic.

Furthermore, we propose a new BKZ simulator to predict the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ after BKZ-β. Some previous works aimed to find a short vector as fast as possible, and did not consider other quantities. However, additional information is needed to analyze the security of lattice-based cryptosystems. In literatures, a series of works on lattice basis reduction [43,18,12,13] have attempted to predict the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ after lattice reduction. In particular, Schnorr's GSA is the first simulator of Gram-Schmidt lengths and the information it provides is used to analyze the random sampling algorithm. We follow this idea, i.e., predicting Gram-Schmidt lengths to analyze other algorithms.

Our simulator is based on the Gaussian heuristic with some modifications, and is computable directly from the lattice dimension and the blocksize. On the other hand, Chen-Nguyen's simulator must compute the values sequentially; it has an inherent problem of accumulative error, if we use the strategy that changes blocksize many times. We also investigate the computational cost of our implementation of the new progressive BKZ, and show our estimation for solving challenge problems in the Darmstadt SVP Challenge and Ideal Lattice Challenge [49]. Our cost estimation is derived by setting the computation model and by curve fitting based on results from computer experiments. Using our improved progressive BKZ, we solved Ideal Lattice Challenge of 600 and 652 dimensions in the exact expected times of $2^{20.7}$ and $2^{24.0}$ seconds, respectively, on a standard PC.

Finally, we compare our algorithm with several previous algorithms. In particular, compared with Chen-Nguyen's BKZ 2.0 algorithm [12,13] and Schnorr's blocksize doubling strategy [47], our algorithm is significantly faster. For example, to find a vector shorter than $1.05 \cdot \mathrm{GH}(L)$, which is required by the SVP Challenge [49], our algorithm is approximately 50 times faster than BKZ 2.0 in a simulator-based comparison up to 160 dimensions.

Roadmap: In Section 2 we introduce the basic facts on lattices. In Section 3 we give an overview of BKZ algorithms, including Chen-Nguyen's BKZ 2.0 [12] and its cost estimation; we also state some heuristic assumptions. In Section 4, we propose the optimized BKZ parameters under the Schnorr's geometric series assumption (GSA). In Section 5, we explain the basic variant of the proposed progressive BKZ algorithm and its simulator for the cost estimation. In Section 6, we discuss the optimized block strategy that improved the speed of the proposed progressive BKZ algorithm. In Section 7, we describe the details of our implementation and the cost estimation for processing local blocks. We then discuss an extended strategy using many random reduced bases [19] besides our progressive BKZ in Section 8. Finally, Section 9 gives the results of our sim-

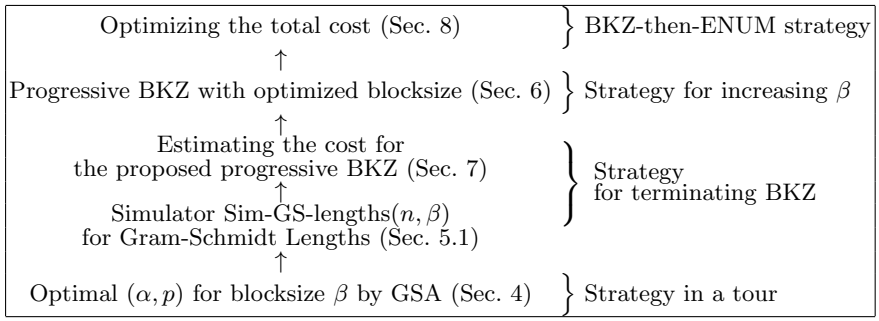

Fig. 1. Roadmap of this paper: optimizing parameters from local to global

ulation to solve the SVP Challenge problems and compares these results with previous works.

2 Lattice and Shortest Vector

A lattice L is generated by a basis B which is a set of linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^m$. We will refer to it as $L(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$. Throughout this paper, we assume $m = O(n)$ to analyze the computational cost, though it is not essential. The length of $\mathbf{v} \in \mathbb{R}^m$ is the standard Euclidean norm $\|\mathbf{v}\| := \sqrt{\mathbf{v} \cdot \mathbf{v}}$, where the dot product of any two lattice vectors $\mathbf{v} = (v_1, \ldots, v_m)$ and $\mathbf{w} = (w_1, \ldots, w_m)$ is defined as $\mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{m} v_i w_i$. For natural numbers i and j with $i < j$, $[i : j]$ is the set of integers $\{i, i+1, \ldots, j\}$. Particularly, $[1 : j]$ is denoted by $[j]$.

The gamma function $\Gamma(s)$ is defined for $s > 0$ by $\Gamma(s) = \int_0^\infty t^{s-1} \cdot e^{-t} dt$. The beta function is $\mathrm{B}(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1} dt$. We denote by $\mathrm{Ball}_n(R)$ the n-dimensional Euclidean ball of radius R, and then its volume $V_n(R) = R^n \cdot \frac{\pi^{n/2}}{\Gamma(n/2+1)}$. Stirling's approximation yields $\Gamma(n/2+1) \approx \sqrt{\pi n}(n/2)^{n/2} e^{-n/2}$ and $V_n(1)^{-1/n} \approx \sqrt{n/(2\pi e)} \approx \sqrt{n/17}$.

2.1 Gram-Schmidt Basis and Projective Sublattice

For a given lattice basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, we define its *Gram-Schmidt orthogonal basis* $B^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ by $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^*$ for $1 \le j < i \le n$, where $\mu_{ij} = (\mathbf{b}_i \cdot \mathbf{b}_j^*)/\|\mathbf{b}_j^*\|^2$ are the Gram-Schmidt coefficients (abbreviated as *GS-coefficients*). We sometimes refer to $\|\mathbf{b}_i^*\|$ as the *Gram-Schmidt lengths* (abbreviated as *GS-lengths*). We also use the Gram-Schmidt variables (abbreviated as *GS-variables*) to denote the set of GS-coefficients μ_{ij} and lengths $||\mathbf{b}_i^*||$. The lattice determinant is defined as $\det(L) := \prod_{i=1}^{n} \|\mathbf{b}_i^*\|$ and it is equal to the volume $\mathrm{vol}(L)$ of the fundamental parallelepiped. We denote the orthogonal projection by $\pi_i : \mathbb{R}^m \mapsto \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$ for $i \in \{1, \ldots, n\}$. In particular, $\pi_1(\cdot)$ is used as the identity map.

We denote the *local block* by the projective sublattice

$$L_{[i:j]} := L(\pi_i(\mathbf{b}_i), \pi_i(\mathbf{b}_{i+1}), \ldots, \pi_i(\mathbf{b}_j))$$

for $j \in \{i, i+1, \ldots, n\}$. We sometimes use B_i to denote the lattice whose basis is $(\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_j))$ of projective sublattice $L_{[i:j]}$. That is, we omit the change of blocksize $\beta = j - i + 1$ if it is clear by context.

2.2 Shortest Vector and Gaussian Heuristic

A non-zero vector in a lattice L that has the minimum norm is called the *shortest vector*. We use $\lambda_1(L)$ to denote the norm of the shortest vector. The notion is also defined for a projective sublattice as $\lambda_1(L_{[i:j]})$ (we occasionally refer to this as $\lambda_1(B_i)$ in this paper).

The *shortest vector problem* (SVP) is the problem of finding a vector of length $\lambda_1(L)$. For a function $\gamma(n)$ of a lattice dimension n, the standard definition of γ-approximate SVP is the problem of finding a vector shorter than $\gamma(n) \cdot \lambda_1(L)$.

An n-dimensional lattice L and a continuous (usually convex and symmetric) set $S \subset \mathbb{R}^m$ are given. Then the *Gaussian heuristic* says that the number of points in $S \cap L$ is approximately $\mathrm{vol}(S)/\mathrm{vol}(L)$.

In particular, taking S as the origin-centered ball of radius R, the number of lattice points is approximately $V_n(R)/\mathrm{vol}(L)$, which derives the length of shortest vector λ_1 by R so that the volume of the ball is equal to that of the lattice:

$$\lambda_1(L) \approx \det(L)^{1/n}/V_n(1)^{1/n} = \frac{(\Gamma(n/2+1)\det(L))^{1/n}}{\sqrt{\pi}}$$

This is usually called the *Gaussian heuristic of a lattice*, and we denote it by $\mathrm{GH}(L) = \det(L)^{1/n}/V_n(1)^{1/n}$.

For our analysis, we use the following lemma on the randomly generated points.

Lemma 1 *Let $x_1, \ldots, x_K$ be K points uniformly sampled from the n-dimensional unit ball. Then, the expected value of the shortest length of vectors from origin to these points is*

$$\mathbf{E}\Big[\min_{i\in[K]} ||x_i||\Big] = K \cdot B\Big(K, \frac{n+1}{n}\Big) := K \cdot \int_0^1 t^{1/n}(1-t)^{K-1}dt.$$

In particular, letting $K = 1$, the expected value is $n/(n+1)$.

Proof. Since the cumulative distribution function of each $\|x_i\|$ is $F_i(r) = r^n$, the cumulative function of the shortest length of the vectors is $F_{\min}(r) = 1 - (1 - F_i(r))^K = 1 - (1 - r^n)^K$. Its probability density function is $P_{\min}(r) = \frac{dF}{dr} = Kn \cdot r^{n-1}(1-r^n)^{K-1}$. Therefore, the expected value of the shortest length of the vectors is

$$\int_0^1 rP_{\min}(r)dr = K \cdot \int_0^1 t^{1/n}(1-t)^{K-1}dt.$$

□

2.3 Enumeration Algorithm [27,45,19]

We explain the enumeration algorithm for finding a short vector in the lattice. The pseudo code of the enumeration algorithm is given in [45,19]. For given lattice basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$, and its Gram-Schmidt basis $(\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$, the enumeration algorithm considers a search tree whose nodes are labeled by vectors. The root of the search tree is the zero vector; for each node labeled by $\mathbf{v} \in L$ at depth $k \in [n]$, its children have labels $\mathbf{v} + a_{n-k} \cdot \mathbf{b}_{n-k}$ $(a_{n-k} \in \mathbb{Z})$ whose projective length $\|\pi_{n-k}(\sum_{i=n-k}^{n} a_i \cdot \mathbf{b}_i)\|$ is smaller than a bounding value $R_{k+1} \in (0, \|\mathbf{b}_1\|]$. After searching all possible nodes, the enumeration algorithm finds a lattice vector shorter than R_n at a leaf of depth n, or its projective length is somehow short at a node of depth $k < n$. It is clear that by taking $R_k = \|\mathbf{b}_1\|$ for all $k \in [n]$, the enumeration algorithm always finds the shortest vector $\mathbf{v}_1$ in the lattice, namely $\|\mathbf{v}_1\| = \lambda_1(L)$.

Because $\|\mathbf{b}_1\|$ is often larger than $\lambda_1(L)$, we can set a better searching radius $R_n = \mathrm{GH}(L)$ to decrease the computational cost. We call this *the full enumeration algorithm* and define the full enumeration cost $\mathrm{FEC}(B)$ as the cost of the algorithm for this basis. With the same argument in [19], we can evaluate $\mathrm{FEC}(B)$ using the following equation.

$$\mathrm{FEC}(B) = \sum_{k=1}^{n} \frac{V_k(\mathrm{GH}(L))}{\prod_{i=n-k+1}^{n} \|\mathbf{b}_i^*\|}.$$

Because full enumeration is a cost-intensive algorithm, several improvements have been proposed by considering the trade-offs between running time, searching radius, and success probability [46,19]. Gama-Nguyen-Regev [19] proposed a cost estimation model of the lattice enumeration algorithm to optimize the bounding functions of $R_1, \ldots, R_n$, which were mentioned above. The success probability p of finding a single vector within a radius c is given by

$$p = \Pr_{(x_1,\ldots,x_n) \leftarrow c \cdot S_n} \Big[\sum_{i=1}^{\ell} x_i^2 < R_\ell^2 \text{ for } \forall\ \ell \in [n] \Big],$$

where S_n is the surface of the n-dimensional unit ball. Then, the cost of the enumeration algorithm can be estimated by the number of processed nodes, i.e.,

$$N = \frac{1}{2} \sum_{k=1}^{n} \frac{\mathrm{vol}\{(x_1, \ldots, x_k) \in \mathbb{R}^k : \sum_{i=1}^{\ell} x_i^2 < R_\ell^2 \text{ for } \forall\ \ell \in [k]\}}{\prod_{i=n-k+1}^{n} \|\mathbf{b}_i^*\|}. \tag{1}$$

Note that the factor $1/2$ is based on the symmetry. Using the methodology in [19], Chen-Nguyen proposed a method to find the optimal bounding functions of $R_1, \ldots, R_n$ that minimizes N subject to p.

In this paper, we use the *lattice enumeration cost*, abbreviated as *ENUM cost*, to denote the number N in equation (1). For a lattice L defined by a basis B and parameters $\alpha > 0$ and $p \in [0, 1]$, we use $\mathrm{ENUMCost}(B; \alpha, p)$ to denote the minimized cost N of lattice enumeration with radius $c = \alpha \cdot \mathrm{GH}(L)$ subject to the success probability p. This notion is also defined for a projective sublattice.

3 Lattice Reduction Algorithms

Lattice reduction algorithms transform a given lattice basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ to another basis whose Gram-Schmidt lengths are relatively shorter.

LLL algorithm [29]: The LLL algorithm transforms the basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ using the following two operations: size reduction $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lfloor \mu_{ji} \rceil \mathbf{b}_j$ for $j \in [i-1]$, and neighborhood swaps between $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ if $\|\mathbf{b}^*_{i+1}\|^2 \leq 1/2\|\mathbf{b}^*_i\|^2$ until no update occurs.

BKZ algorithms [45,46]. For a given lattice basis and a fixed blocksize β, the BKZ algorithm processes the following operation in the local block B_i, i.e., the projected sublattice $L_{[i,i+\beta-1]}$ of blocksize β, starting from the first index $i = 1$ to $i = n - 1$. Note that the blocksize β reluctantly shrinks to $n - i + 1$ for large $i > n - \beta + 1$, and thus we sometimes use β' to denote the dimension of B_i, i.e. $\beta' = \min(\beta, n - i + 1)$.

At index i, the standard implementation of the BKZ algorithm calls the enumeration algorithm for the local block B_i. Let $\mathbf{v}$ be a shorter vector found by the enumeration algorithm. Then the BKZ algorithm inserts $\mathbf{v}$ into $\mathbf{b}_{i-1}$ and $\mathbf{b}_i$, and constructs the degenerated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}, \mathbf{b}_i, \ldots, \mathbf{b}_{\min(i+\beta-1,n)})$. For this basis, we apply the LLL algorithm (or BKZ with a smaller blocksize) so that the basis of shorter independent vectors can be obtained. One set of these procedures from $i = 1$ to $n - 1$ is usually called a *tour*. The original version of the BKZ algorithm stops when no update occurs during $n - 1$ iterations. In this paper, we refer to the BKZ algorithm with blocksize β as the BKZ-β.

HKZ reduced basis: The lattice basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is called Hermite-Korkine-Zolotarev (HKZ) reduced [37, Chapter 2] if it is size-reduced $|\mu_{ji}| \leq 1/2$ for all i and j, and $\pi_i(\mathbf{b}_i)$ is the shortest vector in the projective sublattice $L_{[i:n]}$ for all i. We can estimate the Gram-Schmidt length of the HKZ-reduced basis by using the Gaussian heuristic as $\|\mathbf{b}^*_i\| = \mathrm{GH}(L_{[i:n]})$. Since the HKZ-reduced basis is completely reduced in this sense, we will use this to discuss the lower bound of computing time in Section 8.2.

3.1 Some Heuristic Assumptions in BKZ

Gaussian Heuristic in Small Dimensions: Chen and Nguyen observed that the length $\lambda_1(B_i)$ of the shortest vector in the local block B_i is usually larger than $\mathrm{GH}(B_i)$ in small dimensions i.e., small β' [12]. They gave the averaged values of $\|\mathbf{b}^*_i\| / \det(L)^{1/n}$ for the last indexes of highly reduced bases to modify their BKZ simulator, see [12, Appendix C]. For their 50 simulated values for $\|\mathbf{b}^*_{n-49}\|, \ldots, \|\mathbf{b}^*_n\|$, we define the modified Gaussian heuristic constant by

$$\tau_i := \frac{\lambda_1(\pi_{n-i+1}(L))}{\mathrm{GH}(\pi_{n-i+1}(L))} = \frac{\|\mathbf{b}^*_{n-i+1}\|}{V_i(1)^{-1/i} \cdot \prod_{j=n-i+1}^{n} \|\mathbf{b}^*_j\|^{1/i}}. \tag{2}$$

We show the graph of τ_i in Figure 2. We will use τ_i for $i \leq 50$ to denote these modifying constants; for $i > 50$ we define $\tau_i = 1$ following Chen-Nguyen's simulator [12].

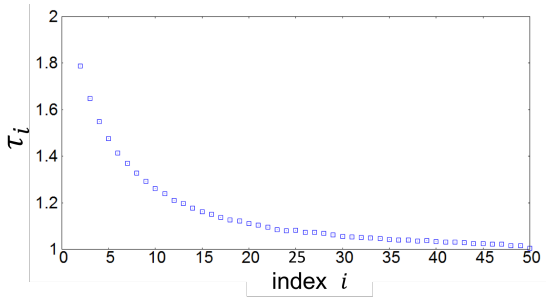

Fig. 2. Modified Gaussian heuristic constant τ_i

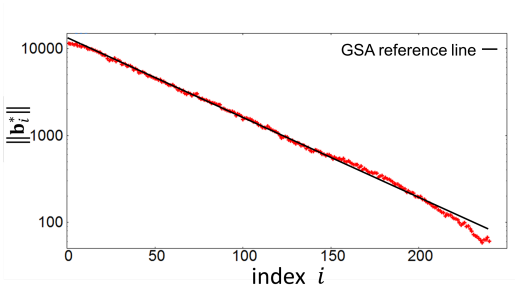

Fig. 3. Semi-log graph of $\|\mathbf{b}_i^*\|$ of a 240-dimensional highly reduced basis

In the rest of this paper, we assume that the shortest vector lengths of β-dimensional local blocks B_i of reduced bases satisfies

$$\lambda_1(B_i) \approx \begin{cases} \tau_\beta \cdot \mathrm{GH}(B_i) & (\beta \leq 50) \\ \mathrm{GH}(B_i) & (\beta > 50) \end{cases}$$

on average.

We note that there exists a mathematical theory to guarantee $\tau_i \to 1$ for random lattices when the dimension goes to infinity [41]. Though it does not give the theoretical guarantee $\tau_i = 1$ for BKZ local blocks, they are very close in our preliminary experiments.

Geometric Series Assumption (GSA): Schnorr [43] introduced *geometric series assumption* (GSA), which says that the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ in the BKZ-reduced basis decay geometrically with quotient r for $i = 1, \ldots, n$, namely, $\|\mathbf{b}_i^*\|^2/\|\mathbf{b}_1\|^2 = r^{i-1}$, for some $r \in [3/4, 1)$. Here r is called *the GSA constant*. Figure 3 shows the Gram-Schmidt lengths of a 240-dimensional reduced basis after processing BKZ-100 using our algorithm and parameters.

It is known that GSA does not hold exactly in the first and last indexes [10]. Several previous works [3,10,43] aimed to modify the reduction algorithm that outputs the reduced basis satisfying GSA. However, it seems difficult to obtain such a reduced basis in practice. In this paper, we aim to modify the

Input: A lattice basis B of n dimensions, blocksize β, and some terminating condition.
Output: A reduced basis B.
1: $B \leftarrow \mathrm{LLL}(B)$;
2: **for** $i = 1$ to $n - 1$
3: Set probability p for local block B_i of fixed blocksize $\beta'_i = \min(\beta, n-i+1)$ and let $M = \lfloor 1/p \rceil$;
4: Generate randomized local blocks $G_1, \ldots, G_M$ from local block B_i, and preprocess $G_1, \ldots, G_M$ (reduction by LLL and small blocksize BKZ);
5: Find a vector $\mathbf{v}$ using lattice enumeration with radius $c = \min\{||\mathbf{b}_i^*||, \sqrt{1.1} \cdot \mathrm{GH}(B_i)\}$ for $G_1, \ldots, G_M$ with probability p;
6: **if** $\mathbf{v}$ satisfies $\|\mathbf{v}\| < \|\mathbf{b}_i^*\|$ **then** update basis B by $\mathbf{v}$;
7: **end-for**
8: **if** terminating condition is satisfied **then** return B **else** goto Step 2;

Fig. 4. Outline of BKZ 2.0

parameters in the first and last indexes so that the proposed simulator performs with optimal efficiency (See section 5.1).

3.2 Chen-Nguyen's BKZ 2.0 Algorithm [12]

We recall Chen-Nguyen's BKZ 2.0 Algorithm in this section. The outline of the BKZ 2.0 algorithm is described in Figure 4.

Speed-up Techniques for BKZ 2.0: BKZ 2.0 employs four major speed-up techniques that differentiate it from the original BKZ:

1. BKZ 2.0 employs the extreme pruning technique [19], which attempts to find shorter vectors in the local blocks B_i with low probability p by randomizing basis B_i to more blocks $G_1, \ldots, G_M$ where $M = \lfloor 1/p \rceil$.

2. For the search radius $\min\{||\mathbf{b}_i^*||, \alpha \cdot \mathrm{GH}(B_i)\}$ in the enumeration algorithm of the local block B_i, Chen and Nguyen set the value as $\alpha = \sqrt{1.1}$ from their experiments, while the previous works set the radius as $\|\mathbf{b}_i^*\|$.

3. In order to reduce the cost of the enumeration algorithm, BKZ 2.0 preprocesses the local blocks by executing the sequence of BKZ algorithm, e.g., 3 tours of BKZ-50 and then 5 tours of BKZ-60, and so on. The parameters blocksize, number of rounds and number of randomized bases, are precomputed to minimize the total enumeration cost.

4. BKZ 2.0 uses the terminating condition introduced in [22], which aborts BKZ within small number of tours. It can find a short vector faster than the full execution of BKZ.

Chen-Nguyen's BKZ 2.0 Simulator: In order to predict the computational cost and the quality of the output basis, they also propose the simulating procedure of the BKZ 2.0 algorithm. Let $(\ell_1, \ldots, \ell_n)$ be the simulated values of

the GS-lengths $\|\mathbf{b}_i^*\|$ for $i = 1, \ldots, n$. Then, the simulated values of the determinant and the Gaussian heuristic are represented by $\prod_{j=1}^{n} \ell_j$ and $\mathrm{GH}(B_i) = V_{\beta'}(1)^{-1/\beta'} \prod_{j=i}^{i+\beta'-1} \ell_i$ where $\beta' = \min\{\beta, n-i+1\}$, respectively.

They simulate a BKZ tour of blocksize β assuming that each enumeration procedure finds a vector of projective length $\mathrm{GH}(B_i)$. Roughly speaking, their simulator updates (ℓ_i, ℓ_{i+1}) to (ℓ'_i, ℓ'_{i+1}) for $i = 1, \ldots, n-1$, where $\ell'_i = \mathrm{GH}(B_i)$ and $\ell'_{i+1} = \ell_{i+1} \cdot (\ell_i / \ell'_i)$. Here, the last 50 GS-lengths are modified using an HKZ reduced basis. The details of their simulator are given in [12, Algorithm 3].

They also present the upper and lower bounds for the number of processed nodes during the lattice enumeration of blocksize β. From [13, Table 4], we extrapolate the costs as

$$\log_2(\mathrm{Cost}_\beta) = 0.000784314\beta^2 + 0.366078\beta - 6.125 \tag{3}$$

Then, the total enumeration cost of performing the BKZ 2.0 algorithm using blocksize β and t tours is given by

$$t \cdot \sum_{i=1}^{n-1} \mathrm{Cost}_{\min\{\beta, n-i+1\}}. \tag{4}$$

To convert the number of nodes into single-threaded time in seconds, we use the rational constant $4 \cdot 10^9/200 = 2 \cdot 10^7$, because they assumed that processing one node requires 200 clock cycles in a standard CPU, and we assume it can work at 4.0GHz.

We note that there are several models to extrapolate $\log_2(\mathrm{Cost}_\beta)$. Indeed, Lepoint and Naehrig [30] consider two models by a quadratic interpolation and a linear interpolation from the table. Albrecht et al. [5] showed another BKZ 2.0 cost estimation that uses an interpolation using the cost model $\log_2(\mathrm{Cost}_\beta) = O(n \log n)$. It is a highly non-trivial task to find a proper interpolation that estimates a precise cost of the BKZ 2.0 algorithm.

We further mention that the upper bound of the simulator is somewhat debatable, because they use the enumeration radius $c = \min\{\sqrt{1.1} \cdot \mathrm{GH}(B_i), \|\mathbf{b}_i^*\|\}$ for $i < n - 30$ in their experiment whereas they assume $c = \mathrm{GH}(B_i)$ for the cost estimation in their upper bound simulation. Thus, the actual cost of BKZ 2.0 could differ by a factor of $1.1^{O(\beta)}$.

4 Optimizing Parameters in Plain BKZ

In this section we consider the plain BKZ algorithm described in Figure 5, and roughly predict the GS-lengths of the output basis, which were computed by the GSA constant r. Using this analysis, we can obtain the optimal settings for parameters (α, p) in Step 4 of the plain BKZ algorithm of blocksize β.

4.1 Relationship of Parameters α, p, β, r

We fix the values of parameters (β, α) and assume that the lattice dimension n is sufficiently large.

Input: A lattice basis B of n dimensions, blocksize β
Output: A reduced basis B.
1: $B \leftarrow \mathrm{LLL}(B)$;
2: $flag = 1$ // set $flag = 0$ when the basis is updated.
3: **for** $i = 1$ to $n - 1$
4: Set (α, p) for local block B_i of fixed blocksize $\beta_i' = \min(\beta, n - i + 1)$;
5: Execute lattice enumeration with probability p and radius $\alpha \cdot \mathrm{GH}(B_i)$;
6: **if** $\mathbf{v}$ satisfies $\|\mathbf{v}\| < \alpha \cdot \mathrm{GH}(B_i)$, **then** update basis B by $\mathbf{v}$ and $flag = 0$;
7: **end-for**
8: **if** $flag = 1$ **then** return B **else** goto Step 2;

Fig. 5. Plain BKZ algorithm

Suppose that we found a vector $\mathbf{v}$ of $\|\mathbf{v}\| < \alpha \cdot \mathrm{GH}(B_i)$ in the local block B_i. We update the basis B_i by inserting $\mathbf{v}$ at i-th index, and perform LLL or small blocksize BKZ on the updated basis.

When the lattice dimension is large, Rogers' theorem [41] says that approximately $\alpha^n/2$ vector pairs $(\mathbf{v}, -\mathbf{v})$ exist within the ball of radius $c = \alpha \cdot \mathrm{GH}(L)$. Since the pruning probability is defined for a single vector pair, we expect the actual probability that the enumeration algorithm finds at least one vector shorter than c is roughly

$$1 - (1 - p)^{\alpha^n/2} \approx p \cdot \frac{\alpha^n}{2}. \tag{5}$$

From relation (5), there may exist one lattice vector in the searching space by setting parameter p as

$$p = \frac{2}{\alpha^\beta}. \tag{6}$$

Remark 1. The probability setting of equation (6) is an optimal choice under our assumption. If p is smaller, the enumeration algorithm finds no short vector with high probability and basis updating at i-th index does not occur, which is a waste of time. On the other hand, if we take a larger p so that there exist $p \cdot \alpha^\beta/2 > 1$ vector pairs, the computational time of the enumeration algorithm increases more than $p \cdot \alpha^\beta/2$ times [19]. Although it can find shorter vectors, this is also a waste of time from the viewpoint of basis updating.

Assume that one vector is found using the enumeration, and also assume that the distribution of it is the same as the random point in the β-dimensional ball of radius $\alpha \cdot \mathrm{GH}(B_i)$. Then, the expected value of $\|\mathbf{v}\|$ is $\frac{\beta}{\beta+1}\alpha \cdot \mathrm{GH}(B_i)$ by letting $K = 1$ in Lemma 1. Thus, we can expect that this is the value $\|\mathbf{b}_i^*\|$ after update.

Therefore, after executing a sufficient number of BKZ tours, we can expect that all the lengths $\|\mathbf{b}_i^*\|$ of the Gram-Schmidt basis satisfy

$$\|\mathbf{b}_i^*\| = \frac{\beta}{\beta + 1}\alpha \cdot \mathrm{GH}(B_i) \tag{7}$$

on average. Hence, under Schnorr's GSA, we have the relation

$$\|\mathbf{b}_i^*\| = \frac{\alpha\beta}{\beta+1} \cdot V_\beta(1)^{-1/\beta} \|\mathbf{b}_i^*\| \prod_{j=1}^{\beta} r^{(j-1)/2\beta}, \tag{8}$$

and the GSA constant is

$$r = \left(\frac{\beta+1}{\alpha\beta}\right)^{\frac{4}{\beta-1}} \cdot V_\beta(1)^{\frac{4}{\beta(\beta-1)}}. \tag{9}$$

Therefore, by fixing (α, β), we can set the probability p and obtain r as a rough prediction of the quality of the output lattice basis. We will use the relations (6) and (9) to set our parameters. Note that any two of β, α, p and r are determined from the other two values.

Remark 2. Our estimation is somehow underestimate, i.e., in our experiments, the found vectors during BKZ algorithm are often shorter than the estimation in equation (7). This gap is mainly from the estimation in (5), which can be explained as follows. Let $(R_1, \ldots, R_\beta)$ be a bounding function of probability p for a vector of length $\|v\|$. Then, the probability p' for a vector of length $\|v'\|$ of a shorter vector is the same as the scaled bounding function $(R'_1, \ldots, R'_\beta)$ where $R'_i = \min\{1.0, R_i \cdot \|v\|/\|v'\|\}$. Here, p' is clearly larger than p due to $R'_i \geq R_i$ for $i \in [\beta]$. Therefore, when the above parameters are used, the quality of the output basis is better than that derived from equation (9) if we perform a sufficient number of tours. Hence, within a few tours, our algorithm can output a basis which has a good quality predicted by our estimation in this section.

4.2 Optimizing Parameters

Now for a fixed parameter pair (β, r), the cost $\mathrm{ENUMCost}(B_i; \alpha, p)$ of the enumeration algorithm in local block B_i satisfying GSA is computable. Concretely, we compute α using the relation (9), fix p by (6), and simulate the Gram-Schmidt lengths of B_i using $\|\mathbf{b}_i^*\| = r^{(i-1)/2}$. Using the computation technique in [19], for several GSA constants r, we search for the optimal blocksize β that minimizes the enumeration cost $\mathrm{ENUMCost}(B_i; \alpha, p)$. The small squares in Figure 6 show the results. From these points, we find the functions $f_1(\beta)$ and $f_2(\beta)$, whose graphs are also in the figure.

We explain how to find these functions $f_1(\beta)$ and $f_2(\beta)$. Suppose lattice dimension n is sufficiently large, and suppose the cost of the enumeration algorithm is roughly dominated by the probability p times the factor at $k = n/2$ in the summation (1). Then $\mathrm{ENUMCost}(B_i; \alpha, p)$ is approximately

$$D = p \cdot \frac{V_{\beta/2}(\alpha \cdot \mathrm{GH}(B_r))}{\prod_{i=\beta/2+1}^{\beta} \|\mathbf{b}_i^*\|} = 2\alpha^{-\beta/2} \frac{V_{\beta/2}(1) V_\beta(1)^{-1/2}}{r^{\beta^2/16}},$$

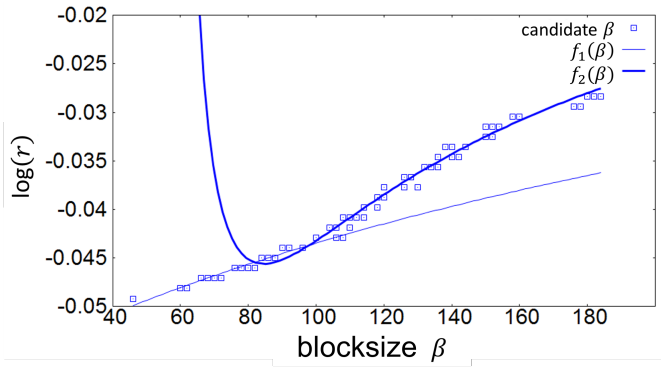

Fig. 6. Relation between β and r that minimizes the computational cost

where from equation (9) we have obtained

$$D \approx Const. \times r^{(\beta^2-2\beta)/16} \cdot \left(\frac{\beta}{e\pi}\right)^{\beta/4}, \text{ and } \frac{\partial \log D}{\partial \beta} \approx \frac{\beta-1}{8}\log r + \frac{1}{4} + \frac{1}{4}\log\frac{\beta}{e\pi}.$$

In order to minimize D, we roughly need the above derivative to be zero; thus, we use the following function of β for our cost estimation with constants c_i

$$\log(r) = 2 \cdot (\log\beta + 1 - \log(e\pi))/(\beta - 1) = \frac{\log c_1\beta}{c_2\beta + c_3}.$$

From this observation, we fix the fitting function model as $f(\beta) = \frac{\log(c_1\beta+c_2)}{c_3\beta+c_4}$.

By using the least squares method implemented in `gnuplot`, we find the coefficients c_i so that $f(\beta)$ is a good approximation of the pairs $(\beta_i, \log(r_i))$. In our curve fitting, we separate the range of β into the interval $[40, 100]$, and the larger one. This is needed for converging to $\log(r) = 0$ when β is sufficiently large; however, our curve fitting using a single natural function did not achieve it. Curves $f_1(\beta)$ and $f_2(\beta)$ in Figure 6 are the results of our curve fitting for the range $[40, 100]$ and the larger one, respectively.

For the range of $\beta \in [40, 100]$, we have obtained

$$\log(r) = f_1(\beta) := -18.2139/(\beta + 318.978) \tag{10}$$

and for the larger blocksize $\beta > 100$,

$$\log(r) = f_2(\beta) := (-1.06889/(\beta - 31.0345)) \cdot \log(0.417419\beta - 25.4889). \tag{11}$$

Note that we will use the relation (10) when the blocksize is smaller than 40.

Moreover, we obtain pairs of β and minimize ENUMCost$(B_i; \alpha, p)$, in accordance with the above experiments. Using the curve fitting that minimizes

$\sum_\beta |f(\beta) - \log_2 \mathrm{ENUMCost}(B_i; \alpha, p)|^2$ using `gnuplot`, we find the extrapolating formula

$$\log_2 \mathrm{MINCost}(\beta) := \begin{cases} 0.1375\beta + 7.153 & (\beta \in [60, 105]) \\ 0.000898\beta^2 + 0.270\beta - 16.97 & (\beta > 105) \end{cases} \tag{12}$$

to $\log_2 \mathrm{ENUMCost}(B_i; \alpha, p)$. We will use this as the standard of the enumeration cost of blocksize β.

Remark 3. Our estimation from the real experiments is $0.25\beta \cdot \mathrm{ENUMCost}(B_i; \alpha, p)$ (See, Section 7.1), which crosses over the estimation of BKZ 2.0 simulator (3) at $\beta = 873$. Thus, the performance of BKZ 2.0 might be better in some extremely high block sizes, while our algorithm has a better performance in the realizable block sizes < 200.

4.3 Parameter Settings in Step 4 in Figure 5

Using the above arguments, we can fix the optimized pair (α, p) for each blocksize β. That is, to process a local block of blocksize β in Step 4 of the plain BKZ algorithm in Figure 5, we compute the corresponding r by equations (10) and (11), and additionally obtain the parameters α by equation (9) and p by equation (6). These are our basic parameter settings.

Modifying blocksize at first indexes: We sometimes encounter the phenomenon in which the actual $\mathrm{ENUMCost}(B_i; \alpha, p)$ in small indexes is much smaller than that in middle indexes. This is because $||\mathbf{b}_i^*||$ is smaller than $\mathrm{GH}(B_i)$ in small indexes. In other words, $\mathbf{b}_i$ is hard to update using the enumeration of blocksize β. To speed up the lattice reduction, we use a heuristic method that enlarges the blocksizes as follows.

From the discussion in the above subsection, we know the theoretical value of the enumeration cost at blocksize β. On the other hand, in the actual computing of BKZ algorithms, the enumeration cost is increased because the sequence $(||\mathbf{b}_i^*||, \ldots, ||\mathbf{b}_{i+\beta-1}^*||)$, which mainly affects the computing cost, does not follow the GSA of slope r exactly. In some experiments, we will verify that the enumeration cost is approximately β times the theoretical value (See, Figure 11 in Section 7.1). Thus, we define the expected enumeration cost in blocksize β as $\beta \cdot \mathrm{MINCost}(\beta)$. With this expectation, we reset the blocksize as the minimum β satisfying $\mathrm{ENUMCost}(B_{[i:i+\beta-1]}; \alpha, p) > \beta \cdot \mathrm{MINCost}(\beta)$.

Modifying (α, p) at last indexes: For large indexes such as $i > n - \beta$, the blocksize of a local block shrinks to $\beta' = \min(\beta, n-i+1)$. In our implementation, we increase the success probability to a new p', while $\mathrm{ENUMCost}(B_i; \alpha', p')$ is smaller than $\beta \cdot \mathrm{MINCost}(\beta)$. We also reset the radius as $\alpha' = (2/p')^{1/\beta}$ from equation (6).

5 Our Proposed Progressive BKZ: Basic Variant

In this section, we explain the basic variant of our proposed progressive BKZ algorithm.

Input: A lattice basis B of n dimensions, starting blocksize β_{start}, and ending blocksize β_{end}.
Output: A reduced basis B.
1: $B \leftarrow \mathrm{LLL}(B)$;
2: **for** $\beta = \beta_{start}$ **to** β_{end} **do**
3: **while** $\mathrm{FEC}(B) > \text{Sim-FEC}(n, \beta)$ **do**
4: **for** $i = 1$ to $n - 1$
5: Set (α, p) for local block B_i of blocksize $\beta' = \min(\beta, n - i + 1)$ using the setting in Section 4.3;
6: Preprocess the basis by the progressive BKZ;
7: Execute lattice enumeration with probability p and radius $\alpha \cdot \mathrm{GH}(B_i)$;
8: **if** $\mathbf{v}$ satisfies $\|\mathbf{v}\| < \alpha \cdot \mathrm{GH}(B_i)$ **then** update basis B by $\mathbf{v}$;
9: **end-for**
10: **end-while**
11: **end-for**

Fig. 7. Our progressive BKZ algorithm (basic variant)

In general, if the blocksize of the BKZ algorithm increases, a shorter vector $\mathbf{b}_1$ can be computed; however, the running cost will eventually increase. The progressive BKZ algorithm starts a BKZ algorithm with a relatively small blocksize β_{start} and increases the blocksize to β_{end} by some criteria. The idea of the progressive BKZ algorithm has been mentioned in several literatures, for example, [12,44,47,24]. The research challenge in the progressive BKZ algorithm is to find an effective criteria for increasing blocksizes that minimizes the total running time.

In this paper we employ the *full enumeration cost* (FEC) in Section 2.3, in order to evaluate the quality of the basis for finding the increasing criteria. Recall that the FEC of basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of n-dimensional lattice L is defined by $\mathrm{FEC}(B) = \sum_{k=1}^{n} \frac{V_k(\mathrm{GH}(L))}{\prod_{i=n-k+1}^{n} \|\mathbf{b}_i^*\|}$, where $\|\mathbf{b}_i^*\|$ represents the GS-lengths. Note that $\mathrm{FEC}(B)$ eventually decreases after performing several tours of the BKZ algorithm using the fixed blocksize β.

Moreover, we construct a simulator that evaluates the GS-lengths by the optimized parameters α, p, β, r for the BKZ algorithm described in the local block discussion in Section 4.3. The simulator for an n-dimensional lattice only depends on the blocksize β of the local block; we denote by Sim-GS-lengths(n, β) the simulated GS-lengths $(\ell_1, \ldots, \ell_n)$. The construction of simulator will be presented in Section 5.1.

For this purpose, we define some functions defined on the simulated GS-lengths $(\ell_1, \ldots, \ell_n)$. $\text{Sim-GH}(\ell_1, \ldots, \ell_n) = V_n(1)^{-1/n} \prod_{j=1}^{n} \ell_j^{1/n}$ is the simulated Gaussian heuristic. The simulated value of full enumeration cost is

$$\text{Sim-FEC}(\ell_1, \ldots, \ell_n) := \sum_{k=1}^{n} \frac{V_k(\text{Sim-GH}(\ell_1, \ldots, \ell_n))}{\prod_{i=n-k+1}^{n} \ell_i}.$$

Further, for $(\ell_1, \ldots, \ell_n) = \text{Sim-GS-lengths}(n, \beta)$, we use the notation $\text{Sim-FEC}(n, \beta) := \text{Sim-FEC}(\ell_1, \ldots, \ell_n)$ in particular. The simulated enumeration cost $\text{Sim-ENUMCost}(\ell_1, \ldots, \ell_\beta; \alpha, p)$ is defined by $\text{ENUMCost}(B; \alpha, p)$ for a lattice basis B that has GS-lengths $\|\mathbf{b}_i^*\| = \ell_i$ for $i \in [\beta]$.

The key point of our proposed progressive BKZ algorithm is to increase the blocksize β if $\text{FEC}(B)$ becomes smaller than $\text{Sim-FEC}(n, \beta)$. In other words, we perform the BKZ tours of blocksize β while $\text{FEC}(B) > \text{Sim-FEC}(n, \beta)$. We describe the proposed progressive BKZ in Figure 7.

Remark 4. In the basic variant of our progressive BKZ described in Section 6.1, we increase the blocksize β in increments of one in Step 2. However, we will present an optimal strategy for increasing the blocksize in Section 5.

5.1 Sim-GS-lengths(n, β): Predicting Gram-Schmidt Lengths

In the following, we construct a simulator for predicting the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ obtained from the plain BKZ algorithm of blocksize β.

Our simulator consists of two phases. First, we generate approximated GS-lengths using Gaussian heuristics; we then modify it for the first and last indexes of GSA in Section 3.1. We will explain how to compute $(\ell_1, \ldots, \ell_n)$ as the output of $\text{Sim-GS-lengths}(n, \beta)$.

First phase: Our simulator computes the initial value of $(\ell_1, \ldots, \ell_n)$.

We start from the last index by setting $\ell_n = 1$, and compute ℓ_i backwards. From equations (2) and (7) we are able to simulate the GS-lengths ℓ_i by solving the following equation of ℓ_i:

$$\ell_i = \max\left\{\frac{\beta'}{\beta'+1}\alpha, \tau_{\beta'}\right\} \cdot \text{GH}(\ell_i, \ldots, \ell_{i+\beta'-1}), \text{ where } \beta' = \min(\beta, n-i+1). \tag{13}$$

Here, α is the optimized radius parameter in Section 4.3 and $\tau_{\beta'}$ is the coefficient of the modified Gaussian heuristic.

This simple simulation in the first phase is sufficient for smaller blocksizes ($\beta < 30$). However, for simulating larger blocksizes, we must modify the GS-lengths of the first and last indexes in Section 3.1.

Second phase: To modify the results of the simple simulation, we consider our two modifying methods described in Section 4.3. We recall that $\text{MINCost}(\beta)$ is the standard value of the enumeration cost of blocksize β.

We first consider the modification for the last indexes $i > n - \beta + 1$, i.e., a situation in which the blocksize is smaller than β. We select the modified probability p_i at index i so that $\text{Sim-ENUMCost}(\ell_i, \ldots, \ell_n; \alpha_i, p_i) = \text{MINCost}(\beta)$, where $\ell_i, \ldots, \ell_n$ is the result of the first simulation, and we use $\alpha_i = (2/p_i)^{n-i+1}$. After all (α_i, p_i) for $n - \beta + 1 \leq i \leq n$ are fixed, we modify the GS-lengths by solving the following equation of ℓ_i again:

$$\ell_i = \max\left\{\frac{\beta'}{\beta'+1}\alpha_i, \tau_{\beta'}\right\} \cdot \text{GH}(\ell_i, \ldots, \ell_n) \text{ where } \beta' = n-i+1.$$

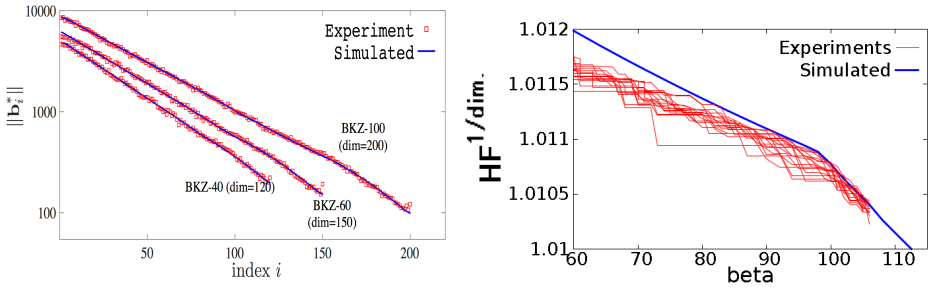

Fig. 8. Left figure: Semi-log graph of $\|\mathbf{b}_i^*\|$ of reduced random lattices from the SVP Challenge problem generator: Simulation (bold lines) vs. Experiment (small squares). **Right figure**: The root Hermite factor of reduced random 300-dimensional bases after BKZ-β. Simulation (bold red lines) vs. Experiment (thin blue lines).

Next, using the modified $(\ell_1, \ldots, \ell_n)$, we again modify the first indexes as follows. We determine the integer parameter $b > 0$ for the size of enlargement. For $b = 1, 2, \ldots$, we reset the blocksize at index i as $\beta_i := \beta + \max\{(b - i + 1)/2, b - 2(i - 1)\}$ for $i \in \{1, \ldots, b\}$. Using these blocksizes, we recompute the GS-lengths by solving equation (13) from $i = \beta_i$ to 1. Then, we compute Sim-ENUMCost$(\ell_1, \ldots, \ell_{\beta+b}; \alpha, p)$. We select the maximum b such that this simulated enumeration cost is smaller than $2 \cdot$ MINCost(β).

Experimental result of our GS-lengths simulator: We performed some experiments on the GS-lengths for some random lattices from the Darmstadt SVP Challenge [49]. We computed the GS-lengths for 120, 150 and 200 dimensions using the proposed progressive BKZ algorithm, with ending blocksizes of 40, 60, and 100, respectively (Note that the starting blocksize is irrelevant to the quality of the GS-lengths). The simulated result is shown in Figure 8. Almost all small squares of the computed GS-lengths are plotted on the bold line obtained by our above simulation. Our simulator can precisely predict the GS-lengths of these lattices. The progress of the first vector, which uses 300-dimensional lattices, is also shown in the figure.

5.2 Expected Number of BKZ Tours at Step 3

At Step 3 in the proposed algorithm (Figure 7) we iterate the BKZ tour with blocksize β as long as the full enumeration cost FEC(B) is larger than the simulated cost Sim-FEC(n, β). In the following we estimate the expected number of BKZ tours (we denote it as $\sharp tours$) at blocksize β.

In order to estimate $\sharp tours$, we first compute $(\ell_1, \ldots, \ell_n)$ and the output of Sim-GS-lengths$(n, \beta - 1)$, and update it by using the modified Chen-Nguyen's BKZ 2.0 simulator described in Section 3.2, until Sim-FEC$(\ell_1, \ldots, \ell_n)$ is smaller than Sim-FEC(n, β). We simulate a BKZ tour by updating the pair (ℓ_i, ℓ_{i+1}) to

(ℓ'_i, ℓ'_{i+1}) for $i = 1, \ldots, n-1$ according to the following rule:

$$\begin{aligned} \ell'_i &= \max\left\{\tfrac{\beta}{\beta+1}\alpha, \tau_\beta\right\} \cdot \mathrm{GH}(\ell_i, \ldots, \ell_{\min(n, i+\beta-1)}) \\ \text{and } \ell'_{i+1} &= \ell_{i+1} \cdot (\ell_i/\ell'_i). \end{aligned}$$

At the simulation of t-th BKZ tour, write the input GS-lengths $(\ell'_1, \ldots, \ell'_n)$; i.e., the output of the $(t-1)$-th BKZ tour. We further denote the output of t-th BKZ tour as $(\ell_1, \ldots, \ell_n)$. Suppose they satisfy

$$\text{Sim-FEC}(\ell'_1, \ldots, \ell'_n) > \text{Sim-FEC}(n, \beta) > \text{Sim-FEC}(\ell_1, \ldots, \ell_n).$$

Then, our estimation of $\sharp tours$ is the interpolated value:

$$\sharp tours = (t-1) + \frac{\text{Sim-FEC}(\ell'_1, \ldots, \ell'_n) - \text{Sim-FEC}(n, \beta)}{\text{Sim-FEC}(\ell'_1, \ldots, \ell'_n) - \text{Sim-FEC}(\ell_1, \ldots, \ell_n)}. \tag{14}$$

Note that we can use this estimation for other BKZ strategies, although we estimate the number of BKZ tours from BKZ-$(\beta-1)$ basis to BKZ-β basis, using BKZ-β algorithm. We will estimate the tours for other combinations of starting and ending blocksizes, and use them in the algorithm.

6 Our Progressive BKZ: Optimizing Blocksize Strategy

We propose how to optimally increase the blocksize β in the proposed progressive BKZ algorithm. Several heuristic strategies for increasing the blocksizes have been proposed. The following sequences of blocksizes after LLL-reduction have been used in the previous literatures:

$20 \to 21 \to 22 \to 23 \to 24 \to \cdots$ Gama and Nguyen [18]
$2 \to 4 \to 8 \to 16 \to 32 \to \cdots$ Schnorr and Shevchenko [47],
$2 \to 4 \to 6 \to 8 \to 10 \to \cdots$ Haque, Rahman, and Pieprzyk [24],
$50 \to 60 \to 70 \to 80 \to 90 \to \cdots$ Chen and Nguyen [12,13]

The timings for changing to the next blocksize were not explicitly given. They sometimes continue the BKZ tour until no update occurs as the original BKZ. In this section we try to find the sequence of the blocksizes that minimizes the total cost of the progressive BKZ to find a BKZ-β reduced basis. To find this strategy, we consider all the possible combinations of blocksizes used in our BKZ algorithm and the timing to increase the blocksizes.

Notations on blocksize strategy: We say a lattice basis B of dimension n is β-reduced when $\mathrm{FEC}(B)$ is smaller than $\text{Sim-FEC}(n, \beta)$. For a tuple of blocksizes $(\beta^{alg}, \beta^{start}, \beta^{goal})$ satisfying $2 \le \beta^{start} < \beta^{goal} \le \beta^{alg}$, the notation

$$\beta^{start} \overset{\beta^{alg}}{\to} \beta^{goal}$$

is the process of the BKZ following algorithm. The input is a β^{start}-reduced basis B, and the algorithm updates B using the tours of BKZ-β^{alg} algorithm with parameters in Section 4.3. It stops when $\mathrm{FEC}(B) < \text{Sim-FEC}(n, \beta^{goal})$.

Input: A lattice basis B of n dimensions,
Blocksize strategy $\{(\beta_j^{alg}, \beta_j^{goal})\}_{j=1,\ldots,D}$
Output: A β_D^{goal}-reduced basis B.
1: $B \leftarrow \mathrm{LLL}(B)$;
2: **for** $j = 1$ **to** D **do**
3: **while** $\mathrm{FEC}(B) > \text{Sim-FEC}(n, \beta_j^{goal})$ **do**
4-9: The same as Step 4-9 in Figure 7 with blocksize β_j^{alg}
10: **end-while**
11: **end-for**

Fig. 9. Our progressive BKZ algorithm with blocksize strategy

$\mathrm{TimeBKZ}(n, \beta^{start} \overset{\beta^{alg}}{\rightarrow} \beta^{goal})$ is the computing time in seconds of this algorithm. We provide a concrete simulating procedure in this and the next sections. We assume that TimeBKZ is a function of $n, \beta^{alg}, \beta^{start}$ and β^{goal}.

To obtain a BKZ-β reduced basis from an LLL reduced basis, many blocksize strategies are considered as follows:

$$\beta_0^{goal} = \mathrm{LLL} \overset{\beta_1^{alg}}{\rightarrow} \beta_1^{goal} \overset{\beta_2^{alg}}{\rightarrow} \beta_2^{goal} \overset{\beta_3^{alg}}{\rightarrow} \cdots \overset{\beta_D^{alg}}{\rightarrow} \beta_D^{goal} (= \beta). \tag{15}$$

We denote this sequence as $\{(\beta_j^{alg}, \beta_j^{goal})\}_{j=1,\ldots,D}$, and regard it as the progressive BKZ given in Figure 9.

6.1 Optimizing Blocksize Strategies

Our goal in this section is to find the optimal sequence that minimizes the total computing time

$$\sum_{i=1}^{D} \mathrm{TimeBKZ}(n, \beta_{i-1}^{goal} \overset{\beta_i^{alg}}{\rightarrow} \beta_i^{goal}) \tag{16}$$

of the progressive BKZ algorithm to find a BKZ-β_D^{goal} basis.

Based on our experimental results, which are given in Section 7, we can estimate the computing time of the BKZ algorithm:

$$\begin{aligned} &\mathrm{TimeBKZ}(n, \beta^{start} \overset{\beta^{alg}}{\rightarrow} \beta^{goal}) \text{ [sec.]} \\ &= \sum_{t=1}^{\sharp tours} \left[1.5 \cdot 10^{-10} \cdot (\beta^{alg})^2 n^3 + 1.5 \cdot 10^{-8} \cdot \beta^{alg} \sum_{i=1}^{n-1} \mathrm{ENUMCost}(B_i; \alpha, p) \right] \end{aligned} \tag{17}$$

when dimension n is small ($n < 400$), and

$$\begin{aligned} &\mathrm{TimeBKZ}(n, \beta^{start} \overset{\beta^{alg}}{\rightarrow} \beta^{goal}) \text{ [sec.]} \\ &= \sum_{t=1}^{\sharp tours} \left[2.5 \cdot 10^{-4} \cdot \frac{n - \beta^{alg}}{250 - \beta^{alg}} \cdot n^2 . + 3.0 \cdot 10^{-8} \cdot \beta^{alg} \sum_{i=1}^{n-1} \mathrm{ENUMCost}(B_i; \alpha, p) \right] \end{aligned} \tag{18}$$

when dimension n is large ($n \geq 400$). The difference is caused by the difference in the types to compute Gram-Schmidt variables in implementation. The former and latter implementation employ `quad_float` and `RR` (320 bits) respectively, where `RR` is the arbitrary precision floating point type in the NTL library [48]. To compute $\sharp tours$ we use the procedure in Section 5.2. The input of the ENUMCost function is from Sim-GS-lengths(n, β^{start}) at the first tour. From the second tour, we use the updated GS-lengths by the Chen-Nguyen's simulator with blocksize β^{alg}.

Using these computing time estimations, we discuss how to find the optimal blocksize strategy (15) that minimizes the total computing time. In this optimizing procedure, the input consists of n and β, the lattice dimension and the goal blocksize. We denote TimeBKZ(n, β^{goal}) to be the minimized time in seconds to find a β-reduced basis from an LLL reduced basis, that is, the minimum of (16) from among the possible blocksize strategies. By definition, we have

$$\text{TimeBKZ}(n, \beta^{goal}) = \min_{\beta', \beta^{alg}} \left\{ \text{TimeBKZ}(n, \beta') + \text{TimeBKZ}(n, \beta' \overset{\beta^{alg}}{\rightarrow} \beta^{goal}) \right\}$$

where we take the minimum over the pair of blocksizes (β', β^{alg}) satisfying $\beta' < \beta^{goal} \leq \beta^{alg}$.

For the given (n, β), our optimizing algorithm computes TimeBKZ$(n, \bar{\beta})$ from small $\bar{\beta}$ to the target $\bar{\beta} = \beta$. As the base case, we define that TimeBKZ$(n, 20)$ represents the time to compute a BKZ-20 reduced basis using a fixed blocksize, starting from an LLL reduced basis:

$$\text{TimeBKZ}(n, 20) := \min_{\beta^{alg}} \left\{ \text{TimeBKZ}(n, \text{LLL} \overset{\beta^{alg}}{\rightarrow} 20) \right\}.$$

6.2 Simulating Time to Find Short Vectors in Random Lattices

In this section, we give our simulating result of finding short vectors for random lattices. For the given lattice dimension n and the target length, we simulate the necessary BKZ blocksize β so that ℓ_1 of Sim-GS-lengths(n, β) is smaller than the target length. Then, we simulate TimeBKZ(n, β) by using the method in Section 6.1.

As an example, in Table 2, we show the optimized blocksize strategy and computing time to find a 102-reduced basis in $n = 600$ dimension. We estimate blocksize 102 is necessary to find a vector shorter than $n \cdot \det(L)^{1/n}$, which is the condition to enter the Hall of Fame in the Approximate Ideal Lattice Challenge [49].

Table 3 shows the blocksize and predicted total computing time in seconds to find a vector shorter than $n \cdot \text{GH}(L)$ (this corresponds to the n-approximate SVP from the learning with errors problem [40].), $n \cdot \det(L)^{1/n}$ (from the Approximate Ideal Lattice Challenge published in Darmstadt [49]), and $\sqrt{n} \cdot \text{GH}(L)$. For comparison, the simulating result of BKZ 2.0 is given to find $n \cdot \det(L)^{1/n}$. Recall that their estimated cost in seconds is given by $\sharp\text{ENUM}/2 \cdot 10^7$. From Table 3, our algorithm is asymptotically faster than BKZ 2.0.

Table 2. The optimized blocksize strategy and computational time in seconds in 600-dimensional lattice.

$\overset{\beta^{alg}}{\rightarrow}\ \beta^{goal}$	LLL	$\overset{32}{\rightarrow} 21$	$\overset{50}{\rightarrow} 36$	$\overset{58}{\rightarrow} 46$	$\overset{65}{\rightarrow} 55$	$\overset{71}{\rightarrow} 61$	$\overset{75}{\rightarrow} 70$	$\overset{81}{\rightarrow} 76$	$\overset{85}{\rightarrow} 84$
$\log_2$(Time [sec.])		15.61	15.86	16.04	16.21	16.31	16.51	16.70	17.07
$\overset{\beta^{alg}}{\rightarrow}\ \beta^{goal}$		$\overset{89}{\rightarrow} 88$	$\overset{91}{\rightarrow} 90$	$\overset{93}{\rightarrow} 92$	$\overset{99}{\rightarrow} 98$	$\overset{101}{\rightarrow} 100$	$\overset{103}{\rightarrow} 102$		
$\log_2$(Time [sec.])		17.42	17.67	17.97	18.89	19.49	20.09		

Table 3. Simulated $\log_2$(Time [sec.]) of our algorithm and BKZ 2.0 for large dimensions to find short vectors. The time is after LLL-reduced basis. Because the estimate for BKZ 2.0 is only the cost for enumeration, our algorithm appears to be slow in small blocksizes.

Goal	$n \cdot \mathrm{GH}(L)$		$n \cdot \det(L)^{1/n}$			$\sqrt{n} \cdot \mathrm{GH}(L)$	
n	β	$\log_2$(Ours)	β	$\log_2$(Ours)	$\log_2$(BKZ 2.0)	β	$\log_2$(Ours)
600	35	15.8	102	20.1	16.0	145	38.4
650	45	16.6	114	24.3	21.9	157	51.0
700	59	17.3	124	28.3	28.2	169	60.4
800	100	20.8	144	38.6	41.3	193	82.1

6.3 Comparing with Other Heuristic Blocksize Strategies

In this section, we compare the blocksize strategy of our progressive BKZ in Figure 9. Using a random 256-dimensional basis, we experimented and simulated the progressive BKZ to find a BKZ-128 reduced basis with the three following strategies:

$$2 \overset{4}{\rightarrow} 4 \overset{8}{\rightarrow} 8 \overset{16}{\rightarrow} 16 \overset{32}{\rightarrow} 32 \overset{64}{\rightarrow} 64 \overset{128}{\rightarrow} 128$$
(Schnorr-Shevchenko's doubling strategy [47])
$$2 \overset{20}{\rightarrow} 20 \overset{21}{\rightarrow} 21 \overset{22}{\rightarrow} 22 \overset{23}{\rightarrow} 23 \overset{24}{\rightarrow} 24 \overset{25}{\rightarrow} \cdots \overset{128}{\rightarrow} 128$$
(Simplest step-by-step in Figure 7)
$$2 \overset{30}{\rightarrow} 20 \overset{35}{\rightarrow} 25 \overset{39}{\rightarrow} 29 \overset{43}{\rightarrow} 33 \overset{47}{\rightarrow} 37 \overset{48}{\rightarrow} \cdots \overset{128}{\rightarrow} 128$$
(Optimized blocksize strategy in Figure 9)

In experiment, our simple and optimized strategy takes about 27.1 minutes and about 11.5 minutes respectively to achieve BKZ-64 basis after the LLL reduction. On the other hand, Schnorr-Schevchenko's doubling strategy takes about 21 minutes.

After then, the doubling strategy switches to BKZ-128 and takes about 14 single-core days to process the first one index, while our strategies comfortably continues the execution of progressive BKZ.

Our simulator predicts that it takes about $2^{25.3}$, $2^{25.1}$ and $2^{37.3}$ seconds to finish BKZ-128 by our simple, optimized, and Schnorr-Schevchenko's doubling strategy, respectively. Our strategy is about 5000 times faster than the doubling strategy.

Interestingly, we find that the computing time of simple blocksize strategy is close to that of optimized strategy in many simulations when the blocksize

5-1: Compute the Gram-Schmidt lengths $||\mathbf{b}_i^*||$ and coefficients μ_{ij} corresponding to the local block B_i of blocksize $\beta' = \min(\beta, n-i+1)$
5-2: Set (α, p) for B_i using the setting in Section 4.3;
6-1: Set near optimized pruning coefficients $(R_1, \ldots, R_\beta)$ for (B_i, α, p);
6-2: Preprocess B_i by the simple version of progressive BKZ in Section 7.1;
6-3: **if** enumeration cost for B_i computed using $(\alpha_p, R_1, \ldots, R_\beta)$ is large **then** optimize the bounding function;
7: $\{\mathbf{v}_1, \ldots, \mathbf{v}_h\} \leftarrow$ (lattice enumeration for B_i using $(\alpha_p, R_1, \ldots, R_\beta)$);
8-1: Construct the degenerated basis
$$(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_g}, \mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta'-1})$$
8-2: Apply the LLL algorithm to the basis
$$(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_g}, \mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta'-1})$$
and erase the zero vectors

Fig. 10. One BKZ tour of our implementation to process the local block. These lines correspond to Step 5-8 in Figure 7.

is larger than about 100. Hence, the simple blocksize strategy would be better than the optimizing blocksize strategy in practice, because the latter needs a heavy precomputing as in Section 6.1.

7 Our Implementation and Cost Estimation for Processing Local Blocks

In this section we describe how to derive the estimation of the computing times of equations (17) and (18) of Step 3-10 in Figure 7. The total computing time is the sum of times to process local blocks (corresponds to Step 5-8 in Figure 7):

$$\begin{aligned}&\mathrm{TimeBKZ}(n, \beta^{start} \overset{\beta^{alg}}{\rightarrow} \beta^{goal}) = \\ &\sum_{t=1}^{\sharp tours} \sum_{i=1}^{n-1} \Big[\text{Time of processing local block } B_i \text{ with parameters } (\alpha, p)\Big].\end{aligned} \tag{19}$$

Because $\sharp tours$ is already given in Section 5.2, we consider the factor of time of processing local block B_i.

For the details of analysis, we introduce a pseudo-code of our implementation in Figure 10. We decompose the running time over the internal summation as follows.

$$\begin{aligned}&\sum_{i=1}^{n-1}(\text{Time of processing local block } B_i \text{ with parameters } (\alpha, p)) = \\ &Time_{GS} + Time_{Optimize} + Time_{Preprocess} + Time_{Enum} + Time_{LLL} + misc,\end{aligned} \tag{20}$$

where $Time_{GS}$ is for Step 5-1, $Time_{Preprocess}$ is for Step 6-2, $Time_{Optimize}$ is for 6-3, $Time_{Enum}$ is for Step 7, and $Time_{LLL}$ is the time for Step 8-2. Note that the miscellaneous part is the time for all the other steps including the memory allocation and vector insertion which are negligible.

In the rest of this section, we introduce our implementation and give a rough estimating formula of computing times with some order notations. (Section 7.1 and 7.2.) Then, we fix the rational coefficients by using the experimental results. Although some of implementing techniques are folklore or trivial, we give them for the completeness of the paper.

How to construct the degenerated basis in Step 8-1: Suppose we have a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_h$ found in Step 7. First, compute the projections of $\mathbf{v}_i$ onto $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ and let $\mathbf{v}_{i_1}$ be the vector with shortest projection. After choosing the g-th vector, $\mathbf{v}_{i_g}$, the next vector $\mathbf{v}_{i_{g+1}}$ is selected as follows. Compute the projections $||\pi'(\mathbf{v}_i)||$ of $\mathbf{v}_i$ onto $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_g}$. If there exists i such that $0 < ||\pi'(\mathbf{v}_i)|| < ||\mathbf{b}^*_{i+g-1}||$, then $\mathbf{v}_{i_{g+1}}$ is $\mathbf{v}_i$ that minimizes $||\pi'(\mathbf{v}_i)||$; otherwise, stop this process and output the degenerated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_g}, \mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta'-1})$.

7.1 Implementation and Time Estimation of Step 6 and 7: $Time_{Preprocess} + Time_{Enum} + Time_{Optimize}$

We give the details of our implementation from Step 6 to 7 in Figure 10. The goal of this section is to justify

$$\begin{aligned} &Time_{Optimize} + Time_{Preprocess} + Time_{Enum} \\ &= W \cdot \beta \cdot \sum_{i=1}^{n-1} \text{Sim-ENUMCost}(\ell'_i, \ldots, \ell'_{i+\beta-1}; \alpha, p) \end{aligned} \tag{21}$$

by using a constant W which we will determine in Section 7.4. Here, $\ell'_i, \ldots, \ell'_{i+\beta-1}$ is a part of output of Sim-GS-lengths(n, β) or its updated values by the simulator in Section 5.1.

Computing bounding coefficients: In Step 6-1, we use Aono's precomputing technique [6] to generate the bounding coefficients $R_1, \ldots, R_\beta$ for pruning in the enumeration algorithm in Section 2.3. We fix this bounding function to predict the enumeration cost in the preprocessing step 6-2 (see the next paragraph). After preprocessing, in Step 6-3, we search better bounding coefficients if the expected number of enumeration searching nodes is larger than 10^8, which corresponds to a few seconds in a single thread. The procedure for finding a better bounding coefficients is the simple algorithm that considers random perturbations of $(R_1, \ldots, R_\beta)$ as the strategy in [13].

$Time_{Optimize}$ is the sum of the computing time in Step 6-1 and 6-3. It is significantly smaller than the cost of lattice vector enumeration. In small blocksizes, Step 6-1 can be done in about 100 milliseconds and Step 6-3 is skipped. Thus, $Time_{Optimize} \approx 0$. Moreover, since the precomputing technique outputs $R_1, \ldots, R_\beta$ as a function of the dimension β, target probability and target GSA

constant, we can reuse them in implementation. Thus, the computing times of these steps are very small. Note that the target GSA constant is computed from the line fitting by the least square method to the points $(i, \log_2 ||\mathbf{b}_i^*||)$ when the GS-lengths $(||\mathbf{b}_i^*||, \ldots, ||\mathbf{b}_{i+\beta-1}^*||)$ is given.

On the other hand, for large blocksizes (larger than about 80), the time for lattice vector enumeration is much larger than that of optimization in Step 6-3. Therefore, we can assume $Time_{Optimize} \ll Time_{Enum}$ in the both situations.

Implementation of preprocess step: In Step 6-2, we use our progressive BKZ algorithm proposed in this paper with small blocksizes. It starts with the blocksize $\bar{\beta} = 15$ and increases $\bar{\beta}$ one by one when $\mathrm{FEC}(B_i) < \text{Sim-FEC}(\beta, \bar{\beta})$.

At the end of each tour, we compute $\mathrm{ENUMCost}(B_i; \alpha, p)/\texttt{persec}$ as the estimation for the time of main enumeration. Here, $\mathrm{ENUMCost}(B_i; \alpha, p)$ is estimated by using near optimized coefficients generated in Step 6-1. `persec` is the number of processed nodes in lattice vector enumeration in one second, which can be determined from our preliminary benchmark. It is about $6.0 \cdot 10^7$ at the maximum in a single threaded implementation. On the other hand, it slows down to about $\sharp threads \times 3.0 \cdot 10^7$ when we use the multithreaded programming. In our implementation, we use 12 threads which can process about $3.0 \cdot 10^8$ nodes in one second. During the preprocessing, we obtain several lattice bases in the ends of tours. We keep the basis that takes minimum $\mathrm{ENUMCost}(B_i; \alpha, p)$ among them and also keep its minimized cost. The preprocessing subroutine terminates when the elapsed (wallclock) time exceeds the kept minimum cost in seconds scaled by the benchmarking result. We use the minimum pair $(B, cost)$ for the preprocessing output.

Because our preprocessing subroutine works in a single thread, $Time_{Preprocess}$ is proportional to the recorded minimum cost. While the actual time for the enumeration decreases by the optimizing bounding coefficients, we does not consider it and assume that $Time_{Preprocess} = A_{Preprocess} \cdot Time_{Enum}$ by a constant $A_{Preprocess}$.

Implementation of enumeration step: In Step 7, we implement our modified version of the lattice vector enumeration subroutine in the NTL library. In order to speed up, we use `double` type to keep the Gram-Schmidt coefficients and lengths during the enumeration while the original version uses `quad_float`. In addition, we use assembly codes optimized for a latest CPU.

In the lattice enumeration subroutine, we find many vectors whose projective lengths are small, although their non-projective lengths are not small enough. In our implementation, we store the first $h = 16$ vectors ordered in the lexicographic order of the pair $(\beta - k, ||\pi_{\beta-k}(\mathbf{v})||)$ satisfying $||\pi_{\beta-k}(\mathbf{v})|| \leq R_{k+1}$. After the enumeration, the output is the stored vectors $\mathbf{v}_1, \ldots, \mathbf{v}_h$.

Our model and cost estimation of the enumeration step: To estimate $Time_{Enum}$, we performed experiments to compare the numbers of processed nodes and the simulated values.

Figure 11 shows the maximum and average of the number of processed nodes during the first tour of our progressive BKZ of blocksize β using random lattices of 300 dimensions. The symbol "+" indicates the actual number of nodes during

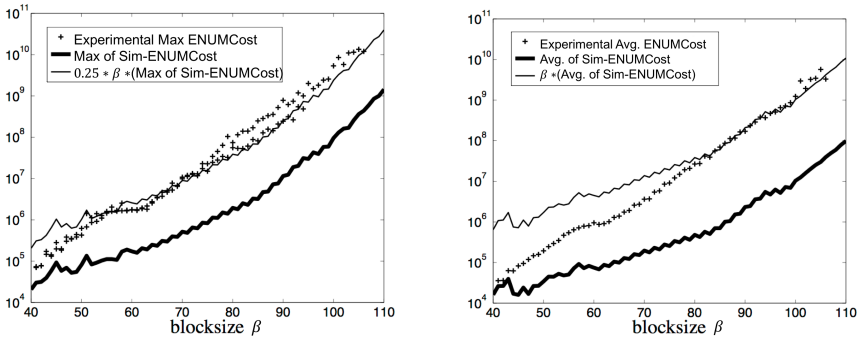

Fig. 11. Maximum (left)/Average (right) of the number of nodes during the first tour at blocksize β

the enumeration subroutine, while the bold curve is the maximum and average of Sim-ENUMCost$(\ell_i, \ldots, \ell_{i+\beta-1}; \alpha, p)$ for $i = 1, \ldots, n-\beta+1$, where $(\ell_1, \ldots, \ell_n)$ is the output of simulator Sim-GS-lengths$(n, \beta - 1)$ in Section 5.1 and (α, p) is from Section 4.3.

As we can see in Figure 11, the numbers of processed nodes in our experiment are larger than the simulated numbers. This phenomenon is mainly caused by the basis updating, i.e., the vector inserting process in Step 8 in Figure 10. By inserting found vectors, the GS-lengths are changed and the corresponding enumeration cost is increased.

From the above experiments, we find the maximum of actual number of nodes is about 0.25β times the maximum of Sim-ENUMCost$(\ell_i, \ldots, \ell_{i+\beta-1}; \alpha, p)$ (See the left-hand side of Figure 11). A similar proportional relation is found in the average number of nodes (the right-hand side of the figure). Therefore, we assume the actual number of processed nodes during one BKZ tour for the basis whose GS-lengths are $(\ell_1, \ldots, \ell_n)$ is

$$A_{Enum} \cdot \beta \cdot \sum_{i=1}^{n-1} \text{Sim-ENUMCost}(\ell_i, \ldots, \ell_{i+\beta'-1}; \alpha, p),$$

where A_{Enum} is a rational constant. Thus, $Time_{Enum}$ in seconds is this value divided by `persec` (the number of processed nodes in one second in a single thread).

Total computing time in seconds in Step 6 and 7: Summarizing the above argument, we have

$$\begin{aligned} &Time_{Optimize} + Time_{Preprocess} + Time_{Enum} \quad [\text{sec.}] \\ &= \frac{(A_{Preprocess} + 1) \cdot A_{Enum} \cdot \beta \cdot \sum_{i=1}^{n-1} \text{Sim-ENUMCost}(\ell'_i, \ldots, \ell'_{i+\beta'-1}; \alpha, p)}{\texttt{persec}}. \end{aligned}$$

Hence, letting $W = (A_{Preprocess}+1)\cdot A_{Enum}\cdot\beta/\texttt{persec}$, we get the equation (21). Note that we regard $A_{Optimize} = 0$.

7.2 Estimating Time of Step 5-1 and 8-2: $Time_{GS} + Time_{LLL}$ when the lattice dimension is small

In this section, we introduce our implementation to compute the GS-variables (i.e., the Gram-Schmidt lengths $||\mathbf{b}_i^*||$ and the coefficients μ_{ij}) used from Step 5-2 to 6-3. Then we show how to update these values in Step 8-2. In our implementation, we use the different precision types to treat GS-variables. If the lattice dimension n is smaller than 400, we compute and keep the GS-variables by using the `quad_float` type variables. Otherwise, we compute and keep them by using `RR` types which will be explained the detail in the next subsection.

In small dimensions $n < 400$, we keep GS-variables in the `quad_float` type and compute them directly. In short, Step 5-1 merely consists of copying the necessary parts $||\mathbf{b}_{i'}^*||$ and $\mu_{i'j'}$ for $i' \in \{i, \ldots, i+\beta'-1\}$ and $j' \in [i']$ that corresponds to the local block B_i.

In Step 8-2, we apply the LLL algorithm to the degenerated basis consisting of the $i+\beta'-1+g$ vectors. Since we can assume that the first part of basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$ is LLL reduced, the number of swaps can be approximated by that of the LLL algorithm for the basis consists of $\beta'+g$ vectors $(\pi_i(\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_g}, \mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta'-1}))$.

Hence, from the standard analysis of LLL [36], the number of swaps is $\Theta((\beta'+g)^2) = \Theta(\beta^2)$, and each swap requires the procedure for updating the Gram-Schmidt coefficients that takes $\Theta(n^2)$ floating point operations when the basis vectors are given by $m = O(n)$ dimensional vectors [5].

Thus, the required number of floating point operations in Step 8-2 is $\Theta(\beta^2 n^2)$, and the total computational time in seconds in one BKZ tour is

$$Time_{GS} + Time_{LLL} = \sum_{i=1}^{n-1} \Theta(n^2 \cdot \beta^2) = A_1 \cdot \beta^2 n^3 [sec.] \tag{22}$$

with a rational constant A_1.

In this standard implementation using `quad_float` type, i.e., `LLL_QP` subroutine in NTL, about 400 dimension is the limit of stable computing in practice and the loss-of-precision error occurs in larger n. If we simply replace `quad_float` with another high precision type such as `RR` to avoid this error, the computing time must increase significantly. Several implementing techniques to reduce the cost of LLL have been proposed. One idea that we employ is to extract small local block by using small precision variables as in the next subsection.

[5] If we treat a larger dimension $m \gg n$, we need to consider an additional term $O(mn)$ of computational cost.

7.3 Estimating Time of Step 5-1 and 8-2: $Time_{GS} + Time_{LLL}$ when the lattice dimension is large

To compute the LLL algorithm correctly, we need to compute the GS-variables in a sufficient accuracy. A naive method using a high precision floating point variables takes a large cost for large dimensional lattices, e.g., `LLL_RR` function in the NTL library. To decrease such costs, we introduce a heuristic algorithm using two types of floating point numbers that is used after the vector insertion in Step 8. Although it is a folklore among the programmers, we precisely define it to analyze the computational cost.

We use the notations $(\mathtt{mu}_{i,j}^{\mathtt{type}}, \mathtt{c}_i^{\mathtt{type}})$ to denote the variables μ_{ij} and $||\mathbf{b}_i^*||$ using $\mathtt{type} \in \{\mathtt{qf}, \mathtt{RR}\}$ by which we can treat floating point numbers. Here, `qf` is the shortened form of `quad_float`.

In our implementation, we use $(\mathtt{mu}_{i,j}^{\mathtt{RR}}, \mathtt{c}_i^{\mathtt{RR}})$ to store the all GS-variables, and we also use $(\mathtt{mu}_{i,j}^{\mathtt{qf}}, \mathtt{c}_i^{\mathtt{qf}})$ as a "temporary cache" that stores the GS-variables of a projective sublattice to process local blocks as shown in Figure 10. Hence, a significant amount of basis updating (Step 8-2 in Figure 9) can be done efficiently within the cache variables with small precisions, and the update of $(\mathtt{mu}_{i,j}^{\mathtt{RR}}, \mathtt{c}_i^{\mathtt{RR}})$ does not occur frequently.

We can assume the GS-variables $(\mathtt{mu}_{i,j}^{\mathtt{RR}}, \mathtt{c}_i^{\mathtt{RR}})$ of the basis are computed at the start point of Step 5-1, since they are computed in the LLL algorithm in Step 1 in Figure 7, or computed in the end of the previous loop.

Implementation: In our experiments, we use `RR` as the `RR` type with the 320 bit precision of the NTL library. The temporary cashe is stored as a $H \times H$ matrix where we set $H = 250$.

Our implementation is described as follows. In the first step of our BKZ algorithm (Step 1 in Figure 9), we compute the GS-variables of the LLL-reduced basis and store them to $(\mathtt{mu}_{i,j}^{\mathtt{RR}}, \mathtt{c}_i^{\mathtt{RR}})$. From these high precision variables, we compute the cached values $(\mathtt{mu}_{i,j}^{\mathtt{qf}}, \mathtt{c}_i^{\mathtt{qf}})$ of size H starting from index I by constructing a new basis defined by the following matrix

$$B_{cache} = \begin{bmatrix} ||\mathbf{b}_I^*|| & & & \\ ||\mathbf{b}_{I+1}^*|| \cdot \mu_{I+1,I} & ||\mathbf{b}_{I+1}^*|| & & \\ \vdots & & \ddots & \\ ||\mathbf{b}_{I+H-1}^*|| \cdot \mu_{I+H-1,I} & \cdots & \cdots & ||\mathbf{b}_{I+H-1}^*|| \end{bmatrix}.$$

Here, the corresponding GS-variables are merely the copies of $(\mathtt{mu}_{i,j}^{\mathtt{RR}}, \mathtt{c}_i^{\mathtt{RR}})$. This temporary cache contains the GS-variables of the index from I to $I + H - 1$. We also keep the information on how the basis matrix is changed by processing the local block (Step 6-2 to 8-2 in Figure 10). The information is a unimodular matrix U that generates the basis B_{cache2} after processing and the basis B_{cache} before processing: $B_{cache2} = U \cdot B_{cache}$. (We can directly obtain U at Step 8-2, before shifting to the next cache.)

In Step 5-1, we check whether the indexes $[i : i + \beta - 1]$ of local block are a subset of the cache indexes $[I : I + H - 1]$. Here we recall that the notation

$[I : J]$ is the set of integers between I and J. If these are not contained, we update the entire basis by multiplying the unimodular matrix U to the part of the basis $(\mathbf{b}_I, \ldots, \mathbf{b}_{I+H-1})$, and then apply the LLL algorithm to $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$. After applying the LLL algorithm, the updated $(\mathtt{mu}_{i,j}^{\mathtt{RR}}, \mathtt{c}_i^{\mathtt{RR}})$ is computed. To process the local block, we once again compute the cache $(\mathtt{mu}_{i,j}^{\mathtt{qf}}, \mathtt{c}_i^{\mathtt{qf}})$ with the starting index $I = i$ and size $H = \min(250, n - I + 1)$. By this process, we can assume $[i : i + \beta - 1] \subset [I : I + H - 1]$ at Step 5-2 and the remaining part can be computed by using the cached variables.

Computational time: We estimate the number of floating point operations for `quad_float` and `RR` variables. First, to estimate the cost of computation in `RR` variables, we consider when the entire basis is updated. Since the index starts at $i = 1$ in Figure 10, the index range of the cache at this time is $[1 : H]$. By the fact that the blocksize is β, the updating process at Step 5-1 is occurred at the index $i = 2+H-\beta$ and the new cache is from the indexes $[2+H-\beta : 2H-\beta+1]$. By the same argument, the j-th updating is in the index $j + (H - \beta)(j - 1)$, and the corresponding indexes are $[j + (H - \beta)(j - 1) : \min(jH - (j - 1)(\beta - 1), n)]$. Hence, the number T of the entire updating in one BKZ tour is the minimum integer T so that $TH - (T - 1)(\beta - 1) > n$ and we have $T \approx (n - \beta)/(H - \beta)$.

In Step 5-1, we need to compute the multiplication by the unimodular matrix U and the LLL reduction in the large precision. We assume that the computational cost of the LLL is approximated by the time for computing the GS-variables in the caches because the updated basis is nearly LLL reduced. It is $\Theta(n^2 \cdot H)$ because $\Theta(n^2)$ floating point operations are required to compute one $\mathbf{b}_i^*$. The number of multiplication operations by U is $O(n \cdot H^2)$. Thus, the cost at Step 5-1 is $\Theta(n^2 \cdot H) + O(n \cdot H^2) = \Theta(n^2 \cdot H)$, and in total $\Theta(T \cdot n^2 \cdot H)$ throughout one BKZ tour.

On the other hand, the cost to update the local temporary cache in `quad_float` variables is $\Theta(\beta^2 H^2)$. We neglect this in our model for the large dimensions because the operations in `RR` are significantly heavier than that in `quad_float` from the preliminary experiments.

Therefore the total cost throughout one BKZ tour is as follows:

$$Time_{GS} + Time_{LLL} = \Theta(T \cdot n^2 \cdot H) = A_2 \cdot \frac{n - \beta}{H - \beta} \cdot n^2 H \text{ [sec.]}. \tag{23}$$

The rational constant A_2 will be fixed by the computer experiments.

7.4 Experimental Coefficient Fitting

Substituting equations (21), (22) and (23) to equation (20), we obtain our formulas for estimating the computational time of the progressive BKZ algorithm. For small dimensions (< 400) using only `quad_float` type of computing the GS-variables, the estimated computational time for finding a BKZ-β^{goal} reduced

basis is as follows:

$$Time_{Sim\text{-}small}(dim, \beta, A_1, W_1) = \sum_{\beta^{start}}^{\beta^{goal}} \sum_{t=1}^{\sharp tours} \Big[A_1 \cdot \beta^2 n^3 + W_1 \cdot \beta \sum_{i=1}^{n-1} \mathrm{ENUMCost}(B_i; \alpha, p) \Big] [\text{sec.}]. \tag{24}$$

The computational time for the large dimensions is as follows:

$$Time_{Sim\text{-}large}(dim, \beta, A_2, W_2) = \sum_{\beta^{start}}^{\beta_{goal}} \sum_{t=1}^{\sharp tours} \Big[A_2 \cdot \frac{n-\beta}{H-\beta} \cdot Hn^2 + W_2 \cdot \beta \sum_{i=1}^{n-1} \mathrm{ENUMCost}(B_i; \alpha, p) \Big] [\text{sec.}]. \tag{25}$$

In this section, we conduct the computer experiments with the simple blocksize strategy:

$$2 \overset{20}{\to} 20 \overset{21}{\to} 21 \overset{22}{\to} 22 \overset{23}{\to} 23 \overset{24}{\to} 24 \overset{25}{\to} \cdots$$

and then we estimate the undefined variables W_1, W_2, A_1 and A_2 by the experimental computing time after BKZ-55, i.e., $\beta^{start} = 55$.

Generating method of test lattice bases: The input bases are the Goldstein-Mayer type random lattices generated by the SVP Challenge problem generator. The instance of SVP Challenge problem has three parameters: lattice dimension n, random seed s, and the bit-length parameter b. The determinant of the generated lattice is equal to a prime $p \approx 2^{bn}$, where the default value of b is 10 to generate the published challenge problems. However, if we use $b = 10$, then $||\mathbf{b}_i^*|| = 1$ holds for some last indexes i after the LLL and BKZ with a small blolcksize. When the basis has basis vectors of $||\mathbf{b}_i^*|| = 1$ in last indexes, the FEC of basis does not indicate the reduction level correctly and fails to increase the blocksize in sharp timings. Indeed, this basis yields no sharp estimation from our preliminary experiments.

To prevent this, we take $b = \max(10, \lceil n/20 \rceil)$ for basis generation so that $||\mathbf{b}_n^*|| > 1$ holds after the LLL reduction.

Finding the coefficients: The experiments are conducted using a server with two Intel Xeon CPU E5-2697@2.70GHz processors. The small squares in Figure 12 indicate the single-core seconds of finishing the BKZ-β algorithm. We let the CPU times of lattice reductions for computing a BKZ-β reduced basis of n dimensional basis be $Time_{Exp}(n, \beta)$. In other words, we exclude the time of memory allocation, generating pruning coefficients, computing Sim-GS-lengths(n, β) and corresponding FEC. We also exclude the computing time of LLL reduced basis of an input lattice.

We find the suitable coefficients (A_1, W_1) by using the standard curve fitting method in semi-log scale, which minimize

$$\sum_{dim \in \{200,300\}} \sum_{\beta=55} \Big| \log\Big(T(dim, \beta, A_1, W_1)\Big) - \log\Big(Time_{Exp}(dim, \beta)\Big) \Big|^2,$$

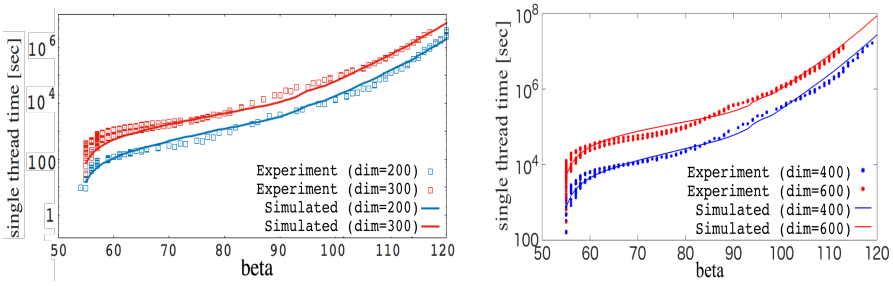

Fig. 12. Result of our parameter fitting for cost estimation. Left Figure: implementation described in Section 7.2. Right Figure: implementation described in Section 7.3. In both graphs, experimental results are plotted by small squares and the simulating results are drawn in bold lines.

where $T(dim, \beta, A_1, W_1) = Time_{Sim\text{-}large}(dim, \beta, A_1, W_1)$ in the small dimensional situation. For the large dimensional situation, we use the result of $dim \in \{600, 800\}$ to fix A_2 and W_2.

We find suitable coefficients

$$\begin{array}{ll} A_1 = 1.5 \cdot 10^{-10} & \text{and } W_1 = 1.5 \cdot 10^{-8} \\ A_2 = 10^{-6} & \text{and } W_2 = 3.0 \cdot 10^{-8}. \end{array} \tag{26}$$

The fitting results are given in Figure 12. Using the equations (24) and (25) with the above coefficients (26), we can estimate the computing times of our progressive BKZ algorithm.

8 Pre/Post-Processing the Entire Basis

In this section, we consider an extended strategy that enhances the speed of our progressive BKZ by pre/post-precessing the entire basis.

In pre-processing we first generate a number of randomized bases for input basis. Each basis is then reduced by using the proposed progressive BKZ algorithm. Finally we perform the enumeration algorithm for each reduced basis with some low probability in the post-processing. This strategy is essentially the same as the extreme pruning technique [19]. However, it is important to note that we do not generate a randomized basis inside the progressive BKZ. Our simulator for the proposed progressive BKZ is so precise that we can also estimate the speedup by the pre/post-precessing using our simulator.

8.1 Algorithm for Finding Nearly Shortest Vectors

In the following, we construct an algorithm for finding a vector shorter than $\gamma \cdot \mathrm{GH}(L)$ with a reasonable probability using the strategy above, and we analyze the total computing time using our simulator for the BKZ algorithm.

Table 4. The cost of solving SVP Challenge using our optimal blocksize strategy

dim.	M	$\log_2$(Time [sec.])	optimal blocksize strategy in Section 6
100	5	10.8	$LLL \overset{23}{\to} 20 \overset{34}{\to} 32 \overset{40}{\to} 37 \overset{43}{\to} 41 \overset{48}{\to} 47 \overset{50}{\to} 49 \overset{59}{\to} 58 \overset{70}{\to} 69 \overset{80}{\to} 79$
120	9	20.3	$LLL \overset{23}{\to} 20 \overset{34}{\to} 29 \overset{40}{\to} 37 \overset{48}{\to} 47 \overset{49}{\to} 48 \overset{54}{\to} 52 \overset{62}{\to} 60 \overset{71}{\to} 70 \overset{80}{\to} 80 \overset{88}{\to} 88 \overset{95}{\to} 95 \overset{99}{\to} 99 \overset{103}{\to} 103 \overset{108}{\to} 107$
140	81	30.3	$LLL \overset{23}{\to} 20 \overset{34}{\to} 30 \overset{40}{\to} 36 \overset{48}{\to} 47 \overset{52}{\to} 50 \overset{57}{\to} 54 \overset{64}{\to} 61 \overset{70}{\to} 68 \overset{78}{\to} 77 \overset{86}{\to} 86 \overset{93}{\to} 93 \overset{97}{\to} 97 \overset{100}{\to} 100 \overset{103}{\to} 103 \overset{106}{\to} 106 \overset{110}{\to} 110 \overset{114}{\to} 114 \overset{118}{\to} 117 \overset{123}{\to} 122$
160	327	41.2	$LLL \overset{23}{\to} 20 \overset{34}{\to} 29 \overset{40}{\to} 36 \overset{48}{\to} 47 \overset{52}{\to} 49 \overset{59}{\to} 57 \overset{64}{\to} 61 \overset{69}{\to} 66 \overset{73}{\to} 71 \overset{80}{\to} 78 \overset{86}{\to} 85 \overset{93}{\to} 93 \overset{95}{\to} 95 \overset{99}{\to} 99 \overset{101}{\to} 101 \overset{104}{\to} 104 \overset{107}{\to} 107 \overset{110}{\to} 110 \overset{113}{\to} 113 \overset{116}{\to} 116 \overset{119}{\to} 119 \overset{122}{\to} 122 \overset{126}{\to} 126 \overset{130}{\to} 129 \overset{133}{\to} 132 \overset{139}{\to} 137$

Concretely, for given lattice basis B of dimension n, the pre-processing part generates M randomized bases $B_i = U_i B$ by multiplying unimodular matrices U_i for $i = 1, \ldots, M$. Next, we apply our progressive BKZ for finding the BKZ-β reduced basis. The cost to obtain the randomized reduced bases is estimated by $M \cdot (\text{TimeRandomize}(n) + \text{TimeBKZ}(n, \beta))$. Here, TimeRandomize includes the cost of generating a random unimodular matrix and matrix multiplication, which is negligibly smaller than TimeBKZ in general. Thus we assume the computational cost for lattice reduction is $M \cdot \text{TimeBKZ}(n, \beta)$.

Finally, in the post-processing part, we execute the standard enumeration algorithm with the searching radius parameter $\alpha = \gamma$ and probability parameter $p = 2 \cdot \gamma^{-n}/M$. As with the similar argument in Section 4.1, there exist about $\gamma^n/2$ short vector pairs in $\text{Ball}_n(\gamma \cdot \text{GH}(L))$. Therefore, the probability that one enumeration finds the desired vector is about $(\gamma^n/2) \cdot (2 \cdot \gamma^{-n}/M) = 1/M$ and the total probability of success is $1 - (1 - 1/M)^M \approx 0.632$.

Consequently, the total computing cost in our model is

$$M \cdot \left(\text{TimeBKZ}(n, \beta) + \frac{\text{ENUMCost}(B; \gamma, p = 2 \cdot \gamma^{-n}/M)}{6 \cdot 10^7} \right) \text{ [sec.]}, \qquad (27)$$

where $\text{TimeBKZ}(n, \beta)$ and $\text{ENUMCost}(B; \gamma, p)$ are defined by Section 6.1 and Section 2.3, respectively. We can optimize this total cost by finding the minimum of formula (27) over parameter (β, M). Here, note that the constant $6 \cdot 10^7$ comes from our best benchmarking record of lattice enumeration. In Table 4, we provide the detailed simulating result with setting $\gamma = 1.05$ to analyze the hardness of the Darmstadt SVP Challenge in several dimensions. A comparison with previous works are given in Section 9 (See the line **C** in Figure 13).

8.2 Lower Bound of the Cost by an Idealized Algorithm

Here we discuss the lower bound of the total computing cost of the proposed progressive BKZ algorithm (or other reduction algorithm) with the pre/post-processing.

The total cost is estimated by the sum of the computational time for the randomization, the progressive BKZ algorithm, and the enumeration algorithm

by the following extremely idealized situations. Note that we believe that they are beyond the most powerful cryptanalysis which we can achieve in the future, and thus we say that this is the lower bound in our model.

(a)The cost for the randomization becomes negligibly small. The algorithm for randomizing the basis would not only be the method of multiplying random unimodular bases, and we could find an ideal randomization at a negligibly small cost. Thus, TimeRandomize$(n) = 0$.

(b)The cost for the progressive BKZ algorithm does not become lower than that of computing the Gram-Schmidt lengths. Even though the progressive BKZ algorithm ideally improved, we always need the Gram-Schmidt basis computation used for the enumeration algorithm or the LLL algorithm. The computation of the Gram-Schmidt basis (even though the computation is performed in an approximation using floating point operations with a sufficient precision) includes $\Theta(n^3)$ floating point arithmetic operations via the Cholesky factorization algorithm (See, for example [37, Chapter 5]). A modern CPU can perform a floating point operation in one clock cycle, and it can work at about 4.0GHz. Thus, we assume that the lower bound of the time in seconds is $(4.0 \cdot 10^9)^{-1} \cdot n^3$.

(c)The reduced basis obtained by the progressive BKZ (or other reduction algorithm) becomes ideally reduced. We define *the simulated γ-approximate HKZ basis* $B_{\gamma\text{-}HKZ}$ by a basis satisfying

$$||\mathbf{b}_i^*|| = \tau_{n-i+1}\mathrm{GH}(L_{[i:n]}) \text{ for } i = 2, \ldots, n \text{ and } ||\mathbf{b}_1|| = \gamma \cdot \mathrm{GH}(L).$$

For any fixed γ and p, we assume this basis minimizes the cost for enumeration over any basis satisfying $||\mathbf{b}_1|| \geq \gamma \cdot \mathrm{GH}(L)$.

Therefore, the lower bound of the total cost of the idealized algorithm in seconds is given by

$$\min_{M \in \mathbb{N}} M \cdot \left((4.0 \cdot 10^9)^{-1} \cdot n^3 + \frac{ENUMCost(B_{\gamma\text{-}HKZ}; \alpha, p/M)}{6 \cdot 10^7} \right). \tag{28}$$

Setting $\gamma = 1.05$, we analyze the lower bound cost to enter the SVP Challenge. (See the line **D** in Figure 13).

9 Simulation Results for SVP Challenge and Comparison

In this section, we give our simulation results using our proposed progressive BKZ algorithm together with the pre/post-processing strategy in Section 8.1 for solving the Darmstadt SVP Challenge [49], which tries to find a vector shorter than $1.05 \cdot \mathrm{GH}(L)$ in the random lattice L of dimension n.

We also simulate the cost estimation of Lindner and Peikert [31] and that of Chen and Nguyen [12] in the same model. The summery of our simulation results and the latest records published in the SVP Challenge are given in Figure 13. The outlines of our estimations **A** to **D** in Figure 13 are given below.

From our simulation, the proposed progressive BKZ algorithm is about 50 times faster than BKZ 2.0 and about 100 times slower than the idealized algorithm that achieves the lower bound in our model of Section 8.2.

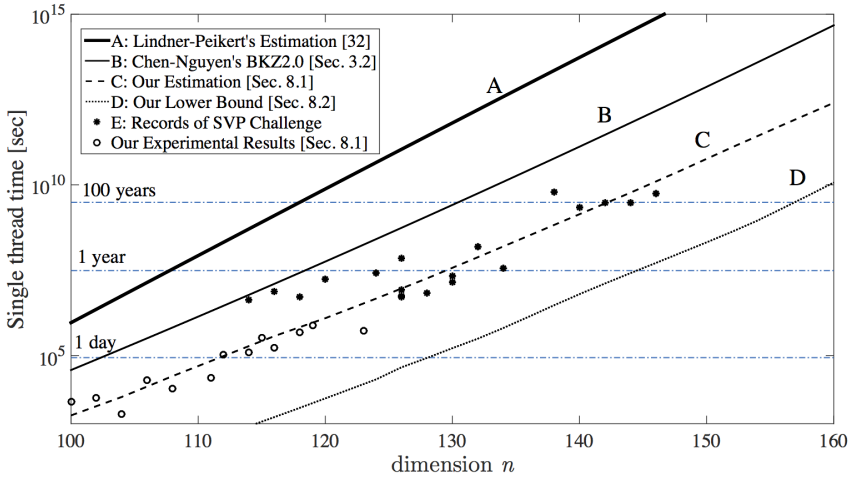

Fig. 13. Comparing cost in seconds. **A**: Lindner-Peikert estimation, **B**: Chen-Nguyen's BKZ 2.0 simulation, **C**: Simulating estimation of our randomized BKZ-then-ENUM algorithm, **D**: Lower bound in the randomized BKZ-then-ENUM strategy. Records in the SVP Challenge are indicated by the black circles "•", and our experimental results are indicated by the white circles "○".

A: Lindner-Peikert's estimation [31]: From the experiments using the BKZ implementation in the NTL library [48], they estimated that the BKZ algorithm can find a short vector of length $\delta^n \det(L)^{1/n}$ in $2^{1.8/\log_2(\delta)-110}$ [sec.] in the n-dimensional lattice. The computing time of Lindner-Peikert's model becomes

$$Time_{\mathrm{LP}} = 2^{1.8/\log_2(\delta)-110} \text{ with } \delta = 1.05^{1/n} \cdot V_n(1)^{-1/n^2},$$

because this δ attains $1.05 \cdot \mathrm{GH}(L) = \delta^n \det(L)^{1/n}$.

B: Chen-Nguyen's BKZ 2.0 [12,13]: We estimated the cost of BKZ 2.0 using the simulator in Section 3.2. Following the original paper [12], we assume that a blocksize is fixed and the estimation is the minimum of (4) over all possible pairs of the blocksize β and the number t of tours. Again we convert the number of nodes into the single-threaded time, we divide the number by $2 \cdot 10^7$.

C: Our estimation: We searched the minimum cost using the estimation (27) over M and β with setting $\gamma = 1.05$.

D: Lower bound in our model: We searched the minimum cost using the estimation (28) over M with setting $\gamma = 1.05$.

Records of SVP Challenge: From the hall of fame in the SVP Challenge [49] and reporting paper [17], we listed up the records that contain the computing time with a single thread in Figure 13, as black circles "•". Moreover we performed experiments on our proposed progressive BKZ algorithm using the

pre/post-processing strategy in Section 8.1 up to 123 dimensions which are also indicated by the white circles "○" in Figure 13.

10 Conclusions and Future Work

We proposed an improved progressive BKZ algorithm with optimized parameters and block-increasing strategy. We also gave a simulator that can precisely predict the Gram-Schmidt lengths computed using the proposed progressive BKZ. We also presented the efficient implementation of the enumeration algorithm and LLL algorithm, and the total cost of the proposed progressive BKZ algorithm was precisely evaluated by the sharp simulator.

Moreover, we showed a comparison with other algorithms by simulating the cost of solving the instances from the Darmstadt SVP Challenge. Our progressive BKZ algorithm is about 50 times faster than the BKZ 2.0 proposed by Chen and Nguyen for solving the SVP Challenges up to 160 dimensions. Finally, we discussed a computational lower bound of the proposed progressive BKZ algorithm under certain ideal assumptions. These simulation results contribute to the estimation of the secure parameter sizes used in lattice based cryptography.

We outline some future works: (1) constructing a BKZ simulator without using our ENUMCost, (2) adopting our simulator with other strategies such as BKZ-then-Sieve strategy for computing a short vector more efficiently, and (3) estimating the secure key length of lattice-based cryptosystems using the lower bound of the proposed progressive BKZ.

References

1. D. Aggarwal, D. Dadush, O. Regev, and N. Stephens-Davidowitz, "Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: extended abstract", STOC 2015, pp. 733–742, 2015.
2. M. Ajtai, "The shortest vector problem in L_2 is NP-hard for randomized reductions", STOC 1998, pp. 10–19, 1998.
3. M. Ajtai, "The worst-case behavior of Schnorr's algorithm approximating the shortest nonzero vector in a lattice", STOC 2003, pp. 396–406, 2003.
4. M. R. Albrecht, R. Fitzpatrick, and F. Göpfert, "On the efficacy of solving LWE by reduction to unique-SVP", ICISC 2013, LNCS, vol. 8565, pp. 293–310, 2013.
5. M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors", J. Mathematical Cryptology, vol. 9, no. 3, pp. 169–203, 2015.
6. Y. Aono, "A faster method for computing Gama-Nguyen-Regev's extreme pruning coefficients", arXiv: 1406.0342, 2014.
7. Y. Aono, X. Boyen, L. Phong, and L. Wang, "Key-private proxy re-encryption under LWE", INDOCRYPT 2013, LNCS, vol. 8250, pp. 1–18, 2014.
8. S. Bai and S. D. Galbraith, "Lattice decoding attacks on binary LWE", ACISP 2014, LNCS, vol. 8544, pp. 322–337, 2014.
9. Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages", CRYPTO 2011, LNCS, vol. 6841, pp. 505–524, 2011.

10. J. Buchmann and C. Ludwig, "Practical lattice basis sampling reduction", ANTS 2006, LNCS, vol. 4076, pp. 222–237, 2006.
11. Y. Chen, "Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe", doctoral dissertation, 2013.
12. Y. Chen and P. Q. Nguyen, "BKZ 2.0: Better lattice security estimates", ASIACRYPT 2011, LNCS, vol. 7073, pp. 1–20, 2011.
13. Y. Chen and P. Q. Nguyen, "BKZ 2.0: Better lattice security estimates", the full version, available at `http://www.di.ens.fr/~ychen/research/Full_BKZ.pdf`.
14. D. Coppersmith and A. Shamir, "Lattice attacks on NTRU", EUROCRYPT 1997, LNCS, vol. 1233, pp. 52–61,1997.
15. U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis", Math. Comp., vol. 44, pp. 463–471, 1985.
16. R. Fischlin and J. P. Seifert, "Tensor-based trapdoors for CVP and their application to public key cryptography (extended abstract)", IMA Cryptography and Coding '99, LNCS, vol. 1746, pp. 244–257, 1999.
17. M. Fukase and K. Kashiwabara, "An accelerated algorithm for solving SVP based on statistical analysis", J. Information Processing, vol. 23, no. 1, pp. 67–80, 2015.
18. N. Gama and P. Q. Nguyen, "Predicting lattice reduction", EUROCRYPT 2008, LNCS, vol. 4965, pp. 31–51, 2008.
19. N. Gama, P. Q. Nguyen, and O. Regev, "Lattice enumeration using extreme pruning", EUROCRYPT 2010, LNCS, vol. 6110, pp. 257–278, 2010.
20. S. Garg, C. Gentry, and S. Halevi, "Candidate multilinear maps from ideal lattices", EUROCRYPT 2013, LNCS, vol. 7881, pp. 1–17, 2013.
21. C. Gentry, "Fully homomorphic encryption using ideal lattices", STOC 2009, pp. 169–178, 2009.
22. G. Hanrot, X. Pujol, and D. Stehlé, "Analyzing blockwise lattice algorithms using dynamical systems", CRYPTO 2011, LNCS, vol. 6841, pp. 447–464, 2011.
23. G. Hanrot and D. Stehlé, "Improved analysis of Kannan's shortest lattice vector algorithm", CRYPTO 2007, LNCS, vol. 4622, pp. 170–186, 2007.
24. M. Haque, M. O. Rahman, and J. Pieprzyk, "Analysing progressive-BKZ lattice reduction algorithm", NCICIT 2013, pp. 73–80, 2013.
25. J. Hoffstein, J. Pipher, and J. H. Silverman, "An introduction to mathematical cryptography", Springer-Verlag New York, 2008.
26. T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi, "Parallel Gauss sieve algorithm: solving the SVP challenge over a 128-dimensional ideal lattice", PKC 2014, LNCS, vol. 8383, pp. 411–428, 2014.
27. R. Kannan, "Improved algorithms for integer programming and related lattice problems", STOC 1983, pp. 193–206, 1983.
28. A. Korkine and G. Zolotareff, "Sur les formes quadratiques", Math. Ann., vol. 6, no. 3, pp. 366–389, 1873.
29. A. K. Lenstra, H. W. Jr. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients", Math. Ann., vol. 261, no. 4, pp. 515–534, 1982.
30. T. Lepoint and M. Naehrig, "A comparison of the homomorphic encryption schemes FV and YASHE", AFRICACRYPT 2014, LNCS, vol. 8469, pp. 318–335, 2014.
31. R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption", CT-RSA 2011, LNCS, vol. 6558, pp. 319–339, 2011.
32. M. Liu and P. Q. Nguyen, "Solving BDD by enumeration: an update", CT-RSA 2013, LNCS, vol. 7779, pp. 293–309, 2013.

33. D. Micciancio, "The shortest vector problem is NP-hard to approximate to within some constant", FOCS 1998, pp. 92–98, 1998.
34. P. Q. Nguyen, "Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto '97", CRYPTO '99, LNCS, vol. 1666, pp. 288–304, 1999.
35. P. Q. Nguyen and O. Regev, "Learning a parallelepiped: cryptanalysis of GGH and NTRU signatures", EUROCRYPT 2006, LNCS, vol. 4004, pp. 271–288, 2006.
36. P. Q. Nguyen and D. Stehlé, "LLL on the average", ANTS 2006, LNCS, vol. 4076, pp. 238-256, 2006.
37. P. Q. Nguyen and B. Vallée, "The LLL algorithm: survey and applications", Springer-Verlag Berlin Heidelberg, 2009.
38. T. Plantard and M. Schneider, "Creating a challenge for ideal lattices", IACR Cryptology ePrint Archive 2013: 039, 2013.
39. T. Plantard, W. Susilo, and Z. Zhang, "Adaptive precision floating point LLL", ACISP 2013, LNCS, vol. 7959, pp. 104–117, 2013.
40. O. Regev, "On lattices, learning with errors, random linear codes, and cryptography", J. ACM, vol. 56, no. 6, article 34, 2009.
41. C. A. Rogers, "The number of lattice points in a set", Proc. London Math. Soc., vol. 3, no. 6, pp. 305–320, 1956.
42. C. P. Schnorr, "A hierarchy of polynomial time lattice basis reduction algorithms", Theor. Comput. Sci., vol. 53, no. 2–3, pp. 201–224, 1987.
43. C. P. Schnorr, "Lattice reduction by random sampling and birthday methods", STACS 2003, LNCS, vol. 2607, pp. 145–156, 2003.
44. C. P. Schnorr, "Accelerated and Improved Slide- and LLL-Reduction", ECCC TR11-050, 2011.
45. C. P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems", Math. Program., vol. 66, no. 1–3, pp. 181–199, 1994.
46. C. P. Schnorr and H. H. Hörner, "Attacking the Chor-Rivest cryptosystem by improved lattice reduction", EUROCRYPT 1995, LNCS, vol. 921, pp. 1–12, 1995.
47. C. P. Schnorr and T. Shevchenko, "Solving subset sum problems of density close to 1 by "randomized" BKZ-reduction", IACR Cryptology ePrint Archive 2012: 620, 2012.
48. V. Shoup, "NTL: a library for doing number theory", `http://www.shoup.net/ntl/`.
49. TU Darmstadt Lattice Challenge, `http://www.latticechallenge.org/`.
50. J. van de Pol and N. P. Smart, "Estimating key sizes for high dimensional lattice-based systems", IMA Cryptography and Coding 2013, LNCS, vol. 8308, pp. 290–303, 2013.