Halt Properties and Complexity Evaluations for Optimal DeepLLL Algorithm Families

Takuto Odagawa1, Koji Nuida2,3

1 Graduate School of Information Science and Technology, The University of Tokyo, Japan
tk55-1234@g.ecc.u-tokyo.ac.jp
2 Institute of Mathematics for Industry (IMI), Kyushu University, Japan
nuida@imi.kyushu-u.ac.jp
3 National Institute of Advanced Industrial Science and Technology (AIST), Japan

Abstract

DeepLLL algorithm (Schnorr, 1994) is a famous variant of LLL lattice basis reduction algorithm, and PotLLL algorithm (Fontein et al., 2014) and S2LLL algorithm (Yasuda and Yamaguchi, 2019) are recent polynomial-time variants of DeepLLL algorithm developed from cryptographic applications. However, the known polynomial bounds for computational complexity are shown only for parameter $\delta < 1$; for “optimal” parameter $\delta = 1$ which ensures the best output quality, no polynomial bounds are known, and except for LLL algorithm, it is even not formally proved that the algorithm always halts within finitely many steps. In this paper, we prove that these four algorithms always halt also with optimal parameter $\delta = 1$, and furthermore give explicit upper bounds for the numbers of loops executed during the algorithms. Unlike the known bound (Akhavi, 2003) applicable to LLL algorithm only, our upper bounds are deduced in a unified way for all of the four algorithms.

Keywords: LLL algorithm, DeepLLL algorithm, computational complexity

1 Introduction

Lattice basis reduction is one of the most important kinds of algorithms from both theoretical and practical viewpoints. Roughly speaking, given a basis of a lattice in finite-dimensional Euclidean space, a lattice basis reduction (or lattice reduction) algorithm aims at outputting a basis of the same lattice consisting of short and nearly orthogonal basis vectors. Major applications of lattice reduction include security analysis of cryptosystems. It has been shown [12] that most of the currently deployed public key cryptosystems, such as RSA cryptosystem [10] and elliptic curve cryptosystems [6, 8], can be broken by quantum algorithms in polynomial time. Accordingly, “post-quantum” cryptosystems that are still secure even after the development of large-scale quantum computers have been an intensively studied topic in the area of cryptography. Among them, lattice-based cryptography [1] is one of the main design principles, which is based on the computational hardness of the shortest vector problem (SVP) and/or the closest vector problem (CVP) on high-dimensional lattices. Lattice reduction is a fundamental building block of the known algorithms for SVP and CVP, therefore it is important from the viewpoint of cryptography to evaluate the computational complexity of lattice reduction.

LLL algorithm [7] is the most famous lattice reduction algorithm, which is known to output, in polynomial time with respect to the dimension of the lattice, a lattice basis having a certain good property (see [9] for

*This research was mainly done when the first author was an undergraduate student at Department of Mathematical Engineering and Information Physics, School of Engineering, The University of Tokyo and when the second author was with Graduate School of Information Science and Technology, The University of Tokyo.
the details). On the other hand, its variant named DeepLLL algorithm has been proposed [11] and is known to output a lattice basis with better property than the case of LLL algorithm. However, it is not proved that DeepLLL algorithm in general halts within polynomial time. To resolve this drawback, some variants of DeepLLL algorithm with (provably) polynomial computational complexity are actively studied mainly in the area of cryptography, such as PotLLL algorithm [4] and $S^2$LLL algorithm [13].

In LLL algorithm and its variants mentioned above, the guaranteed quality of the output basis is controlled by a parameter commonly denoted by $\delta$ (it is denoted by $\eta$ in [13], but here we unify the notation into the $\delta$) in a way that the output basis becomes better when $\delta$ becomes larger. The range of parameter is $1/4 < \delta \leq 1$ for LLL, DeepLLL, and PotLLL algorithms, while $0 < \delta \leq 1$ for $S^2$LLL algorithm. Therefore, purely from the viewpoint of guaranteed output quality, the parameter choice $\delta = 1$ is optimal. However, as opposed to the case $\delta < 1$ where polynomial (in the dimension of the lattice) upper bounds for the computational complexity have been given for LLL, PotLLL, and $S^2$LLL algorithms, no such polynomial bounds are known for the optimal case $\delta = 1$. In fact, for DeepLLL, PotLLL, and $S^2$LLL algorithms with $\delta = 1$, even it has not been proved (to the authors’ best knowledge) that these algorithms always halt within finitely many steps. (See below for more details.) In order to estimate the best possible performances of those algorithms (for the sake of e.g., closely analyzing the security of lattice-based cryptosystems), it is worthy to evaluate the complexity of those algorithms with optimal parameter $\delta = 1$ in detail.

1.1 Our Contributions

In this paper, we prove that all of LLL algorithm and its variants mentioned above (i.e., DeepLLL, PotLLL, and $S^2$LLL algorithms) with optimal parameter $\delta = 1$ always halt within finitely many steps, and furthermore give explicit upper bounds for the numbers of loops executed during the algorithms.

Before explaining our main result, we introduce some notations (see Section 2.1 for the definitions). Let $n \geq 2$ be the dimension of an input lattice $L$. We suppose that $L$ is full-rank for the sake of simplicity. Let $M$ denote the maximum ($L^2$, a.k.a. Euclidean) norm of vectors in an input basis $B$ of the lattice $L$, and let $\text{vol}(L)$ denote the volume of $L$. We put $\alpha = M^n / \text{vol}(L)$ (note that $\alpha \geq 1$). Then our main result is stated as follows (which is the same as Theorem 2 below).

**Theorem 1.** In each of LLL algorithm and its variants mentioned above with parameter $\delta = 1$, the total number of main loops executed during the algorithm is at most

$$(n - 1) \cdot \left(3 \prod_{j=2}^{n-1} (2 + \sqrt{j + 3})\right)^n \alpha^{n-1},$$

which is also upper bounded by

$$(n - 1) \cdot \left(\frac{2}{\sqrt{5}} + 1\right)^{n(n-2)} \alpha^{n-1} \left(3 \cdot (n + 2)! \right)^{n/2} / 8 \leq \alpha^{n-1} \left(\left(\frac{2}{\sqrt{5}} + 1\right) e^{-1/2} n^{1/2 + o(1)}\right)^n (\text{when } n \to \infty).$$

In particular, these algorithms always halt within finitely many steps.

We give a proof of this theorem in Section 3.2. As this bound is hyperexponential in $n$, it is intuitively estimated that the computational complexity (not just the number of loops) also has the same asymptotic bound, assuming that each loop can be executed in polynomial time with respect to $n$.

Intuitively speaking, our proof focuses on the index $k$ for the main loop of each of the four algorithms, which is either increased or decreased at each of the loop. The algorithm starts from $k = 2$, and it halts when $k$ becomes larger than $n$. The given lattice basis $B$ is not changed when $k$ is increased, while we can show that some earlier part of $B$ is shrunk when $k$ is decreased. Now the theorem is proved by a recursive argument based on the fact that the number of lattice points within a given threshold of norms is explicitly
upper bounded (which we show in Section 3.1). One of the main advantages of our result is that essentially the same proof is applied to all of the four algorithms, in contrast to the known upper bound in [2] which is only applicable to LLL algorithm (see Section 1.2 for details). Another advantage of our result is that we obtain an explicit bound for the number of loops in the algorithms, not only an asymptotic bound in the limit case $n \to \infty$. On the other hand, our theorem gives the first upper bounds for the case of DeepLLL, PotLLL, and $S^2$LLL algorithms with $\delta = 1$, but our bound does not improve the known bound in [2] for LLL algorithm with $\delta = 1$.

Although all of the known theoretical upper bounds for the case $\delta = 1$ including our result are hyperexponential in $n$ which is essentially different from the polynomial bounds for the case $\delta < 1$, it is expected that the practical complexity for the case $\delta = 1$ would be not drastically larger than the case $\delta < 1$; we also give some experimental observation in Section 4. Hence, it is an important future research topic to improve the known theoretical upper bounds for the case $\delta = 1$.

1.2 Related Work

Considering the optimal parameter $\delta = 1$, Akhavi [2] gave an asymptotic upper bound $O(A^3 \log M)$ for the computational complexity of LLL algorithm, where $A$ is an arbitrary constant with $A > (4/3)^{1/12}$. This is the state-of-the-art result, to the authors’ best knowledge. Our bound in this paper is basically worse than Akhavi’s bound unless $M$ (hence $\alpha$) is significantly small. However, Akhavi’s approach is specific to the case of LLL algorithm. In more detail, roughly speaking, Akhavi’s approach is focusing on the timing of exchanging the first vector $b_1$ in the basis $B$ and analyzing the behavior of the volume of the sublattice spanned by the first few basis vectors, and is crucially based on the typical property of LLL algorithm that the exchange of basis vectors is performed only for two consecutive vectors. As this property is not satisfied by DeepLLL algorithm and its variants, it seems difficult to extend Akhavi’s approach to the case of the latter algorithms. In contrast, our approach proposed in this paper is applicable in a unified way to all of the four algorithms considered in this paper. It is expected that the high flexibility of our approach would enable us to deduce a similar upper bound when some further variant of (Deep)LLL algorithm will be developed.

On the other hand, it is mentioned in Footnote 6 of [2] that LLL algorithm (with $\delta = 1$) for input basis consisting of integer vectors has computational complexity $O(M^\alpha)$, which is again already better than our upper bound unless $\alpha$ is significantly small. This bound is based on the fact that the quantity $\text{Pot}(B)$ for the basis $B$ (see Eq.2 below for the definition) is monotonically decreasing during LLL algorithm and $\text{Pot}(B)$ is a positive integer for any basis $B$ consisting of integer vectors. We note that the same upper bound for integer basis vectors is also available for PotLLL algorithm, as $\text{Pot}(B)$ is again monotonically decreasing during the algorithm. However, $\text{Pot}(B)$ is not monotonically decreasing in DeepLLL and $S^2$LLL algorithms (we give examples of this fact in Section 3.3), therefore the same approach does not work for those algorithms. In contrast, our approach in this paper is generally applicable without assuming that the input basis consists of integer vectors nor assuming that $\text{Pot}(B)$ is monotonically decreasing (or there is some integer quantity that is monotonically decreasing during the algorithm).

2 Preliminaries

2.1 Notations and Basic Definitions

In this paper, $\|b\|$ denotes the (Euclidean) norm of a real vector $b$. The lattice generated by linearly independent vectors $b_1, \cdots, b_n \in \mathbb{R}^m$ is defined by

$$L(b_1, \cdots, b_n) = \left\{ \sum_{i=1}^{n} v_i b_i \mid v_i \in \mathbb{Z} \; (i = 1, \cdots, n) \right\}.$$  

The matrix $B = [b_1, \cdots, b_n] \in \mathbb{R}^{m \times n}$ indicates a basis of the lattice $L = L(b_1, \cdots, b_n)$. The tuple of vectors obtained by Gram–Schmidt orthogonalization (GSO) for $B$ is denoted by $B^* = [b_1^*, \cdots, b_n^*]$. Namely, we
have
\[b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^* \text{ for } i = 1, \ldots, n, \text{ where } \mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2} \text{ for } 1 \leq j < i \leq n\]

(here, as usual, the summation \(\sum_{i=p}^{q}\) becomes zero when \(p > q\); and \(\langle \cdot, \cdot \rangle\) denotes the standard inner product for vectors). Set \(B_i = \|b_i^*\|^2\) for \(i = 1, \ldots, n\).

The volume of the lattice \(L\) with basis \(B\) as above is given by
\[
\text{vol}(L) = \sqrt{\text{det}(B \cdot B)} = \prod_{i=1}^{n} \|b_i^*\|, \tag{1}
\]

where \(B^\top\) denotes the transpose of a matrix \(B\). Let \(\pi_j\) be the orthogonal projection from \(\mathbb{R}^m\) to the orthogonal complement of the subspace \(\text{span}\{b_1, \ldots, b_{j-1}\}\); in particular, we have \(\pi_1 = \text{id}\). Note that the vector \(b_i^*\) obtained by GSO as above is equal to \(\pi_i(b_i)\). We define a permutation \(\sigma_{i,k}\) for the basis vectors by
\[
\sigma_{i,k}(B) = [b_1, \ldots, b_{i-1}, b_k, b_i, \ldots, b_{k-1}, b_{k+1}, \ldots, b_n].
\]

In the rest of this paper, to simplify the arguments, we focus only on full-rank lattices, i.e., the case of \(n = m\). Our results might be easily extendible to the general case where \(n \leq m\). Our argument below is based on the well-known discreteness of lattices; for \(r \geq 0\), the number of points \(b \in L\) with \(\|b\| \leq r\) is finite (its refinement will be given in Lemma 1 later).

### 2.2 The LLL Algorithm and Its Variants

We recall some properties of LLL algorithm \([7]\) and its variants such as DeepLLL algorithm \([11]\). LLL algorithm takes a basis \(B\) of a lattice (which is supposed to be full-rank in this paper) as input and has an auxiliary parameter \(\delta \in (1/4, 1]\) controlling a trade-off between the computational complexity and the quality of the output. The algorithm outputs a new basis of the same lattice with a certain property called \(\delta\)-LLL-reduced. Here we omit the definition of \(\delta\)-LLL-reduced property as it is not relevant to our argument in this paper (see e.g., \([13]\) for the details), and only mention that if \(\delta\) increases, then the \(\delta\)-LLL-reduced property becomes stronger while the upper bound for computational complexity of the algorithm is also getting larger. In this paper, we write \(\delta\)-LLL to mean LLL algorithm with parameter \(\delta\). The construction of LLL algorithm will be described in Section 2.2 below.

Among the known variants of LLL algorithm, here we deal with DeepLLL algorithm \([11]\), PotLLL algorithm \([4]\), and \(S^2\)LLL algorithm \([13]\) (see Section 3.2 for their constructions). Similarly to the case of LLL, those algorithms also have their own trade-off parameter \(\delta\); the range is \(1/4 < \delta \leq 1\) for DeepLLL and PotLLL, and \(0 < \delta \leq 1\) for \(S^2\)LLL (note that the parameter is denoted by \(\eta\) in the original paper \([13]\)). We use the words \(\delta\)-DeepLLL, \(\delta\)-PotLLL, and \(\delta\)-\(S^2\)LLL in a way similar to \(\delta\)-LLL. Definitions of the properties satisfied by the outputs of those algorithms are also omitted herein.

Roughly speaking, DeepLLL is modified from LLL in a way that DeepLLL may move a basis vector to a position not adjacent to the original while LLL only exchanges two adjacent basis vectors. In comparison to DeepLLL, the strategy of PotLLL is that permutations for the basis \(B\) are iterated in the direction of decreasing the following quantity:
\[
\text{Pot}(B) = \prod_{i=1}^{n} \text{vol}(L(b_1, \ldots, b_i))^2 = \prod_{i=1}^{n} B_i^{n-i+1}. \tag{2}
\]

We note that when \(B\) consists of integer vectors, this value is always a non-negative integer, as each \(\text{vol}(L(b_1, \ldots, b_i))^2\) is an integer due to Eq. (1). We also note the following relation for \(i < k\) (see Lemma 1 of \([4]\)):
\[
\text{Pot}(\sigma_{i,k}(B)) = \text{Pot}(B) \prod_{j=i}^{k-1} \frac{\|\pi_j(b_k)\|^2}{B_j} \tag{3}
\]
On the other hand, the iteration in $S^2$LLL is performed in a way that the following quantity, instead of $\text{Pot}(B)$, is monotonically decreased:

$$SS(B) = \sum_{i=1}^{n} B_i.$$  

We note the following relation for $i < k$ (see Eq.(5) of [13]):

$$S_{ik} := SS(B) - SS(\sigma_{i,k}(B)) = k - 1 \sum_{j=i}^{k-1} \mu_{ij}^2 B_j \left( \| \pi_j(b_k) \|_2^2 - 1 \right).$$  

\section{Our Complexity Evaluation}

In this section, we prove that LLL algorithm and its aforementioned variants with the largest possible parameter $\delta = 1$ always halt within finitely many steps, and also provide upper bounds for the number of loops executed before the algorithm halts. Although the halt property for the case of LLL algorithm has been known, our proof strategy is significantly different from the known proof in [2] and is applicable also to the variants of LLL algorithm (in contrast to the strategy of [2] specific to the case of LLL algorithm).

In the constructions of those algorithms described below, “Size-reduce” for basis $B = [b_1, \cdots, b_n]$ means the procedure to update $b_i \leftarrow b_i - \lceil \mu_{ij} \rceil b_j$ for $i = 1, \cdots, n$ and $j = 1, \cdots, i - 1$ (in this order), where $\lceil \cdot \rceil$ denotes rounding to the nearest integer; see Algorithm 24 of [5] for the details. We note that after the update, the value of $\mu_{ij}$ becomes $\mu_{ij} - \lceil \mu_{ij} \rceil$, which thus has absolute value at most $1/2$. (Such a basis is said to be size-reduced.) “Update GSO” means the procedure to also update the tuple $B^*$ associated to the updated basis $B$. Moreover, in the proofs below, $\text{SWAP}_{\rho,\kappa}$ means the step of $B \leftarrow \sigma_{\rho,\kappa}(B)$ in the algorithm.

\subsection{Number of Lattice Points with Bounded Norm}

Our proof below is based on the fact that the number of lattice points with bounded norm is finite. Here we give the following refinement of this fact.

\begin{lemma}
Let $L$, $B = [b_1, \cdots, b_n]$, and $B^* = [b_1^*, \cdots, b_n^*]$ be as in Section 2.4. For $r \geq 0$, the number of points $x \in L$ with $\|x\| \leq r$ is at most

$$\prod_{i=1}^{n} \left( \frac{2r}{\|b_i^*\|} + 1 \right) \leq \frac{(M + 2r)^n}{\text{vol}(L)}$$

where $M = \max\{\|b_1\|, \cdots, \|b_n\|\}$.

\begin{proof}
Let $x = \sum_{i=1}^{n} v_i b_i$ ($v \in \mathbb{Z}^n$) be a lattice point satisfying $\|x\| \leq r$. Since

$$\|x\|^2 = \left\| \sum_{i=1}^{n} v_i b_i^* + \sum_{j=1}^{i-1} \mu_{ij} b_j^* \right\|^2 = \left\| \sum_{j=1}^{n} \left( \sum_{i=j+1}^{n} v_i \mu_{ij} + v_j \right) b_j^* \right\|^2 = \sum_{j=1}^{n} \left( \sum_{i=j+1}^{n} v_i \mu_{ij} + v_j \right)^2 \|b_j^*\|^2,$$

\end{proof}

\end{lemma}
we have

\[ \sum_{i=j+1}^{n} v_i \mu_{ij} + v_j \leq \frac{r}{\|b_j^*\|} \text{ for } j = 1, \ldots, n. \]

We give an upper bound for the number of \( v \in \mathbb{Z}^n \) satisfying the inequality above. First, as \( |v_n| \leq \frac{r}{\|b_n^*\|} \), \( v_n \) has at most \( 2r/\|b_n^*\| + 1 \) possibilities. On the other hand, when \( v_n, v_{n-1}, \ldots, v_{k+1} \) are fixed, we have a condition \( |\gamma + v_k| \leq r/\|b_k^*\| \) where \( \gamma = \sum_{i=k+1}^{n} v_i \mu_{ik} \) which is also a fixed value, and hence \( v_k \) has at most \( 2r/\|b_k^*\| + 1 \) possibilities. By repeating this, the number of those \( v \in \mathbb{Z}^n \) is at most \( \prod_{i=1}^{n} (2r/\|b_i^*\| + 1) \).

Now the inequality in the statement holds since

\[ \prod_{i=1}^{n} \left( \frac{2r}{\|b_i^*\|} + 1 \right) = \frac{\prod_{i=1}^{n} (2r + \|b_i^*\|)}{\prod_{i=1}^{n} \|b_i^*\|} \leq \left( \frac{2r + M}{\text{vol}(L)} \right)^n. \]

Hence the lemma holds.

3.2 The Results

We recall the construction of LLL algorithm, DeepLLL algorithm, PotLLL algorithm, and \( S^2 \)LLL algorithm as in Algorithms 1, 2, 3, and 4 respectively. Here, for the argmin in PotLLL algorithm and the argmax in \( S^2 \)LLL algorithm, we suppose that the largest index achieving the minimum and the maximum values, respectively, is chosen when more than one candidates exist.

\begin{algorithm}
\caption{LLL Algorithm}
\textbf{Input:} \( B \) and parameter \( 1/4 < \delta \leq 1 \)
\textbf{Output:} \( \delta \)-LLL-reduced basis \( B \)

1: \( k \leftarrow 2 \)
2: \textbf{while } \( k \leq n \) \textbf{do}
3: \hspace{1em} \text{Size-reduce } \( B = [b_1, \ldots, b_n] \)
4: \hspace{1em} \textbf{if } \( B_k \geq (\delta - \mu_{k,k-1}^2)B_{k-1} \) \textbf{then}
5: \hspace{2em} \( k \leftarrow k + 1 \)
6: \hspace{1em} \textbf{else}
7: \hspace{2em} \( B \leftarrow \sigma_{k-1,k}(B) \), Update GSO
8: \hspace{1em} \( k \leftarrow \max\{k-1, 2\} \)
9: \hspace{1em} \textbf{goto Step 3}
10: \hspace{1em} \textbf{end if}
11: \hspace{1em} \textbf{end while}
12: \textbf{return } B
\end{algorithm}

Let \( M \) denote the value \( \max\{\|b_1\|, \ldots, \|b_n\|\} \) at the beginning of the algorithm. We first note the following properties.

**Lemma 2.** At any step of these algorithms, we have \( \max\{\|b_1\|, \ldots, \|b_n\|\} \leq M \).

\textbf{Proof.} By the construction of the algorithms, the content of \( B \) is changed only by either size-reduction or \( \text{SWAP}_{i,k} \) for some \( i, k \). Now the norm of each of \( b_1, \ldots, b_n \) is not increased by size-reduction by the definition, while \( \max\{\|b_1\|, \ldots, \|b_n\|\} \) is not changed by \( \text{SWAP}_{i,k} \) as it just permutes the elements in \( B \). Hence the claim holds. \( \square \)

**Lemma 3.** For operations \( \text{SWAP}_{\rho,k} \) and \( \text{GSO} \) in the algorithms, let \( B_\rho^{\text{new}} \) denote the value of \( B_\rho \) after the update. Then \( B_\rho^{\text{new}} < B_\rho \).

\[ \text{}\]
Algorithm 2: DeepLLL Algorithm

**Input:** $B$ and parameter $1/4 < \delta \leq 1$

**Output:** $\delta$-DeepLLL-reduced basis $B$

1: $k \leftarrow 2$
2: while $k \leq n$ do
3:   Size-reduce $B$
4:   $C \leftarrow \|b_k\|^2$
5:   for $i = 1, \ldots, k - 1$ do
6:     if $C \geq \delta \|b^*_i\|^2$ then
7:        $C \leftarrow C - \mu^2_{ki} \|b^*_i\|^2$
8:     else
9:        $B \leftarrow \sigma_{i,k}(B)$, Update GSO
10:       $k \leftarrow \max\{i, 2\}$
11:   end if
12: end for
13: $k \leftarrow k + 1$
14: end while
15: return $B$

Algorithm 3: PotLLL Algorithm

**Input:** $B$ and parameter $1/4 < \delta \leq 1$

**Output:** $\delta$-PotLLL-reduced basis $B$

1: $k \leftarrow 2$
2: while $k \leq n$ do
3:   Size-reduce $B$
4:   $i \leftarrow \arg\min_{1 \leq j < k - 1} \text{Pot}(\sigma_{j,k}(B))$
5:   if $\delta \text{Pot}(B) \leq \text{Pot}(\sigma_{i,k}(B))$ then
6:      $k \leftarrow k + 1$
7:   else
8:      $B \leftarrow \sigma_{i,k}(B)$, Update GSO
9:      $k \leftarrow \max\{i, 2\}$
10: end if
11: end while
12: return $B$
Algorithm 4 $S^2$LLL Algorithm

**Input:** $B$ and parameter $0 < \delta \leq 1$
**Output:** $\delta$-$S^2$LLL-reduced basis $B$

1: $k \leftarrow 2$
2: while $k \leq n$ do
3: Size-reduce $B$
4: $i \leftarrow \arg\max_{1 \leq j \leq k-1} S_{jk}$
5: if $S_{ik} \leq (1 - \delta)SS(B)$ then
6: $k \leftarrow k + 1$
7: else
8: $B \leftarrow \sigma_{i,k}(B)$, Update GSO
9: $k \leftarrow \max\{i, 2\}$
10: goto Step 3
11: end if
12: end while
13: return $B$

**Proof.** We also use the superscript “new” for other objects in a similar way. For LLL algorithm, SWAP$_{\rho,k}$ is executed only for $\rho = k - 1$ and when $B_k < (\delta - \mu_{k,k-1}^2)B_{k-1}$. The latter condition is equivalent to

$$(B_{k-1} \geq \delta B_{k-1} > B_k + \mu_{k,k-1}^2 B_{k-1} = B_{k-1}^{\text{new}})$$

(note that for $b_k = b_k^* + \sum_{j=1}^{k-1} \mu_{kj} b_j^*$, we have $\mu_{kj}^{\text{new}} = \langle b_k, b_j^* \rangle B_j^{-2} = \mu_{kj}$ for $j \leq k - 2$, therefore $(b_{k-1})^{\text{new}} = b_k^* + \mu_{k,k-1} b_{k-1}^* = B_{k-1}^{\text{new}} = B_{k-1} + \mu_{k,k-1}^2 B_{k-1}$). Hence the claim holds for the case of LLL algorithm.

For DeepLLL algorithm, when SWAP$_{\rho,k}$ is executed, the value of $C$ at Step 6 with $i = \rho$ is $\|b_k\|^2 - \sum_{j=1}^{\rho-1} \mu_{kj}^2 B_j$. Therefore, the negation of the condition in Step 5 with $i = \rho$ is

$$(B_{\rho} \geq \delta B_{\rho} > \|b_k\|^2 - \sum_{j=1}^{\rho-1} \mu_{kj}^2 B_j = B_{\rho}^{\text{new}}).$$

Hence the claim holds for the case of DeepLLL algorithm.

For PotLLL algorithm, when SWAP$_{\rho,k}$ is executed, we have $\text{Pot}(\sigma_{\rho+1,k}(B)) > \text{Pot}(\sigma_{\rho,k}(B))$ where we put $\sigma_{i,k}(B) = B$ for the case $\rho = k - 1$; this follows from the negation of the condition in Step 5 when $\rho = k - 1$, and follows from the choice of the maximal index in the argmin when $\rho \leq k - 2$. By Eq.(1), this implies that

$$1 > \frac{\|\pi_{\rho}(b_k)\|^2}{B_{\rho}},$$

i.e., $B_{\rho} > \|\pi_{\rho}(b_k)\|^2 = B_{\rho}^{\text{new}}$. Hence the claim holds for the case of PotLLL algorithm.

For $S^2$LLL algorithm, when SWAP$_{\rho,k}$ is executed, for the case $\rho = k - 1$, it follows from the negation of the condition in Step 5 that $S_{k-1,k} > (1 - \delta)BS(B)$ ($\geq 0$). On the other hand, for the case $\rho \leq k - 2$, by the choice of the maximal index in the argmax, we have $S_{\rho,k} > S_{\rho+1,k}$. In both cases, by Eq.(3), we have

$$\mu_{k,k}^2 B_{\rho} \left( \frac{B_{\rho}}{\|\pi_{\rho}(b_k)\|^2} - 1 \right) > 0,$$

therefore $B_{\rho} > \|\pi_{\rho}(b_k)\|^2 = B_{\rho}^{\text{new}}$. Hence the claim holds for the case of $S^2$LLL algorithm. \hfill \square

The main observation in this section is the following. Here we put $\alpha = M^n/\text{vol}(L) \geq 1$. 

8
Lemma 4. Let $w_1, \ldots, w_{n-1}$ be any sequence of real numbers satisfying

$$w_1 \geq 3^n \alpha - 2, \quad w_k \geq (1 + \sqrt{k + 3})^n \alpha - 2 \quad \text{for } k = 2, \ldots, n - 1.$$ 

Then the total number of while blocks in each of the algorithms is at most

$$(n - 1) \cdot \left(1 + \sum_{j=1}^{n-1} w_j\right) + 1.$$

Proof. For $1 \leq k_0 \leq n - 1$, let $W_{k_0}$ denote the set of while blocks in which $\text{SWAP}_{k_0, k}$ for some $k$ is executed. For $0 \leq k_0 \leq n - 1$, let $W_{\leq k_0} = \bigcup_{k=1}^{k_0} W_k$ (note that $W_0 = \emptyset$).

Let $1 \leq k_0 \leq n - 1$. We consider any time interval in the algorithm in which no while blocks in $W_{\leq k_0 - 1}$ appear, and evaluate the number, say $N_{k_0}$, of while blocks in $W_{k_0}$ in the interval. Now $\text{SWAP}_{i, k}$ with $i \leq k_0 - 1$ is not executed during the interval, therefore the values of $b_1, \ldots, b_{k_0-1}$ and $B_1, \ldots, B_{k_0-1}$ are not changed during the interval. Moreover, by Lemma 3 the value of $B_{k_0}$ is monotonically decreased at each while block in $W_{k_0}$. Therefore, at each while block in $W_{k_0}$ during the interval, $b_{k_0}$ is changed to some vector that did not previously appear as the value of $b_{k_0}$. This implies that there are at least $N_{k_0} + 1$ different possibilities of the non-zero lattice point $b_{k_0}$. On the other hand, as

$$\|b_{k_0}\|^2 = B_{k_0} + \sum_{j=1}^{k_0-1} \mu_{k_0,j}^2 B_j \leq B_{k_0} + \frac{1}{4} \sum_{j=1}^{k_0-1} B_j \leq \frac{k_0 + 3}{4} \cdot M^2,$$

(see Lemma 2), Lemma 1 implies that there are at most

$$\frac{(M + 2 \cdot M \sqrt{(k_0 + 3)/4})^n}{\text{vol}(L)} - 1 = (1 + \sqrt{k_0 + 3})^n \alpha - 1$$

possibilities of the non-zero lattice point $b_{k_0}$. Hence we have

$$N_{k_0} + 1 \leq (1 + \sqrt{k_0 + 3})^n \alpha - 1,$$

therefore

$$N_{k_0} \leq (1 + \sqrt{k_0 + 3})^n \alpha - 2.$$

By applying the result above with $k_0 = 1$ to the whole algorithm, it follows that

$$|W_1| \leq N_1 \leq 3^n \alpha - 2.$$

On the other hand, let $2 \leq k_0 \leq n - 1$ and consider the interval between two consecutive while blocks in $W_{\leq k_0 - 1}$, or the interval from the beginning of the algorithm until the first while block in $W_{\leq k_0 - 1}$, or the interval from the end of the last while block in $W_{\leq k_0 - 1}$ until the end of the algorithm. By applying the result above to the interval, it follows that there are at most $N_{k_0}$ while blocks in $W_{k_0}$ in the interval. As there are $|W_{\leq k_0 - 1}| + 1$ such intervals, it follows that

$$|W_{k_0}| \leq N_{k_0} \cdot (|W_{\leq k_0 - 1}| + 1) \leq \left(1 + \sqrt{k_0 + 3}\right)^n \alpha - 2 \cdot \left(1 + \sum_{j=1}^{k_0-1} |W_j|\right).$$

Hence it holds recursively that $|W_{k_0}| \leq w_{k_0}$ for any $1 \leq k_0 \leq n - 1$. 


Among the while blocks in the algorithm, those not in $W_{\leq n-1}$ satisfy that $k$ is incremented in the while block. As there are at most $n-1$ consecutive such while blocks (except the last while block in the algorithm where the $k$ is incremented from $n$ to $n+1$), the total number of while blocks in the algorithm is at most

$$(n-1) \cdot (|W_{\leq n-1}| + 1) + 1 \leq (n-1) \cdot \left( 1 + \sum_{j=1}^{n-1} w_j \right) + 1.$$ 

Hence the claim holds. $\square$

Now our main result is given as follows (this is the same as the theorem described in the introduction).

**Theorem 2.** Assume that $n \geq 2$. In each of LLL algorithm and its variants above, including the case of the maximal parameter $\delta = 1$, the total number of while blocks is at most

$$(n-1) \cdot \left( \frac{3 \prod_{j=2}^{n-1} (2 + \sqrt{j+3})}{\sqrt{5}} \right)^n \alpha^{n-1} \leq (n-1) \cdot \left( \frac{2}{\sqrt{5}} + 1 \right)^{n(n-2)} \alpha^{n-1} \left( \frac{3 \cdot (n+2)!}{8} \right)^{n/2}$$

$$\leq \alpha^{n-1} \left( \frac{2}{\sqrt{5}} + 1 \right)^{n-1/2} n^{1+o(1)} \left( \alpha^{n-1} \right)^{n^2} (\alpha^{n-1} \right)^{(1 + (1 + \sqrt{m+3})^n - 2)}$$

where $\alpha = M^n / \text{vol}(L)$. In particular, these algorithms always halt within finitely many steps.

**Proof.** First we show recursively that the sequence $w_1, \ldots, w_{n-1}$ given by

$$w_1 = 3^n \alpha - 2, \quad w_k = \left( 3 \prod_{j=2}^{k-1} \left( 2 + \sqrt{j+3} \right) \right)^n \alpha^k - 2$$

satisfies the condition in Lemma 4. The case of $w_1$ is obvious. When $k \geq 2$ and the claim holds for $w_1, \ldots, w_{k-1}$, it follows recursively that

$$\sum_{i=1}^{m} w_i \leq \left( 3 \prod_{j=2}^{m} \left( 2 + \sqrt{j+3} \right) \right)^n \alpha^m - 2$$

for $m = 1, \ldots, k-1$.

Indeed, this is obvious when $m = 1$. When $m \geq 2$, we have

$$\sum_{i=1}^{m} w_i = \sum_{i=1}^{m-1} w_i + w_m$$

$$\leq \left( 3 \prod_{j=2}^{m-1} \left( 2 + \sqrt{j+3} \right) \right)^n \alpha^{m-1} + \left( 3 \prod_{j=2}^{m-1} \left( 2 + \sqrt{j+3} \right) \right)^n \alpha^m - 2$$

$$\leq \left( 3 \prod_{j=2}^{m-1} \left( 2 + \sqrt{j+3} \right) \right)^n \alpha^m \left( 1 + (1 + \sqrt{m+3})^n - 2 \right)$$

where we used the relation $\alpha \geq 1$. By using the fact

$$1 + (1 + \sqrt{m+3})^n \leq (1 + (1 + \sqrt{m+3})^n) = (2 + \sqrt{m+3})^n ,$$

we have

$$\sum_{i=1}^{m} w_i \leq \left( 3 \prod_{j=2}^{m-1} \left( 2 + \sqrt{j+3} \right) \right)^n \alpha^m (2 + \sqrt{m+3})^n - 2 = \left( 3 \prod_{j=2}^{m} \left( 2 + \sqrt{j+3} \right) \right)^n \alpha^m - 2 ,$$

10
as desired.

By using the result above, the right-hand side of the inequality in Lemma 4 to be verified becomes

\[
(1 + \sqrt{k+3})^n \alpha - 2 \cdot \left( 1 + \sum_{j=1}^{k-1} w_j \right)
\]

\[
\leq (1 + \sqrt{k+3})^n \alpha - 2 \cdot \left( 3 \prod_{j=2}^{k-1} (2 + \sqrt{j+3}) \right)^n \alpha^{k-1} - 1
\]

\[
\leq (1 + \sqrt{k+3})^n \alpha \cdot \left( 3 \prod_{j=2}^{k-1} (2 + \sqrt{j+3}) \right)^n \alpha^{k-1} - 2 = w_k
\]

where we used the relations \((1 + \sqrt{k+3})^n \alpha \geq 0\) and \((3 \prod_{j=2}^{k-1} (2 + \sqrt{j+3}) \alpha^{k-1} \geq 2\). Hence the condition in Lemma 4 is satisfied by these \(w_k\). Therefore, the total number of while blocks is at most the value in the statement of Lemma 4. Now by the argument above, this value becomes

\[
(n-1) \cdot \left( 1 + \sum_{j=1}^{n-1} w_j \right) + 1 \leq (n-1) \cdot \left( 3 \prod_{j=2}^{n-1} (2 + \sqrt{j+3}) \right)^n \alpha^{n-1} - 1 + 1
\]

\[
\leq (n-1) \cdot \left( 3 \prod_{j=2}^{n-1} (2 + \sqrt{j+3}) \right)^n \alpha^{n-1},
\]

which is the left-hand side of the inequality in the statement of this theorem, as desired.

Moreover, by using the fact that \(2 + \sqrt{j+3} \leq (2/\sqrt{5} + 1)\sqrt{j+3}\) for any \(j \geq 2\), the value above is smaller than or equal to

\[
(n-1) \cdot \left( 3 \left( \frac{2}{\sqrt{5}} + 1 \right)^{-2} \prod_{j=2}^{n-1} \sqrt{j+3} \right)^n \alpha^{n-1}
\]

\[
= (n-1) \cdot 3^n \left( \frac{2}{\sqrt{5}} + 1 \right)^{n-2} \alpha^{n-1} \left( \frac{n+2}{4!} \right)^{n/2}
\]

\[
= (n-1) \cdot \left( \frac{2}{\sqrt{5}} + 1 \right)^{n-2} \alpha^{n-1} \left( \frac{3 \cdot (n+2)!}{8} \right)^{n/2},
\]

as desired. By using Stirling’s Formula \(m! \leq \sqrt{2\pi m} m^{m+1/2} e^{-m+1/(12m)}\), this value is further bounded by

\[
(n-1) \cdot \left( \frac{2}{\sqrt{5}} + 1 \right)^{n(n-2)} \alpha^{n-1} \left( \frac{3 \cdot \sqrt{2\pi} (n+2)^{n+5/2} e^{-n-2+1/(12(n+2))}}{8} \right)^{n/2}
\]

which at \(n \to \infty\) is upper bounded by

\[
\left( \frac{2}{\sqrt{5}} + 1 \right)^{n(n-2)} \alpha^{n-1} \left( (n+2)^{n+5/2 + o(1)} e^{-n} \right)^{n/2}
\]

\[
= \alpha^{n-1} \left( \left( \frac{2}{\sqrt{5}} + 1 \right)^{1-2/n} (n+2)^{1/2+5/(4n)+o(1)} e^{-1/2} \right)^{n^2}
\]

\[
\leq \alpha^{n-1} \left( \left( \frac{2}{\sqrt{5}} + 1 \right)^{1/2+5/(4n)+o(1)} e^{-1/2} n^{1/2+o(1)} \right)^{n^2}.
\]

Hence the theorem holds. \(\square\)
Remark 1. By assuming that each while block can be executed within polynomial time in \( n \), it is intuitively estimated that the computational complexity of each of the algorithms has the same asymptotic bound as Theorem 2 as the bound in Theorem 2 is already hyperexponential in \( n \).

3.3 Comparisons to the Previous Results

When \( \text{vol}(L) \) is almost constant, i.e., \( \alpha \approx M^n \), the bound for the computational complexity of 1-LLL, 1-DeepLLL, 1-PotLLL, and 1-S\(^2\)LLL obtained by Theorem 2 can be roughly written as \((Mn)^{O(n^2)}\). Comparing to the bound \( O(A^{n^3} \log M) \) for 1-LLL given in [2] (where \( A \) is a constant with \( A > (4/3)^{1/12} \)), our bound becomes better for the case of smaller \( M \) (e.g., when \( M = o(2^n/n) \)), while our bound does not improve the previous one for the case of larger \( M \) (e.g., when \( M = \Omega(2^n) \)). The latter case may often occur in practical situations. For example, for any basis \( \mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \) provided as a problem instance of Darmstadt SVP Challenge [3], \( \mathbf{B} \) is an upper triangular matrix where the first component of each column \( \mathbf{b}_j \) is \( 2^{\Theta(n)} \), the diagonal components (except for the first column) are all 1, and the other components are all 0. Now we have \( M \sim \text{vol}(L) = 2^{\Theta(n)} \) and our bound becomes \((2^{\Theta(n)} n)^{O(n^2)}\), which does not improve the bound \( O(A^{n^3} \log M) \). Moreover, as mentioned in the introduction, the complexity of 1-LLL and 1-PotLLL for input basis consisting of integer vectors is further bounded by \( O(Mn^2) \); our bound does not improve this bound either.

We emphasize that the main advantage of our result is its generality compared to the previous results. Namely, in contrast to our result applicable to all of the four algorithms in a unified way, the proof of the bound \( O(A^{n^3} \log M) \) in [2] is specific to the case of 1-LLL. On the other hand, the bound \( O(M^{n^2}) \) for the case of integer inputs in 1-LLL and 1-PotLLL mentioned above is deduced from the fact that now \( \text{Pot}(\mathbf{b}) \) takes integer values and is monotonically decreasing during those algorithms. The same argument for 1-S\(^2\)LLL by using SS(\( \mathbf{B} \)) instead of \( \text{Pot}(\mathbf{B}) \) does not work due to the difference that SS(\( \mathbf{B} \)) is not necessarily an integer even for integer basis \( \mathbf{B} \). For the case of 1-DeepLLL, if the value \( \text{Pot}(\mathbf{B}) \) were also monotonically decreasing during the algorithm, then the same argument would yield a simple bound better than ours. However, in fact \( \text{Pot}(\mathbf{B}) \) is not monotonically decreasing in 1-DeepLLL, as shown in the following example, therefore the previous argument might not be straightforwardly applicable to 1-DeepLLL.

Example 1. We give examples showing that the value \( \text{Pot}(\mathbf{B}) \) (for integer inputs) is not monotonically decreasing in general for 1-DeepLLL and 1-S\(^2\)LLL. We set \( n = 3 \) and consider the following size-reduced basis

\[
\mathbf{B} = \begin{pmatrix} 0 & -3 & 2 \\ 3 & -2 & -2 \\ -2 & 0 & -2 \end{pmatrix}.
\]

By applying GSO to \( \mathbf{B} \), we obtain

\[
\mathbf{B}^* = \begin{pmatrix} 0 & -3 & 8/7 \\ 3 & -8/13 & -12/7 \\ -2 & -12/13 & -18/7 \end{pmatrix}.
\]

Hence we have

\[
\text{Pot}(\mathbf{B}) = \left\| \begin{pmatrix} 0 \\ 3 \\ -2 \end{pmatrix} \right\|^{2 \cdot 3} \left\| \begin{pmatrix} -3 \\ -8/13 \\ -12/13 \end{pmatrix} \right\|^{2 \cdot 2} \left\| \begin{pmatrix} 8/7 \\ -12/7 \\ -18/7 \end{pmatrix} \right\|^{2 \cdot 1} = 13^3 \cdot \left( \frac{133}{13} \right)^2 \cdot \frac{76}{7} = 13 \cdot 7 \cdot 19^2 \cdot 76 = 2,496,676.
\]

On the other hand, when \( \mathbf{B} \) is input to 1-DeepLLL, \( k = 2 \) is incremented to \( k = 3 \) in the first while block (as \( \|\mathbf{b}_2\| = \|\mathbf{b}_1\| \)), and then SWAP\(_{1,3}\) is executed in the second while block (as \( \|\mathbf{b}_4\|^2 = 12 < 13 = \|\mathbf{b}_1\|^2 \)).
The result is
\[ \sigma_{1,3}(B) = \begin{pmatrix} 2 & 0 & -3 \\ -2 & 3 & -2 \\ -2 & -2 & 0 \end{pmatrix}, \]
which is size-reduced. By applying GSO, we obtain
\[ (\sigma_{1,3}(B))^* = \begin{pmatrix} 2 & 1/3 & -5/2 \\ -2 & 8/3 & -1 \\ -2 & -7/3 & -3/2 \end{pmatrix}. \]
Hence we have
\[
\text{Pot}(\sigma_{1,3}(B)) = \left\| \begin{pmatrix} 2 \\ -2 \\ -2 \end{pmatrix} \right\|^2 \cdot \left\| \begin{pmatrix} 1/3 \\ 8/3 \\ -7/3 \end{pmatrix} \right\|^2 \cdot \left\| \begin{pmatrix} -5/2 \\ -1 \\ -3/2 \end{pmatrix} \right\|^2 = 12^3 \cdot \left( \frac{38}{3} \right)^2 \left( \frac{19}{2} \right) = 3 \cdot 4^3 \cdot 2 \cdot 19^2 \cdot 19 = 2,633,856 > \text{Pot}(B),
\]
which shows that the value \( \text{Pot}(B) \) is not monotonically decreasing in 1-DeepLLL.

Similarly, for 1-\( S^2 \)LLL, we set \( n = 3 \) and consider the following size-reduced basis
\[ B = \begin{pmatrix} 3 & 1 & 1 \\ 1 & -1 & -2 \\ -1 & 2 & -2 \end{pmatrix}. \]
By applying GSO to \( B \), we obtain
\[ B^* = \begin{pmatrix} 3 & 1 & 23/66 \\ 1 & -1 & -161/66 \\ -1 & 2 & -46/33 \end{pmatrix}. \]
Hence we have
\[
\text{Pot}(B) = \left\| \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix} \right\|^2 \cdot \left\| \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} \right\|^2 \cdot \left\| \begin{pmatrix} 23/66 \\ -161/66 \\ -46/33 \end{pmatrix} \right\|^2 = 11^3 \cdot 6^2 \cdot \left( \frac{529}{66} \right) = 11^2 \cdot 6 \cdot 529 = 384,054 .
\]
On the other hand, when \( B \) is input to 1-\( S^2 \)LLL, \( k = 2 \) is incremented to \( k = 3 \) in the first \textbf{while} block (as now \( \langle b_1, b_2 \rangle = 0 \) and \text{SWAP}_{1,2} \) does not change the values of \( B_i \)'s, therefore \( S_{12} = \text{SS}(B) - \text{SS}(\sigma_{1,2}(B)) = 0 \). For the next \textbf{while} block, we have
\[ \sigma_{1,3}(B) = \begin{pmatrix} 1 & 3 & 1 \\ -2 & 1 & -1 \\ -2 & -1 & 2 \end{pmatrix}, \sigma_{2,3}(B) = \begin{pmatrix} 3 & 1 & 1 \\ 1 & -2 & -1 \\ -1 & -2 & 2 \end{pmatrix}, \]
which are size-reduced, and
\[ (\sigma_{1,3}(B))^* = \begin{pmatrix} 1 & 8/3 & 46/45 \\ -2 & 5/3 & -23/18 \\ -2 & -1/3 & 161/90 \end{pmatrix}, (\sigma_{2,3}(B))^* = \begin{pmatrix} 3 & 2/11 & 46/45 \\ 1 & -25/11 & -23/18 \\ -1 & -19/11 & 161/90 \end{pmatrix}. \]
Then we have
\[
\text{SS}(\sigma_{1,3}(B)) = \left\| \begin{pmatrix} 1 \\ -2 \\ -2 \end{pmatrix} \right\|^2 + \left\| \begin{pmatrix} 8/3 \\ 5/3 \\ -1/3 \end{pmatrix} \right\|^2 + \left\| \begin{pmatrix} 46/45 \\ -23/18 \\ 161/90 \end{pmatrix} \right\|^2 = \frac{2239}{90},
\]
which shows that the value \( \text{Pot}(B) \) is not monotonically decreasing in 1-S^2LLL.

4 Experimental Comparison of the Cases \( \delta < 1 \) and \( \delta = 1 \)

In this section, we describe our computer experiments to compare practical behaviors of LLL algorithm and its variants for two cases \( \delta = 1 \) and \( \delta < 1 \) (more precisely, \( \delta = 0.99 \) which is one of the popular parameter choices in cryptographic applications).

We implemented the four algorithms dealt with in Section 3 (LLL, DeepLLL, PotLLL, S^2LLL) and also variants of DeepLLL algorithm where the for loop is executed only for the range \( k - 5 \leq i \leq k - 1 \) (denoted here by “Deep-5”) or \( k - 10 \leq i \leq k - 1 \) (denoted here by “Deep-10”). The machine environment is: Windows WSL 2, Intel(R) Core(TM) i7-8650 CPU @ 1.90 GHz 2.11 GHz, 16 GB main memory. Instead of the real execution times, we counted the numbers of exchanges SWAP_\* for each of the 1000 steps are randomly selected. Hence \( B_2 \) is obtained by randomly generating a lattice with vol(L) = 1. More precisely, \( B_2 \) is generated by starting from the identity matrix and iterating, 1000 times in total, addition of ±1 times some column to some other column, where the two columns and the sign ±1 at each of the 1000 steps are randomly selected.

Table 4 shows the maximum values, among the five inputs for each algorithm, dimension, and input type, of the ratios of numbers of basis exchanges for the case \( \delta = 1 \) relative to the case \( \delta = 0.99 \) with the same input; “N/A” means that the algorithm did not halt within our experiment time. The exact results of the experiments are shown in Tables 2, 3, 4, and 5 at the appendix. For the five algorithms other than \( S^2LLL \), the ratios calculated in our experiments are fairly small (smaller than 3, and mostly close to 1) and seem to be not drastically growing as \( n \) increases. For the case of \( S^2LLL \), the ratio grows more rapidly especially for the input type \( B_2 \), but the growth seems to be linear or at most quadratic in \( n \). These results suggest that, contrary to the significant difference of our hyperexponentially growing upper bound for the case \( \delta = 1 \) from the known polynomial upper bound (except for DeepLLL) for the case \( \delta < 1 \), the computational complexity for the two cases \( \delta = 1 \) and \( \delta < 1 \) in practice may be closer and accordingly, there might be hope to improve the upper bound for the case \( \delta = 1 \) much further.

Acknowledgements. This research was partially supported by the Ministry of Internal Affairs and Communications SCOPE Grant Number 182103105 and by JSPS KAKENHI Grant Number JP19H01804.
Table 1: Maximum ratios of numbers of basis exchanges for the case $\delta = 1$ relative to the case $\delta = 0.99$ with the same input; here “N/A” means that the algorithm did not halt within our experiment time.

| Algorithm | Input | Dimension $n$ |
|-----------|-------|---------------|
|           |       | 10  | 15  | 20  | 25  | 30  | 35  | 40  |
| LLL       | B₁    | 1.025 | 1.037 | 1.066 | 1.077 | 1.106 | 1.102 | 1.097 |
|           | B₂    | 1.035 | 1.050 | 1.083 | 1.115 | 1.065 | 1.114 | 1.100 |
| Deep      | B₁    | 1.043 | 1.096 | 1.222 | 1.159 | 1.237 | N/A   | N/A   |
|           | B₂    | 1.077 | 1.131 | 1.241 | 1.331 | 1.450 | 1.473 | N/A   |
| Deep-5    | B₁    | 1.055 | 1.101 | 1.126 | 1.126 | 1.126 | 1.126 | 1.126 |
|           | B₂    | 1.069 | 1.299 | 1.223 | 2.193 | 1.385 | 2.544 | 1.144 |
| Deep-10   | B₁    | 1.043 | 1.088 | 1.095 | 1.299 | 1.299 | 1.299 | 1.299 |
|           | B₂    | 1.077 | 1.447 | 1.564 | 2.033 | 1.689 | 1.169 | 1.498 |
| Pot       | B₁    | 1.042 | 1.038 | 1.063 | 1.070 | 1.084 | 1.078 | 1.098 |
|           | B₂    | 1.041 | 1.042 | 1.080 | 1.133 | 1.058 | 1.080 | 1.093 |
| $S^2$     | B₁    | 1.604 | 2.346 | 3.256 | 4.263 | 5.373 | 6.479 | 7.629 |
|           | B₂    | 2.075 | 3.375 | 4.976 | 6.872 | 8.882 | 11.381 | 13.228 |

References

[1] Ajtai, M., Dwork, C., “A public-key cryptosystem with worst-case/average-case equivalence.” In: Proceedings of STOC 1997, pp.284–293, ACM, 1997.

[2] Akhavi, A., “The optimal LLL algorithm is still polynomial in fixed dimension.” Theoretical Computer Science, vol.297, pp.3–23, 2003.

[3] SVP Challenge. [https://www.latticechallenge.org/svp-challenge/](https://www.latticechallenge.org/svp-challenge/)

[4] Fontein, F., Schneider, M., Wagner, U., “PotLLL: a polynomial time version of LLL with deep insertions.” Designs, Codes and Cryptography, vol.73, pp.355–368, 2014.

[5] Galbraith, S.D., Mathematics of Public Key Cryptography. Cambridge University Press, Cambridge, 2012.

[6] Koblitz, N., “Elliptic curve cryptosystems.” Mathematics of Computation, vol.48, no.177, pp.203–209, 1987.

[7] Lenstra, A.K., Lenstra, H.W., Lovász, L., “Factoring polynomials with rational coefficients.” Mathematische Annalen, vol.261, no.4, pp.515–534, 1982.

[8] Miller, V.S., “Use of elliptic curves in cryptography.” In: Proceedings of CRYPTO 1985, LNCS vol.218, pp.417–426, Springer, 1985.

[9] Nguyen, Q., Vallée, B., The LLL Algorithm. Information Security Cryptography, 2010.

[10] Rivest, R.L., Shamir, A., Adleman, L.M., “A method for obtaining digital signatures and public-key cryptosystems.” Communications of the ACM, vol.21, no.2, pp.120–126, 1978.

[11] Schnorr, C.P., “Block reduced lattice bases and successive minima.” Combinatorics, Probability and Computing, vol.3, no.4, pp.507–522, 1994.

[12] Shor, P.W., “Algorithms for quantum computation: discrete logarithms and factoring.” In: Proceedings of FOCS 1994, pp.124–134, IEEE, 1994.

[13] Yasuda, M., Yamaguchi, J., “A new polynomial-time variant of LLL with deep insertions for decreasing the squared-sum of Gram-Schmidt lengths.” Designs, Codes and Cryptography, vol.87, pp.2489–2505, 2019.
Appendix

Here we include (as mentioned in Section 4) the tables showing the detailed numbers of exchanges for basis vectors in LLL algorithm and its variants with two choices of parameters $\delta = 0.99$ and $\delta = 1$. We note that for some inputs for Deep-5 and Deep-10, the number of basis exchanges for the case $\delta = 1$ is even smaller than that for the case $\delta = 0.99$ as opposed to the intuition. This would be explained by noting that for DeepLLL algorithm and its variants (i.e., Deep-5 and Deep-10), the running time depends not only on the number of basis exchanges but also on the distribution of the index $i$ in for loop at which the basis exchange occurred, the latter being different in general for the two cases $\delta = 0.99$ and $\delta = 1$. 
Table 2: Numbers of exchanges for basis vectors in LLL and its variants with $\delta = 0.99$ (upper rows) and $\delta = 1$ (lower rows) for dimensions $n \in \{10, 15\}$; here “Deep”, “Pot”, and “$S^2$” stand for DeepLLL, PotLLL, and $S^2$LLL, respectively

| $n$ | Input Seed | $M$ | # of Exchanges |
|-----|-------------|-----|----------------|
|     |             |     | LLL   Deep Deep-5 Deep-10 Pot $S^2$ |
| 0   | $2.12 \times 10^{120}$ | 1862 | 717 | 807 | 717 | 706 | 484 |
|     |             | 1901 | 742 | 851 | 742 | 726 | 720 |
| 1   | $1.41 \times 10^{120}$ | 2018 | 778 | 853 | 778 | 766 | 485 |
|     |             | 2046 | 801 | 865 | 801 | 785 | 778 |
| B1  | $1.86 \times 10^{120}$ | 2036 | 771 | 854 | 771 | 743 | 488 |
|     |             | 2066 | 790 | 882 | 790 | 761 | 748 |
| 2   | $1.89 \times 10^{120}$ | 2052 | 774 | 849 | 774 | 757 | 480 |
|     |             | 2087 | 787 | 858 | 787 | 767 | 760 |
| 3   | $2.07 \times 10^{120}$ | 1998 | 767 | 870 | 767 | 742 | 482 |
|     |             | 2047 | 800 | 905 | 800 | 773 | 762 |
| 4   | $3.46 \times 10^{18}$  | 1508 | 486 | 579 | 486 | 485 | 240 |
|     |             | 1556 | 517 | 601 | 517 | 504 | 498 |
| B2  | $1.10 \times 10^{17}$  | 1520 | 507 | 589 | 507 | 502 | 251 |
|     |             | 1559 | 540 | 606 | 540 | 513 | 499 |
| 2   | $1.33 \times 10^{16}$  | 1357 | 457 | 522 | 457 | 419 | 215 |
|     |             | 1383 | 470 | 558 | 470 | 436 | 428 |
| 3   | $8.59 \times 10^{15}$  | 1325 | 432 | 517 | 432 | 428 | 237 |
|     |             | 1371 | 451 | 529 | 451 | 433 | 432 |
| 4   | $2.16 \times 10^{18}$  | 1583 | 521 | 610 | 521 | 508 | 253 |
|     |             | 1627 | 561 | 641 | 561 | 516 | 514 |
| B1  | $2.12 \times 10^{120}$ | 4465 | 1774 | 1337 | 1805 | 1570 | 710 |
|     |             | 4623 | 1883 | 1384 | 1964 | 1611 | 1598 |
| 2   | $1.41 \times 10^{120}$ | 4460 | 1691 | 1145 | 1514 | 1555 | 696 |
|     |             | 4600 | 1853 | 1163 | 1607 | 1612 | 1564 |
| 3   | $1.86 \times 10^{120}$ | 4489 | 1765 | 1338 | 1839 | 1556 | 686 |
|     |             | 4553 | 1832 | 1185 | 1860 | 1597 | 1539 |
| 4   | $1.89 \times 10^{120}$ | 4602 | 1888 | 1314 | 1495 | 1608 | 696 |
|     |             | 4748 | 1998 | 1338 | 1379 | 1560 | 1633 |
| B2  | $2.07 \times 10^{120}$ | 4624 | 1891 | 1340 | 1598 | 1600 | 704 |
|     |             | 4794 | 2066 | 1475 | 1712 | 1661 | 1588 |
| 0   | $3.04 \times 10^{13}$  | 3241 | 1270 | 471 | 1241 | 1062 | 341 |
|     |             | 3402 | 1344 | 612 | 1307 | 1107 | 1060 |
| 1   | $1.03 \times 10^{12}$  | 3385 | 1428 | 674 | 1304 | 1162 | 345 |
|     |             | 3473 | 1520 | 518 | 1441 | 1176 | 1154 |
| B2  | $1.59 \times 10^{13}$  | 3436 | 1322 | 547 | 1305 | 1120 | 328 |
|     |             | 3609 | 1495 | 543 | 1386 | 1166 | 1107 |
| 3   | $2.58 \times 10^{12}$  | 3786 | 1473 | 687 | 1487 | 1257 | 360 |
|     |             | 3948 | 1605 | 735 | 1239 | 1299 | 1198 |
| 4   | $2.58 \times 10^{12}$  | 3195 | 1246 | 688 | 804 | 1027 | 312 |
|     |             | 3282 | 1272 | 622 | 1163 | 1061 | 1029 |
Table 3: Numbers of exchanges for basis vectors in LLL and its variants with $\delta = 0.99$ (upper rows) and $\delta = 1$ (lower rows) for dimensions $n \in \{20, 25\}$; here “Deep”, “Pot”, and “$S^2$” stand for DeepLLL, PotLLL, and $S^2$LLL, respectively

| $n$ | Input Seed | $M$               | # of Exchanges |
|-----|------------|-------------------|----------------|
|     |            | LLL Deep Deep-5 Deep-10 Pot $S^2$ |
| 20  | 0          | $2.12 \times 10^{120}$ | 7995 3788 1345 2287 2860 903 |
|     |            |                   | 8149 4202 1515 2414 3009 2830 |
|     | 1          | $1.41 \times 10^{120}$ | 7952 3736 1371 2147 2868 889 |
|     |            |                   | 8476 4564 1352 2352 3032 2841 |
|     | B1         | $1.86 \times 10^{120}$ | 7948 4066 1367 2429 2896 885 |
|     |            |                   | 8310 4597 1185 1942 3049 2831 |
|     | 2          | $1.89 \times 10^{120}$ | 7892 3724 1314 2433 2826 871 |
|     |            |                   | 8223 4146 1338 2469 2948 2836 |
|     | 3          | $2.07 \times 10^{120}$ | 8188 4437 1340 2567 2976 892 |
|     |            |                   | 8556 4841 1475 2258 3164 2903 |
|     | 4          | $4.89 \times 10^{12}$ | 8569 3243 428  956 2095 423 |
|     |            |                   | 8636 4027 454 1495 2226 2044 |
| 25  | 0          | $2.12 \times 10^{120}$ | 12183 7995 1345 2623 4605 1077 |
|     |            |                   | 12521 9036 1515 2414 4872 4591 |
|     | 1          | $4.56 \times 10^{10}$ | 6178 3490 387  1435 2217 446 |
|     |            |                   | 6462 3959 408  1464 2300 2203 |
|     | B2         | $2.32 \times 10^{10}$ | 6454 3659 452  1323 2361 452 |
|     |            |                   | 6831 4068 553  1588 2435 2249 |
|     | 2          | $1.85 \times 10^{8}$ | 5616 3127 377  1337 2001 396 |
|     |            |                   | 5948 3373 422  1250 2125 1933 |
|     | 3          | $1.61 \times 10^{10}$ | 5956 3028 566  1219 1994 434 |
|     |            |                   | 6348 3296 489  1506 2153 2011 |
|     | 4          | $2.07 \times 10^{120}$ | 12074 8587 1340 2567 4555 1072 |
|     |            |                   | 12826 9731 1475 2258 4876 4535 |
|     | B1         | $1.18 \times 10^{120}$ | 11828 7673 1307 2429 4394 1040 |
|     |            |                   | 12626 8587 1185 1942 4700 4215 |
|     | 2          | $1.86 \times 10^{120}$ | 11950 7738 1314 2524 4405 1046 |
|     |            |                   | 12471 8492 1338 2469 4556 4366 |
|     | 3          | $2.07 \times 10^{120}$ | 12074 8587 1340 2567 4555 1072 |
|     |            |                   | 12826 9731 1475 2258 4876 4535 |
|     | 4          | $2.48 \times 10^{12}$ | 8610 7051 269  607 3252 480 |
|     |            |                   | 9329 8608 240  1234 3686 3298 |
Table 4: Numbers of exchanges for basis vectors in LLL and its variants with \( \delta = 0.99 \) (upper rows) and \( \delta = 1 \) (lower rows) for dimensions \( n \in \{30, 35\} \); here “Deep”, “Pot”, and “S\(^2\)” stand for DeepLLL, PotLLL, and S\(^2\)LLL, respectively; and “N/A” means that the algorithm did not halt within our experiment time.

| \( n \) | Input Seed | \( M \) | \# of Exchanges |
|------|------------|-----------|----------------|
|      |            | LLL | Deep | Deep-5 | Deep-10 | Pot | S\(^2\) |
| 0    | 2.12 \times 10^{120} | 16649 | 14587 | 1345 | 2623 | 6640 | 1218 |
|      |            | 17321 | 17425 | 1515 | 2414 | 6933 | 6439 |
| 1    | 1.41 \times 10^{120} | 16392 | 14938 | 1371 | 2147 | 6751 | 1212 |
|      |            | 18124 | 18261 | 1352 | 2789 | 7138 | 6512 |
| B\(_1\) | 2 | 1.86 \times 10^{120} | 16152 | 14273 | 1367 | 2429 | 6473 | 1183 |
|      |            | 17663 | 17661 | 1185 | 1942 | 6977 | 6194 |
| 3    | 1.89 \times 10^{120} | 16436 | 14535 | 1314 | 2524 | 6418 | 1184 |
|      |            | 17332 | 17865 | 1338 | 2469 | 6772 | 6284 |
| 4    | 2.07 \times 10^{120} | 16313 | 14898 | 1340 | 2567 | 6570 | 1226 |
|      |            | 17679 | 17482 | 1475 | 2258 | 7124 | 6398 |
| 30   | 9.53 \times 10^{6} | 11442 | 11944 | 177  | 514  | 4727 | 534  |
|      |            | 12184 | 15218 | 171  | 868  | 4999 | 4540 |
| B\(_2\) | 2 | 1.13 \times 10^{7} | 11268 | 12261 | 286  | 593  | 4756 | 503  |
|      |            | 11868 | 15072 | 147  | 828  | 4897 | 4231 |
| 3    | 1.26 \times 10^{7} | 11930 | 12651 | 256  | 820  | 5046 | 544  |
|      |            | 12560 | 15470 | 217  | 568  | 5324 | 4832 |
| 4    | 3.47 \times 10^{6} | 11371 | 12579 | 199  | 662  | 4996 | 525  |
|      |            | 12012 | 15400 | 196  | 780  | 5118 | 4638 |
| 35   | 2.12 \times 10^{120} | 21221 | N/A  | 1345 | 2623 | 8897 | 1340 |
|      |            | 22548 | N/A  | 1515 | 2414 | 9487 | 8569 |
| B\(_1\) | 2 | 1.41 \times 10^{120} | 21432 | N/A  | 1371 | 2147 | 9191 | 1333 |
|      |            | 23010 | N/A  | 1352 | 2789 | 9695 | 8548 |
| 3    | 1.86 \times 10^{120} | 20835 | N/A  | 1367 | 2429 | 8797 | 1316 |
|      |            | 22882 | N/A  | 1185 | 1942 | 9486 | 8274 |
| 4    | 2.07 \times 10^{120} | 20816 | N/A  | 1340 | 2567 | 9031 | 1353 |
|      |            | 22940 | N/A  | 1475 | 2258 | 9623 | 8593 |
| 0    | 6.70 \times 10^{6} | 15685 | 24239 | 135  | 426  | 6954 | 582  |
|      |            | 16961 | 33381 | 214  | 498  | 7512 | 6624 |
| 1    | 1.38 \times 10^{6} | 14205 | 22975 | 131  | 554  | 6550 | 560  |
|      |            | 15825 | 31369 | 195  | 567  | 6863 | 5896 |
| B\(_2\) | 2 | 1.81 \times 10^{6} | 13789 | 21574 | 165  | 612  | 6018 | 530  |
|      |            | 15055 | 25067 | 179  | 442  | 6480 | 5888 |
| 3    | 1.88 \times 10^{6} | 15201 | 21214 | 206  | 489  | 6652 | 550  |
|      |            | 15393 | 31244 | 188  | 551  | 6984 | 6259 |
| 4    | 1.38 \times 10^{6} | 14941 | 22677 | 136  | 545  | 6610 | 579  |
|      |            | 15645 | 29246 | 346  | 614  | 6899 | 6299 |
Table 5: Numbers of exchanges for basis vectors in LLL and its variants with $\delta = 0.99$ (upper rows) and $\delta = 1$ (lower rows) for dimension $n = 40$; here “Deep”, “Pot”, and “$S^2$” stand for DeepLLL, PotLLL, and $S^2$LLL, respectively; and “N/A” means that the algorithm did not halt within our experiment time.

| $n$ | Input Seed | $M$ | # of Exchanges |
|-----|------------|-----|----------------|
| 0   | $2.12 \times 10^{120}$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     |            | 26073 | N/A  | 1345   | 2623    | 11530 | 1449 |
|     |            | 27687 | N/A  | 1515   | 2414    | 12232 | 11010 |
| 1   | $1.41 \times 10^{120}$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     |            | 26116 | N/A  | 1371   | 2147    | 11690 | 1447 |
|     |            | 27728 | N/A  | 1352   | 2789    | 12246 | 10923 |
| 2   | $1.86 \times 10^{120}$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     |            | 25890 | N/A  | 1367   | 2429    | 11119 | 1421 |
|     |            | 28086 | N/A  | 1185   | 1942    | 12213 | 10471 |
| 3   | $1.89 \times 10^{120}$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     |            | 25780 | N/A  | 1314   | 2524    | 11321 | 1419 |
|     |            | 27344 | N/A  | 1338   | 2469    | 11964 | 10825 |
| 4   | $2.07 \times 10^{120}$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     |            | 25525 | N/A  | 1340   | 2567    | 11608 | 1465 |
|     |            | 28013 | N/A  | 1475   | 2258    | 12277 | 10940 |
|     | $1.32 \times 10^5$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     | 0          | 17063 | N/A  | 195    | 387     | 7869  | 572  |
|     |            | 18537 | N/A  | 223    | 407     | 8372  | 7513 |
|     | $1.96 \times 10^5$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     | 1          | 17715 | N/A  | 146    | 317     | 7666  | 600  |
|     |            | 18595 | N/A  | 162    | 365     | 8031  | 7530 |
|     | $2.70 \times 10^5$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     | 2          | 17799 | N/A  | 139    | 424     | 8542  | 610  |
|     |            | 18434 | N/A  | 138    | 492     | 8958  | 7731 |
|     | $1.34 \times 10^5$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     | 3          | 17886 | N/A  | 188    | 297     | 8274  | 592  |
|     |            | 19561 | N/A  | 143    | 445     | 9040  | 7831 |
|     | $2.10 \times 10^5$ | LLL | Deep | Deep-5 | Deep-10 | Pot | $S^2$ |
|     | 4          | 15726 | N/A  | 211    | 397     | 7592  | 540  |
|     |            | 17298 | N/A  | 217    | 450     | 7881  | 7028 |