Des. Codes Cryptogr.
DOI 10.1007/s10623-014-9957-1

LLL for ideal lattices: re-evaluation of the security of Gentry–Halevi's FHE scheme

Thomas Plantard · Willy Susilo · Zhenfei Zhang

Received: 28 May 2013 / Revised: 3 March 2014 / Accepted: 4 March 2014

Abstract The LLL algorithm, named after its inventors, Lenstra, Lenstra and Lovász, is one of the most popular lattice reduction algorithms in the literature. In this paper, we propose the first variant of LLL algorithm that is dedicated for ideal lattices, namely, the iLLL algorithm. Our iLLL algorithm takes advantage of the fact that within LLL procedures, previously reduced vectors can be re-used for further reductions. Using this method, we prove that the iLLL is at least as fast as the LLL algorithm, and it outputs a basis with the same quality. We also provide a heuristic approach that accelerates the re-use method. As a result, in practice, our algorithm can be approximately eight times faster than LLL algorithm for typical scenarios where lattice dimension is between 100 and 150. When applying our algorithm to the Gentry–Halevi's fully homomorphic challenges, we are able to solve the toy challenge within 24 days using a 2.66 GHz CPU, while with the classical LLL algorithm, it takes 32 days. Further, assuming a 4.0 GHz CPU, we predict to reduce the basis in 15.7 years for the small challenges, while previous best prediction was 45 years.

Keywords Lattice theory · Lattice reduction · LLL · Ideal lattice · Fully homomorphic encryption

Mathematics Subject Classification 94A60

Communicated by I. Shparlinski.

T. Plantard · W. Susilo (✉) · Z. Zhang
Centre for Computer and Information Security Research, School of Computer Science & Software Engineering (SCSSE), University of Wollongong, Wollongong, NSW, Australia
e-mail: wsusilo@uow.edu.au

T. Plantard
e-mail: thomaspl@uow.edu.au

Z. Zhang
e-mail: zz920@uow.edu.au

Published online: 23 March 2014

1 Introduction

A lattice $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^n$. It is usually represented by a set of integer linear combinations of vectors $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$, $\mathbf{b}_i \in \mathbb{R}^n$, $d \leq n$. $\mathbf{B}$ is a basis of $\mathcal{L}$, if $\mathbf{b}_i$-s are linearly independent, and d is called the dimension of the $\mathcal{L}$. For a given lattice $\mathcal{L}$, there exists an infinite number of bases for $d \geq 2$. Given a 'bad' basis (basis with large coefficients), to find a 'good' basis (bases with small coefficients and almost orthogonal) is known as "lattice reduction".

The concept of an ideal lattice was first introduced by Micciancio in [19]. For any vectors within the lattice, the cyclic rotation of the vector (over a certain polynomial ring $\mathbb{Z}[X]/F[X]$) is also in the lattice. Such a lattice enables important applications in modern cryptography such as constructing an asymptotic efficient digital signature scheme [18] or a fully homomorphic encryption scheme [8]. In those schemes, the public key is usually a bad basis, and to find a good basis through the bad basis will break the cryptosystem. The principal ideal lattice is a special ideal lattice that can be represented by 3 integers α, δ and d, where δ is the determinant of the lattice, α is a root and d is the dimension. This type of lattice with this kind of representation is usually adopted in practice, in order to improve the efficiency of the system [8,30].

The first polynomial time lattice reduction algorithm, known as LLL, was devised by Lenstra, Lenstra and Lovász [16] in 1982. For an arbitrary basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$, where all the Euclidean norms of $\mathbf{b}_i$ are bounded by 2^β, the LLL algorithm is guaranteed to terminate in $O(d^6\beta^3)$ time. In theory, the LLL algorithm is able to find vectors that are exponential approximations of the shortest non-zero vectors of the lattice.

The most popular variant of LLL-type reduction algorithms was that of Nguyen and Stehlé [24]. It was named L^2 since its worst case complexity $O(d^5\beta^2)$ was quadratic in β. Within L^2, there is a method named FP which makes uses of several heuristics as described in their paper. In practice, FP is considered as the most efficient implementation so far.

The first fully homomorphic encryption scheme was proposed by Gentry [7–9], using ideal lattices. To date, there exist three types of variants: schemes based on ideal lattices [11,30,31], schemes based on integers [5,33] and schemes based on (ring) learning with errors [2,3,12,13], among which, Gentry and Halevi's scheme [11] is one of the most efficient implementations. The authors also proposed four different challenges for their scheme. To the best of our knowledge, the best known attack against those challenges was devised by Chen and Nguyen [4,23].

Our contribution In this paper, we propose the first variant of LLL algorithm that is dedicated to ideal lattices. We obtain the following results:

- In theory, our algorithm shares the same worst-case bit complexity with the LLL algorithm. But our algorithm is at least as fast the corresponding LLL algorithm.
- On average cases, we reduce the complexity from $O((d^3\beta + d^2\beta^2)\mathcal{M}(d))$ (as in L^2) to $O((d^3\beta + d\beta^2)\mathcal{M}(d))$, where $\mathcal{M}(d)$ is the cost of integer multiplication with two d bit integers.
- In practice, our algorithm out-performs all known lattice reduction algorithms in terms of running time.
- In terms of the quality of the output basis, our algorithm produces an LLL reduced basis. Furthermore, in a vast majority of cases, our modification will not affect the output of the corresponding algorithm. With the same input, our iLLL algorithm will output the same result as the LLL algorithm in most tests.

– This leads us to a new result to the Gentry and Halevi fully homomorphic encryption challenge. We solve the toy challenge within 24 days, while we estimate to solve the small challenge in 16.7 years.

Paper organization In the next section, we will discuss the background for this paper. In Sect.3, we present our iLLL algorithm, and prove its correctness. In Sect.4, we compare the complexity of iLLL with the classical LLL algorithm. In Sect.5, we extend our technique over bases other than principal ideal lattice bases. In Sect.6, we present the implementation results. Finally, Sect.7 concludes the paper.

2 Background

2.1 Lattice theory

In this subsection, we review some concepts of lattice theory that will be used throughout this paper. The lattice theory, also known as the geometry of numbers, was introduced by Minkowski in 1896 [21]. We refer readers to [17,20] for a more complex account.

Definition 1 (*Lattice*) A lattice $\mathcal{L}$ is a discrete sub-group of $\mathbb{R}^n$, or equivalently the set of all the integral combinations of $d \leq n$ linearly independent vectors over $\mathbb{R}$.

$$\mathcal{L} = \mathbb{Z}\mathbf{b}_1 + \mathbb{Z}\mathbf{b}_2 + \cdots + \mathbb{Z}\mathbf{b}_d, \quad \mathbf{b}_i \in \mathbb{R}^n$$

$\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ is called a basis of $\mathcal{L}$ and d is the dimension of $\mathcal{L}$, denoted as $\dim(\mathcal{L})$. $\mathcal{L}$ is a full rank lattice if d equals n.

For a given lattice $\mathcal{L}$, there exists an infinite number of bases. However, its determinant, denoted by $\det(\mathcal{L}) = \sqrt{\det(\mathbf{B} \cdot \mathbf{B}^T)}$, is unique, where $\mathbf{B}^T$ is the transposition of $\mathbf{B}$.

For a lattice $\mathcal{L}$, the ith successive minima with respect to $\mathcal{L}$, denoted by λ_i, is the smallest real number, such that there exist i non-zero linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_i \in \mathcal{L}$ with

$$\|\mathbf{b}_1\|, \|\mathbf{b}_2\|, \ldots, \|\mathbf{b}_i\| \leq \lambda_i,$$

where $\|\cdot\|$ denotes the Euclidean norm of the corresponding vector. In addition, if the lattice is random, then the value of ith minima is estimated by:

$$\lambda_i(\mathcal{L}) \sim \sqrt{\frac{d}{2\pi e}} \det(\mathcal{L})^{\frac{1}{d}}. \tag{1}$$

We note that the minimas are independent from i.

For any vector $\mathbf{v}$, denote $v(x)$ the polynomial form of $\mathbf{v}$. Let $Rot(\mathbf{v}, i)$ be $x^i v(x) \mod f(x)$. Then $\{Rot(\mathbf{v}, i)\}\ 1 \leq i \leq d$ forms a rotation basis. For instance, if $f(x) = x^n + 1$ and $\mathbf{v} = (v_1, v_2, \ldots, v_d)$, then the rotation basis ($\mathbf{B}$) is of the following form:

$$\mathbf{B} = \begin{pmatrix} v_1 & v_2 & v_3 & \ldots & v_d \\ -v_d & v_1 & v_2 & \ldots & v_{d-1} \\ -v_{d-1} & -v_n & v_1 & \ldots & v_{d-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -v_2 & -v_3 & -v_4 & \ldots & v_1 \end{pmatrix}$$

Definition 2 (*Ideal lattice* [19]) Let R be a polynomial ring $\mathbb{Z}[X]/f(x)$, where $f(x) \in \mathbb{Z}[X]$ is a monic irreducible polynomial of degree n. Let $\mathbf{v} \in \mathbb{Z}^n$. The ideal lattice over R with respect to $\mathbf{v}$, denoted by $\mathcal{L}(Rot(\mathbf{v}, f))$ is the set of all integer linear combinations of $\mathbf{v}$ and its rotation vectors.

Note that for ideal lattices, we have $n = d$ for all bases. Since we are dealing with ideal lattices throughout this paper, we have $n = d$.

For a principal ideal lattice, its Hermite normal form (HNF) basis (**H**) is in the following form. Such a basis can be represented by three integers $\{\alpha, \delta, d\}$, where δ is the determinant of the lattice, and α is the root of unity over the ring. From the point of view of cryptography, one sometimes also uses $\mathbf{H}'$, which shares a similar property with **H**. We note that the bit complexity to reduce those two bases is the same.

$$\mathbf{H} = \begin{pmatrix} \delta & 0\ 0 \dots 0\ 0 \\ (-\alpha) & 1\ 0 \dots 0\ 0 \\ (-\alpha)^2 \mod \delta & 1 \dots 0\ 0 \\ \vdots & \vdots\ \vdots\ \ddots\ \vdots\ \vdots \\ (-\alpha)^{d-1} \mod \delta & 0\ 0 \dots 0\ 1 \end{pmatrix}, \quad \mathbf{H}' = \begin{pmatrix} \delta & 0 & 0 \dots & 0 & 0 \\ -\alpha & 1 & 0 \dots & 0 & 0 \\ 0 & -\alpha & 1 \dots & 0 & 0 \\ \vdots & \vdots & \vdots\ \ddots & \vdots & \vdots \\ 0 & 0 & 0 \dots & -\alpha & 1 \end{pmatrix}.$$

In this paper, we also deal with lattice bases with other special forms, such as cyclic bases formed by the bottom several rows of **H**, or ideal lattices where the top several elements on the diagonal of their HNF bases are not 1. We note that one can still use our technique when the vectors whose last non-zero element is 1 start to appear. For simplicity, we use principal ideal lattices to demonstrate our technique in the rest of the paper, and we assume that the principal ideal lattices are given in the form of HNF bases.

Definition 3 (*Hermite factor*) Let $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$ a basis of $\mathcal{L}$. The Hermite factor with respect to **B** is defined as $\frac{\|\mathbf{b}_1\|}{\det(\mathcal{L})^{\frac{1}{d}}}$.

Note that the Hermite factor indicates the quality of a reduced basis.

Definition 4 (*Gram–Schmidt orthogonalization*) Let $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$ be a basis of $\mathcal{L}$. The Gram–Schmidt orthogonalization (GSO) of **B** is the following basis $\mathbf{B}^* = (\mathbf{b}_1^*, \dots, \mathbf{b}_d^*)$:

$$\begin{aligned} \mathbf{b}_1^* &= \mathbf{b}_1, \\ \mathbf{b}_i^* &= \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*, \qquad (2 \le i \le d), \\ \mu_{i,j} &= \frac{\mathbf{b}_i \cdot \mathbf{b}_j^*}{\mathbf{b}_j^* \cdot \mathbf{b}_j^*}. \end{aligned}$$

Definition 5 (*Gram determinants*)] Let $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$ be a basis of $\mathcal{L}$. Let $\mathbf{B}^* = (\mathbf{b}_1^*, \dots, \mathbf{b}_d^*)$ be the corresponding GSO. The Gram determinants of **B**, noted $\{\Delta_1^*, \dots, \Delta_d^*\}$ is defined as:

$$\Delta_i^* = \det(\mathbf{b}_1^*, \dots, \mathbf{b}_i^*).$$

The product of all Gram determinants is defined as: $D = \prod_{i=1}^{d-1} \Delta_i^*$. For any basis, D is upper-bounded by $2^{\beta d(d-1)}$, while for the HNF bases of principal ideal lattices, D is further bounded by $2^{\beta(d-1)}$.

2.2 The LLL algorithm

In this subsection, we give a general overview of the LLL algorithm. We refer the reader to the book by Nguyen and Vallée [26] for a more complex account.

The LLL algorithm is described in Algorithm 1 and 2. Algorithm 1 is for size reduction, and it is sometimes referred to as the Gram–Schmidt reduction. Algorithm 2 outputs a (δ, η)-reduced basis. The basis is also referred to as the LLL-reduced basis.

Definition 6 (*η-size reduced*) Let $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ be a basis of $\mathcal{L}$. $\mathbf{B}$ is η-size reduced, if $|\mu_{i,j}| \leq \eta$ for $1 \leq j < i \leq d$. $\eta \geq 0.5$ is the reduction parameter.

Definition 7 (*(δ, η)-reduced basis*) Let $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ be a basis of $\mathcal{L}$. $\mathbf{B}$ is (δ, η)-reduced, if the basis is η-size reduced and it satisfies Lovász condition as follows: $\delta \|\mathbf{b}_{i-1}^*\|^2 \leq \|\mathbf{b}_i^*\|^2 + \mu_{i,i-1}^2 \|\mathbf{b}_{i-1}^*\|^2$ for $2 \leq i \leq d$. $\frac{1}{4} < \delta \leq 1$ and $\frac{1}{2} \leq \eta < \sqrt{\delta}$ are two reduction parameters.

It is quite straightforward to see that if Algorithm 2 terminates, then its output basis satisfies the Lovász condition. Hence, the basis is (δ, η)-reduced. Note that in the classical LLL, $\eta = 0.5$, while in L^2 (see next subsection), it is essential that η is slightly greater than 0.5.

With regard to the running time of the algorithm, it has been shown that the loop invariant D is unchanged except during the exchange procedure, while during the exchange, D is decreased by a factor of δ. Hence, the total number of exchanges is upper bounded by $\left\lfloor \frac{\beta d(d-1)}{\log_2 \delta} \right\rfloor$, which implies there are maximum $O(d^2\beta)$ loop iterations. In addition, since the size reduction algorithm requires $O(d^2)$ operations, the total number of operations is $O(d^4\beta)$. Finally, each operation involves integer multiplications with a cost of $\mathcal{M}(d\beta)$ due to rational arithmetics.[1] Hence, the original LLL algorithm terminates in polynomial time $O(d^6\beta^3)$.

Algorithm 1 Size Reduction

Input: $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d)$, its GSO, an index κ and a reduction parameter η.
Output: A new basis $\mathbf{B}$, where $\mathbf{b}_\kappa$ is size reduced, and the updated GSO.

```
for i = (κ − 1) → 1 do
  if μ ≤ η then
    b_κ ← b_κ − ⌈μ_{κ,i}⌋ · b_i;
    Update GSO;
  end if
end for
return B.
```

[1] In this paper, we follow the LLL algorithm by using $\mathcal{M}(d)$ to be $O(d^2)$ assuming a naive integer multiplication, although one can replace it with $O(d^{1+\varepsilon})$ to obtain the exact bit complexity using fast integer arithmetics as indicated in Table 1.

Algorithm 2 LLL

Input: $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d)$ and reduction parameters (δ, η).
Output: A (δ, η)-reduced basis $\mathbf{B}$.

```
Compute GSO;
κ ← 2;
while κ ≤ d do
  size reduce (B, κ, η);
  if δ‖b*_{κ−1}‖² ≤ ‖b*_κ‖² + μ²_{κ,κ−1}‖b*_{κ−1}‖² (Lovász condition) then
    κ ← κ + 1;
  else
    Exchange b_κ and b_{κ−1};
    κ ← max(κ − 1, 2);
    Update GSO;
  end if
end while
return B.
```

With regard to the quality of a reduced basis for an arbitrary lattice, the following theorem provides an upper bound.

Theorem 1 *For a lattice $\mathcal{L}$, if $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ form an LLL-reduced basis of $\mathcal{L}$, then,*

$$\forall i, \quad \|\mathbf{b}_i\| \leq \left(\frac{1}{\delta - \eta^2}\right)^{\frac{d-1}{2}} \lambda_i(\mathcal{L}). \tag{2}$$

Hence, if $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ forms an (δ, η)-reduced basis, then $\|\mathbf{b}_i\| \leq 2^d \det(\mathcal{L})^{\frac{1}{d}}$ for $1 \leq i \leq d$.

2.3 LLL variants

In this subsection, we list some of the LLL-type algorithms that improve complexity with regard to β. For other improvements with respect to d, we refer readers to [15,22,29].

In 2005, Nguyen and Stehlé [24] proposed an improvement to the LLL, which is the first variant whose worst-case time complexity is quadratic with respect to β. This algorithm is therefore named L^2. We briefly recall the difference between L^2 and classical LLL. L^2 uses floating point arithmetics where the multiplication can be carried out with precision $O(d)$. Hence, it reduces the cost of integer multiplication from $\mathcal{M}(d\beta)$ to $\mathcal{M}(d)$. However, to assure correctness of the floating point, the L^2 uses a 'lazy' size reduction algorithm which incurs a cost of $O(d^2 + d\beta)$, compared to $O(d^2)$ as in LLL. Finally, both algorithms use maximum $O(d^2\beta)$ loop iterations. Therefore, it terminates with a worst-case time complexity of $O(d^5\beta^2 + d^6\beta)$ for any basis with naive multiplication.

As for a principal ideal lattice basis, it is proved that L^2 terminates in $O(d^4\beta^2 + d^5\beta)$, since there are $O(d\beta)$ loop iterations for these bases instead of $O(d^2\beta)$ for bases of random lattices (see Remark 3, [24]).

The FP is the fastest variant of L^2 in practice. It uses some heuristics, such as multi-levels of floating-point precisions. The classical L^2 uses a fixed precision $l = O(d)$, while FP will try some small precisions (for instance, C int length, which allows fast computation) first, before it finally uses l. This procedure with small precision is called "early reduction". Since the complexity depends largely on the precision, the early reductions are in general very fast. Meanwhile, they produce a basis that is already reduced to some extent. So the cost of final reduction with precision $= l$ is greatly reduced.

In [34], van Hoeij and Novocin proposed a gradual sub-lattice reduction algorithm based on LLL that deals with knapsack-type bases. Unlike other LLL-type reduction algorithms, it only produces a basis of a sub-lattice. This algorithm uses a worst-case $O(d^7 + d^3\beta^2)$ time complexity.

In 2011, Novocin, Stehlé and Villard [27] proposed a new improved LLL-type algorithm that is quasi-linear in β. This led to the name $\tilde{\mathrm{L}}^1$. It is guaranteed to terminate in time $O(d^6\beta + d^{\omega+2}\beta^2)$ for any basis, where ω is a valid exponent from matrix multiplications. To bound ω, we have $2 < \omega \leq 3$. A typical setting in [27] is $\omega = 2.3$.

In [28], Plantard et al. proposed a recursive reduction that can be applied to ideal lattice. Their algorithm is proved to finish in $O(d^3\beta^2+d^5\beta)$ for principal ideal lattice bases. However, we note that this work assumes a uniform distribution, hence, did not provided a worst-case complexity.

In this paper, we compare our algorithm primarily with L^2 and its fastest implementation, FP, since those two are the best in theory and in practice.

2.4 Fully homomorphic encryption challenges

In the fully homomorphic encryption challenges [10], the authors published four sets of public keys with respect to different parameters (toy, small, medium and large). To solve the challenge, one needs to recover some short vectors from the public keys. In toy, small and medium challenges, this can be achieved by performing an LLL reduction over the principal ideal lattice bases. Since in those lattices, the corresponding Hermite factor is much greater than the upper bound of the LLL algorithm. Therefore, performing an LLL reduction will solve the challenge. As a result, the remaining problem is the running time of the LLL algorithm. The previous best results/prediction for the running time of LLL can be found in Table 2.

3 LLL for ideal lattices

Ideal lattice maintains this special property: if a vector $\mathbf{v} \in \mathcal{L}$, then all its rotation vectors over R exist in this lattice as well. This useful property can accelerate the reduction. The LLL algorithm uses a stepping method. At the κth step, the first κ vectors in the basis are involved. For a certain step κ, the top $\kappa - 1$ vectors are (δ, η)-reduced. So it will first size-reduce $\mathbf{b}_\kappa$ with $(\mathbf{b}'_1, \ldots, \mathbf{b}'_{\kappa-1})$, where $(\mathbf{b}'_1, \ldots, \mathbf{b}'_{\kappa-1})$ denotes the reduced basis of $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$, and then perform the LLL reduction on the whole κ vectors.

However, instead of size-reducing the input vector $\mathbf{b}_\kappa$ whose bit-length is in β, one can use a rotation of a previous vector, providing that this new vector, denoted as $\mathbf{v}$, together with $\{\mathbf{b}_i\}$, $1 \leq i \leq d, i \neq \kappa$ also form a basis of $\mathcal{L}$. This technique is named "re-use".

Recall that in the $\kappa - 1$ step, one has already performed a size reduction on $\mathbf{b}_{\kappa-1}$ with $(\mathbf{b}'_1, \ldots, \mathbf{b}'_{\kappa-2})$. Denote $\mathbf{v}'$ the reduced vector. Then one can simply shift $\mathbf{v}'$ to the right to obtain $\mathbf{v}$. $(\mathbf{b}'_1, \ldots, \mathbf{b}'_{\kappa-1}, \mathbf{v})$ also form a basis of $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_\kappa)$. Moreover, since $\mathbf{v}$ is already size-reduced to some extent, it will often be shorter than $\mathbf{b}_\kappa$, and as a result, re-use will often accelerate the reduction.

To use the re-use technique recursively, we obtain iLLL algorithm. In the following, we first show our iLLL algorithm. Then we prove that the algorithm is correct.

3.1 The algorithm

The iLLL algorithm is described in Algorithm 3. We note that the only difference between Algorithm 3 and the LLL algorithm is the re-use technique. In Algorithm 3, κ indicates the current vector that iLLL is working on. κ_1 indicates if the current size-reduced vector should be re-used later.

Algorithm 3 The iLLL Algorithm

Input: The HNF basis $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d)$ of an ideal lattice and reduction parameters (δ, η)
Output: An (δ, η)-reduced basis $\mathbf{B}$.

```
Compute GSO.
κ ← 2, κ1 ← 2.
while κ ≤ d do
  Size reduce (B, κ, η);
  if κ = κ1 and κ < d then
    v ← Right shift (b_κ);
    if ‖v‖ < ‖b_{κ+1}‖ then
      b_{κ+1} ← v
    end if
    κ1 ← κ1 + 1;
  end if
  if δ‖b*_{κ−1}‖² ≤ ‖b*_κ‖² + μ²_{κ,κ−1}‖b*_{κ−1}‖² then
    κ ← κ + 1;
  else
    Exchange b_κ and b_{κ−1};
    κ ← max(κ − 1, 2);
    Update GSO;
  end if
end while
return B.
```

3.2 Correctness

To start with, for each step, we have the following Lemma:

Lemma 1 *Let* $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$. *Let* $\mathbf{B}_{i-1} = (\mathbf{b}'_1, \ldots, \mathbf{b}'_{i-1})$ *where* $(\mathbf{b}'_1, \ldots, \mathbf{b}'_{i-1})$ *forms a* (δ, η)*-reduced basis of* $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$. *Let* $\mathbf{v} \in \mathcal{L}$ *where the* i*th coefficient of* $\mathbf{v}$ *is* 1. *Then* $\mathbf{B}' = (\mathbf{b}'_1, \ldots, \mathbf{b}'_{i-1}, \mathbf{v}, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_d)$ *form a basis of* $\mathcal{L}(\mathbf{B})$.

Proof Firstly, all row vectors of $\mathbf{B}'$ can be obtained by linear operations of row vectors of $\mathbf{B}$. Meanwhile, all row vectors of $\mathbf{B}'$ are linearly independent. $\mathbf{b}'_1, \ldots, \mathbf{b}'_{i-1}, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_d$ are linear independent since the top $i-1$ vectors are LLL reduced form $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. Also, $\mathbf{v}$ is independent with $\mathbf{b}'_1, \ldots, \mathbf{b}'_{i-1}, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_d$ since the ith element of all row vectors of $\mathbf{B}'$ is 0 except for $\mathbf{v}$. Hence, $\mathbf{B}'$ is a basis of $\mathcal{L}(\mathbf{B})$. □

Then we prove the correctness in Theorem 2.

Theorem 2 *For an input basis* $\mathbf{B}$*, our iLLL algorithm outputs an LLL-reduced basis of* $\mathcal{L}(B)$.

Proof For the quality of the basis, it is quite straightforward that our algorithm produces an LLL-reduced basis, since it checks Lovász condition at the last iteration. Furthermore, we have shown that our algorithm produces a basis that spans the same lattice as the input basis in Lemma 1, since the shifted vector is the only one whose $(\kappa+1)$th coefficient is non-zero. Hence, our iLLL algorithm outputs an LLL-reduced basis of $\mathcal{L}(B)$. □

4 Analysis

Our algorithm shares the same worst-case complexity with the LLL algorithm, however, we show that our algorithm is in theory always faster than LLL algorithm. Further, on average cases, we obtain a complexity of $O(d^5\beta + d^3\beta^2)$.

4.1 Comparison with LLL and worst-case complexity

Recall that the LLL algorithm uses a stepping method. For the κth step ($\kappa > 2$), the basis is of the following form:

$$\mathbf{B}_{\kappa,LLL} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,\kappa} & 0\,0\dots 0 \\ x_{2,1} & x_{2,2} & \dots & x_{2,\kappa} & 0\,0\dots 0 \\ \vdots & \vdots & \dots & \vdots & \vdots\,0\dots\vdots \\ x_{\kappa-1,1} & x_{\kappa-1,2} & \dots & x_{\kappa-1,\kappa} & 0\,0\dots 0 \\ x_{\kappa,1} & x_{\kappa,2} & \dots & x_{\kappa,\kappa} & 0\,0\dots 0 \\ X_{\kappa+1} & 0 & \dots & 0 & 1\,0\dots 0 \\ X_{\kappa+2} & 0 & \dots & 0 & 0\,1\dots 0 \\ \vdots & \vdots & \dots & \vdots & \vdots\,\vdots\dots\vdots \\ X_d & 0 & \dots & 0 & 0\,0\dots 1 \end{pmatrix}$$

where the top κ vectors are LLL-reduced. Then, for the next step, since the top κ vectors are already reduced, there will be no exchange initially. The LLL will directly size-reduce $\mathbf{b}_{\kappa+1}$, and then operate on $(\mathbf{b}_1, \dots, \mathbf{b}_{\kappa+1})$.

$$\mathbf{B}_{\kappa,iLLL} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,\kappa} & 0\,0\dots 0 \\ x_{2,1} & x_{2,2} & \dots & x_{2,\kappa} & 0\,0\dots 0 \\ \vdots & \vdots & \dots & \vdots & \vdots\,\vdots\dots\vdots \\ x_{\kappa-1,1} & x_{\kappa-1,2} & \dots & x_{\kappa-1,\kappa} & 0\,0\dots 0 \\ x_{\kappa,1} & x_{\kappa,2} & \dots & x_{\kappa,\kappa} & 0\,0\dots 0 \\ 0 & v_1 & \dots & v_{\kappa-1} & 1\,0\dots 0 \\ X_{\kappa+2} & 0 & \dots & 0 & 0\,1\dots 0 \\ \vdots & \vdots & \dots & \vdots & \vdots\,\vdots\dots\vdots \\ X_d & 0 & \dots & 0 & 0\,0\dots 1 \end{pmatrix}$$

By comparison, the only change we have made is to replace $\mathbf{b}_{\kappa+1}$ with $\mathbf{v}$ as shown in $\mathbf{B}_{\kappa,iLLL}$. This modification indeed accelerates the size-reduction, since $\mathbf{v}$ is in general significantly shorter than $\mathbf{b}_{\kappa+1}$. Moreover, since our algorithm does not work on large coefficients (i.e. $X_{\kappa+1}$), it requires less precision of floating-point to size-reduce.

To sum up, in theory, we proved that the iLLL will always be faster than the LLL algorithm due to the fact that $\mathbf{v}$ is not longer than $\mathbf{b}_{\kappa+1}$. However, in worst cases, it is possible that $\|\mathbf{v}\| \sim \|\mathbf{b}_{\kappa+1}\|$ if all κ vectors are not well reduced. In this case, our algorithm shares the same worst-case complexity as the LLL algorithm. It is also worth pointing out that we can never be more costly than LLL, since if $\|\mathbf{v}\| > \|\mathbf{b}_{\kappa+1}\|$, one simply does not adopt the re-use, and we obtain an exact LLL algorithm.

4.2 Average-case complexity

Now we analyze the average case complexity of our algorithm. We assume a uniform distribution of α over $(0, \delta)$, then the lattice is, therefore, a random ideal lattice as in [14]. If a lattice is random, then its minima λ_i follow Eq. 1. We also assume that random ideal lattices share same properties with (normal) random lattices in terms of the minimas. Hence, Eq. 1 will hold for our case under this assumption.

For the κth step ($\kappa > 2$), the basis our algorithm is shown as in the last subsection, where $\|\mathbf{b}_i\| \sim 2^{\frac{\kappa-1}{2}} 2^{\frac{\beta}{\kappa-1}}$ for $i < \kappa$ and $\|\mathbf{b}_\kappa\| \sim 2^{\frac{\kappa-2}{2}} 2^{\frac{\beta}{\kappa-2}}$. The loop invariant for current step D_κ is then bounded by $\prod_{i=1}^{\kappa} \|\mathbf{b}_i\|^{2(\kappa-i+1)} = 2^{\kappa(\kappa-1)^2-1} 2^{2\beta\kappa+\frac{\beta}{(\kappa-1)(\kappa-2)}}$. When the κth step terminates, $\mathbf{b}_i$ will be reduced to $2^{\frac{\kappa}{2}} 2^{\frac{\beta}{\kappa}}$ for $i \leq \kappa$. Hence, one obtains $O(\beta)$ loop iterations on average cases for each step. We note that this observation is quite natural, since there are $O(d\beta)$ loop iterations in total, hence, on average there are $O(\beta)$ loop iterations for each κ.

1. For the κth step, the κth step terminated in $O(\beta(\kappa^2 + \beta)\mathcal{M}(\kappa))$ operations:
   (a) There are maximum $O(\beta)$ loop iterations.
   (b) For each loop iteration, there is maximum $O(1 + \frac{\beta}{\kappa(\kappa-1)})$ iterations within the size reduction.
   (c) In each size reduction, there are $O(\kappa^2)$ arithmetic operations.
   (d) The cost of arithmetic operations is determined by integer multiplications with bit length $O(\kappa)$.
2. Assuming a naive integer multiplication, one obtains $O(\sum_{\kappa=2}^{d}(\beta\kappa^4+\beta^2\kappa^2)) = O(d^5\beta+d^3\beta^2)$.

Table 1 shows a comparison of time complexity between iLLL and some of LLL-type algorithms using fast multiplications.

5 Extensions

5.1 Heuristics

The main improvement in our provable algorithm is to replace $\mathbf{b}_{\kappa+1}$ with a vector from previous reductions through the size reduction algorithm. In fact, any vector in the lattice can be used as the replacement, as long as the last non-zero element of this vector is 1.

Table 1 Comparison of time complexity

Algorithms	Time complexity
LLL [16]	$O(d^{5+\varepsilon}\beta^{2+\varepsilon})$
LLL for ideal lattice	$O(d^{4+\varepsilon}\beta^{2+\varepsilon})$
L^2 [24]	$O(d^{4+\varepsilon}\beta^2 + d^{5+\varepsilon}\beta)$
L^2 for ideal lattice [24]	$O(d^{3+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$
$\tilde{L}^1$ [27]	$O(d^{\omega+1+\varepsilon}\beta^{1+\varepsilon} + d^{5+\varepsilon}\beta)$
Rec-L^2 [28] (average case)	$O(d^{2+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$
iLLL (average case)	$O(d^{2+\varepsilon}\beta^2 + d^{4+\varepsilon}\beta)$

The remaining issue is to find $\mathbf{v}$ more efficiently than size-reduce $\mathbf{b}_{\kappa+1}$ with $(\mathbf{b}_1, \ldots, \mathbf{b}_\kappa)$. Algorithm 4 describes a probabilistic yet very efficient method to find $\mathbf{v}$.

Algorithm 4 Heuristic Re-use

Input: $(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_\kappa)$ a (δ, η)-reduced basis
Output: $\mathbf{v}$ that can be used for the $\kappa + 1$ step of iLLL.

```
if The last non-zero coefficient of b_1 = 1 then
  Shift b_1 such that the last coefficient is 1.
  v ← b_1
else
  for i = 1 → κ do
    Shift b_i such that the last coefficient is non-zero.
  end for
  v ← zero vector
  for i = 1 → κ do
    for j = i + 1 → κ do
      Find x, y and z such that x = y b_{i,κ} + z b_{j,κ} {The XGCD algorithm.}
      if x = 1 then
        v' ← y b_i + z b_j.
        if v ≠ 0 and ‖v‖ ≥ ‖v'‖ then
          v ← v'
        end if
      end if
    end for
  end for
end if
return v.
```

This algorithm first checks if we can directly use the first vector from the previous step with simple shifting. It requires the last non-zero element of $\mathbf{b}_1$ to be 1. The successful rate is high when β is small and κ is big.

If it fails, then it checks if one can construct a vector by linear combination of two vectors. This can be done using the extended GCD algorithm. In the re-use technique, it finds all possible vectors where the last non-zero coefficient is 1, and return the shortest one.

This procedure takes $O(\kappa^2 \mathcal{M}(\beta))$ on worst-cases reductions and $O(\kappa^2 \mathcal{M}(\beta/\kappa))$ on average case reductions. We note it is negligible in terms of time complexity, compared with the cost of each iteration. In practice, it can be done in less than 1 s for dimensions as large as 500. Hence, it finds a vector much faster than using size-reduction algorithm. However, we note that if one uses this optimization, when the first vector does not qualify, it is possible that the returned vector will be slightly longer than the one from size-reduction. Hence, it will make the next step a bit more costly. We also note that this technique is heuristic, since sometimes it is possible that one cannot find a suitable vector. Nonetheless, our practical tests show that this method is in general faster than iLLL.

5.2 Re-usable basis

In fact, our algorithm can also deal with bases other than principal ideal bases. For instance, general ideal lattices or Coppersmith–Shamir bases, which are used to attack the NTRU encryption scheme.

It is true that for some ideal lattice HNF bases, the diagonal coefficients do not follow the same form as a principal ideal lattice. The first several coefficients on the diagonal are not 1.

We can adopt our technique when 1 starts to appear. This should appear very soon, since the diagonal coefficients are decreasing rapidly to 1 with the increase in dimension.

$$\mathbf{B}_{CS} = \left(\begin{array}{cccc|cccc} q & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & q & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & q & 0 & 0 & \dots & 0 \\ \hline h_0 & h_1 & \dots & h_{N-1} & 1 & 0 & \dots & 0 \\ h_{N-1} & h_0 & \dots & h_{N-2} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ h_1 & h_2 & \dots & h_0 & 0 & 0 & \dots & 1 \end{array}\right)$$

The Coppersmith–Shamir basis are of the above form. It is not a principal ideal lattice basis, however, its bottom part does allow one to apply our re-use technique, since the right non-zero coefficient is 1 and the left part of the basis is a rotated basis.

To this end, we formally define a reusable basis as a basis where our re-use technique can be applied.

Definition 8 (*i-Reusable basis*) $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_d)$ be a basis of $\mathcal{L}$. Let $i < d$ an integer. $\mathbf{B}$ is a reusable basis if $\forall \kappa > i$, there exist vectors $\mathbf{v}, \mathbf{v}' \in \mathbb{Z}^n$ and a permutation matrix $\mathbf{P} \in \mathbb{Z}^{n \times n}$ with regard to the following:

- $\mathbf{v} = \mathbf{v}'\mathbf{P}$;
- $\mathbf{v}'$ is a linear combination of $(\mathbf{b}_1, \dots, \mathbf{b}_{\kappa-1})$
- $\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_{\kappa-1}, \mathbf{v}) = \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_{\kappa})$.

It is quite straightforward to see that all ideal lattice HNF bases are reusable by simple shifting, while the Coppersmith–Shamir basis is an n-reusable basis with certain permutation. Hence, our iLLL is also applicable to these bases.

6 Implementation result and practice analysis

In this section, we show some implementation results. The implementation was conducted with MAGMA [1] on Xeon E5640 CPUs @ 2.66 GHz. The memory was always sufficient since the algorithm only requires a polynomial space. We first show the average behavior of our algorithm by comparing us with L^2 and FP on bases of random ideal lattice. Subsequently, we apply iLLL to Gentry–Halevi's fully homomorphic encryption challenge, and present the results. Finally we summarize our advantage in practice.

6.1 Test results

We tested our algorithm with bases of random ideal lattice over three scenarios: $\beta \sim 10d$, $\beta \sim 20d$ and $\beta \sim 380d$. The first scenario for $\beta \sim 10d$ is the classical setting for the SVP challenges [32], and the last one $\beta \sim 380d$ is the requirement for Gentry–Halevi's fully homomorphic encryption scheme. For comparison, we also tested $\beta \sim 20d$ to observe the difference. To ensure the randomness of the ideal lattice, we require α to be a uniform distribution between 1 and γ.

For each dimension of each test, we generated 10 bases with 10 different seeds. Then we present the average time to reduce the bases using both L^2 and FP methods.

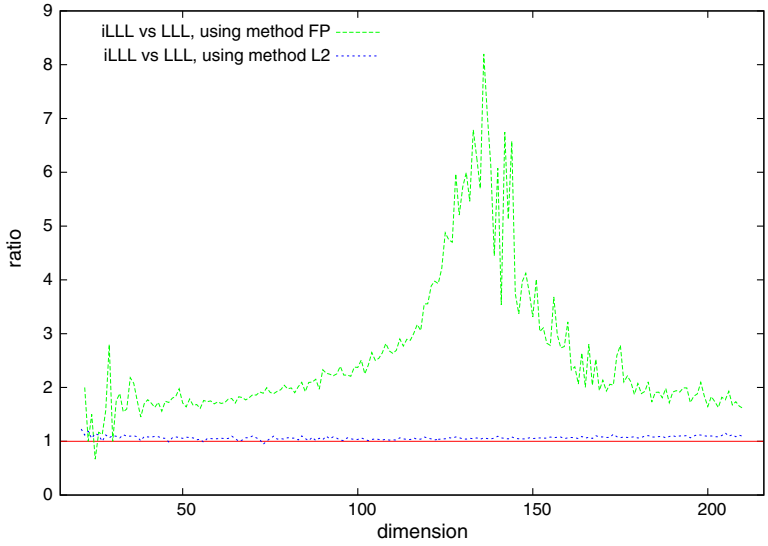

Fig. 1 Testing results: $\beta = 10d$

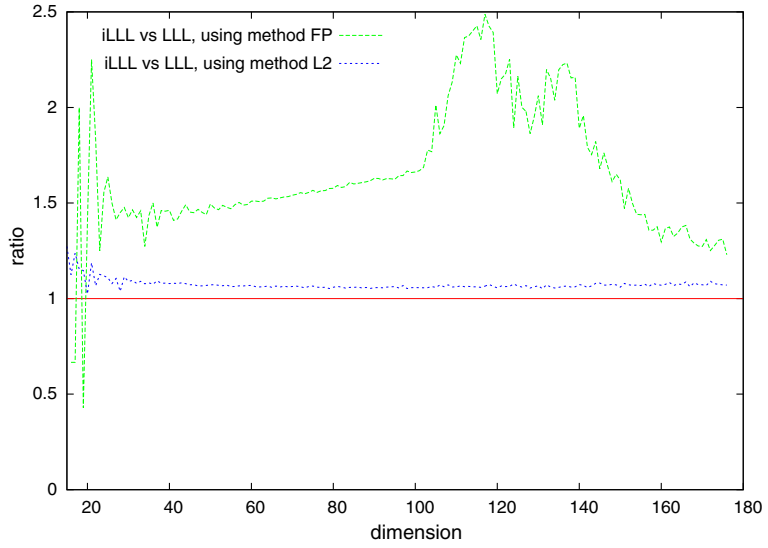

Fig. 2 Testing results: $\beta = 20d$

Figures 1, 2 and 3 show the advantage for each scenario. The ratio is computed from the running time of LLL divide by the running time of iLLL. As observed, the curves are always above one, which implies that iLLL is always faster than the counterpart. We observe a small advantage for L^2 as we expected, and the advantage is stable for all three scenarios, while it grows with the increase of β/d. With $\beta \sim 380d$, iLLL is 20 % faster. The time difference in these cases is due to the re-use technique. In another words, our improvement is due to the unnecessary size reduction in LLL. As the increase of β/d, the length of the reusable vector increases, which results in an increasing advantage.

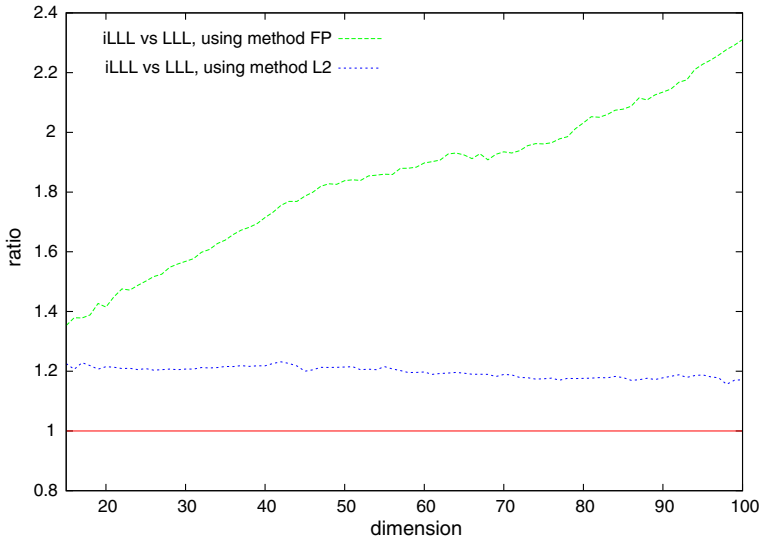

Fig. 3 Testing results: $\beta = 380d$

As for the FP method, apart from the starting point and dimension between 100 to 150, iLLL can be approximately twice faster than LLL. When $100 < d < 150$, we enjoy a massive advantage. We do not present the result for $\beta \sim 380d$ in those dimensions, instead we refer the reader to Fig. 7, which is essentially using the same setting. In some cases, when $d \sim 150$ and $\beta \sim 10d$, iLLL can be as much as eight times faster than LLL. This is due to the floating-point setting in FP. We shall discuss this in Sect. 6.3.

Interestingly, for small dimensions $d < 80$, iLLL and LLL always produce the same basis in these tests,, while as the dimension grows, some differences may appear. Nevertheless, in most cases (>90 %), the same basis is produced. We note that this phenomenon is quite natural, since for a certain step, if we assume the size-reduced vector is indeed a random vector, then two algorithms will produce same result with a probability of $\left(1 - 2\left(\frac{\eta-0.5}{\eta}\right)^2\right)^{\frac{(d-1)d}{2}}$. This probability is obtained as follows:

- Let $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1}, \mathbf{b}_\kappa)$ and $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1}, \mathbf{b}'_\kappa)$ be two (δ, η)-reduced bases of the same lattice $\mathcal{L}$, where $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$ are also reduced under the same parameters.
- Let $\mathbf{v} \leftarrow \mathbf{b}_d - \sum_{i=1}^{\kappa-1} \lfloor \mu_{i,\kappa} \rceil \cdot \mathbf{b}_{i,\kappa}$ and $\mathbf{v}' \leftarrow \mathbf{b}'_\kappa - \sum_{i=1}^{\kappa-1} \lfloor \mu'_{i,\kappa} \rceil \cdot \mathbf{b}_{i,\kappa}$, where $\mu_{i,\kappa}$ and $\mu'_{i,\kappa}$ are GSO for the respective basis.
- Since both basis are (δ, η)-reduced, we have $\|\mu_{i,\kappa}\| \leq \eta$ and $\|\mu'_{i,\kappa}\| \leq \eta$. Hence, if $\mathbf{b}_\kappa$ and $\mathbf{b}'_\kappa$ are random vectors, the probability of $\|\mu_{i,\kappa} + \mu'_{i,\kappa}\| < 1$ for a single pair is $1 - 2\left(\frac{\eta-0.5}{\eta}\right)^2$.
- We need the condition holds for $\kappa - 1$ pairs, therefore, if $\mathbf{b}_\kappa$ and $\mathbf{b}'_\kappa$ are random vectors, then, $\mathbf{b}_\kappa = \mathbf{b}'_\kappa$ with a probability of $\left(1 - 2\left(\frac{\eta-0.5}{\eta}\right)^2\right)^{\kappa-1}$.
- If the above result holds for each step, then one obtains

$$\left(1 - 2\left(\frac{\eta - 0.5}{\eta}\right)^2\right)^{\sum_{i=2}^{d} i-1} = \left(1 - 2\left(\frac{\eta - 0.5}{\eta}\right)^2\right)^{\frac{(d-1)d}{2}}.$$

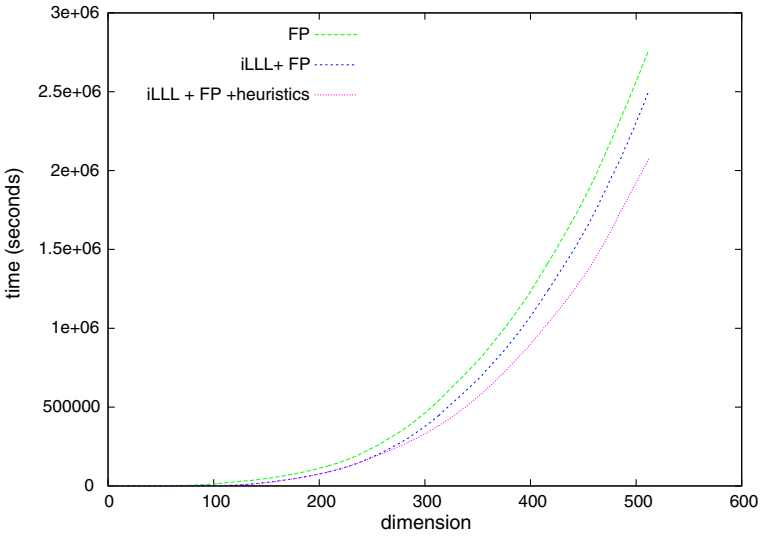

Fig. 4 Toy challenge

Remark 1 In practice, we observe that in most cases our algorithm returns the same basis as LLL. We believe that this is due to the fact that the reduced vectors in an LLL procedure are very close to random vectors. Hence, since $\eta = 0.501$ by default in MAGMA [1] we achieve a high rate, i.e., for a lattice with dimension 200, we obtain the same basis in over $>85\,\%$ of the tests. We also note that for some extreme cases, where the reduced vector has some special distribution, this observation may not be valid.

6.2 Challenge results

We tested iLLL on the Gentry–Halevi's fully homomorphic encryption challenge for dimension 512 and dimension 2048. For the dimension 512, the lattice basis was obtained from the challenge website [10]. For the dimension 2048, we generated the basis as per Gentry–Halevi's paper [11], since the basis was not available from the website. Figures 4 and 5 show the result for dimension 512 and dimension 2048, respectively. We also include the test results for our heuristic method, which accelerates the reduction a bit further (Fig. 6).

As observed from Fig. 4, iLLL is faster than LLL in practice for the small challenges. The classical FP finishes in 32 days. In comparison, with iLLL we are able to finish within 28 days. In addition, our heuristics accelerate a bit further to 24 days. Overall, we are around 33 % faster than the classical FP algorithm.

As for the 2048 challenge, within 6 months, we are able to reach dimension 559 and 560 for iLLL and its heuristics, respectively, while LLL reaches dimension 512. Furthermore, as the dimension grows, the gap between the running time of iLLL and LLL grows as well, which indicates the time that iLLL gains is increasing as the dimension grows. Nevertheless, we anticipate to accelerate the reduction by 30 %, which is the same case with the toy challenge. However, due to the fact that the size reduction of a single vector becomes less important as the dimension grows, compared with the whole cost of a single step, the actual ratio of advantage is diminishing.

Figure 7 shows a comparison of the running time of iLLL and LLL algorithms. The green curve shows the ratio between the total running time of LLL and iLLL up to a certain

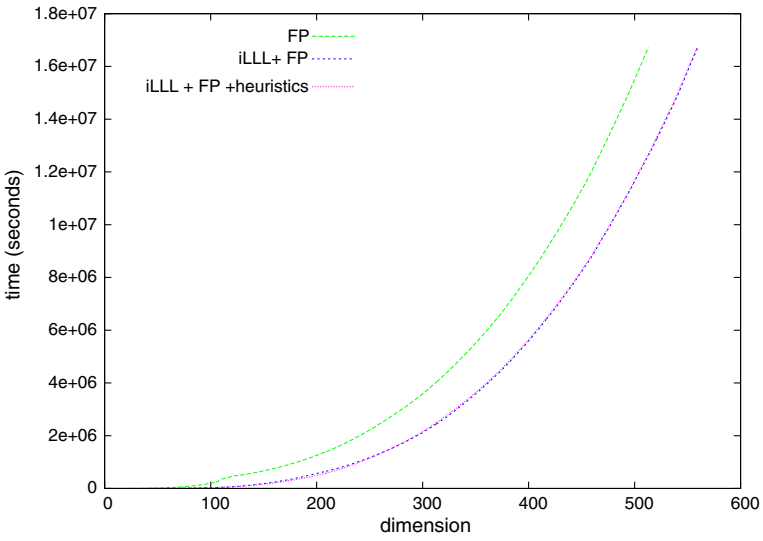

Fig. 5 Small challenge

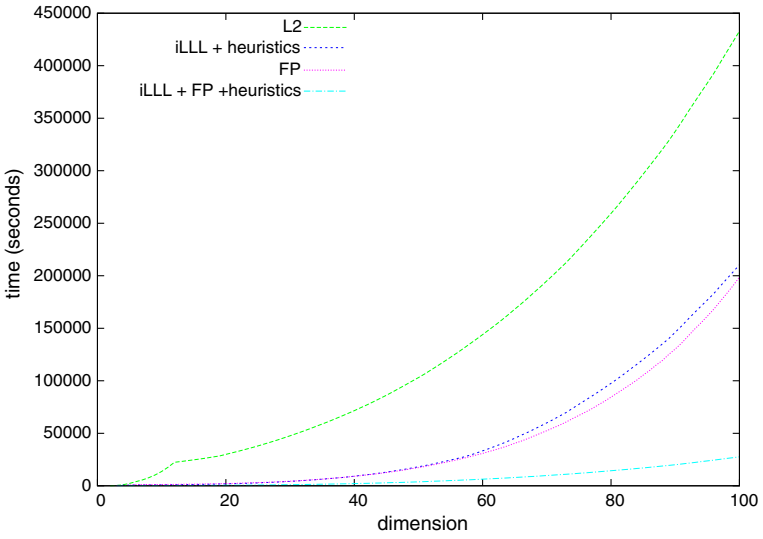

Fig. 6 Comparison of L^2 and FP for small challange

dimension, while the blue curve shows LLL vs iLLL for each dimension. Since the curves is always higher than 1, it is straightforward to see that iLLL is always faster than LLL. It is also worth pointing out that at dimension around 100, iLLL can be 10 times faster than LLL.

6.3 Analysis

In the previous subsection, we have seen that the iLLL algorithm is faster than the LLL algorithm in practice. If we look at each step (Fig. 8 shows one example of this), one can see

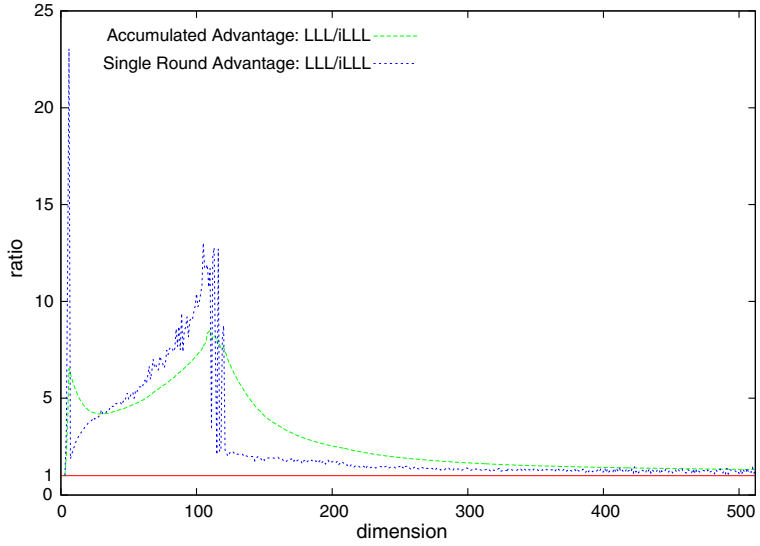

Fig. 7 Timing results for small challenge: LLL vs iLLL

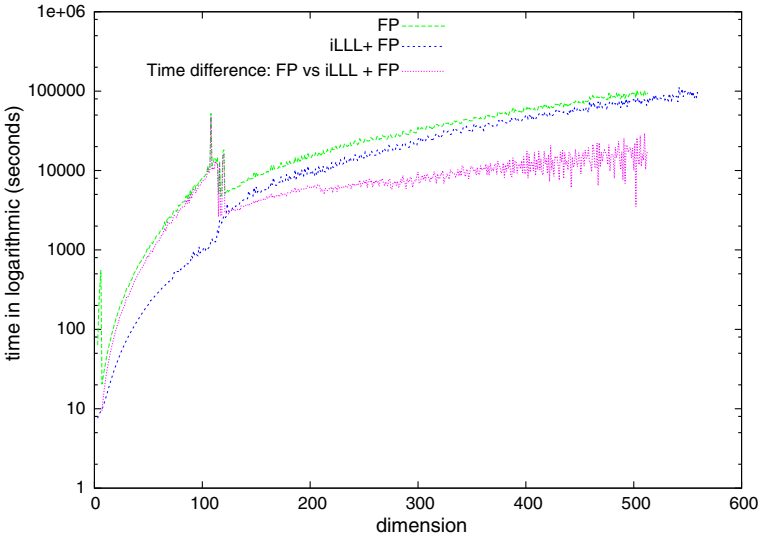

Fig. 8 Time of each step for dimension 2048

that in each step, iLLL is faster. This is due to the fact that instead of reducing a vector of the length β at the κth step, we can use a vector of length β/κ by simply re-using the previous results. The result follows our theoretical analysis. As stated before, although the time we are able to gain continues to increase, the advantage is actually diminishing as the dimension grows. This is because the cost of size-reducing the large vector is less and less important compare to the cost of reducing the whole basis as the dimension grows.

Interestingly, we enjoy a massive advantage when the dimension is less than 150. Similar phenomenon is also observed in the tests with random lattices. This is mainly due to the

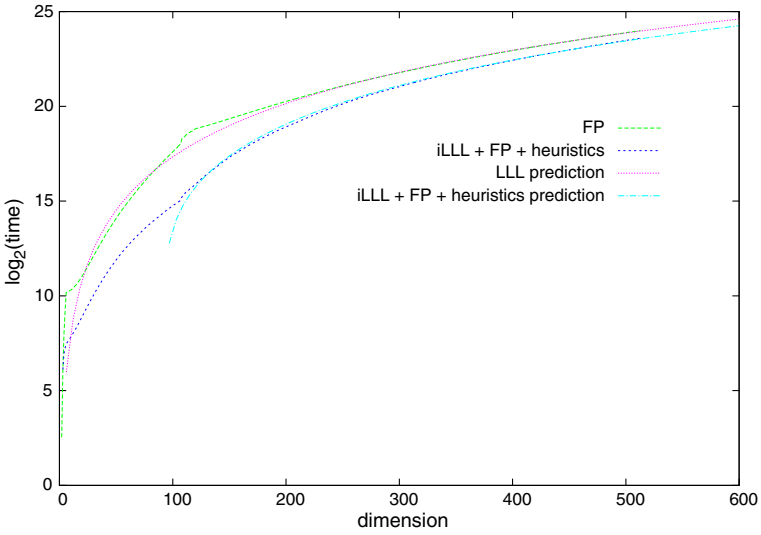

Fig. 9 Estimated time for dimension 2048

implementation of L^2. In theory, L^2 uses a floating point precision p which is in function of d. However, in practice, FP uses several levels of precisions. It starts with the smallest precision, 53, which is with respect to C programming language of type double/int in the program. It then increases the precision to the minimum value of p and the precision for the next stage. This procedure will be continued until it reaches p. Since the cost relies heavily on the length of floating-points, this optimization guarantees that the reduction is generally performed with the lowest cost.

It must be noted, however, that since the coefficients of the basis for each new step is large, FP will need to use a large precision to provide provable reductions. As a comparison, since we are dealing with small coefficients on average cases, when the dimension is smaller, we only require a default precision of 53. This is the reason our algorithm is faster in the beginning.

6.4 New estimation

In this subsection, we show our new estimation for the small challenge of Gentry–Halevi's fully homomorphic encryption. We start with predicting L^2 as in Fig. 9. We predict that the curve follows $\frac{d^{2.95}}{65} + \frac{5d^{2.8}}{13}$, which indicates that L^2 will finish in $2^{29.6}$ s which equals to 25.8 years. We note that this estimation is quite natural, since the previous research [6,25] has shown that LLL runs in cubic with regard to d when d is big and $\beta \sim O(d^2)$.

Then we look at the time difference between iLLL and LLL as shown by the third curve in Fig. 8. The result implies that the time we are able to gain with re-use is linear in dimension, when the curve becomes stable after dimension 130. This indicates that the accumulated time contains a quadratic term. Now we predict the running time of iLLL. We use the heuristic method, since in general it is faster. We predict that the curve follows $\frac{d^{2.95}}{65} + \frac{5d^{2.8}}{13} - \frac{77d^2}{5}$. This gives us $2^{29.47}$ s, which equals to 23.6 years. To conclude, we summarize our results in Table 2.

Table 2 Practical result on Gentry–Halevi's challenge

Gentry–Halevi's challenge	dim 512 (days)	dim 2048 (years)
Previous best results/prediction [4]	30	45
cre LLL implementation @2.66 GHz	32	25.8
iLLL implementation @2.66 GHz	24	23.6
iLLL prediction @4.0 GHz	16	15.7

Remark 2 The previous best prediction [4] was using L^2. We note that it is not specified in [4] what kind of a platform (CPU, library, etc.) their test was conducted on. Therefore, we performed the same LLL reduction as in [4]. We believe the difference between [4] and our approach is due to the implementation. This is the reason why we have compared our own LLL/iLLL implementation against the Gentry–Halevi's challenge.

7 Conclusion

In this paper, we presented the iLLL algorithm for ideal lattice. We take the advantage of ideal lattices that if a vector is in the lattice, then its rotation vector over the ring is still in the lattice. Therefore, previously reduced vectors can be re-used for further reduction. This ensures our iLLL algorithm is always faster than the LLL algorithm, while providing the same quality (if not exactly same) basis. In practice, it is also faster, especially when the coefficients of the lattice are significantly larger than the dimension. To show the advantage of our method, we applied our algorithm to Gentry–Halevi's fully homomorphic encryption challenge. We obtained a better estimation than the previously best known results.

Acknowledgments This work is supported by ARC Future Fellowship FT0991397.

References

1. Bosma W., Cannon J., Playoust C.: The Magma algebra system. I. The user language. J. Symb. Comput. **24**(3–4), 235–265 (1997).
2. Brakerski Z., Vaikuntanathan V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky R. (ed.) FOCS, pp. 97–106. IEEE, Los Alamitos (2011).
3. Brakerski Z., Gentry C., Vaikuntanathan V.: Fully homomorphic encryption without bootstrapping. Electron. Colloquium Comput. Complex. (ECCC) **18**, 111 (2011).
4. Chen Y., Nguyen P.Q.: BKZ 2.0: better lattice security estimates. In: Lee D.H., Wang X. (eds.) ASIACRYPT. Lecture Notes in Computer Science, vol. 7073, pp. 1–20, Springer, Heidelberg (2011).
5. Coron J.-S., Mandal A., Naccache D., Tibouchi M.: Fully homomorphic encryption over the integers with shorter public keys. In: Rogaway P. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 6841, pp. 487–504. Springer, Heidelberg (2011).
6. Gama N., Nguyen P.Q.: Predicting lattice reduction. In: Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology, EUROCRYPT'08, pp. 31–51. Springer, Berlin (2008).
7. Gentry C.: A fully homomorphic encryption scheme. PhD thesis, Stanford University (2009).
8. Gentry C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher M. (ed.) STOC, pp. 169–178. ACM, New York (2009).
9. Gentry C.: Computing arbitrary functions of encrypted data. Commun. ACM **53**(3), 97–105 (2010).
10. Gentry C., Halevi S.: Public challenges for fully-homomorphic encryption (online). http://researcher.watson.ibm.com/researcher/view_project.php?id=1548.

11. Gentry C., Halevi S.: Implementing Gentry's fully-homomorphic encryption scheme. In: Paterson K.G. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 6632, pp. 129–148. Springer, Berlin (2011).
12. Gentry C., Halevi S., Peikert C., Smart N.P.: Ring switching in BGV-style homomorphic encryption. In: Visconti I., Prisco R.D. (eds.) SCN. Lecture Notes in Computer Science, vol. 7485, pp. 19–37. Springer, Berlin (2012).
13. Gentry C., Halevi S., Smart N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval D., Johansson T. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 7237, pp. 465–482. Springer, Berlin (2012).
14. Goldstein D., Mayer A.: On the equidistribution of hecke points. Forum Math. **15**, 165–189 (2006).
15. Koy H., Schnorr C.-P.: Segment lll-reduction of lattice bases. In: Silverman J.H. (ed.) CaLC. Lecture Notes in Computer Science, vol. 2146, pp. 67–80. Springer, Berlin (2001).
16. Lenstra A.K., Lenstra H.W., Lovász L.: Factoring polynomials with rational coefficients. Math. Ann. **261**, 513–534 (1982).
17. Lovász L.: An Algorithmic Theory of Numbers, Graphs and Convexity. CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 50. SIAM Publications, Philadelphia (1986).
18. Lyubashevsky V., Micciancio, D.: Asymptotically efficient lattice-based digital signatures. In: Canetti R. (ed.) TCC. Lecture Notes in Computer Science, vol. 4948, pp. 37–54. Springer, Berlin (2008).
19. Micciancio D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. Comput. Complex. **16**(4), 365–411 (2007).
20. Micciancio D., Goldwasser S.: Complexity of Lattice Problems: A Cryptographic Perspective. Kluwer, Dordrecht (2002).
21. Minkowski H.: Geometrie der Zahlen. B.G. Teubner, Leipzig (1896).
22. Morel I., Stehlé D., Villard G.: H-lll: using householder inside lll. In: ISSAC, pp. 271–278. ACM, New York (2009).
23. Nguyen P.Q.: Breaking fully-homomorphic-encryption challenges. In: Lin D., Tsudik G., Wang X. (eds.) CANS. Lecture Notes in Computer Science, vol. 7092, pp. 13–14. Springer, Berlin (2011).
24. Nguyen P.Q., Stehlé D.: Floating-point LLL revisited. In: Advances in Cryptology—Eurocrypt 2005. Lecture Notes in Computer Science, vol. 3494, pp. 215–233. Springer, Berlin (2005).
25. Nguyen P.Q., Stehlé D.: LLL on the average. In: 7th International Symposium on Algorithmic Number Theory (ANTS 2006), pp. 238–256 (2006).
26. Nguyen P.Q., Vallée B.: The LLL Algorithm: Survey and Applications, 1st edn. Springer, Berlin (2009).
27. Novocin A., Stehlé D., Villard G.: An lll-reduction algorithm with quasi-linear time complexity: extended abstract. In: Fortnow L., Vadhan S.P. (eds.) STOC, pp. 403–412. ACM, New York (2011).
28. Plantard T., Susilo W., Zhang Z: Lattice reduction for knapsack. In: SAC, Windsor (2012).
29. Schnorr C.-P.: Fast lll-type lattice reduction. Inf. Comput. **204**(1), 1–25 (2006).
30. Smart N.P., Vercauteren F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen P.Q., Pointcheval D. (eds.) Public Key Cryptography. Lecture Notes in Computer Science, vol. 6056, pp. 420–443. Springer, Berlin (2010).
31. Stehlé D., Steinfeld R.: Faster fully homomorphic encryption. In: Abe M. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 6477, pp. 377–394. Springer, Berlin (2010).
32. SVP Challenge (online). http://www.latticechallenge.org/svp-challenge/index.php.
33. van Dijk M., Gentry C., Halevi S., Vaikuntanathan V.: Fully homomorphic encryption over the integers. In: Gilbert H. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 6110, pp. 24–43. Springer, Berlin (2010).
34. van Hoeij M., Novocin A.: Gradual sub-lattice reduction and a new complexity for factoring polynomials. CoRR. http://arxiv.org/abs/1002.0739 (2010).