Lattice Basis Reduction Algorithms and the Subset Sum Problem

LATTICE BASIS REDUCTION ALGORITHMS AND THE SUBSET SUM PROBLEM

BY

YANG BO, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING & SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Master of Science (2016) McMaster University

(Computing & Software) Hamilton, Ontario, Canada

TITLE:	Lattice Basis Reduction Algorithms and the Subset Sum Problem
AUTHOR:	Yang Bo B.Sc., (Information Security) University of Toronto Mississauga, Mississauga, Canada
SUPERVISOR:	Dr. Sanzheng Qiao
NUMBER OF PAGES:	vii, 64

Abstract

It is well-known that the subset sum problem is NP-complete, which is the basis for the subset sum based public-key cryptosystems. Some attacks on such cryptosystems have been developed. Those methods reduce the subset sum problem to the problem of finding a shortest Euclidean-norm nonzero vector in a point lattice [10, 24, 8, 12, 26].

In this thesis, we propose a new hybrid lattice basis reduction algorithm by integrating the recent polynomial time Type-I reduction algorithm by Wen Zhang [29] in 2015 with other techniques. We show that the Type-I algorithm is well suited for high dimensional lattices and apply the hybrid algorithm to the subset sum problem. Our experiments demonstrate that our method can solve relatively small size subset sum problems with higher success rates than the famous existing methods such as the Lagarias-Odlyzko attack and the Radziszowski-Kreher attack.

Acknowledgments

I would like to express my sincerest gratitude to my supervisor Dr.Sanzheng Qiao, for his guidance and knowledge during the research and preparation of this thesis. Furthermore, I would like to thank Dr.Wenbo He and Dr.Reza Samavi, for their careful reviews, valuable comments and suggestions. I also thank my friends, for the valuable discussions and suggestions about the algorithm. Finally, many thanks to my family for their help and support.

Contents

List of Figures

Chapter 1

Introduction

After Rivest, Shamir, and Adleman [25] published the first public-key cryptosystem, the RSA cryptosystem, several subset sum based cryptosystems were proposed [18, 3, 27, 28, 23, 7, 22]. The security of such cryptosystems is based on the difficulty of solving the subset sum problems which is a (worst case) NP-complete problem [9, 6]. In 1984 and 1985, Brickell [1] and Lagarias and Odlyzko [10] independently proposed an algorithm for the subset sum problems. Both methods show that it is possible to solve all subset sum problems of density $< 0.6463\ldots$. In 1988, Radziszowski and Kreher [24] evaluated the performance of an improved variant of the Lagarias-Odlyzko attack. Then Coster, LaMacchia, Odlyzko and Schnorr [4] and Joux and Stern [8] improved this bound to $0.9408\ldots$. In all the above algorithms, the subset sum problem is reduced to the problem of finding a shortest(Euclidean norm) nonzero vector in a lattice, and the LLL algorithm is used to find a shortest nonzero lattice vector. In 1991, LaMacchia [12] used the Seysen's algorithm for finding a shortest nonzero lattice vector, combined with the GCD reduction in solving the subset sum problem. In 1994, Schnorr and Euchner [26] proposed an improved LLL algorithm

(L^3FP algorithm) to attack the subset sum problem. Both LaMacchia and Schnorr and Euchner solved the subset sum problem with relatively large size(≥ 42). This thesis studies a new polynomial time approach of lattice basis reduction developed by Wen Zhang [29] in 2015, the Type-I reduction algorithm, for finding a shortest nonzero lattice vector in attacking the subset sum problem. This algorithm is initially developed to find the relatively orthogonal bases of a lattice. However, it can also be successfully applied to solving the subset sum problem, especially when combined with other known reduction approaches. Our experiments show that the Type-I algorithm, when integrated with other techniques, including the LLL, the GCD reduction [2], the weight reduction [24] and the sort operations, is able to solve relatively small size(≤ 43) subset sum problems with higher success rates than the Largarias-Odlyzko attack [10] and the Radziszowski-Kreher attack [24].

1.1 Thesis Outline

In Chapter 2, we introduce basic concepts related to our work. For example, the definitions of a lattice, the unimodular matrix, the volume of lattice, the orthogonality defect. The relation between two arbitrary bases for a lattice is also shown in this Chapter.

Chapter 3 first discusses some notions of the reduced lattice basis. Then we introduce two polynomial time lattice basis reduction algorithms, the LLL algorithm and the Type-I algorithm. After analyzing the complexities of algorithms and some properties of the reduced bases, the experimental comparisons are shown in the last section of Chapter 3.

Chapter 4 is the main part of this thesis. We first introduce the definitions of the

subset sum problem and the density. After that, we present previous theoretical and experimental results of the subset sum problem. In section 4.4, we propose a new hybrid algorithm,the Type-I & LLL algorithm, for solving the subset sum problem and we show the experimental results of the new algorithm in section 4.5.

In the last Chapter, we remark the advantages and disadvantages of the Type-I and the Type-I & LLL algorithms. We also propose some possible improvements, which can extend the Type-I & LLL algorithm to attack the subset sum problems with large sizes.

Chapter 2

Lattices and Bases

In this chapter, we introduce some concepts of the lattice theory which will be used for later work.

Let A be a set of linearly independent vectors $a_1, a_2, \ldots, a_n$ in R^m, $m \geq n$. lattice $L(A)$ is an infinite set of discrete points, each point is a linear combination of vectors $a_1, a_2, \ldots, a_n$ with integer coefficients. In other words :

$$L(A) = \{x \,|\, x = \sum_{i=1}^{n} z_i a_i, \quad \text{where } z_i \in \mathbb{Z} \text{ and } 1 \leq i \leq n\}.$$

We say that A form a basis for L. A lattice can have multiple bases. For example, Figure 2.1 illustrates a lattice L and its basis A, where

$$A = [a_1, a_2] = \begin{bmatrix} 2 & 2.7 \\ 0 & 1 \end{bmatrix}. \tag{2.1}$$

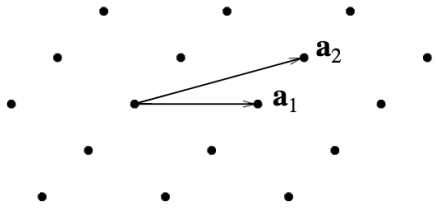

Figure 2.1: Lattice L and its basis A

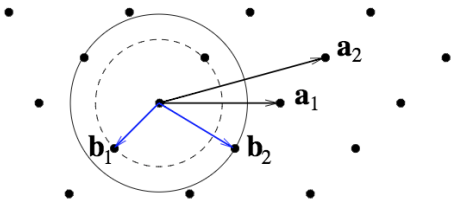

Figure 2.2: Lattice L and its bases A and B

Figure 2.2 shows another basis B for L, where

$$B = [b_1, b_2] = \begin{bmatrix} 0.7 & 1.3 \\ 1 & -1 \end{bmatrix}. \tag{2.2}$$

Two basis matrices are related by a unimodular matrix defined as follows.

Definition 2.1. *Let U be a nonsingular matrix with integer entries, we say U is unimodular if $det(U) = \pm 1$.*

Notice that if U is unimodular, then U^{-1} is also unimodular. To see this, consider the following equation:

$$Uu_i^{'} = e_i, \tag{2.3}$$

where $u_i^{'}$ is the i^{th} column vector of U^{-1} and e_i is the i^{th} column vector of the identity

matrix I.

Recall the Cramer's rule, the solution to (2.3) is given by

$$u_{ij}^{'} = \frac{det([u_1, \ldots, u_{j-1}, e_i, u_{j+1}, \ldots, u_n])}{det(U)}, \tag{2.4}$$

where $u_i^{'} = [u_{i1}^{'}, \ldots, u_{in}^{'}]^T$ and u_i is the i^{th} column vector of U. Clearly, the numerator of (2.4) is an integer since every column vector u_i of U and e_i are integer vectors. Recall that $det(U) = \pm 1$, this proves that U^{-1} is a integer matrix. Matrix U^{-1} is unimodular because:

$$det(U)det(U^{-1}) = det(UU^{-1}) = det(I) = 1, \tag{2.5}$$

so $det(U^{-1}) = \pm 1$.

Theorem 2.1. *Let A and B be two bases, then $L(A) = L(B)$ if and only if there exists a unimodular matrix U, such that $B = AU$.*

Proof. Assume A and B are two bases for the same lattice $L(A) = L(B)$. There exists integer matrices U and $U^{'}$ such that $B = AU$ and $A = BU^{'}$. Combining these two equations, we get $A = AUU^{'}$, which is equivalent to $UU^{'} = I$ or $U^{'} = U^{-1}$. By equation (2.5), we get $det(U) = det(U^{-1}) = \pm 1$ since U and U^{-1} are integer matrices, so their determinants are also integers.

Next, assume $B = AU$ for some unimodular matrix U, we also can get $A = BU^{-1}$. Equations $B = AU$ and $A = BU^{-1}$ imply $L(B) \subset L(A)$ and $L(A) \subset L(B)$, respectively. So two matrices A and B generate the same lattice. □

Recall matrices A from (2.1) and B from (2.2), there exists a unimodular matrix

U such that $B = AU$ where

$$U = \begin{bmatrix} -1 & 2 \\ 1 & -1 \end{bmatrix}. \tag{2.6}$$

Theorem 2.2. *Let L be the lattice generated by a matrix $A = [a_1, a_2, \ldots, a_n] \in \mathbb{R}^{m \times n}$. Then the volume of L is defined as* $\mathrm{vol}(L) = \sqrt{\det(A^T A)}$.

If matrix A and B are two bases for lattice L and $B = AU$, where U is a unimodular matrix then

$$\begin{aligned} \mathrm{vol}(L) &= \sqrt{\det(B^T B)} \\ &= \sqrt{\det((AU)^T (AU))} \\ &= \sqrt{\det(U^T A^T A U)} \\ &= \sqrt{\det(U^T)\det(A^T)\det(A)\det(U)} \\ &= \sqrt{\det(A^T)\det(A)} \\ &= \sqrt{\det(A^T A)}. \end{aligned}$$

Because the determinant of the unimodular matrix U is ± 1 and $\det(U^T) = \det(U)$. Hence the volume of a lattice is independent of the choice of basis.

Intuitively, as shown in Figure 2.2, the basis $\{b_1, b_2\}$ is "more orthogonal" than the basis $\{a_1, a_2\}$. In the following, we define the orthogonality defect, a measurement of orthogonality, of a basis.

Definition 2.2. *The orthogonality defect of a basis $A = [a_1, a_2, \ldots, a_n]$ for $L(A)$ is defined as* $\gamma(A) = \frac{\prod_{i=1}^{n} \|a_i\|_2}{\mathrm{vol}(L)}$.

The orthogonality defect $\gamma(A)$ is always larger than or equal to 1. From Hardama's Inequality, $\det(A) \leq \prod_{i=1}^{n} \|a_i\|_2$, hence

$$\begin{aligned}\gamma(A) &= \sqrt{\Big(\frac{\prod_{i=1}^{n} \|a_i\|_2}{\mathrm{vol}(L)}\Big)^2} \\ &= \sqrt{\frac{\prod_{i=1}^{n} \|a_i\|_2 \prod_{i=1}^{n} \|a_i\|_2}{\det(A^T)\det(A)}} \\ &\geq 1.\end{aligned}$$

The equality is achieved if and only if the vectors are orthogonal. The concept of orthogonality defect is used to measure the degree of orthogonality for a given matrix. For example, as shown in Figure 2.2, the orthogonality defect $\gamma(A)$ of basis $A = [a_1, a_2]$ equals 2.8792 and the orthogonality defect $\gamma(B)$ of basis $B = [b_1, b_2]$ equals 1.0010.

Chapter 3

Reduced Lattice Bases and Lattice Basis Reduction Algorithm

In this chapter, we first introduce some definitions of reduced lattice basis and the related concepts. Then we discuss two polynomial-time lattice basis reduction algorithms, the LLL reduction algorithm and the Type-I reduction algorithm.

3.1 Minkowski Reduced Basis

As we mentioned in Chapter 2, a lattice may have many different lattice bases and some of them are considered better than others. In this thesis, we use Euclidean norm as a distance function and say a basis consisting of "shorter" vectors is better. Lattice reduction algorithm can find "good" lattice basis for a particular lattice. There is a classical lattice reduction theory due to Minkowski [19].

Definition 3.1. *Using the Euclidean norm as a distance function, we say that* $\lambda_i(L)$, $1 \leq i \leq n$, *is the* i^{th} *successive minimum with respect to a lattice* L, *if* $\lambda_i(L)$ *is the*

lower bound of the radius λ of the sphere

$$\|Az\|_2 \leq \lambda \qquad \textit{where } A \textit{ is a basis of } L \textit{ and } z \in \mathbb{Z}^n$$

that contains i linearly independent lattice points [15].

Actually, $\lambda_i(L)$ is the lowest bound of $\max(\|a_1\|_2, \|a_2\|_2, \ldots, \|a_i\|_2)$ over all sets of linearly independent vectors $a_1, a_2, \ldots, a_i$ of lattice L [15], where $\|a_i\|_2$ denotes the Euclidean length of vector a_i. In particular, the Euclidean length of a shortest nonzero vectors in lattice L is $\lambda_1(L)$.

Definition 3.2. *The basis $\{a_1, a_2, \ldots, a_n\}$ for lattice L is called Minkowski reduced if $\|a_i\|_2 = \lambda_i(L)$, where $1 \leq i \leq n$.*

A Minkowski reduced basis can be obtained by 1. Select a shortest nonzero vector in the lattice as a_1, 2. Choose the subsequent basis vectors a_i such that a_i is a shortest vector in the lattice and $\{a_1, a_2, ..., a_i\}$ can be extended to a basis [12]. Currently, the problem of finding a shortest nonzero lattice vector in a lattice is non-polynomial.

3.2 The LLL Reduced Basis and the LLL Algorithm

In 1982, Lenstra, Lenstra and Lovász [13] introduced a polynomial-time algorithm(the LLL algorithm) which can reduce a given lattice basis $A = \{a_1, a_2, \ldots, a_n\}$ for lattice L into an LLL-reduced basis $A^L = \{a_1^L, a_2^L, \ldots, a_n^L\}$.

Definition 3.3. *The basis* $\{a_1, a_2, \ldots, a_n\}$ *for lattice* L *is called LLL reduced if*

$$|\mu_{i,j}| \leq \frac{1}{2} \qquad for\ 1 \leq i < j \leq n, \quad \mu_{i,j} = \frac{< a_j, a_i^* >}{< a_i^*, a_i^* >}, \tag{3.1}$$

$$(\omega - \mu_{i-1,i}^2)\|a_{i-1}^*\|_2^2 \leq \|a_i^*\|_2^2 \qquad for\ 1 < i \leq n, \quad \frac{1}{4} < \omega < 1, \tag{3.2}$$

where $A^* = [a_1^*, a_2^*, \ldots, a_n^*]$ *is the Gram-Schmidt orthogonalized basis generated from* A*, that is* $a_j^* = a_j - \sum_{i=1}^{j-1} \mu_{i,j} a_i^*$ *and* $< a_i^*, a_j^* >$ *denotes the inner product of vectors* a_i^* *and* a_j^**,* ω *is a constant.*

The LLL algorithm transforms a lattice basis A into an LLL reduced lattice basis A^L by two types of operations, the size-reduction and the exchange. In the size-reduction operation, the LLL enforces (3.1) on $\mu_{i,j}$, $1 \leq i < j \leq n$, by performing the transformation $a_j = a_j - \lceil \mu_{i,j} \rfloor a_i$, where $\lceil \cdot \rfloor$ denotes the nearest integer function and updates the corresponding $\mu_{i,j}$. If a_j^* and a_{j-1}^*, where $1 < j \leq n$, violate (3.2), the exchange operation is called. The LLL swaps a_j and a_{j-1}, $\mu_{i,j}$ and $\mu_{i,j-1}$, for $1 \leq i < j - 1 < n$. After that, the LLL updates a_j^*, a_{j-1}^* and the corresponding $\mu_{j-1,l}$, $j - 1 < l \leq n$ and $\mu_{j,k}$, $j < k \leq n$. For details of the LLL algorithm, please refer to [13].

Lenstra, Lenstra and Lovász have shown that the LLL reduced basis is a reasonable approximation of Minkowski reduced basis [13]. If L is the lattice and A^L is the LLL reduced basis for L, then

$$\gamma^{1-i}\lambda_i^2(L) \leq \|a_i^L\|_2^2 \leq \gamma^{n-1}\lambda_i^2(L), \qquad \text{for } 1 \leq i \leq n, \tag{3.3}$$

where $\gamma = (\omega - \frac{1}{4})^{-1}$ and $\lambda_1(L), \lambda_2(L), \ldots, \lambda_n(L)$ are Minkowski minima. In particular, if $\omega = \frac{3}{4}$, we have

$$2^{1-i}\lambda_i^2(L) \leq \|a_i^L\|_2^2 \leq 2^{n-1}\lambda_i^2(L), \qquad \text{for } 1 \leq i \leq n.$$

From (3.3), it's not hard to see if $i = 1$, then we can get a bound of the length of the first column vector a_1^L in A^L

$$\|a_1^L\|_2 \leq (\frac{4}{4\omega - 1})^{n-1}\|v\|_2, \quad \text{where } v \text{ is a shortest vector in } L \text{ and } v \neq 0.$$

If we set $\omega = \frac{3}{4}$, we have

$$\|a_1^L\|_2 \leq 2^{n-1}\|v\|_2, \qquad \text{where } v \text{ is a shortest vector in } L \text{ and } v \neq 0.$$

Therefore, the length of a_1^L is at most the exponential multiple of the length of a shortest vector v in lattice L. In practice, the LLL algorithm usually performs much better than the above bound. Lenstra, Lenstra and Lovász also proved that the computational complexity of the LLL algorithm is $O(n^4 \log B)$ if $L \subset \mathbb{Z}^n$ is a lattice with basis $a_1, a_2, \ldots, a_n$, $B \in \mathbb{R}$, $B \geq 2$ and $\|a_i\|_2^2 \leq B$ for $1 \leq i \leq n$ [13].

In 2008, Luk and Tracy [16] developed a matrix interpretation of the LLL algorithm, which leads to a new implementation of the LLL algorithm. Consider a nonsingular matrix $A \in \mathbb{R}^{n \times n}$, its QR decomposition is

$$A = QR,$$

where $Q \in \mathbb{R}^{n \times n}$ is orthogonal and $R \in \mathbb{R}^{n \times n}$ is upper triangular. Then (3.1) and (3.2)

can be rewritten as follows.

Definition 3.4. *A nonsingular upper triangular matrix R is reduced if*

$$|r_{i,i}| \geq 2|r_{i,j}|, \qquad for\ all\ 1 \leq i < j \leq n, \tag{3.4}$$

$$r_{i,i}^2 + r_{i-1,i}^2 \geq \omega r_{i-1,i-1}^2, \qquad for\ all\ 2 \leq i \leq n, \quad \frac{1}{4} < \omega < 1. \tag{3.5}$$

Unlike the LLL algorithm, the new algorithm transforms the upper triangular matrix R into a reduced upper triangular matrix R^L and (3.4), (3.5) are enforced by two types of operations, the Decrease operation and the Swap-Restore operation. All column operations are recorded in a unimodular matrix Z.

Algorithm 1 Decrease (j, i)

Require: Upper triangular matrix R, unimodular matrix Z
Ensure: Modified R,Z that satisfy (3.4)
1: $\gamma = \lceil \frac{r_{i,j}}{r_{i,i}} \rfloor$
2: $Z_{ij} = I_n - \gamma e_i e_j^T$
3: $R = RZ_{ij}$
4: $Z = ZZ_{ij}$
5: **return** R, Z

If $r_{i,i}$ and $r_{i,j}$ fail to satisfy (3.4), where $1 \leq i < j \leq n$, then the Decrease is called. After computing the integer nearest $\frac{r_{i,j}}{r_{i,i}}$ in line 1, an $n \times n$ unimodular matrix Z_{ij} is created by an $n \times n$ identity matrix I_n and the i^{th} and j^{th} unit vector e_i and e_j. Actually, Z_{ij} equals the $n \times n$ identity matrix except that its $(i,j)^{th}$ entry equals γ. It is easy to check that Z_{ij} is a unimodular matrix. Both R and Z are modified by Z_{ij} in lines 3 and 4 to satisfy (3.4). Note that the Decrease operation may increase

the length of vectors in the basis. For example, let

$$A = \begin{bmatrix} 3 & -\frac{4}{3} & 1 \\ 0 & \frac{5}{3} & 1 \\ 0 & 0 & 3 \end{bmatrix}.$$

After the Decrease operation with parameters $j = 3$ and $i = 2$, the matrix A becomes

$$A = \begin{bmatrix} 3 & -\frac{4}{3} & \frac{7}{3} \\ 0 & \frac{5}{3} & -\frac{2}{3} \\ 0 & 0 & 3 \end{bmatrix}.$$

Clearly, the length of the third column of A is increased by the Decrease operation.

Algorithm 2 Swap-Restore (j)

Require: Upper triangular matrix R, orthogonal matrix Q, unimodular matrix Z
Ensure: Modified R,Q,Z that satisfy (3.5)
1: Create a plane reflection 2×2 matrix J_j
2: $Q_j = diag[I_{j-2}, J_j, I_{n-j}]$
3: $QT_j = diag[I_{j-2}, J_j^T, I_{n-j}]$
4: $\Pi_j = diag[I_{j-2}, P, I_{n-j}]$
5: $R = Q_j R \Pi_j$
6: $Z = Z\Pi_j$
7: $Q = QQT_j$
8: **return** R, Q, Z

If $r_{j,j}, r_{j-1,j}$ and $r_{j-1,j-1}$ violate (3.5), the Swap-Restore operation is performed. The 2×2 plane reflection matrix J_j is the matrix with form

$$J_j = \begin{bmatrix} c & -s \\ s & c \end{bmatrix},$$

where $c = \frac{r_{j-1,j-1}}{r}, s = -\frac{r_{j,j-1}}{r}$ and $r = \sqrt{r_{j-1,j-1}^2 + r_{j,j-1}^2}$. In [14], Luk and Qiao show that the purpose of J_j is to let

$$J_j \begin{bmatrix} r_{j-1,j-1} & r_{j-1,j} \\ 0 & r_{j,j} \end{bmatrix} P = \begin{bmatrix} \hat{r}_{j-1,j-1} & \hat{r}_{j-1,j} \\ 0 & \hat{r}_{j,j} \end{bmatrix}, \qquad \text{where } P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

In lines 2 and 3, matrices Q_j and Π_j are created. In line 4, the j^{th} and $j-1^{th}$ column of R are swapped by multiplying Π_j then R is restored back to the upper triangular matrix by multiplying Q_j. The unimodular matrix Z and orthogonal matrix Q are correspondingly updated in lines 5 and 6.

Algorithm 3 MatrixLLL Algorithm

Require: Lattice basis A, parameter ω
Ensure: Unimodular matrix Z, such that AZ is a LLL reduced basis

```
1:  n = size(A, 2)
2:  compute Q and R such that A = QR by QR decomposition
3:  Z = I_n
4:  j = 2
5:  while j ≤ n do
6:      if |r_{j-1,j}| > (1/2)|r_{j-1,j-1}| then
7:          [R, Z] = Decrease (j, j − 1)
8:      if r_{j,j}^2 + r_{j-1,j}^2 < ω r_{j-1,j-1}^2 then
9:          [R, Q, Z] = Swap-Restore (j)
10:         if j > 2 then
11:             j = j − 1
12:     else
13:         for i = j − 2 to 1 do
14:             if |r_{i,j}| > (1/2)|r_{i,i}| then
15:                 [R, Z] = Decrease (j, i)
16:         j = j + 1
17: return Z
```

The MatrixLLL algorithm (Algorithm 3) and the LLL algorithm [13] have similar

iterative structure, except the MatrixLLL generates a reduced upper triangular matrix R^L. At the end of the MatrixLLL algorithm, a unimodular matrix Z is returned. The columns of AZ form a reduced basis as defined in [13]. For detailed proofs, please refer to [16] and [14].

3.3 The Type-I Reduced Basis and the Type-I Reduction Algorithm

Since Lenstra, Lenstra and Lovász published the LLL algorithm in 1982, various improvements of the algorithm have been proposed. However, almost all of these improvements are based on the Gram-Schmidt orthogonalization or the QR decomposition and have the same iteration structure as the original LLL algorithm. In 2015, Wen Zhang [29] proposed a new lattice basis reduction algorithm, the Type-I algorithm. This algorithm does not require the Gram-Schmidt orthogonalization nor the QR decomposition and has a different iteration structure.

Definition 3.5. *The basis matrix* $A = [a_1, a_2, a_3, \ldots, a_n] \in \mathbb{R}^{m\times n}$ *for lattice* L *is called Type-I reduced if*

$$\|a_1\|_2 \leq \|a_2\|_2 \leq \ldots \leq \|a_n\|_2, \tag{3.6}$$

$$\delta\|a_j\|_2^2 \leq \|a_j - \left(\frac{a_i^T a_j}{\|a_i\|_2^2} + \frac{1}{2}\right)a_i\|_2^2, \qquad 1 \leq i < j \leq n, \quad \frac{1}{4} < \delta < 1. \tag{3.7}$$

The Type-I algorithm also transforms a lattice A into a Type-I reduced basis by two types of operations, the exchange operation and the size-reduction operation. However, the trigger conditions of these two types of operations are different from

the trigger conditions in the LLL.

Algorithm 4 Type-I Algorithm

Require: Lattice basis A, parameter δ
Ensure: Type-I reduced basis A

$n = \text{size}(A, 2)$
$done = \text{False}$
while not $done$ **do**
    $done = \text{True}$
    for $i = 1$ to $n - 1$ **do**
        for $j = i + 1$ to n **do**
            if $\|a_i\|_2^2 > \|a_j\|_2^2$ **then**
                $done = \text{False}$
                Swap a_i and a_j
            $m = \lfloor \frac{a_i^T a_j}{\|a_i\|_2^2} \rceil$
            $tmp = a_j - m a_i$
            if $\|tmp\|_2^2 < \delta \|a_j\|_2^2$ **then**
                $done = \text{False}$
                $a_j = tmp$
return A

It is not hard to see that in Algorithm 4, the Type-I algorithm does not stop unless both (3.6) and (3.7) are satisfied. For each pair of a_i and a_j where $1 \leq i < j \leq n$, the Type-I algorithm exchanges them to meet (3.6)(the exchange operation) if needed. The Type-I enforces (3.7) by subtracting $\lfloor \frac{a_i^T a_j}{\|a_i\|_2^2} \rceil a_i$ from a_j(the size-reduction

operation) since

$$\begin{aligned}
\delta\|a_j\|_2^2 &\leq \|a_j - \lfloor\frac{a_i^T a_j}{\|a_i\|_2^2}\rceil a_i\|_2^2 \\
&= \|a_j - \left(\frac{a_i^T a_j}{\|a_i\|_2^2} + k\right) a_i\|_2^2 \\
&= \|a_j - \frac{a_i^T a_j}{\|a_i\|_2^2} a_i\|_2^2 - 2k(a_j - \frac{a_i^T a_j}{\|a_i\|_2^2} a_i)a_i + k^2\|a_i\|_2^2 \\
&= \|a_j - \frac{a_i^T a_j}{\|a_i\|_2^2} a_i\|_2^2 + k^2\|a_i\|_2^2 \\
&\leq \|a_j - \frac{a_i^T a_j}{\|a_i\|_2^2} a_i\|_2^2 + \frac{1}{4}\|a_i\|_2^2 \\
&\leq \|a_j - \left(\frac{a_i^T a_j}{\|a_i\|_2^2} + \frac{1}{2}\right) a_i\|_2^2,
\end{aligned}$$

where $k \in \mathbb{R}$, $\lfloor\frac{a_i^T a_j}{\|a_i\|_2^2}\rceil = \frac{a_i^T a_j}{\|a_i\|_2^2} + k$ and $k^2 \leq \frac{1}{4}$. Note that $a_j - \frac{a_i^T a_j}{\|a_i\|_2^2} a_i$ is orthogonal to a_i. The size-reduction operation of the Type-I algorithm calculates the integer nearest $\frac{a_i^T a_j}{\|a_i\|_2^2}$, which is different from what the size-reduction operation does in the LLL. In addition, the column vectors a_j will be replaced by $a_j - \lfloor\frac{a_i^T a_j}{\|a_i\|_2^2}\rceil a_i$ only if the length of $a_j - \lfloor\frac{a_i^T a_j}{\|a_i\|_2^2}\rceil a_i$ is shorter. Thus, unlike the LLL, the Type-I algorithm never increases the length of vectors in the basis matrix.

Recall from the previous section that the LLL algorithm guarantees the column vector a_1^L of the LLL reduced basis A^L satisfies:

$$\|a_1^L\|_2 \leq 2^{n-1}\|v\|_2, \qquad \text{where } v \text{ is a shortest vector in } L(A^L) \text{ and } v \neq 0.$$

However, the Type-I algorithm cannot guarantee the length of column vectors in the Type-I reduced basis, but it guarantees that the acute angle between any pair of the reduced basis vectors is at least 60°.

Proposition 3.1. *Let $A^I = [a_1^I, a_2^I, \ldots, a_n^I]$ be a Type-I reduced basis, then the acute angle between any pair of a_i^I and a_j^I, $1 \leq i < j \leq n$ is at least $60°$.*

Proof. Let $c = \frac{(a_i^I)^T a_j^I}{\|a_i^I\|_2^2}$, from (3.7) we have

$$\begin{aligned}\delta\|a_j^I\|_2^2 &\leq \|a_j^I - \left(c + \frac{1}{2}\right)a_i^I\|_2^2 \\ &= \|a_j^I - ca_i^I\|_2^2 + \frac{1}{4}\|a_i^I\|_2^2 \\ &= \|a_j^I\|_2^2 - c^2\|a_i^I\|_2^2 + \frac{1}{4}\|a_i^I\|_2^2,\end{aligned}$$

since $a_j^I - ca_i^I$ is orthogonal to a_i and $(a_i^I)^T a_j^I = c\|a_i^i\|_2^2$. It then implies

$$\begin{aligned}c^2\|a_i^I\|_2^2 &\leq (1-\delta)\|a_j^I\|_2^2 + \frac{1}{4}\|a_i^I\|_2^2 \\ &\leq \frac{1}{4}\|a_i^I\|_2^2\end{aligned}$$

if $\delta = 1$, which implies that $|c| \leq \frac{1}{2}$. Let θ be the angle between a_i^I and a_j^I where $1 \leq i < j \leq n$, then

$$\begin{aligned}|cos\theta| &= \frac{|(a_i^I)^T a_j^I|}{\|a_i^I\|_2\|a_j^I\|_2} \\ &\leq \frac{|(a_i^I)^T a_j^I|}{\|a_i^I\|_2^2} \\ &\leq \frac{1}{2}.\end{aligned}$$

This shows that the acute angle between any pair of reduced lattice basis vectors is at least $60°$ when $\delta = 1$. □

We can expect that the acute angle between any pair of vectors in the reduced lattice basis is at least 60° when δ is close to one. Wen Zhang has proved that the complexity of the Type-I reduction is $O(n^3\log(\frac{M^n}{\mathrm{vol}(L)}))$.

Proposition 3.2. *Let $L \subset \mathbb{R}^n$ be a lattice with basis vectors $a_1, a_2, \ldots, a_n \in \mathbb{R}^m$, where $n \leq m$ and $M = \max_{1\leq i\leq n}\|a_i\|_2^2$. Then the computational complexity of the Type-I reduction algorithm(Algorithm 4) is $O(n^3\log(\frac{M^n}{\mathrm{vol}(L)}))$.*

Proof. First we introduce the quantity

$$D = \|a_1\|_2^2\|a_2\|_2^2 \ldots \|a_n\|_2^2.$$

It is not hard to see that the number of the iterations of the while-loop is bounded by the number of times that the exchange and the size-reduction are executed. We now estimate the number of the iterations. Note that each time the size-reduction operation is executed, the corresponding $\|a_j\|_2^2$ will be reduced by a factor smaller than $\delta \leq 1$, while the other $\|a_1\|_2^2, \|a_2\|_2^2, \ldots, \|a_{j-1}\|_2^2, \|a_{j+1}\|_2^2, \ldots, \|a_n\|_2^2$ remain unchanged. Consequently, D will be reduced by a factor smaller than δ. At the beginning of the algorithm, we have $D \leq M^n$ and throughout the algorithm $D \geq \mathrm{vol}(L)$. Therefore, the number of times that the Type-I algorithm passes through the size-reduction operation is at most $O(\log(\frac{M^n}{\mathrm{vol}(L)}))$. After (3.7) is satisfied, the Type-I algorithm just needs one more time of the while-loop to enforce (3.6) since the Type-I only needs to go through every pair of vectors once to sort them into increasing order of length. Hence, the number of the iteration of the while-loop is bounded by $O(\log(\frac{M^n}{\mathrm{vol}(L)}))$. It is not hard to verify that the operations of the exchange operation and the size-reduction operation require $O(n)$ flops. As a result, the operations in each iteration

of the while-loop require at most $O(n^3)$ flops. Then the computational complexity of the Type-I reduction is $O(n^3 \log(\frac{M^n}{\mathrm{vol}(L)}))$. □

In [29], Wen Zhang also mentions the Type-I algorithm can be implemented by the Gram matrix $A^T A$, rather than the original lattice basis matrix A to improve the efficiency. Let the original basis matrix $A = [a_1, a_2, a_3, \ldots, a_n] \in \mathbb{R}^{m \times n}$, then the Gram matrix is

$$G = A^T A = \begin{bmatrix} a_{11} & a_{21} & a_{31} & \ldots & a_{m1} \\ a_{12} & a_{22} & a_{32} & \ldots & a_{m2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & a_{3n} & \ldots & a_{mn} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \ldots & a_{mn} \end{bmatrix}$$

$$= \begin{bmatrix} \|a_1\|_2^2 & a_1^T a_2 & a_1^T a_3 & \ldots & a_1^T a_n \\ a_1 a_2^T & \|a_2\|_2^2 & a_2^T a_3 & \ldots & a_2^T a_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 a_n^T & a_2 a_n^T & a_3 a_n^T & \ldots & \|a_n\|_2^2 \end{bmatrix}.$$

The Gram matrix $G \in \mathbb{R}^{n \times n}$ and the elements in G have some special properties, such as $G_{i,j} = G_{j,i} = a_i^T a_j$ and $G_{i,i} = \|a_i\|_2^2$, where $1 \leq i, j \leq n$.

Algorithm 5 is the Type-I algorithm implemented with Gram matrix G. We record all column operations on A by a unimodular matrix Z. Recall Theorem 2.1 from Chapter 2, $L(A) = L(AZ)$ is guaranteed. The Gram matrix G is calculated in lines 5 to 7. Notice that we only compute the upper triangular part of G because $G_{i,j} = G_{j,i} = a_i^T a_j$. Lines 13 to 20 implement the exchange operation. If a_i is exchanged with a_j, where $1 \leq i < j \leq n$, then the GType-I records this exchange operation in the unimodular matrix Z and all related entries of G are updated. In the

Algorithm 5 GType-I Algorithm

Require: Lattice basis A, parameter δ
Ensure: Type-I reduced basis AZ

$n = \text{size}(A, 2)$
$G = n \times n$ zero matrix
$Z = n \times n$ identity matrix
for $i = 1$ to n **do**
    for $j = i$ to n **do**
        $G_{i,j} = a_i^T a_j$
done =False
while not *done* **do**
    done = True
    for $i = 1$ to $n - 1$ **do**
        for $j = i + 1$ to n **do**
            if $G_{i,i} > G_{j,j}$ **then**
                done =False
                Swap z_i and z_j
                Swap $G_{i,i}$ and $G_{j,j}$
                Swap $G_{(i,j+1:end)}$ and $G_{(j,j+1:end)}$
                Swap $G_{(1:i-1,i)}$ and $G_{(1:i-1,j)}$
                for $k = j - 1$ to $i + 1$ **do**
                    Swap $G_{i,k}$ and $G_{k,j}$
            $m = \lfloor \frac{G_{i,j}}{G_{i,i}} \rceil$
            $tmp = G_{j,j} - 2mG_{i,j} + m^2 G_{i,i}$
            if $tmp < \delta G_{j,j}$ **then**
                done =False
                $z_j = z_j - mz_i$
                $G_{j,j} = tmp$
                $G_{(i:j-1,j)} = G_{(i:j-1,j)} - mG_{(i,i:j-1)}^T$
                for $k = i - 1$ to 1 **do**
                    $G_{k,j} = G_{k,j} - mG_{k,i}$
                $G_{(j,j+1:n)} = G_{(j,j+1:n)} - mG_{(i,j+1:n)}$
return AZ

size-reduction operation(lines 21 to 30) of the GType-I algorithm, the variable *tmp* stores $\|a_j - ma_i\|_2^2$ instead of the vector $a_j - ma_i$. The expression $\|a_j - ma_i\|_2^2$ can be written as follows:

$$\begin{aligned}\|a_j - ma_i\|_2^2 &= \|a_j\|_2^2 - 2ma_ia_j + m^2\|a_i\|_2^2 \\ &= G_{j,j} - 2mG_{i,j} + m^2G_{i,i}.\end{aligned}$$

If *tmp* is less than $\|a_j\|_2^2$, the column vector a_j is updated to $a_j - ma_i$. The GType-I algorithm records this operation by $z_j = z_j - mz_i$. Because the vector a_j is changed, all related entries of the Gram matrix G should be updated. After the entry $G_{j,j}$ is updated by variable *tmp*, for $1 \leq i \leq k < j \leq n$, the elements

$$\begin{aligned}G_{k,j} &= a_k^T a_j \\ &= a_k^T(a_j - ma_i) \\ &= a_k^T a_j - ma_i a_k^T \\ &= G_{k,j} - mG_{i,k}.\end{aligned}$$

For $1 \leq k < i < j \leq n$, the entries

$$\begin{aligned}G_{k,j} &= a_k^T a_j \\ &= a_k^T(a_j - ma_i) \\ &= a_k^T a_j - ma_k^T a_i \\ &= G_{k,j} - mG_{k,i},\end{aligned}$$

and for $1 \leq i < j < k \leq n$, the elements

$$\begin{aligned} G_{j,k} &= a_j a_k^T \\ &= (a_j - m a_i) a_k^T \\ &= a_j a_k^T - m a_i a_k^T \\ &= G_{j,k} - m G_{i,k}. \end{aligned}$$

The GType-I returns AZ instead of A, because all exchange and size-reduction operations are recorded in the unimodular matrix Z and the Type-I reduced basis equals AZ. By implementing the Type-I algorithm with Gram matrix G, the computational complexity is further reduced to $O(n^2 \log(\frac{M^n}{\mathrm{vol}(L)}))$ since the operations of the exchange and the size-reduction only require $O(1)$ flops while the number of the iterations of the while-loop remains the same, that is $O(\log(\frac{M^n}{\mathrm{vol}(L)}))$.

3.4 Experimental Results of the LLL and the Type-I Algorithm

In previous sections, we have introduced the LLL and the Type-I reduction algorithms. We also detailed some properties of the LLL and the Type-I reduced bases. In this section, we will compare the running times of these two algorithms, the orthogonality defects of the reduced bases and the Euclidean norms of the shortest vectors. In our experiments, we set $n = 10, 15, 20, 25, \ldots, 95, 100$. For each value of n, we use MATLAB function **rand(n,n)** times 100 to generate 20 random $n \times n$ bases, where all entries of these bases are real numbers between 0 and 100. We apply

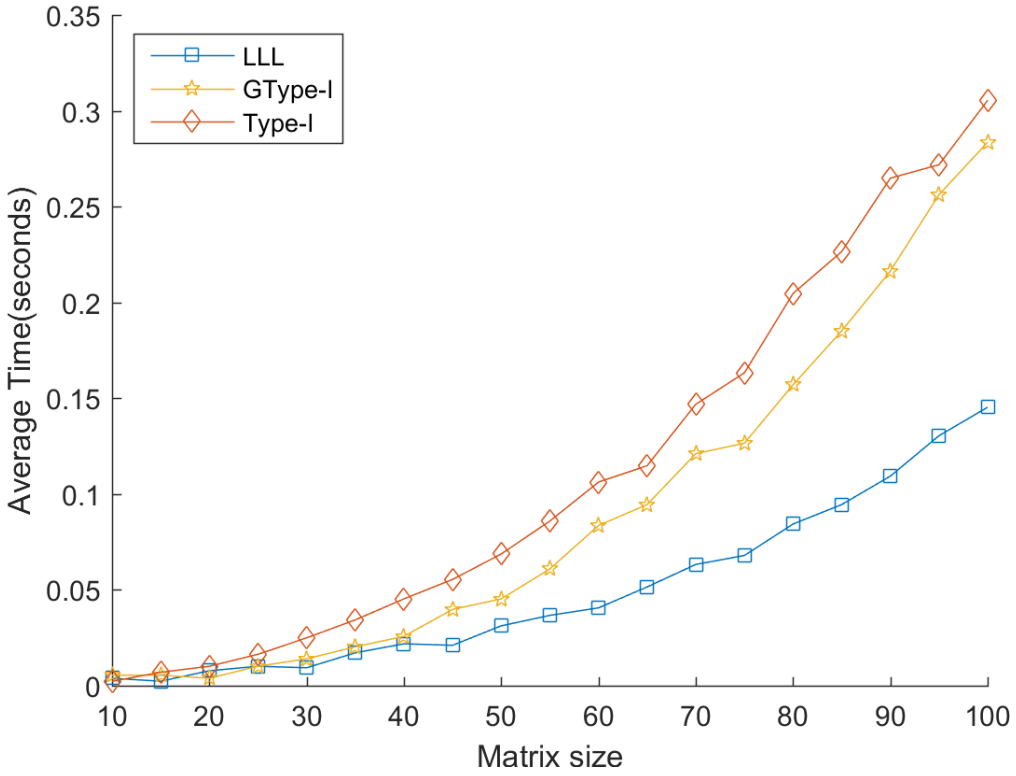

Figure 3.1: Running Times of Applying the LLL, GType-I and the Type-I on $n \times n$ Bases.

both the LLL algorithm with $\omega = 0.99$ and the Type-I algorithm with $\delta = 0.99$ on the 20 random bases only once then calculate the average values of the running time, the orthogonality defect and the Euclidean norm of a shortest vector in the reduced basis.

Figure 3.1 shows the average running times of applying the LLL the GType-I and the Type-I algorithms on the $n \times n$ bases. When $n = 10$, three algorithms approximately took same time to reduce a basis. However, the LLL algorithm performed better than the GType-I and Type-I algorithms as the value of n grows.

We measure the quality of the reduced basis by the orthogonality defect and the

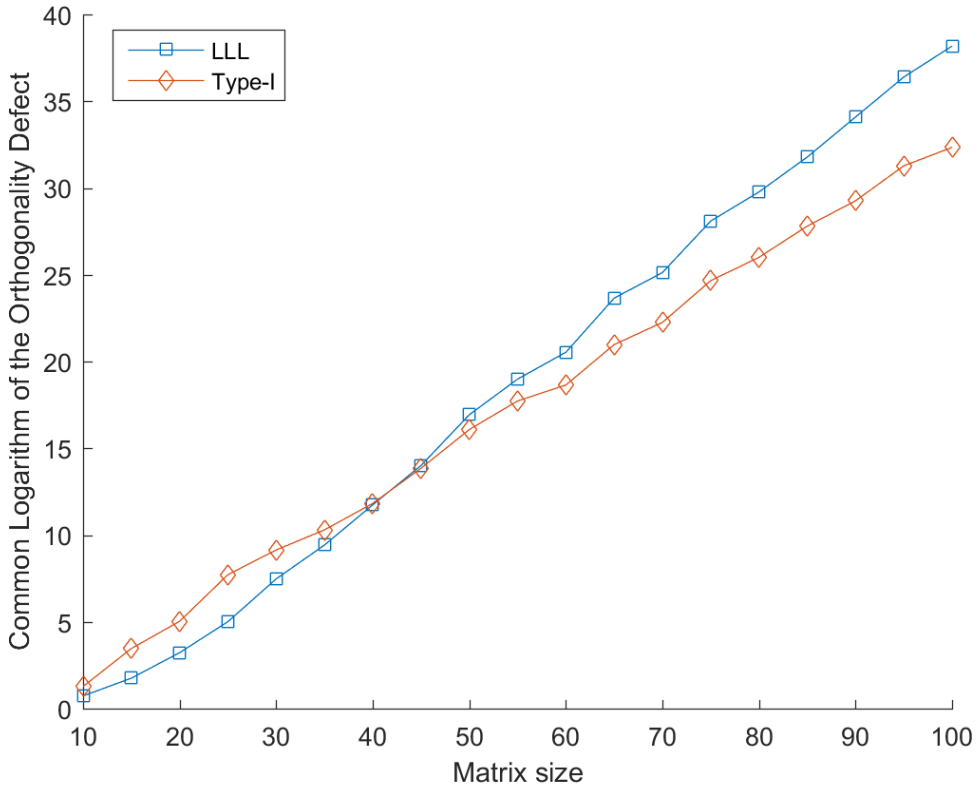

Figure 3.2: Common Logarithm of the Orthogonality Defects of the LLL and the Type-I Reduced Bases

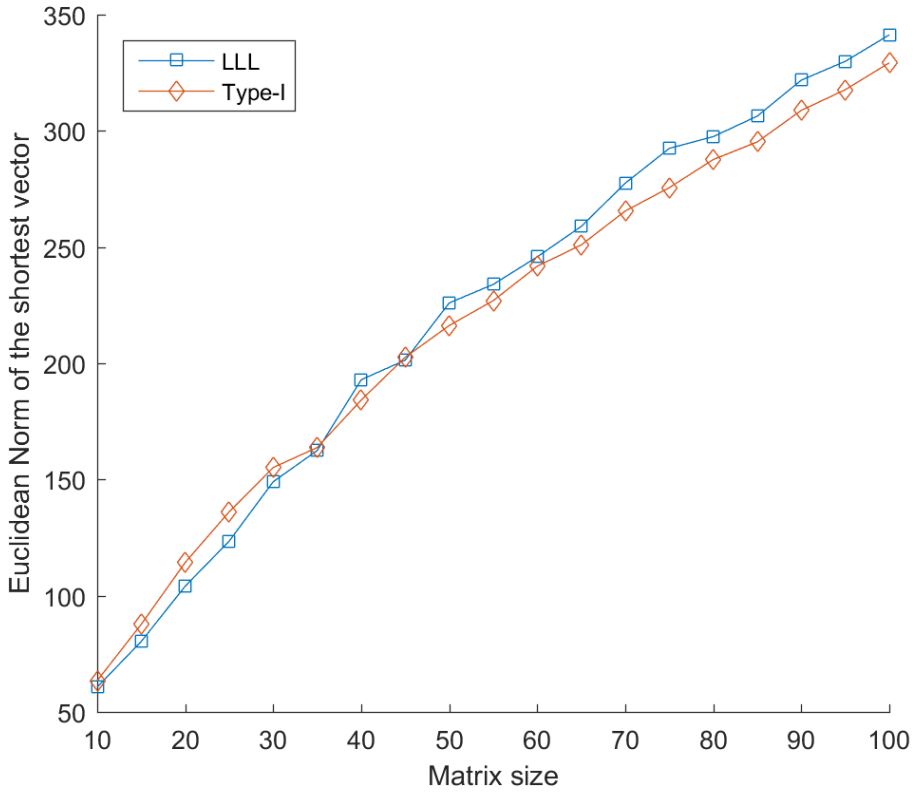

Figure 3.3: Euclidean Norm of a Shortest Vector in the LLL and the Type-I Reduced Bases

Euclidean norm of a shortest vector in the reduced basis. In Figure 3.2, we use the common logarithm of the orthogonality defect as Y-axis, which is denoted as $\log_{10}\gamma(A)$ where A is a matrix. We can see clearly that when $n < 45$, $\log_{10}\gamma(A^L) < \log_{10}\gamma(A^I)$, where $\log_{10}\gamma(A^L)$ is the common logarithms of average values of the orthogonality defect of the LLL reduced matrix and $\log_{10}\gamma(A^I)$ is the common logarithms of average values of the orthogonality defect of the Type-I reduced basis. But when $n \geq 45$, $\log_{10}\gamma(A^I) \leq \log_{10}\gamma(A^L)$, which means the Type-I reduced bases are more orthogonal than the LLL reduced bases. The difference$(\log_{10}\gamma(A^I) - \log_{10}\gamma(A^L))$ becomes larger when n approaches 100. The Euclidean norm of a shortest vector in reduced bases is shown in Figure 3.3. The LLL almost performs as well as the Type-I algorithm. When $10 \leq n \leq 45$, the LLL finds shorter vectors than the Type-I algorithm. However, when $45 < n \leq 100$, the Type-I algorithm finds short vectors. Overall, we can say that the LLL runs faster than the Type-I, but in high dimensional, the Type-I algorithm generates better reduced bases than the LLL algorithm.

Chapter 4

Solving the Subset Sum Problem

In the previous chapter, we introduced two lattice basis reduction algorithms and their implementations, the length bounds of shortest vectors and the time complexities of two lattice basis reduction algorithms. In this chapter, we introduce the subset sum problem and outline some methods for solving the problem. At last, we will show how to use the Type-I reduction algorithm in conjunction with the LLL algorithm and other techniques to attack the subset sum problem.

4.1 The Subset Sum Problem

Definition 4.1. *Given a set of positive integers(the weights)* $W = \{w_1, w_2, \ldots, w_n\}$ *and a positive integer* s*, the subset sum problem is to find a set* $E = \{e_1, e_2, \ldots, e_n\}$*, where* $e_i \in \{0, 1\}, 1 \leq i \leq n$*, such that*

$$s = \sum_{i=1}^{n} w_i e_i.$$

In other words, the subset sum or the knapsack problem is to find, given the weight W and the sum s, some subset W' of W, such that the sum of the elements in W' equals s.

The subset sum problem, in general, is NP-complete [9, 6]. Many subset sum based cryptosystems are built on the difficulty of solving the subset sum problem, but almost all of these cryptosystems have been shown to be insecure [20, 21, 11]. There are two famous independent attacks that attempt to solve the certain type of subset sum problems, one due to Lagarias and Odlyzko [10] and one due to Radziszowski and Kreher [24].

Definition 4.2. *The density d of a weight set $W = \{w_1, w_2, \ldots, w_n\}$ is defined by*

$$d = \frac{n}{\log_2 \max\limits_{1 \le i \le n} w_i}. \tag{4.1}$$

Although Brickell and Lagarias-Odlyzko attacks depend in theory only on the density of the subset sum problem, in practice, the success rate of these methods is still bounded by the performance of lattice reduction technique used in attack.

4.2 Previous Results of the Subset Sum Problem

In 1984 and 1985, two algorithms have been proposed independently, one by Brickell [1] and another by Lagarias and Odlyzko [10]. Different from the majority of attacks on the subset sum based cryptosystems which exploit the specific construction of the cryptosystems, these algorithms show that almost all low-density subset problems may be solved in polynomial time if a polynomial time algorithm(lattice oracle) for

finding a shortest non-zero vector in a lattice could be invoked. The density of subset problems d is defined in (4.1), we are only interested in the problems where $d \leq 1$, since if $d > 1$, there will, in general, be many subsets of weights for the same sum s, therefore, such sets cannot be used for transmitting information.

Both Brickell and Lagrias-Odlyzko attacks reduce the subset sum problem to the problem of finding a shortest nonzero vector in a lattice, a lattice basis reduction algorithm, the LLL algorithm, is applied to produce a reduced basis for a lattice that contains a shortest nonzero vector. Given a lattice basis as input, consider a lattice oracle with high probability finds in polynomial time a shortest nonzero vector in the given lattice. We do not know how to construct such an oracle, but it is shown in [10, 24] that the LLL algorithm acts like such an oracle in relatively low dimensions. In [10], the analysis shows that it is possible to solve all subset problems of density $d < 0.6463\ldots$ in polynomial time, but no higher than that. In 1992, Coster, LaMacchia, Odlyzko and Schnorr [4] and Joux and Stern [8] independently demonstrated via different techniques that the bound could be improved to $d < 0.9408\ldots$.

The Lagarias-Odlyzko attack proceeds as follows. Let M be a positive integer and $\{w_1, w_2, \ldots, w_n\}$ be a set of random integers with $0 < w_i \leq M$ for $1 \leq i \leq n$. Let the set $E = \{e_1, e_2, \ldots, e_n\} \in \{0,1\}^n$ and $E \neq \{0, 0, \ldots, 0\}$ be fixed and depend only on n, also let

$$s = \sum_{i=1}^{n} e_i w_i, \qquad\qquad t = \sum_{i=1}^{n} w_i.$$

We may assume

$$\frac{1}{n}t \leq s \leq \frac{n-1}{n}t, \tag{4.2}$$

because if $s < \frac{t}{n}$ then any $w_i \geq \frac{t}{n}$ cannot be in the subset W'. If $s > \frac{n-1}{n}t$ then every $w_i \geq \frac{t}{n}$ must be in subset W'. In both cases, the complexity of solving the subset sum problem is reduced. Now, define a matrix $A = [a_1, a_2, \ldots, a_n, a_{n+1}] \in R^{n+1 \times n+1}$ as follows:

$$A = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 \\ Nw_1 & Nw_2 & Nw_3 & \ldots & Nw_n & Ns \end{bmatrix}, \tag{4.3}$$

where N is a positive integer and $N > \sqrt{n}$. Let L be the lattice generated by A, that is,

$$L = \{\sum_{i=1}^{n+1} z_i a_i, \quad \text{where } z_i \in \mathbb{Z} \text{ and } 1 \leq i \leq n+1\}.$$

Notice that the solution vector $\hat{e} = [e_1, e_2, \ldots, e_n, 0]^T$ is in L, since

$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 \\ Nw_1 & Nw_2 & Nw_3 & \ldots & Nw_n & Ns \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \\ -1 \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \\ 0 \end{bmatrix}, \tag{4.4}$$

and $\hat{e}$ is believed a "short" vector in L. Then the Lagarias-Odlyzko attack uses the LLL algorithm as the lattice oracle to find the solution vector $\hat{e}$.

Lagarias and Odlyzko also derive the theoretical bound of the attack. Let P be the

probability that there exist another vector $\hat{x} = [x_1, x_2, \ldots, x_n, x_{n+1}]^T$ which satisfies:

$$\begin{aligned} \hat{x} &\in L, \\ \|\hat{x}\|_2 &\leq \|\hat{e}\|_2, \\ \hat{x} &\notin \{0, \hat{e}, -\hat{e}\}. \end{aligned} \tag{4.5}$$

Following the proof in [5], we only consider the situation that

$$\sum_{i=1}^{n} e_i \leq \frac{1}{2}n,$$

because if $\sum_{i=1}^{n} e_i > \frac{1}{2}n$, we can let $s' = t - s$, $e' = [e'_1, e'_2, \ldots, e'_n]^T$ where $e'_i = 1 - e_i$, for $1 \leq i \leq n$, then the solution vector will be $\hat{e}' = [e'_1, e'_2, \ldots, e'_n, 0]^T$. Clearly, $\sum_{i=1}^{n}(1 - e_i) \leq \frac{1}{2}n$. It is not hard to see that $\hat{x}$ satisfies (4.5) only if $x_{n+1} = 0$, otherwise $\|\hat{x}\|_2 \geq |x_{n+1}| \geq N > \sqrt{n} \geq \|\hat{e}\|_2$, contradicting the hypothesis (4.5). Similar to (4.4), we can get a new equation for $\hat{x}$,

$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 \\ Nw_1 & Nw_2 & Nw_3 & \ldots & Nw_n & Ns \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ -y \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 0 \end{bmatrix},$$

where y is an integer. It is clear that $\sum_{i=1}^{n} w_i x_i = sy$, it follows:

$$\begin{aligned} s|y| &= \left| \sum_{i=1}^{n} w_i x_i \right| \\ &\leq \|\hat{x}\|_2 \sum_{i=1}^{n} w_i \\ &\leq t\sqrt{\frac{1}{2}n}, \end{aligned}$$

notice that $\sum_{i=1}^{n} w_i = t$ and $\|\hat{x}\|_2 \leq \|\hat{e}\|_2 \leq \sqrt{\frac{1}{2}n}$. Therefore, by (4.2) above, we get

$$|y| \leq n\sqrt{\frac{1}{2}n}. \tag{4.6}$$

Notice that P denotes the probability of there exist a vector $\hat{x}$ which satisfies (4.5), let $x = [x_1, x_2, \ldots, x_n]^T$ denote an element of $\mathbb{Z}^n$ since $\hat{x} \in L$. By the definition of lattice, $x_1, x_2, \ldots, x_n$ are all integers, then $\|\hat{x}\|_2 = \|x\|_2$ and as a special case we have $\|\hat{e}\|_2 = \|e\|_2$, where $e = [e_1, e_2, \ldots, e_n]^T$. In [4], the probability P is estimated by

$$\begin{aligned} P &\leq Pr\left(\exists \ x, y \ \text{ s.t. } \ \|x\|_2 \leq \|e\|_2, |y| \leq n\sqrt{\frac{1}{2}n}, x \notin \{0, e, -e\}, \sum_{i=1}^{n} x_i w_i = sy \right) \\ &\leq Pr\left(\sum_{i=1}^{n} x_i w_i = sy : 0 < \|x\|_2 < \|e\|_2, |y| \leq n\sqrt{\frac{1}{2}n}, x \notin \{0, e, -e\} \right) \\ &\quad \cdot |\{x : \|x\|_2 \leq \|e\|_2\}| \cdot \left| \{y : |y| \leq n\sqrt{\frac{1}{2}n}\} \right|. \end{aligned} \tag{4.7}$$

For estimating the first part of (4.7) easily, in [4] authors rewrite $\sum_{i=1}^{n} x_i w_i = sy$ as:

$$\sum_{i=1}^{n} w_i z_i = 0, \quad \text{where } \ z_i = x_i - y e_i.$$

We have $z = [z_1, z_2, \ldots, z_n]^T \neq 0$ because $x \neq 0$ and $\|x\|_2 \leq \|e\|_2$. We assume without loss of generality that $z_1 \neq 0$, let $z' = -\left(\frac{\sum_{i=2}^{n} w_i z_i}{z_1}\right)$ then

$$\begin{aligned} Pr\left(\sum_{i=1}^{n} w_i z_i = 0\right) &= Pr\left(a_1 = z'\right) \\ &= \sum_{j=1}^{M} Pr\left(w_1 = z' | z' = j\right) Pr\left(z' = j\right) \\ &= \sum_{j=1}^{M} Pr\left(w_1 = z'\right) Pr\left(z' = j\right) \\ &= \sum_{j=1}^{M} \frac{1}{M} Pr\left(z' = j\right) \\ &\leq \frac{1}{M}. \end{aligned} \tag{4.8}$$

The second part of (4.7) is estimated in [10] as:

$$\begin{aligned} |\{x : \|x\|_2 \leq \|e\|_2\}| &\leq \left|\{x : \|x\|_2 \leq \sqrt{\frac{1}{2}n}\}\right| \\ &\leq 2^{c_0 n}. \qquad \text{where } c_0 = 1.54724\ldots. \end{aligned} \tag{4.9}$$

It is clear that the last part of (4.7) can be estimated by

$$2n\sqrt{\frac{1}{2}n} + 1. \tag{4.10}$$

Combining (4.8) (4.9) (4.10), we can get

$$P \leq n\left(2n\sqrt{\frac{1}{2}n} + 1\right)\frac{2^{c_0 n}}{M}, \qquad \text{where } c_0 = 1.54724\ldots,$$

this implies if $M = 2^{cn}$ with $c > c_0$, $\lim_{n\to\infty} P = 0$. Also, if the density of a subset sum problem is less than $0.6463\ldots$, then

$$\frac{n}{\log_2 \max\limits_{1\le i\le n} w_i} < 0.6463\ldots \Longrightarrow \max_{1\le i\le n} w_i > 2^{\frac{n}{0.6463\ldots}} \Longrightarrow M > 2^{c_0 n}. \tag{4.11}$$

Therefore, all subset sum problems with density less than $0.6463\ldots$ can be solved in polynomial time if given a lattice oracle.

In 1991, two improvements of Lagarias-Odlyzko attack have been presented by Coster, LaMacchia, Odlyzko and Schnorr [4] and by Joux and Stern [8], both of them increase the bound of density to $d < 0.9408\ldots$. In [4], the modification suggested is to replace $a_{n+1} = [0, 0, \ldots, 0, Ns]^T$ by $a'_{n+1} = [\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2}, Ns]^T$, therefore the new matrix is

$$A' = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & \frac{1}{2} \\ 0 & 1 & 0 & \ldots & 0 & \frac{1}{2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & \frac{1}{2} \\ Nw_1 & Nw_2 & Nw_3 & \ldots & Nw_n & Ns \end{bmatrix}.$$

Let L' be the lattice generated by A'. Similar to (4.4), the new equation is:

$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & \frac{1}{2} \\ 0 & 1 & 0 & \ldots & 0 & \frac{1}{2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & 0 & \ldots & 1 & \frac{1}{2} \\ Nw_1 & Nw_2 & Nw_3 & \ldots & Nw_n & Ns \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \\ -1 \end{bmatrix} = \begin{bmatrix} e_1 - \frac{1}{2} \\ e_2 - \frac{1}{2} \\ \vdots \\ e_n - \frac{1}{2} \\ 0 \end{bmatrix},$$

therefore, L' does not contain the solution vector $\hat{e}$, instead it contains the vector $\hat{e}' = [e'_1, e'_2, \ldots, e'_n, 0]^T = [e_1 - \frac{1}{2}, e_n - \frac{1}{2}, \ldots, e_n - \frac{1}{2}, 0]^T$. Similar to (4.5), now we are interested in the probability P' that there exists a vector $\hat{x}'$ which satisfies the following:

$$\hat{x}' \in L,$$
$$\|\hat{x}'\|_2 \leq \|\hat{e}'\|_2 \leq \frac{1}{2}\sqrt{n},$$
$$\hat{x}' \notin \{0, \hat{e}', -\hat{e}'\},$$

because $e'_i \in \{-\frac{1}{2}, \frac{1}{2}\}$ for $1 \leq i \leq n$, then $\|\hat{e}'\|_2^2 \leq \frac{1}{4}n$. In [4], it is shown that the probability P' is bounded by:

$$P' \leq n(4n\sqrt{n}+1)\frac{2^{c'_0 n}}{M} \qquad \text{where } c'_0 = 1.0628\ldots,$$

by using the similar techniques to [10, 17, 5]. Since $\frac{1}{c'_0} = 0.9408\ldots$, similar to (4.11), we get

$$\frac{n}{\log_2 \max\limits_{1\leq i\leq n} w_i} < 0.9408\ldots \Longrightarrow \max_{1\leq i\leq n} w_i > 2^{\frac{n}{0.9408\ldots}} \Longrightarrow M > 2^{c'_0 n}.$$

Therefor any subset sum problem with density $d < 0.9408\ldots$ may be solved by a lattice oracle in polynomial time.

4.3 Previous Empirical Methods for Solving the Subset Sum Problem

Lagarias and Odlyzko [10] presented the results of first empirical attacks on the general subset sum problem along with their theoretical results in 1985. Figure 4.1 is a flowchart representation of the Lagarias-Odlyzko attack. They first fixed the vector e as follows:

$$e = [e_1, e_2, \ldots, e_n]^T \qquad \text{where } e_i = 1, \text{ if } 1 \leq i \leq \frac{n}{2} \text{ and } e_i = 0, \text{ otherwise.}$$

Then a basis A (4.3) is generated using e and the set of random integers $\{w_1, w_2, \ldots, w_n\}$. After that, a multiprecision version of the LLL algorithm was applied to A. The LLL parameter ω was set to 1 instead of 0.75, they claimed when $\omega = 1$ the LLL algorithm was not proved to run in polynomial time, but in practice, it just takes three times as long as the LLL algorithm with $\omega = 0.75$, and it seems to find much shorter vectors than the LLL algorithm with $\omega = 0.75$. After the LLL reduction, their algorithm tried to find the solution vectors $\pm\hat{e} = \pm[e_1, e_2, \ldots, e_n, 0]^T$ in the reduced basis. In cases of failure, the LLL algorithm was run again to another permutation of the basis A (4.3) and this was repeated for up to five trials since different initial orderings generated different LLL-reduced bases. If all these five trials failed, the basis A was redefined with $s = \sum_{i=1}^{n} w_i - s$ and the process was repeated that is shown in Figure 4.1.

In 1988, Radziszowski and Kreher [24] described an improvement of Lagarias and Odlyzko method. The new method has two significant advantages: running time is an order of magnitude shorter and can attack higher density subset sum problems. Figure 4.2 presents the procedure of the Radziszowski-Kreher attack. The input basis

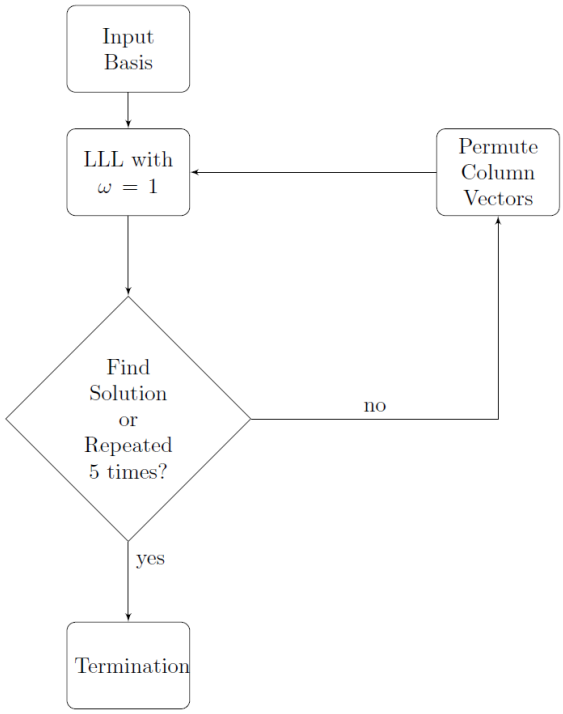

Figure 4.1: Lagarias-Odlyzko Attack Process

B of the Radziszowski-Kreher attack equals A (4.3) except that the last row of B is $[w_1, w_2, w_3, \ldots, w_n, s]$. A modified multiprecision algorithm was applied to B which is called the MP algorithm. The MP algorithm is built on one important equation: let basis B be the input matrix, then

$$\begin{bmatrix} w_1 & w_2 & \ldots & w_n \end{bmatrix} \begin{bmatrix} b_{1,i} \\ b_{2,i} \\ \vdots \\ b_{n,i} \end{bmatrix} = k_i s + b_{n+1,i}, \tag{4.12}$$

where $w_1, w_2, \ldots, w_n$ are weights of the subset sum problem, $k_i = 0$ if $1 \leq i \leq n$ and $k_i = -1$ if $i = n + 1$. Let B^L be the reduced basis after running the LLL algorithm with input B, there exist some integers k_i where $1 \leq i \leq n + 1$, such that (4.12) remains true since the only operations performed in the LLL algorithm are the Decrease and Swap-Restore and equation (4.12) is an invariant of both of them. As it is shown in [24], the MP algorithm can reduce the number of required multiprecision calculations by a divide and conquer approach, all calls to the LLL algorithm($\omega = 0.99$) in the MP are executed entirely in single precision arithmetic. Let $t =$ an integer smaller than the machine word(a word is the amount of data that a machine can process at one time) size, for example, $t = 28$ for a $32-$bit computer, which can reduce the possibility of raising the overflow errors during the calculation. $top = \lfloor \log_2 \text{Max}(a_{n+1,1}, a_{n+1,2}, \ldots, a_{n+1,n}, a_{n+1,n+1}) \rfloor + 1$, $k = \text{Max}(top - t, 0)$ and the last row of B is changed to $[\frac{b_{n+1,1}}{2^k}, \frac{b_{n+1,2}}{2^k}, \ldots, \frac{b_{n+1,n+1}}{2^k}]$. In essence, the MP algorithm call the LLL algorithm approximate top/t times on B. The resulting basis of each call of the LLL will be the input of next call. Please refer to [24] for detailed explanations.

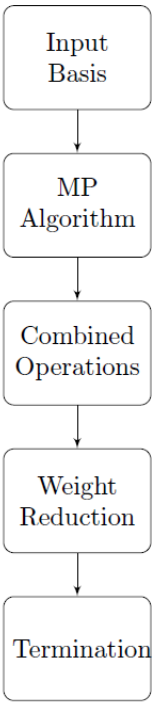

Figure 4.2: Radziszowski-Kreher Attack Process

After applied the MP algorithm to B, as it is shown in Figure 4.3, a combinational process includes solution check, weight-reduction, sort and the LLL algorithm was run on that LLL-reduced basis. They defined the total weight(the squared Frobenius norm) of the basis B as follows:

$$\text{weight}(B) = \sum_{i=1}^{n+1} \|b_i\|_2^2.$$

The weight-reduction procedure searches pairs of vectors b_i and b_j in the input basis

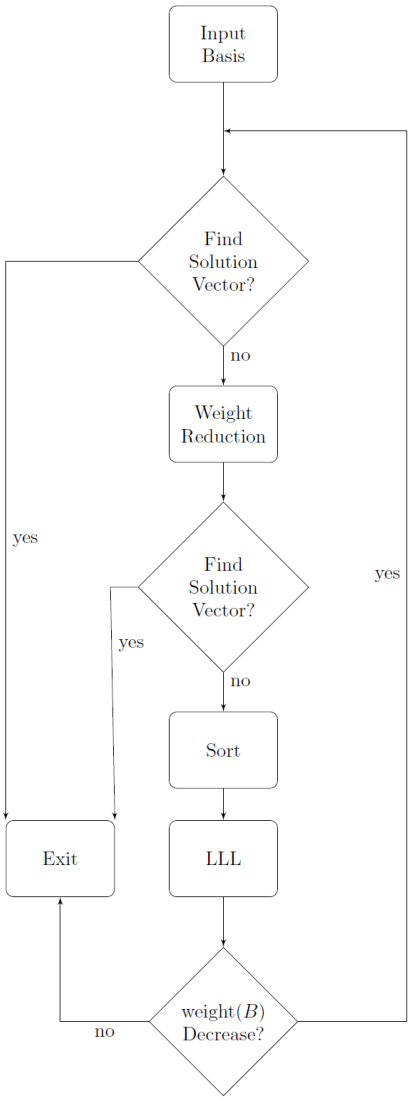

Figure 4.3: Combined Operations

satisfies:

$$\|b_i + b_j\|_2 < \max(\|b_i\|_2, \|b_j\|_2) \ \textit{or} \ \|b_i - b_j\|_2 < \max(\|b_i\|_2, \|b_j\|_2),$$

when such pair is found, then the longer(in terms of square-norm) of b_i and b_j is replaced by $b_i + b_j$ or $b_i - b_j$. Another important feature of the Radziszowski-Kreher attack is: each time before the LLL algorithm is run, the vectors in the output basis of the weight-reduction procedure is sorted in increasing order of length. Therefore, the length of the shortest vector in the basis is guaranteed not to increase by the application of the LLL algorithm. Because if vector b_1 is a shortest vector in the output basis of sort procedure, the LLL can replace b_1 with vector b_2 in lattice if and only if

$$\|b_2\|_2 < \sqrt{\omega}\|b_1\|_2, \quad \omega = 0.99 \text{ in Radziszowski-Kreher attack.}$$

This may not hold for every b_i, but it is true for b_1. If such combinational process cannot reduce weight(B) and the combinational procedure can not find solution vectors $\pm\hat{e}$, the Radziszowski-Kreher attack ran the final weight-reduction and the solution check processes, and then algorithm terminated. Similar to the Lagarias-Odlyzko attack, if solution vectors $\pm\hat{e}$ were not found after the whole process, the attack is repeated on the redefined basis B which with $s = \sum_{i=1}^{n} w_i - s$.

4.4 Using the Type-I Algorithm to Solve the Subset Sum Problem

Both the Lagarias-Odlyzko attack [10] and the Radziszowski-Kreher attack [24] use the LLL algorithm almost exclusively to perform the lattice reduction. As shown by the analysis of the LLL and the Type-I algorithms in Chapter 3, the Type-I algorithm alone does not perform well for solving the subset sum problem. In this section, we propose a hybrid method, which integrates the Type-I, the LLL and other techniques, to find a shortest vector. We define an initial matrix similar to the one used by Coster, LaMacchia, Odlyzko and Schnorr in [4]:

$$A = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & \frac{1}{2} \\ 0 & 1 & 0 & \dots & 0 & \frac{1}{2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & \frac{1}{2} \\ w_1 & w_2 & w_3 & \dots & w_n & s \end{bmatrix}. \tag{4.13}$$

Note that the solution vectors of (4.13) should have the form $\hat{e} = [e_1 - \frac{1}{2}, e_2 - \frac{1}{2}, \dots, e_n - \frac{1}{2}, 0]^T$ or $-\hat{e} = [-e_1 + \frac{1}{2}, -e_2 + \frac{1}{2}, \dots, -e_n + \frac{1}{2}, 0]^T$. As pointed out in [12], the algorithms of [10, 24, 4] search vectors of the form $(e_1, e_2, \dots, e_n, 0)$. A problem with these methods is that often the last element of the short vectors produced by the LLL reduction not equals zero, which means that the sum $\sum_{i=1}^{n} w_i x_i$ does not describe any relation with the target subset sum problem. One way to reduce the possibility that the LLL algorithm(or the Type-I algorithm) produces such vectors in the reduced basis is to scale all w_i and s by some constant N. This method

increases the length of any vector having a nonzero last element by about $\sqrt{N}$ if N is sufficiently large. This is the reason of the last row of the initial matrix defined in [10, 24, 4] has the form $[Nw_1, Nw_2, Nw_3, \ldots, Nw_n, Ns]$. However, this approach has one disadvantage: it increases the size of numbers that are already large, therefore the multiprecision arithmetic is required. In [12], a new approach is introduced to eliminate the consideration of all lattice vectors with $e_{n+1} \neq 0$: the GCD-reduction.

The GCD-reduction performs column operations on the lattice basis to make the entire $n + 1^{th}$ row contains only one nonzero element, the GCD of the weights $w_1, w_2, \ldots, w_n$. This idea of implementing the GCD-reduction is described by Brun [2]. We use matrix A (4.13) as the input of the GCD-reduction algorithm, the first step is to sort the column vectors of A in order of decreasing $a_{n+1,i}$ where $1 \leq i \leq n+1$. Then a_2 is repeatedly subtracted from a_1 until $a_{n+1,2} > a_{n+1,1}$. The column vectors are then sorted in decreasing order of $a_{n+1,i}$ again and the process repeats. Eventually, the only nonzero element will be $a_{n+1,1}$ and $a_{n+1,1} = GCD(w_1, w_2, \ldots, w_n)$. After the GCD-reduction procedure, we can remove the vector a_1 and all $a_{n+1,i}$ where $2 \leq i \leq n + 1$ from A, which reduces an n+1 dimensional lattice basis A to an n dimensional lattice basis A_G. Thus, the solution vectors of A_G are $\hat{e}_G = [e_1 - \frac{1}{2}, e_2 - \frac{1}{2}, \ldots, e_n - \frac{1}{2}]^T$ or $-\hat{e}_G = [-e_1 + \frac{1}{2}, -e_2 + \frac{1}{2}, \ldots, -e_n + \frac{1}{2}]^T$. Figure 4.4 shows that we apply the GCD-reduction to A before the Type-I reduction because the GCD-reduction requires all $a_{n+1,i}$ where $1 \leq i \leq n + 1$ be positive integers. If we applied the Type-I algorithm first, then the output basis would include negative integers in the $(n+1)^{th}$ row. After the GCD-reduction is applied to the basis A, the process enters the I&LLL phase.

Figure 4.5 shows the details in the I&LLL phase. The input to this phase is the lattice basis A_G which is the result of the GCD-reduction. We apply the Type-I

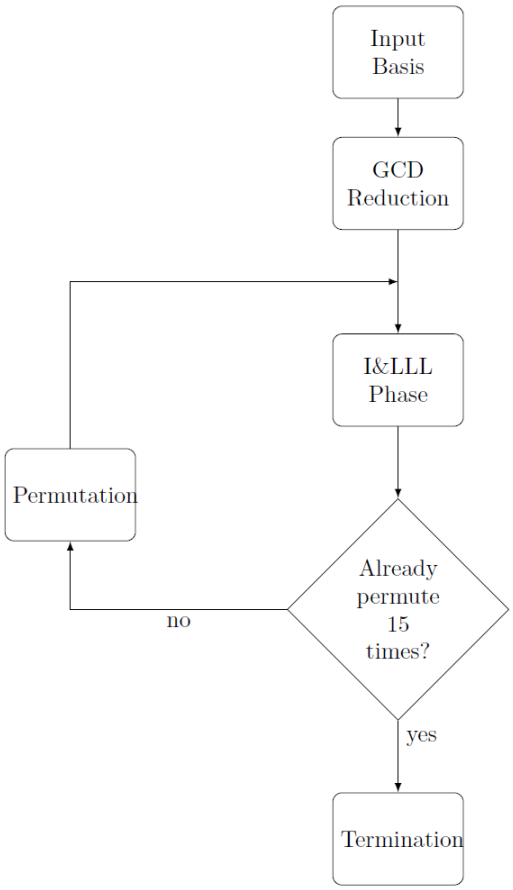

Figure 4.4: Type-I & LLL Attack Process

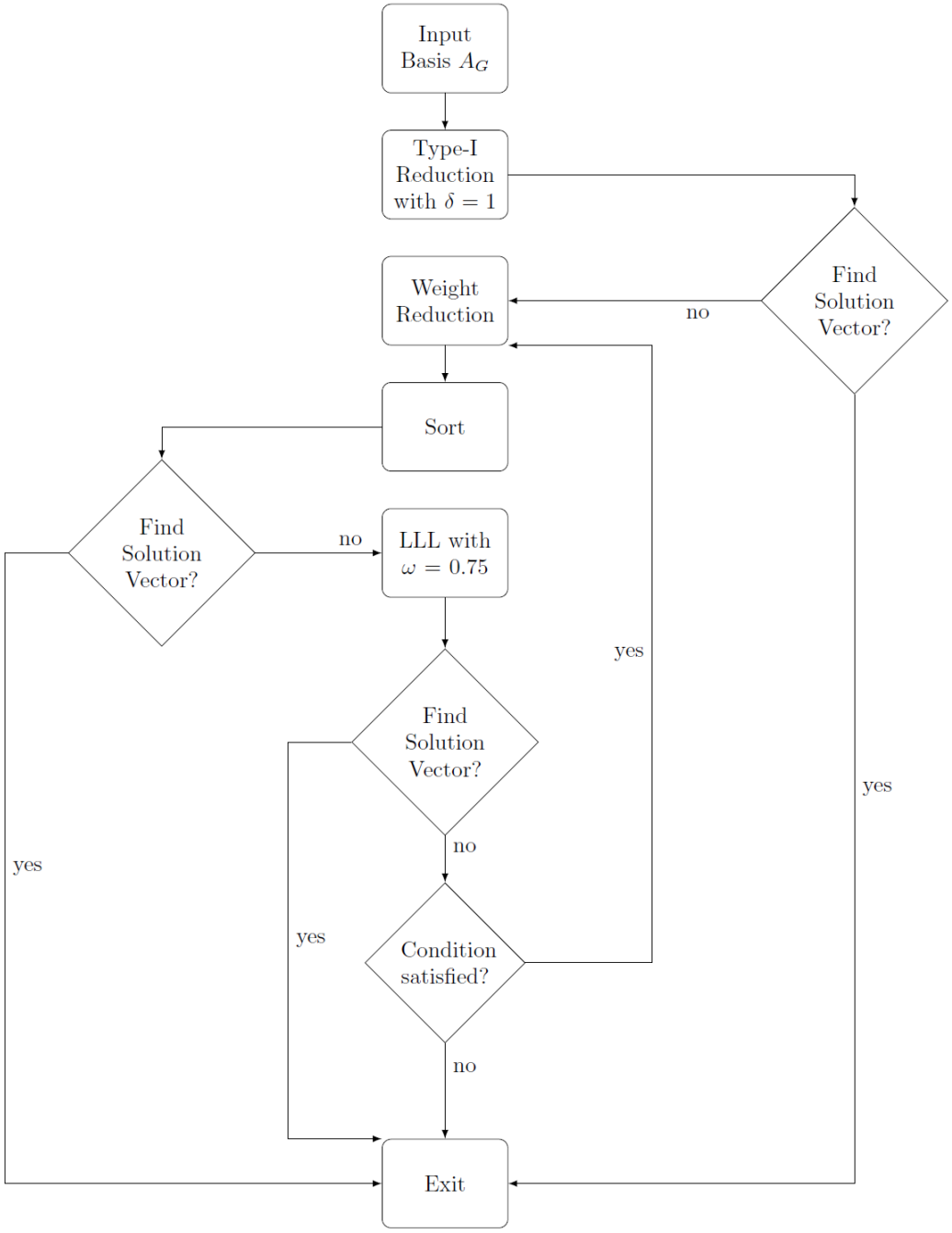

Figure 4.5: I&LLL Phase

reduction with parameter $\delta = 1$ to A_G. In our experiments, the Type-I & LLL algorithm still terminates even $\delta = 1$. Recalling the property of the Type-I reduction, when $\delta = 1$, the acute angle between any pair of the Type-I reduced basis vectors is guaranteed at least 60°. When the value of n is small ($n \leq 15$), with high probability that solution vectors $\pm\hat{e}_G$ will be in this Type-I reduced basis A_I, thus we search for the solution vectors in A_I. If solution vectors are not found, then a number of iterations of the weight-reduction-sort-LLL operation are applied to the basis A_I. By calling the sort(sort just sorts the basis vectors by length in increasing order) after the weight-reduction, the shortest vector in A_I is the first column of the basis A_I. Hence the length of it is guaranteed not to increase by following iterations of the weight-reduction-sort-LLL operation. Recalling the Radziszowski-Kreher attack from Section 4.3, they show that iterations of the weight-reduction-sort-LLL (Figures 4.2, 4.3) yield better results than a single application of the LLL algorithm. The Type-I & LLL algorithm incorporates this approach. The weight-reduction-sort-LLL loop stops when certain conditions are satisfied and we will specify the conditions later. In each iteration of the weight-reduction-sort-LLL, we use the LLL algorithm with $\omega = 0.75$. Although in [10, 24, 4], the LLL algorithm is run with parameter $\omega \approx 0.99$, Lenstra, Lenstra and Lovász [13] show that ω could be any value between in the range $\frac{1}{4} < \omega < 1$. After each iteration of the weight-reduction-sort-LLL, we compute the boolean value of the following expression, which is first used by LaMacchia in [12]. Let $A_{wsl,in}$ be the input basis of each iteration of the weight-reduction-sort-LLL loop and $A_{wsl,out}$ be the resulting basis of each iteration of the

weight-reduction-sort-LLL loop,

$$\begin{aligned}&(min\{\|a_i\|_2 \in A_{wsl,out}\} < min\{\|a_i\|_2 \in A_{wsl,in}\}) \\ &\quad \text{OR } ((min\{\|a_i\|_2 \in A_{wsl,out}\} = min\{\|a_i\|_2 \in A_{wsl,in}\}) \\ &\quad \text{AND } (max\{\|a_i\|_2 \in A_{wsl,out}\} < max\{\|a_i\|_2 \in A_{wsl,in}\})).\end{aligned} \tag{4.14}$$

If the value of (4.14) is true, then the output basis $A_{wsl,out}$ is the input again into the weight-reduction-sort-LLL loop. Thus, we perform the weight-reduction-sort-LLL loop until either the length of the shortest vector in basis has increased after the weight-reduction-sort-LLL loop or if the length of the shortest vector has not changed and the length of the longest vector has not decreased. The looping structure in Figure 4.5 and the exit condition in (4.14) make the LLL algorithm perform as many Decrease operations as possible [12]. In [13], Lagarias and Odlyzko show that applying the LLL algorithm multiple times on input bases with different orders can improve the success rate. The Type-I & LLL algorithm incorporates this approach. If no solution vectors are found in the I&LLL phase, we randomly permute the column vectors in A_G then apply the I&LLL phase again on it. Based on our experiments and the trade-off between the running time and the success rates, we set the maximum number of permutations to 15. If all these 15 trials fail, then we redefine the initial basis A with $s = \sum_{i=1}^{n} w_i - s$ and apply the attack (Figure 4.4) again on the modified basis A.

4.5 The Experimental Result of the Type-I & LLL Attack

In the previous section, we have detailed the operations of the Type-I & LLL algorithm, a combination of the Type-I and the LLL algorithm. Also, we have utilized the GCD-reduction, the weight-reduction and the sort approach. In this section, we will present the results of our experiments.

All algorithms of the Type-I & LLL, including the GType-I(Algorithm 5) and the LLL algorithm(Algorithm 3) in Chapter 3 are implemented in 64-bit MATLAB R2015b. All lattice bases are stored in double precision floating point format. Recall from Section 4.2 that if the density of the subset sum problem is greater than 1, then generally, there are many subsets W' of weights with the same sum s. Therefore such set cannot be used for transmitting information. In our experiment, we set the size of the subset sum problem $10 \leq n \leq 43$ and keep the density $d \leq 1$. Although the Type-I and the LLL algorithms both need dot multiplication of vectors which may lead rounding error, this implementation of the Type-I & LLL algorithm is satisfactory for our purpose if the value of $n \leq 43$. When $n \geq 43$, the success rates drop rapidity.

Following the test results of [10, 24], we attempt to solve some random subset sum problems of various values of n. For each value of n, we choose a set of b values, where each b represents the number of bits in the binary representation of the maximum weight. The Type-I & LLL algorithm is run on some randomly generated subset sum problems for each pair of (n, b). If it is possible, we choose the pairs of n and b corresponding to the (n, b) pairs that are used in [10, 24].

Table 4.1: Test Result of the Type-I & LLL Algorithm for $10 \leq n \leq 30$

20 trials for each density d for each size n					
size n	bits b	density d	number of successes	success rate	average time (in seconds)
10	10	1.000	20	1.00	0.0273
	20	0.500	20	1.00	0.0437
	30	0.333	20	1.00	0.0945
	40	0.250	20	1.00	0.1218
	50	0.200	19	0.95	0.1421
14	14	1.000	20	1.00	0.0679
	16	0.875	20	1.00	0.0710
	20	0.700	20	1.00	0.0890
	30	0.466	20	1.00	0.0750
	40	0.350	20	1.00	0.1125
	50	0.280	20	1.00	0.1179
20	20	1.000	20	1.00	0.1328
	21	0.952	20	1.00	0.1906
	24	0.833	20	1.00	0.1828
	30	0.667	20	1.00	0.1445
	36	0.555	20	1.00	0.1851
	40	0.500	20	1.00	0.1960
	50	0.400	20	1.00	0.1859
26	26	1.000	20	1.00	0.6265
	27	0.963	20	1.00	0.3711
	30	0.866	20	1.00	0.4101
	35	0.743	20	1.00	0.2109
	36	0.722	20	1.00	0.2820
	40	0.650	20	1.00	0.2570
	44	0.590	20	1.00	0.3257
	50	0.520	18	0.90	0.8773
30	30	1.000	19	0.95	1.6625
	31	0.968	20	1.00	1.2156
	36	0.833	20	1.00	0.6507
	40	0.750	20	1.00	0.5835
	44	0.682	20	1.00	0.3906
	50	0.600	16	0.80	1.6086

Table 4.2: Test Result of the Type-I & LLL Algorithm for $34 \leq n \leq 43$

20 trials for each density d for each size n					
size n	bits b	density d	number of successes	success rate	average time in seconds
34	34	1.000	16	0.80	4.1265
	35	0.971	16	0.80	3.4781
	38	0.895	19	0.95	2.8398
	40	0.850	20	1.00	1.9851
	43	0.790	20	1.00	1.5102
	45	0.755	20	1.00	1.1851
37	37	1.000	16	0.80	23.3078
	38	0.974	13	0.65	5.3063
	40	0.925	12	0.60	17.4039
	42	0.881	16	0.80	18.8875
	45	0.822	19	0.95	9.9117
	47	0.787	19	0.95	9.7227
42	42	1.000	4	0.20	13.8984
	44	0.955	3	0.15	14.8679
	46	0.913	2	0.10	15.6937
	48	0.875	8	0.40	14.2898
43	43	1.000	1	0.05	11.3602
	44	0.977	2	0.10	10.7156
	46	0.935	5	0.25	8.4602
	48	0.896	6	0.30	12.8297

Tables 4.1 and 4.2 illustrate the test results of the Type-I & LLL algorithm performs on random subset sum problems with different (n, b). For each problem, we define a $0-1$ set K, where $\lfloor \frac{1}{2}n \rfloor$'s 1 and $n - \lfloor \frac{1}{2}n \rfloor$'s 0 in K. After that, we use MATLAB function **randi()** to randomly generate a set $\{w_1, w_2, \ldots, w_{n-1}\}$, where $1 \leq w_i \leq 2^b$ and $w_i \leq w_{i+1}$, also, let $w_n = 2^b$ to guarantee the density is $\frac{n}{b}$. Then we use the MATLAB function **randperm()** to randomly permute all elements in K and let $s = [k_1, k_2, \ldots, k_n]^T[w_1, w_2, \ldots, w_n]$. After that, we generate a initial matrix A (4.13) with $w_1, w_2, \ldots, w_n$ and s and apply the Type-I & LLL algorithm on A. All experiments are run on 64-bit Windows 10 Pro operating system with the 6^{th} Gen 2.4GHz Intel Core i5-6300U processor and 8GB memory. The columns in Tables 4.1 and 4.2 show the value of following variables:

n: The size of the subset sum problem. The initial matrix A of the Type-I & LLL algorithm has size $(n+1) \times (n+1)$.

b: The number of bits in the binary representation of the maximum weight. Each weight is chosen randomly from 1 to 2^b.

d: The density of the subset sum problem, $d = \frac{n}{b}$.

number of successes: How many subset sum problem are solved by the Type-I & LLL algorithm.

success rate: The success rate measured as a fraction of the number of attempted problems.

average time: The average amount of CPU time(in seconds) required to run the Type-I & LLL algorithm on 20 subset problems, which are generated with a fixed pair (n, b).

From Tables 4.1 and 4.2, it is not hard to see that the Type-I & LLL algorithm has

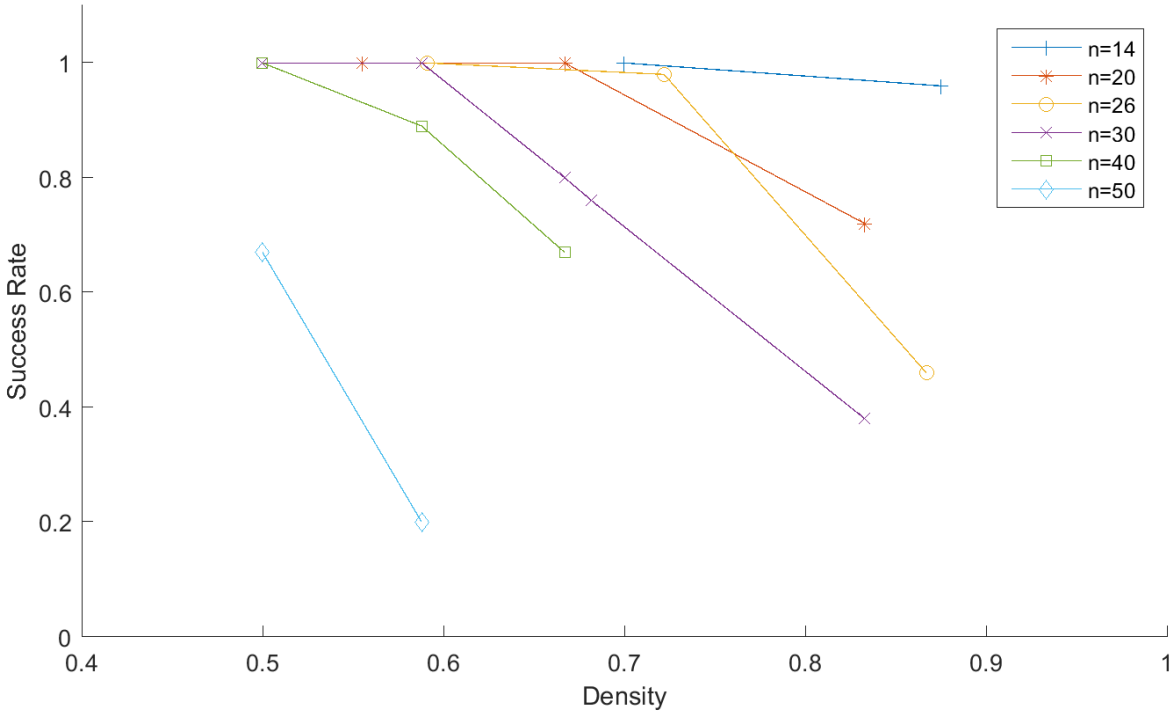

Figure 4.6: Lagarias-Odlyzko Results: Success Rate vs. Density

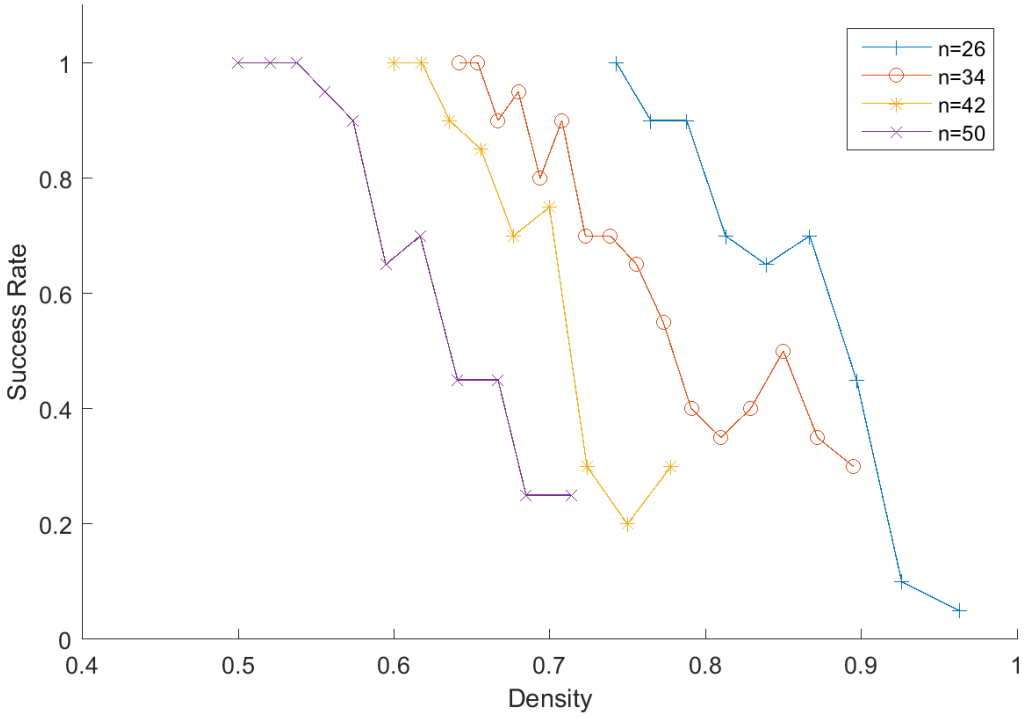

Figure 4.7: Radziszowski-Kreher Results($26 \leq n \leq 50$): Success Rate vs. Density

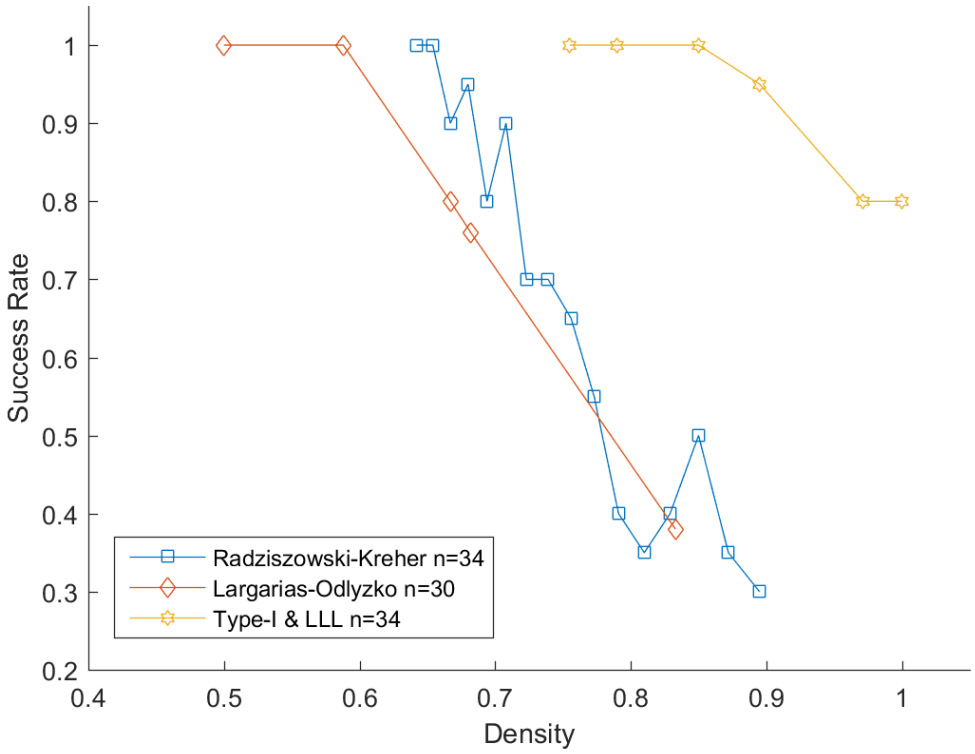

Figure 4.8: Results of the Lagarias-Odlyzko($n = 30$), the Radziszowski-Kreher($n = 34$) and the Type-I & LLL($n = 34$)

great performance for the subset sum problems with small size $n \leq 37$. Figure 4.6 graphically shows the performance of the Lagarias-Odlyzko method, Lagarias and Odlyzko only solve approximate 45% of the problems for $n = 26$, density ≈ 0.9. However, Table 4.1 shows that the Type-I & LLL algorithm solved almost all problems with $n = 26$ and $d \leq 1$. In fact, the Type-I & LLL algorithm works as well as a lattice oracle for all problems with $n \leq 30$. A true lattice oracle should have been able to solve all subset sum problem with the density $d < 0.9408$. However, the Type-I & LLL algorithm still performs well for problems with $n \leq 37$ and $0.9408 < d \leq 1$. Figure 4.7 graphically shows the experimental results in [24]. We can see that if the density of the subset sum problems greater than 0.85, then Radziszowski and Kreher are only able to solve at most approximate 45% of the problems for $n = 26$.

Figure 4.8 graphically compares the results of applying the Radziszowski-Kreher, the Largarias-Odlyzko and the Type-I & LLL algorithm to the subset sum problem with $n = 30$ or $n = 34$. On the one hand, when the density of the subset sum problem approaches 1, the success rates of all these three algorithms decrease. On the other hand, the success rate of the Type-I & LLL algorithm drops much slower than others. When $0.75 \lesssim$ density ≤ 1, the Type-I & LLL algorithm has the highest success rate.

Although the Type-I & LLL algorithm performs particularly well for the subset problem with small values of n, its performance retrogresses quickly if $n \geq 42$. As it is shown in Table 4.2, at $n = 42$, the Type-I & LLL solves at most 40% of the problems for $0.875 \leq d \leq 1$. The maximum success rate drops to 30% if $n = 43$ and $0.896 \leq d \leq 1$.

Chapter 5

Conclusions and Future Work

In this thesis, we introduce a new lattice reduction algorithm, the Type-I algorithm, which does not require the Gram-Schmidt orthogonalization or the QR decomposition. Also, we propose a hybrid method, which integrates the Type-I algorithm, the LLL algorithm, the GCD-reduction and the weight-reduction for solving the subset sum problem. After the concepts of lattices and bases are introduced in Chapter 2, we present the LLL and the Type-I reduction algorithm in Chapter 3. Also, we compare them by the running time, the orthogonality defect and the Euclidean norm of the shortest vector in the reduced bases. When ω in the LLL is set to 0.99 and δ in the Type-I is also set to 0.99, the Type-I generates more orthogonal reduced bases and finds shorter shortest lattice vectors than the LLL algorithm, if the dimension of lattices is high($\geq$ 45). However, when the dimension of lattices is less than 45, the LLL reduced bases are better than the Type-I reduced bases. Also, The LLL algorithm runs much faster than the Type-I algorithm for almost all experimental bases. In Chapter 4, we introduce the subset sum problem and discuss some previous methods for solving the problem. Then we propose the Type-I & LLL algorithm for

solving the subset sum problem. The experimental results are shown in section 4.5 and compared with the results of the Lagarias-Odlyzko method and the Radziszowski-Kreher method. The Type-I & LLL almost performs as well as a lattice oracle for all problems with size $n \leq 30$. Although the performance declines over the range $34 \leq n \leq 43$, the success rate of the Type-I & LLL algorithm is still much higher than those of the Lagarias-Odlyzko attack and the Radziszowski-Kreher attack(Figure 4.8).

The future work includes how to extend the experiments in order to attack the subset sum problems of large sizes. There are several ways to improve this. One way is using a multi-precision LLL algorithm. For example, we can use the MP algorithm [24] instead of the MatrixLLL algorithm(Algorithm 3) in the I&LLL phase. The implementation of the LLL in the MP algorithm uses floating-point approximations to the rational values of $\mu_{i,j}$ and $\|a_i^*\|_2^2$, where $1 \leq i < j \leq n$. The MP algorithm almost eliminates the accumulated errors in the floating-point $\mu_{i,j}$, since at each iteration of the while loop, $\mu_{i,j}$ are initialized directly from the definition [24]. An improved LLL algorithm(L^3FP algorithm) [26] can also be used to take the place of the MatrixLLL. The L^3FP algorithm produces considerably shorter vectors than the original LLL algorithm but may be inefficient in the worst case. Another way of extending the Type-I & LLL algorithm is to use symbolic computation. The symbolic computation is only used in the LLL part of the I&LLL phase because the Type-I algorithm does not require the Gram-Schmidt orthogonalization or the QR decomposition.

Finally, it is possible to modify the initial matrix to take into account the additional information related to the creation of the specific subset sum problem. In [4], it is shown how to modify the input basis for a subset sum problem if it is known

that

$$\sum_{i=1}^{n} e_i = \beta n.$$

The input matrix can be changed to

$$A = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & \beta \\ 0 & 1 & 0 & \dots & 0 & \beta \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & \beta \\ w_1 & w_2 & w_3 & \dots & w_n & s \end{bmatrix},$$

so that a solution vector with length $\sqrt{n\beta(1-\beta)}$ exists (In our experiments, we choose $\beta = \frac{1}{2}$). Although no information of $\sum e_i$ is known for general subset sum problems, some subset sum based cryptosystem, such as the Chor-Rivest system [3], do use subsets with relatively few weights. The Type-I & LLL algorithm could be modified to use the tailored lattice basis described in [4] when attacking such systems.

Bibliography

[1] Ernest F Brickell. Solving low density knapsacks. In *Advances in Cryptology*, pages 25–37. Springer, 1984.

[2] Viggo Brun. Algorithmes euclidiens pour trois et quatre nombres. In *Treizieme congres des mathematiciens scandinaves, tenua Helsinki*, pages 18–23, 1957.

[3] Benny Chor and Ronald L Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *Information Theory, IEEE Transactions on*, 34(5):901–909, 1988.

[4] Matthijs J Coster, Brian A LaMacchia, Andrew M Odlyzko, and Claus P Schnorr. An improved low-density subset sum algorithm. In *Advances in Cryptology—EUROCRYPT'91*, pages 54–67. Springer, 1991.

[5] Alan M Frieze. On the lagarias-odlyzko algorithm for the subset sum problem. *SIAM Journal on Computing*, 15(2):536–539, 1986.

[6] Michael R Garey and David S Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1979.

[7] Hussain Ali Hussain, Jafar Wadi Abdul Sada, and Saad M Kalipha. New multi-stage knapsack public-key cryptosystem. *International Journal of Systems Science*, 22(11):2313–2320, 1991.

[8] Antoine Joux and Jacques Stern. Improving the critical density of the lagarias-odlyzko attack against subset sum problems. In *Fundamentals of Computation Theory*, pages 258–264. Springer, 1991.

[9] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[10] Jeffrey C Lagarias and Andrew M Odlyzko. Solving low-density subset sum problems. *Journal of the ACM (JACM)*, 32(1):229–246, 1985.

[11] Ming Kin Lai. Knapsack cryptosystems: The past and the future. *Term paper, April*, page 36, 2001.

[12] BA LaMacchia. Basis reduction algorithms and subset sum problems. sm thesis, dept. of elect. *Eng. and Comp. Sci., Massachusetts Institute of Technology, Cambridge, MA*, 1991.

[13] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[14] Franklin T Luk and Sanzheng Qiao. A pivoted lll algorithm. *Linear Algebra and its Applications*, 434(11):2296–2307, 2011.

[15] Franklin T Luk, Sanzheng Qiao, and Wen Zhang. A lattice basis reduction algorithm. *Institute for Computational Mathematics Technical Report 10*, 4, 2010.

[16] Franklin T Luk and Daniel M Tracy. An improved lll algorithm. *Linear algebra and its applications*, 428(2):441–452, 2008.

[17] JE Mazo and Andrew M Odlyzko. Lattice points in high-dimensional spheres. *Monatshefte für Mathematik*, 110(1):47–61, 1990.

[18] Ralph C Merkle and Martin E Hellman. Hiding information and signatures in trapdoor knapsacks. *Information Theory, IEEE Transactions on*, 24(5):525–530, 1978.

[19] Hermann Minkowski. *Geometrie der zahlen*, volume 40. Рипол Классик, 1968.

[20] Phong Q Nguyen and Jacques Stern. Lattice reduction in cryptology: An update. In *Algorithmic number theory*, pages 85–112. Springer, 2000.

[21] Andrew M Odlyzko. The rise and fall of knapsack cryptosystems. *Cryptology and computational number theory*, 42:75–88, 1990.

[22] Tatsuaki Okamoto, Keisuke Tanaka, and Shigenori Uchiyama. Quantum public-key cryptosystems. In *Advances in Cryptology—CRYPTO 2000*, pages 147–165. Springer, 2000.

[23] Glenn Orton. A multiple-iterated trapdoor for dense compact knapsacks. In *Advances in Cryptology—EUROCRYPT'94*, pages 112–130. Springer, 1994.

[24] Stanislaw Radziszowski and Donald Kreher. Solving subset sum problems with the l^ 3 algorithm. *The Charles Babbage Research Centre: The Journal of Combinatorial Mathematics and Combinatorial Computing*, 3, 1988.

[25] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[26] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.

[27] Jozef Vyskoč. Knapsack in cryptography. *Computers and artificial intelligence*, 6(6):535–540, 1987.

[28] HS Wilf. Backtrack: An 0 (1) expected time graph coloring algorithm. *Inform. Process Lett*, 18(1):19–122, 1984.

[29] Wen Zhang. New lattice reduction algorithms with polynomial complexity. 2015. Private Communication.