UNIVERSITÉ PARIS DIDEROT
(Paris 7)
SORBONNE PARIS CITE

ÉCOLE NORMALE SUPÉRIEURE
Équipe Crypto

THÈSE DE DOCTORAT

Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe

Spécialité : INFORMATIQUE

Présentée et soutenue publiquement le 13 novembre 2013 par

YUANMI CHEN

pour obtenir le grade de

Docteur de l'Université Paris Diderot

Composition du Jury :

Directeur de thèse :	Phong-Quang NGUYEN	-	INRIA, France
Rapporteurs :	Antoine JOUX	-	UVSQ, France
	Nigel SMART	-	University of Bristol, UK
Examinateurs :	Jean-Sebastien CORON	-	Université de Luxembourg, Louxembourg
	Nicolas GAMA	-	UVSQ, France
	Guillaume HANROT	-	ENS-Lyon, France
	Vadim LYUBACHEVSKY	-	INRIA, France

Remerciements

Je tiens tout d'abord à remercier Phong Nguyen d'avoir dirigé cette thèse. Pendant ces années, non seulement il m'a fait explorer le merveilleux sujet de l'analyse des cryptosystèmes à base de réseau, m'a encouragé à participer aux conférences et séminaires, mais aussi, lui-même, en tant que grand exemple à suivre, m'a impressionné par sa qualité de recherche et sa personnalité. Je me souviendrai de sa rigueur pour la recherche, de son ésprit ouvert pour sa carrière et son optimisme pour toujours.

Je suis aussi très reconnaissante aux rapporteurs, Antoine Joux et Nigel Smart, qui ont accepté de lire et commenter ce manuscrit, ainsi qu'aux autres membres du jury, Jean-Sebastien Coron, Nicolas Gama, Guillaume Hanrot et Vadim Lyubachevsky.

Je voudrais aussi remercier tous les membres d'équipe CRYPTO à l'ENS, Michel Abdalla Bruno Blanchet, Céline Chevalier, Pierre-Alain Fouque, Elizabeth Quaglia, Vadim Lyubachevsky, David Naccache, Duong Hieu Phan, David Pointcheval, Oded Regev, Damien Vergnaud et Hoeteck Wee; ceux qui sont partis avant moi, comme Aurélie Bauer, Olivier Blazy, Charles Bouillaget, David Cadé, Angelo De Caro, Patrick Derbez, Léo Ducas, Dario Fiore, Georg Fuchsbauer, Gaëtan Leurent, Jérémy Jean, Miriam Paiola, Mehdi Tibouchi et Mario Strefler, ceux qui restent, Aurore Guillevic, Sonia Belaïd, Fabrice Ben Hamouda, Sorina Ionica, Tancrède Lepoint, Thomas Prest et Sylvain Ruhault.

Je remercie aussi tout le personnel administratif de l'ENS ainsi que le SPI, en particulière Joëlle Isnard pour sa disponibilité, Valérie Mongiat et Isabelle Delais pour leur aide, Michelle Angely et Lise-Marie Bivard pour leur amabilité, et Jacques Beigbeder pour son efficacité et son expertise.

Merci à l'ANR pour le financement de ma thèse, et aussi pour m'avoir fournit la possibilité d'enseigner en monitorat et de participer aux nombreuses formations à l'Université Pierre et Marie Curie pendant 3 ans.

I would also like to thank Wang Xiaoyun, professor for hosting me in in Tsinghua University in 2011 and 2012. And thanks to all members of the team that I have met there – Han Lidong, Jia Keting, Li Jianwei, Liu Mingjie, Sun Yue, Tian Chengliang, Wang An, Wei Wei, Yu Hongbo, for welcoming and helping me during my stay.

Thanks to those who I met for a short time in the conferences, Malika Izabachéne, Richard Lindner, Xuejia Lai, Andrew Novincin and Elaine Shi and for the useful exchange of ideas.

Finally, many thanks to my friends and family, who are always willing to give me unconditional support for what ever choice I made. Especially to my beloved husband, Shun Zhang, for everything you have done for me.

Contents

CHAPTER 1

General Introduction

Contents

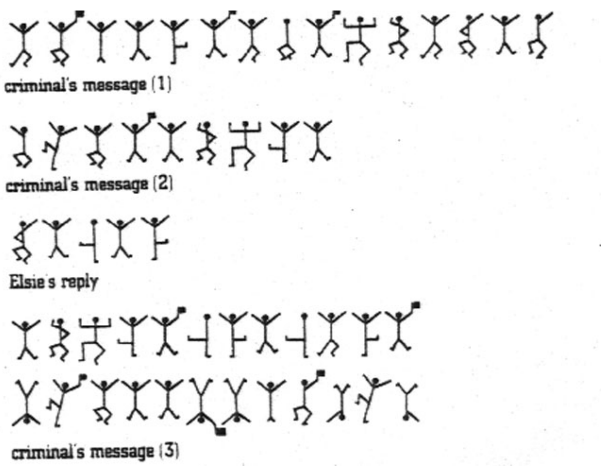
1.1 From code breaking to cryptanalysis

Initially, the use of cryptography consisted of encrypting messages. The usual encryption methods include substituting each letter in the alphabet, or transposing the order of writings. In the era when all computations had to rely on manual labour, such encryption was enough to resist the attacks by an intruder. However, as there was no notion of security and no thorough analysis of encryption methods, no one knew exactly how difficult it was to break a seemingly well-concealed message.

1.1.1 Examples of some classical systems

Games between cryptographers and code-breakers are a common plot in stories. Among the most classical and naive systems are the *substitution code* and the *transposing code*. For example, in Arthur Conan Doyle's novels, *Sherlock Holmes* encountered an encrypted message in the story *The adventure of the dancing men*. The encrypted message between the criminal and *Elsie* was formed by a sequence of dancing men. Holmes was able to find out the concealed content by comparing the frequency of symbols with that of natural English texts.

Figure 1.1: The dancing men stand for different English letters in these secret messages

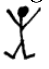


In another famous novel "The journey to the center of the earth" by Jules Verne, the entrance of the passage to the center of the earth was encrypted using a transposition cipher. It took Professor Lidenbrock and his nephew Alex two days to decrypt.

In those days, cracking a ciphertext depended much on one's luck and knowledge of languages, apart from mathematics. People generally could not predict what was essential to break an encryption scheme. No wonder when the United Kingdom wanted to recruit talented people to decrypt telegrams from the German army during World War II, some crossword experts, chess champions were also invited to Bletchley park to participate to their cryptanalysis actions.

1.1.2 The role of cryptanalysis

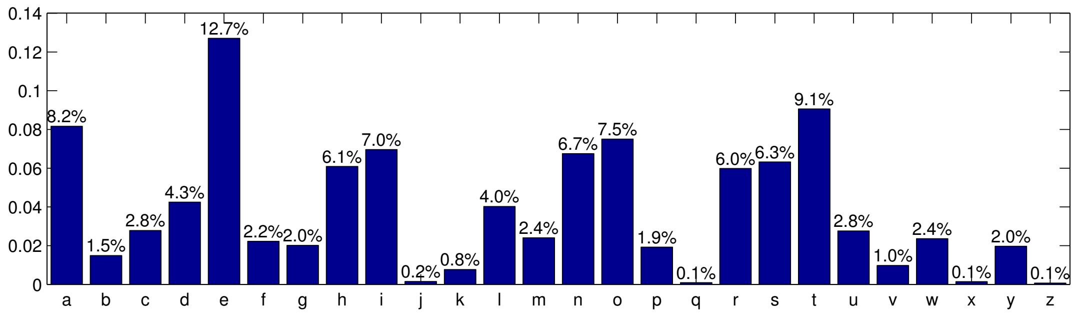
The role of cryptanalysis is to provide an reliable analysis of the security of the encryption method, usually by making use of complicated mathematical tools, thereby provide confidence in the security of the system, and provide arguments to set up parameters, or propose possible ways of improving the security.

For instance, the previous story well illustrated several typical characteristics of substitution ciphers, In the story, Sherlock Holmes guessed correctly that each dancing man was a representation of an English letter, and the whole message was an English sentence with flags separating words. By analysing the frequency of symbols, it is not hard to notice that the symbol  comes up much more often than any other symbol, and therefore it most probably stands for the letter 'e', which appears in predominating high frequency in common English sentences. But since there was too few symbols in the beginning, Holmes was not able to recover the hidden message, until he saw the third message which consisted of only five letters. Having the two E's coming second and fourth in a word of five letters, which serves as a reply to an appeal, the most probable match would be 'never'. This discloses the letter 'N', 'V', and 'R'. Consequently, Holmes was able to decode the whole message by guessing from the finitely many possibilities of words, and thereby decode the malicious messages sent by the criminal. Finally, he wrote such an invitation message in the dancing men code, which successfully allured the criminal and arrested him.

After all, the breaking of a single ciphertext is not the finishing point of cryptanalysis. More important is to explore essential features of substitution code in general.

1. *Frequency analysis* is an effective mean to attack the substitution code. In a normal English text, the frequency of certain letters is considerably higher than others. When these letters are replaced by symbols, their frequencies remain unchanged. Consequently, we can compare these frequencies to that of normal English texts, which will reveal the correspondence between the symbols and the letters they stand for. Apart from English, in every natural language, each symbol in alphabet has a characteristic frequency, which can serve to analyze a substitution code.

Figure 1.2: Frequency of letters in English texts [47]



2. Substitution code can not change the length of a message, neither is it able to conceal special patterns of the text, and in some cases, this will leak important clues for decryption. A recognizable pattern or a particularly short message would usually facilitate attacks.

Upon the analysis of the substitution cipher, it can be concluded that it is not secure to use substitutions directly over a plaintext, when each letter in the alphabet has an apparent characteristic frequency, and the total size of the alphabet is small. The *unicity distance* describes minimum numbers of letters required to break the substitution cipher. For simple substitution, the unicity distance is 27.6. The effectiveness of frequency analysis increases when the length of encrypted message grows. In practice, the number of letters required to break a substitution cipher is usually 2-4 times of the theoretical lower bound, which means 50 to 100 letters for a substitution ciphertext. This also effectively points out how we should improve it.

In order to resist the frequency analysis to a reasonable degree, one possible extension of substitution code is to use a more complicated substitution strategy, for example, instead of substituting each single letter, it is possible to substitute each pair of consecutive letters (digraphs) in the message. Examples of digraphic ciphers include playfair ciphers and four-square ciphers. These ciphers have stronger resistance against frequency attacks. Another way is to change the substitution rule constantly so that there is no longer a fixed mapping between the symbols in the ciphertext and the plaintext. This was the principal underlying idea for motor machines, which is a type of electro-mechanic device. The substitution rule is dependent on the combination of rotors, whose position changes with typing. A famous representative of rotor machine is the Enigma machine used by German army during World War II, which required tremendous effort to break down the code.

1.2 Modern Cryptology

1.2.1 Basic concepts of modern cryptology

Cryptology came into public study long after World War II, and it soon became a domain closely related to applied mathematics. In one hand, the notion of security gradually developed into increasingly rigorous form. On the other, the content of cryptography was also enriched to achieve various purposes, cryptography is no longer restricted to encryption alone.

Following is some of the most important notions which underlies modern cryptography:

Public design It has become widely recognized that modern design of cryptography should comply with Kerckhoffs's principle: A cryptosystem should remain secure even if everything about the system, except the key, is public knowledge. This has two advantages, first of all, we will only have to keep secret a small amount of information, which is the key, and in case it falls in hands of the adversaries, we would only need to change the key, not the entire cryptosystem. In addition, this allows public analysis of the cryptosystems, so that we have a better idea of its security level, and its potential weakness.

Security level The popularity of computers and internet greatly shaped the evolution of cryptography. On one hand, the design is often based on the communication requirement of the digital world. On the other hand, the security requirement is naturally based on computational power, which is growing rapidly. For any cryptosystems, we need to clarify its security level before it can be applied in practice. The *security level* is defined with regard to best known attack to the system. If the best attack against a crypto-system requires 2^n operations, then it is said to have n-bit security. The type of operation should not be too complex, especially it should not be too far from a CPU clock cycle.

Today, the clock rate of a CPU is about 2^{31} Hz, and a single core can perform approximately 2^{26} multiplications of 128-bit integers per second. And the number of PCs sold is about 200 millions per year. While it is already possible to have supercomputer of about 2^{21} cores, in a single day it is feasible to carry out more than 2^{60} multiplications of 128-bit integers. With the Moore's law which predicts an exponential increase in computational power, this number is expected to increase at a constant speed. Therefore a secure cryptosystem should have at least 80 - 128 security bits.

Primitives and protocols Cryptographic primitives are well-studied fundamental algorithms which are considered as reliable. They serve as building blocks for more complex security applications. A cryptographic protocol describes how these cryptographic primitives should be used in the application, and how the different parties should use the information and interact with each other. A cryptographic primitive would need to undergo strict analysis before it is recognized as secure. They usually accomplish a basic task such as encryption, or signature alone. While a protocol makes comprehensive use of different cryptographic primitives and tools, inventing new rules of communications. Such protocols ensures both security and efficiency, for example, electronic-voting, auction protocols are becoming more prevalent.

1.2.2 Cryptography in different security aspects

Modern cryptology is not restricted to encryption. It also provides a lot of other functions like verification of identity, authentication of messages, etc. In general, they aim to guarantee *confidentiality, integrity* and *authenticity* of data. Cryptosystems try to achieve one or several of the following goals:

Confidentiality Only the holder of the secret key can decrypt the message. All other unauthorized parties are not supposed to obtain any knowledge, even one-bit of information of the message except for the message length.

Authenticity There should be a way to guarantee that the message is indeed sent by the claimed sender. Any one else would not be able to forge an identity.

Integrity If the content of message is modified by anyone other than the sender, then the receiver should be able to detect this fact.

Non-repudiation The sender of the messages can not deny having sent them.

There are usually two different parties involved: the first party performs an operation to protect and transform the information into a disguised form, be it encryption, signature or other operations; the second party, who receives the disguised information, uses another operation to interpret it. Depending on whether the two parties share the key, cryptosystems are divided into following two categories: symmetric cryptography and asymmetric cryptography.

1.2.3 Symmetric cryptography

In symmetric cryptography, the secret key is shared between two parties in the conversation. The party must agree upon this secret key before the conversation takes place in a public channel. It is called symmetric because every participant in the conversation have symmetric role and share the same messages. It is also called *secret-key cryptography* because its key should be kept only by trusted parties.

Usually, symmetric cryptosystems enjoy the advantage of being efficient and inexpensive for hardware and software implementations. Particularly, for encryption, the ciphertext would not be significantly longer than the original text. On the other hand, it has a drawback: Especially, we should always share the secret key before any conversation can take place, thus if there is no prior communication between different parties, they can not start their conversation directly by symmetric cryptosystems.

The security of symmetric cryptography is usually upper bounded by the size of its key, due to brute-force attack. That is to say, we can always enumerate all possible keys and check if decryption makes sense. That is the reason why single-DES system whose key size is fixed to be 56-bit is no longer considered secure any more. Several specialized machines had managed to break down DES codes in practice, and some of them can do this in less than one day. As the successor of DES, the AES system has a key size of 128 bits.

1.2.4 Asymmetric cryptography

This kind of cryptosystems require a pair of keys, one of which is public, and another is secret. While any one can verify a signature or encrypt a message with the public key, only the holder of the secret key can decrypt the cypher text, and sign a message. The secret key and the public key are mathematically related, but it should be computationally infeasible to obtain the secret key from the knowledge of the public key. These systems are also called *public-key cryptography*.

In contrast with symmetric cryptography, where the secret key is shared between all participants in a conversion, in asymmetric cryptography, the secret key is not known to all parties in a communication. The holder of the secret key has a central position in the interaction. For example, in the case of public-key encryption, every body can use the public key to encrypt messages, but only the owner of the secret key can decrypt them. The roles of the participants in communication are asymmetric. Apart from encryption, it also has applications in signature, e-voting, etc.

For this type of communication, the different parties in communication are not required to share any secret in advance. But the price is that the encrypted messages in public-key systems are usually much larger than the plaintext, and the computational cost is higher. For practical reason, asymmetric systems are often used together with symmetric key systems for long conversations. As the first step, different parties will agree on a secret using asymmetric cryptosystems. This secret is later used as secret key of symmetric cryptosystems when they carry on with their conversation using symmetric cryptosystems.

In public-key cryptology, only one party owns the secret key, and one never needs to reveal this secret key to other parties during any kind of cryptographic operations. Therefore, the fact of holding a secret key is often regarded as an identity prover, especially in various signature schemes. In symmetric cryptography, the life cycle of a key is usually limited to the duration of the conversation, because the secret key is shared between different parties highlights the problem of secret key disclosure. But the public-secret key pair in asymmetric cryptography is kept much longer. This is necessary as a signature should be verifiable for the same public key during some reasonable duration.

1.3 Public-key cryptography

Public-key cryptography depends on mathematical problems which has no efficient solution. With these problems, one creates a one-way function $f(\cdot)$ which can be obtained with the public key, so that everyone can calculate the mapping $x \to f(x)$. However, it is not possible to compute the inverse of the function, unless we have the some additional hint. This additional hint is the secret key.

The first public-key system was proposed by Diffie and Hellman in 1976. Their system uses the property that calculating discrete logarithms in certain groups is difficult. Soon after the RSA scheme was proposed, its security was related to the hardness of integer factorization. Not all difficult mathematical problems can yield public-key cryptosystems. Up to now, the most successful cryptosystems fall into the following categories of problems, the discrete Logarithmic problem over finite fields or elliptic curves, the RSA problem, lattice-based hard problems, coding-theory problems and multi-variable problems.

1.3.1 Security analysis

Unlike symmetric cryptography, the hardness of the problem is not dependent only on the bit-size of the secret key alone. More importantly, it relies on the difficulty of the mathematical problem. In most of the cases, we can attack a cryptosystem by solving one instance of the underlying mathematical problem. That is to say that the security of the system is at most as hard as the underlying mathematical problem. We would hope to prove the inverse, such that we can reduce the mathematical problem to the task of attacking a system. But this direction does not always hold. For example, it is conjectured that the RSA scheme is as difficult as factoring big integers, and we already have some partial proofs [1, 12] which seems to suggest an affirmative answer to this conjecture.

For integer factorization and discrete logarithms, we do not yet know their complexity classes. In fact, even though a problem is NP-hard, it does not mean that every instance of this problem is difficult to solve. Therefore, estimating the precise security of any cryptosystem is more complicated. There are two ways to do the security estimation:

Security proof by reduction If any algorithm breaking down the cryptosystem can be efficiently transformed to solve an instance of a mathematical problem which is conjectured to be hard. Then breaking this cryptosystem is considered to be as hard as solving the underlying mathematical problem.

Security by cryptanalysis For many other cryptosystems, providing a direct security proof is difficult. However, all cryptosystems will be subject to cryptanalysis. If the research community is unable to raise an effective attack after many years of study, it brings confidence to the security.

With respect to security notion, we should always be clear about the assumptions. According to the amount of information the attacker has access,

Ciphertext only attack The adversary has access to the ciphertext and he tries to recover the plaintext from the ciphertext. This is the most general assumption, since encrypted messages are transmitted by a public channel, and the ciphertext can easily fall in hands of any potential eavesdropper.

Known-plaintext attack The adversary possesses some plaintext-ciphertext pairs $\{(p_1, c_1), (p_2, c_2), \ldots, (p_i, c_i)\}$. With this information, he tries to compute the key, or he obtains information of the plaintext of a new ciphertext c_{i+1}.

This can happen when the encrypted message is of some fixed format, or when some previously classified information became public in later years. But still, the attacker can not deliberately choose any plaintext nor ciphertext that he wants to know.

Chosen-plaintext attack The adversary can choose some plaintext $\{p_1, \ldots, p_i\}$ and ask the cryptosystem to send back their corresponding ciphertext $\{c_1, \ldots, c_i\}$. Using these information, the adversary tries to launch attacks.

Chosen-ciphertext attack The adversary can choose some ciphertext $\{c_1, \ldots, c_i\}$ and ask the cryptosystem to send back their corresponding plaintext $\{p_1, \ldots, p_i\}$, and tries to launch attacks.

1.3.2 Hard problems and NP-completeness

Not all computationally difficult mathematical problems can yield one-way trapdoor functions. In order for a series of problems to be used as trapdoor functions, there must be a way to express the problems as a trapdoor function, and we should be able to efficiently sample from the set of problems. Here is some examples of hard problems:

Discrete Logarithms The discrete logarithm can be defined for arbitrary finite cyclic groups. As cyclic groups are isomorphic to $\mathbb{Z}/n\mathbb{Z}$, we use the group $\mathbb{Z}/n\mathbb{Z}$ to describe the problem. Let g be a generator in the group $\mathbb{Z}/n\mathbb{Z}$, For any integer x, computing $y = g^x \mod p$ requires $O(\log(n))$ multiplications. Conversely, when given y, computing x such that $y = g^x \mod p$ is called the discrete logarithmic problem(DLP). In fact, if we know n can be factored into products of smaller integers, than the problem can be reduced to several smaller problems using Chinese remainder theorem. Therefore, it is usually required that n has at least one large prime factor p. In this case, this problem is known to be difficult, and it is the basis for many cryptosystems, such as the Diffie-Hellman key agreement, ElGamal encryption, ElGamal signature scheme, the Digital Signature Algorithm. The most effective algorithm is number field sieve, which solves DLP in

$$L_n(1/3, c + o(1)),$$

where n is the number to be factored, c is a constant, and $L_n(s, c) = \exp(c(\log n)^s(\log\log n)^{1-s})$ [77].

RSA-problem The RSA problem is the basis of the security of RSA encryption and signature schemes. Let N be a product of two long primes $N = pq$, and e, d be two integers such that $ed \equiv 1 \mod (p-1)(q-1)$. In the RSA setting, N and e is public, but not p, q or d. Then for any integer m, anyone can compute $C \equiv m^e \mod N$ in $O(\log(e))$ multiplications of bit size $O(\log(N))$. But given a random integer C, it is computationally difficult to find m which satisfy this relation. Unless we have the secret key, which is the value of d, then we can compute m with the equation $m \equiv C^d \mod N$ in $O(\log(d))$ multiplications of bit size $O(\log(N))$. Solving this problem is suggested to be as hard as factoring N.

Interestingly, although these two problems are considered to be difficult in computation, and no effective algorithm was found to solve them in polynomial time, they are not in the NP-complete problem set either. We do not know what is their exact complexity class. They belong to the NP complexity class, because we can verify any potential solution in polynomial time, but there is no evidence that a polynomial-time algorithm for finding their solutions exists.

If large scale quantum computer can be created, DLP and RSA problem can be solved much faster. In fact, with Shor's algorithm [82], these two problems can be solved in polynomial time on a sufficiently large quantum computer. This is to say, for crypto schemes whose security relies on DLP and RSA problems, an attacker with access to a large enough quantum computer will easily break their cryptosystems. Therefore, the approach of the quantum computing era calls for reliable alternate cryptographic systems which remains to be secure against attackers with quantum computers.

1.4 Lattice-based cryptography

Lattice algorithms have been used in cryptology since the invention of the LLL reduction algorithm in 1982. Its first applications were in cryptanalysis to address problems like finding small roots for polynomials in Coppersmith method, and solving subset-sum problems. Later in 1996, NTRU, the first cryptosystem based on lattice problems was proposed [41]. This system works very efficiently, and is incorporated as part of IEEE standard. Meanwhile, new crypto schemes are emerging. The technical introduction of lattices will be given in Chapter 2, and here is a brief introduction to some hard lattice problems:

Smallest Vector Problem (SVP**)** Given a lattice, find its shortest non-zero vector. On the average, if we are given a bad basis, the best known algorithm for solving this algorithm is exponential in the lattice dimension. There are certain approximation version or variations of this problem, such as the GapSVP problem, Approx-SVP problem. The cryptosystems whose security rely on hardness of SVP include GGH cryptosystem, NTRU encryption schemes and so on.

Closest Vector Problem(CVP**)** Given a lattice, and a point in the Euclidean space, find a vector belonging to the lattice which is closest to this point. This problem is also difficult, unless we have a good basis. The cryptosystems whose security rely on hardness of CVP include the learning with error problem.

1.4.1 Security analysis

The most notable and classical hard problems are *discrete logarithm problem* and *integer factorization problem*. Recently, a lot of new cryptosystems base their security on lattice-related hard problems. For example, the subset sum problem, the closest vector problem etc. Despite its larger key size and inefficiency of computation compared to systems based on discrete-log or integer factorization problems, there are a number of advantages:

Post-quantum cryptography Unlike for integer factorization problem and discrete logarithm problem, up to now, there is no quantum algorithm that works more efficiently than classical algorithms in terms of solving lattice hard problems. Therefore there is hope that lattice based cryptography will survive in the post-quantum era.

Worst case to average-case reduction When we talk about complexity classes of hard mathematical problems, we are usually referring to the hardest instances of those problems. It might happen that these hardest instances indeed require large computation to solve, but some other instances might be easy. This raises security concerns, as we are unable to guarantee the security of a concrete instance, it might fall into the category of easy instances. In practice, most instances of integer factoring and discrete logarithm are difficult, but there is no theoretical proof against the possibility of an algorithm solving problems with considerable probability. But some of lattice-based cryptography does not suffer from this drawback: following Ajtai's work [3], there is a worst case to average-case reduction, which means that if any algorithm can solve random instances with non-negligible probability, then it can also be used to solve the hardest instances of another problem. Later, this reduction has also been somewhat extended to apply to ideal lattice problems [45].

1.4.2 Fully homomorphic encryption

An attractive functionality provided by lattice-based cryptography is the possibility to construct fully homomorphic encryption schemes. This is an important advantage of lattice based cryptography, as we do not know how to build them from discrete logarithm or integer factoring problems.

Homomorphic encryption allows specific operations on ciphertexts, without knowing the plaintexts. Upon decryption, the plaintext of the manipulated ciphertext would correspond exactly to applying the same operation to the original plaintext. If this scheme allows an arbitrary circuit evaluation, then this is called *fully homomorphic encryption*. The first fully homomorphic encryption system was proposed by Gentry [25], and since then, a lot of improvement was proposed, and recently, an open source implementation is also available (HElib [37]).

1.4.3 Lattice reduction

Lattices are usually represented by a basis. All non-trivial lattices of dimension > 1 has infinitely many basis, and some base are better than others in that they can better reveal essential properties of the lattices. With a good basis, we can compute the solution of SVP and CVP with less effort.

Lattice reduction is the process of transforming any basis of a lattice to a better one. Generally this is done in a progressive manner. We first do some week reductions before applying stronger reductions. The significance of reduction is two fold. First of all, it yields better bases, on which we can perform algorithms like enumeration to compute solutions to problems such as SVP and CVP. In some cases a strongly reduced basis already provides an approximate solution.

Therefore, it is of great importance to study the performance of reduction algorithms, and analyze how to optimally perform reduction and enumeration to solve SVP and CVP. The significance of this study is both in theory and in practice for the security estimates of the majority of lattice-based cryptographic primitives. It can be compared to the significance of analyzing integer factoring to RSA cryptography.

In 1982, LLL reduction was the first polynomial time algorithm which reduces the basis and gives an worst-case bound on the output quality of the basis. For many applications, we need stronger applications, and there comes the idea of block-wise reduction by Schnorr in 1987 [78]. The most popular block-wise reduction in practice is BKZ reduction, which was implemented as an open source code in NTL library [83]. However, in spite of the application of this algorithm, and its wide-spread usage as a benchmark for security estimates for cryptosystems, its practical performance is not well studied. The improvement of the BKZ algorithm and its analysis form an important part of this thesis.

1.5 Structure of this thesis

The main focus of this thesis is to provide security estimates for lattice based cryptography based on the analysis of reduction algorithms. The major work is devoted to the analysis and improvement of BKZ lattice reduction algorithm, which spans chapter 2 to chapter 6. Chapter 7 is another independent work on a successful attack to a FHE scheme with a classic time-space trade-off trick.

1.5.1 New security estimates based on improved BKZ reduction

BKZ reduction is the most practical lattice reduction algorithm up to now. It is frequently used in security estimates for various cryptosystems. In spite of its importance and wide popularity, not much analysis is given, both for its practical running time and output. Our goal is to understand the behavior of BKZ reduction in practice, and thereupon provide improvement to the algorithm. Consequently, the security estimates are renewed for many of the schemes.

The mathematical basics of lattice are presented in chapter 2, where we are also going to go over the hard problems and the existing algorithms. The most essential ingredient of BKZ reduction is the enumeration procedure. BKZ reduction makes a polynomial time of calls to the enumeration procedure, and the time cost of each enumeration is exponential in the dimension of enumeration. However, since the invention of extreme pruning [24], it is possible to tremendously speed up the procedure by making it probabilistic. We develop the analysis of pruning and especially we propose an effective way to generate pruning strategy for different parameters quickly. The in-depth analysis of enumeration algorithm is given in chapter 3. In chapter 4, we propose a new BKZ algorithm with incorporates several improvement, including the previous discussion of enumeration. We also provide a way to simulate the average behavior of the BKZ algorithm. With this simulation tool, we further studies BKZ algorithm and propose a new BKZ procedure with recursion, which is presented in Chapter 5. The new BKZ algorithm aims to renew many of the security estimates, including the famous NTRU and some of the FHE schemes.

1.5.2 Cryptanalysis of a fully homomorphic encryption scheme

Chapter 6 describes an attack to a FHE scheme proposed by [17]. The first fully homomorphic encryption (FHE) scheme proposed by Gentry in 2009 [25] is followed by numerous other proposals, some of them using different underlying hard problems. One of FHE scheme makes use of approximate common divisor problem (ACD) [89]. Especially, [17] proposed a much more efficient variant of ACD problem where they choose relatively small size for noise. Unfortunately, this choice overlooked an attack using a classical algorithm which finds the secret key with square-root time than brute enumeration. This algorithm uses a time-space trade-off trick, and evaluate several at the same time. Then, this method is generalized for various applications, where enumeration has a super-cubic structure.

CHAPTER 2

Introduction to Lattices

Contents

In this chapter, we introduce background on lattices. We will first give basic definitions about lattices, and present the main lattice problems and finally we will briefly present lattice reduction.

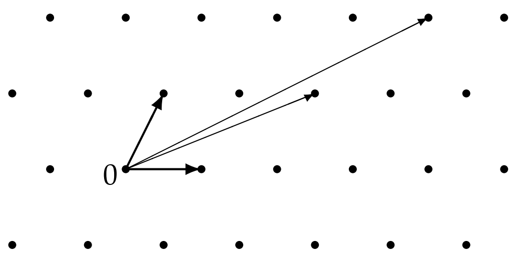
2.1 Lattices and bases

There are several ways to define a lattice. Here is the main definition.

Definition 2.1.1 (Lattice). *A lattice $\mathcal{L}$ of $\mathbb{R}^m$ is a discrete additive subgroup of* $(\mathbb{R}^m, +)$.

A lattice is discrete, which means that for each point in the lattice there is a neighborhood which contains only the point. Because a lattice is also a group, this implies that there exists a positive constant $\epsilon > 0$ such that for any two lattice points $\mathbf{v}_1 \neq \mathbf{v}_2$ we have $\|\mathbf{v}_1 - \mathbf{v}_2\| > \epsilon$. Intuitively, a lattice can be viewed as a set of points regularly distributed in the space $\mathbb{R}^m$. Below is a representation of a lattice in $\mathbb{R}^2$ (fig.2.1).

Figure 2.1: A lattice in $\mathbb{R}^2$



As a group, a lattice will always include the zero element $\{\mathbf{0}\}$. A lattice is trivial if it does not contain any other element. Otherwise, it contains an infinity of points. For a set of d vectors $\mathbf{v}_1, \ldots, \mathbf{v}_d$, if there exist d real numbers $c_1, \ldots, c_d$ not all zero such that $c_1\mathbf{v}_1 + \cdots + c_d\mathbf{v}_d = 0$, then we say that these vectors are linearly dependent over $\mathbb{R}$. Otherwise they are linearly independent over $\mathbb{R}$.

2.1.1 Lattice basis

Definition 2.1.2 (Rank). *The maximum number of linearly independent vectors in a lattice $\mathcal{L}$ is the lattice dimension or rank.*

In $\mathbb{R}^m$, there are at most m linearly independent vectors, therefore the rank of a lattice $\mathcal{L}$ in $\mathbb{R}^m$ is at most m. When the rank of $\mathcal{L}$ is equal to m, $\mathcal{L}$ is called *full-rank*.

Definition 2.1.3 (Basis). *Let $\mathcal{L}$ be a lattice in $\mathbb{R}^m$, and let $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be a set of linearly independent vectors in $\mathcal{L}$. If for any vector $\mathbf{v}$ in $\mathcal{L}$, there exist a set of integers $x_1, \ldots, x_n$ such that $\mathbf{v} = x_1\mathbf{b}_1 + \cdots + x_n\mathbf{b}_n$, then $\mathbf{B}$ is called a basis of the lattice $\mathcal{L}$.*

We also define $span(\mathbf{B}) = \{c_1\mathbf{b}_1 + \cdots + c_n\mathbf{b}_n | c_i \in \mathbb{R}\}$ to be the vector space spanned by $\mathbf{B}$. When $\mathbf{B}$ is a maximum set of linear independent vectors in $\mathcal{L}$, then all lattice points $\mathbf{v}$ are included in $span(\mathbf{B})$, because otherwise $\mathbf{B}$ can be augmented by $\mathbf{v}$. And moreover, there exists a unique representation $(x_1, \ldots, x_n) \in \mathbb{R}^n$, such that $\sum_{i=1}^n x_i\mathbf{b}_i = \mathbf{v}$. This representation is unique, because $\mathbf{B}$ is linearly independent.

Theorem 2.1.4 (Existence of basis). *A non-zero lattice $\mathcal{L}$ has at least one basis. The number of elements of a basis equals to the rank of the lattice.*

Proof. We provide a proof by induction on the lattice rank n, which is a variation of the proof given in [61]. Starting by $n = 1$, a lattice of rank 1 has a basis $\{\mathbf{v}\}$ where $\mathbf{v}$ is its shortest non-zero vector. All other vectors are necessarily multiples of $\mathbf{v}$ because $\mathcal{L}$ does not contain linearly independent set with size > 1, and because $\mathbf{v}$ is the minimum non-zero vector. Assume that the theorem holds for $n = k$, that is any lattice of rank k has at least one basis, and all of these bases have size k. Let $n = k + 1$, in the following we are going to find a basis for an arbitrary lattice $\mathcal{L}_{k+1}$of rank $k + 1$.

Let $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_k, \mathbf{c}_{k+1}\}$ be a set of linearly independent points in $\mathcal{L}_{k+1}$ whose size is $k + 1$. According to the induction assumption, we should already have a basis for $\mathcal{L}_k = \mathcal{L} \cap span(\{\mathbf{c}_1, \ldots, \mathbf{c}_k\})$, which we call $\mathbf{B}_k = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$. Then $\{\mathbf{b}_1, \ldots, \mathbf{b}_k, \mathbf{c}_{k+1}\}$ is also a linearly independent set. For any vector $\mathbf{v} \in \mathcal{L}$, there is a unique representation $(x_1, \ldots, x_n) \in \mathbb{R}^n$ such that $\sum_{i=1}^{k} x_i\mathbf{b}_i + x_{k+1}\mathbf{c}_{k+1} = \mathbf{v}$. Especially, since the point $\sum_{i=1}^{k} x_i\mathbf{b}_i$ is in $\mathcal{L}_k$, and $\mathbf{B}_k$ is a basis of $\mathcal{L}_k$, we have $\forall 0 \leqslant i \leqslant k$, x_i's are all integers . Consider the set of all possible values of x_{k+1}:

$$S = \{x | \exists x_i \text{ s.t. } \sum_{i=1}^{k} x_i\mathbf{b}_i + x\mathbf{c}_{k+1} \in \mathcal{L}\}$$

Clearly S is an additive group which contains at least 0 and 1. For every $x \in S$, we have $\sum_{i=1}^{k}(x_i - \lfloor x_i \rfloor)\mathbf{b}_i + x\mathbf{c}_{k+1}$ is also in $\mathcal{L}$. Since $\mathcal{L}$ is a lattice, it has finite element in any compact set. This is to say, there are finitely many points in $\mathcal{L}$, whose norms are bounded by $\sqrt{\|\mathbf{c}_{k+1}\|^2 + \sum_{i=1}^{k} \|\mathbf{b}_i\|^2}$. Therefore there are finite elements $0 \leqslant x \leqslant 1$ which belong to S, and it must contain a minimum positive element $s > 0$, which we suppose is achieved by lattice point $\mathbf{v}_0 \in \mathcal{L}$. All other values in $t \in S$ are multiples of s, otherwise $t' = t - \lfloor t/s \rfloor \cdot s$ would be a smaller positive element than s. In the following we prove that $\mathbf{B} \cup \{\mathbf{v}_0\}$ is a basis for $\mathcal{L}$. In fact, for any vector $\mathbf{v} \in \mathcal{L}$ which has a representation over $\mathbf{b}_1, \ldots, \mathbf{b}_k, \mathbf{v}_0$ as $(x_1, \ldots, x_k, x_{k+1})$, where x_{k+1} is some multiple of s, we note $x_{k+1} = k \cdot s, k \in \mathbb{Z}$. Then $\mathbf{v} - k \cdot \mathbf{v}_0$ should be in $\mathcal{L}_k$, and can be expressed as integer combination of $\mathbf{B}_k$. Therefore $\mathbf{v}$ can be expressed by integer combination of vectors in $\mathbf{B}_k \cup \mathbf{v}_0$, which proves that any lattice of rank $k + 1$ has a basis of size $k + 1$. By induction this holds for arbitrary n, which completes the proof. □

Remark. *A lattice is not a vector space, although they have many points in common. An integer combination of lattice points is still a lattice point. But since integers do not form a field, we cannot normalize a vector in a lattice as in a vector space.*

All lattices have bases, and reciprocally, given a set of linearly independent vectors, they can generate a lattice. This follows from the following theorem.

Theorem 2.1.5. *Given any linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$, the set*

$$\mathcal{L}(\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}) = \left\{ \sum_{i=1}^{n} x_i\mathbf{b}_i : x_i \in \mathbb{Z} \right\}$$

is a lattice generated by $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$.

Proof. We reproduce the proof given in [61]. It is clear that $\mathcal{L}(\mathbf{B})$ is a group, it remains to prove its discreteness. Consider the parallelepiped P:

$$P = \left\{ \sum_{i=1}^{n} x_i\mathbf{b}_i : |x_i| < 1 \right\}$$

Since the $\mathbf{b}_i$'s are linearly independent, $\mathcal{L}(\mathbf{B}) \cap P = \{0\}$. Since P is convex and non-degenerated in $span(\mathbf{B})$, there exists $\epsilon > 0$ such that the ball B centered at $\mathbf{0}$ with radius ϵ such that $B \cap P$ intersect $\mathcal{L}$

only on the point $\mathbf{0}$, by group properties this holds for all points in the set, which proves that this is a lattice. □

The theorem shows that any set of linearly independent vectors generates a lattice. This gives an alternative definition of lattices which is equivalent to the first one. And in cryptology, the most common representation for lattices is basis.

A basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ can also be written in matrix representation, by filling $\mathbf{b}_i$ its its i-th row: $\mathbf{B} = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{pmatrix}$. Here we assume $\mathbf{b}_i$ to be row vectors.

Definition 2.1.6 (Unimodular matrix). *A unimodular matrix* $\mathbf{U}$ *is a square integer matrix having determinant* 1 *or* -1.

Theorem 2.1.7. *The set of all n-by-n unimodular matrices is* $GL_n(\mathbb{Z})$*, the set of all n-by-n integer matrices which are invertible over integers.*

Proof. Let $\mathbf{M}$ be an arbitrary unimodular matrix. The entry at i-th column and j-th row of $\mathbf{M}^{-1}$ is given by $m_{i,j} = (-1)^{i+j} \det(\mathbf{M}^*_{j,i}) / \det(\mathbf{M})$, where $\mathbf{M}^*_{j,i}$ is the (j, i) minor of matrix $\mathbf{M}$. Therefore $\mathbf{M}$ is invertible over the integers. Meanwhile, for any matrix $\mathbf{M}$ that are invertible over integers, we have both $\det(\mathbf{M})$ and $\det(\mathbf{M}^{-1})$ are both integers too. Since $\det(\mathbf{M}) \cdot \det(\mathbf{M}^{-1}) = 1$, the only possible values for $\det(\mathbf{M})$ is 1 or -1. □

For non-trivial lattices of rank strictly larger than 1, bases are not unique. In fact, if $\mathbf{B}_{n\times m}$ is a basis of n-dimension lattice $\mathcal{L} \in \mathbb{R}^m$, and $\mathbf{U}_{n\times n}$ is a unimodular matrix, then $\mathbf{B}' = \mathbf{U}\mathbf{B}$ is also a basis of the same lattice.

Lemma 2.1.8. *Let* $\mathbf{B}_0$ *be a basis for* $\mathcal{L}$ *of rank n, then the set of all bases of this lattice is* $\{\mathbf{U}\mathbf{B}_0 : \mathbf{U} \in \mathsf{GL}_n(\mathbb{Z})\}$.

Proof. First of all, if $\mathbf{B}$ is also a basis for $\mathcal{L}$, then there is a representation with integer coefficients of $\mathbf{B}$ over $\mathbf{B}_0$, and vice versa. This means that there is an integer matrix $\mathbf{U}$ which satisfies $\mathbf{B} = \mathbf{U}\mathbf{B}_0$, and is invertible over the integers. Therefore $\mathbf{U} \in GL_n(\mathbb{Z})$. On the other hand, for any $\mathbf{U} \in GL_n(\mathbb{Z})$, define $\mathbf{B} = \mathbf{U}\mathbf{B}_0$, then $\mathbf{B}_0 = \mathbf{U}^{-1}\mathbf{B}$, thus every point in $\mathcal{L}$ is an integer combination of $\mathbf{B}$, and $\mathbf{B} \subset \mathcal{L}$, we conclude that $\mathbf{B}$ is a basis of $\mathcal{L}$. □

Given a basis $\mathbf{B}$ of a lattice $\mathcal{L}$, multiplying it by a unimodular matrix yields a basis $\mathbf{B}' = \mathbf{U}\mathbf{B}$ of the same lattice, this is called a basis transformation. Among all the bases of a lattice, the value $|\det(\mathbf{B}\mathbf{B}^T)|$ is constant, because unimodular matrices have determinant 1 or -1.

2.1.2 Lattice volume and Gaussian heuristic

Definition 2.1.9 (Volume). *We define the volume of a lattice* $\mathcal{L}$ *as*

$$vol(\mathcal{L}) = \sqrt{\det(\mathbf{B}\mathbf{B}^T)},$$

where $\mathbf{B}$ *is an arbitrary basis of the lattice* $\mathcal{L}$.

Geometrically, this is the volume of the parallelepiped spanned by the basis $\mathbf{B}$. The volume of a lattice measures the average density of a lattice, in the following sense. For any $\mathbf{v} \in \mathcal{L}$, we define the parallelepiped $P(\mathbf{B}) = \{\sum_{i=1}^{n} x_i\mathbf{b}_i : -1/2 < x_i \leqslant 1/2\}$, then the volume of this parallelepiped equals to $vol(\mathcal{L})$. Meanwhile, we have

$$\bigcup_{\mathbf{v}\in\mathcal{L}} (P(\mathbf{B}) + \mathbf{v}) = span(\mathbf{B}),$$

and

$$\forall \mathbf{v}_1 \neq \mathbf{v}_2 \in \mathcal{L}, P(\mathbf{v}_1) \cap P(\mathbf{v}_2) = \emptyset,$$

which shows that this is a tiling of the space $span(\mathbf{B})$. This means that in $span(\mathbf{B})$, the average space occupied by a lattice point is $vol(\mathcal{L})$ on average. From the point of view of group theory, this parallelepiped is simply the quotient group $\mathbb{R}^n/\mathcal{L}$. Furthermore, for any measurable set $S \in span(B)$ with volume $vol(S)$, it is intuitively expected to contain approximately $vol(S)/vol(\mathcal{L})$ such parallelepipeds, therefore heuristically this should also be the number of points in $S \cap \mathcal{L}$. This estimate for number of lattice points is called the Gaussian Heuristic.

Heuristic 2.1.10 (Gaussian heuristic 1)**.** *Let $\mathcal{L}$ be a n dimension lattice in $\mathbb{R}^n$, and S be a measurable subset of $\mathbb{R}^n$ with finite volume, the* Gaussian Heuristic *predicts that the number of lattice points in S:*

$$\#\{S \cap \mathcal{L}\} \approx vol(S)/vol(\mathcal{L})$$

But being a heuristic estimate, it can be very close to the real answer, or we can deliberately create some sets that make the estimate arbitrarily far from truth. For example, the lattice $\mathcal{L} = \mathbb{Z}^n$ has volume 1, and the set

$$S = \left\{(x_1, \ldots, x_n) : \begin{matrix} 1/4 < x_1 < 3/4, & \\ 0 < x_i < a, & \forall i > 1 \end{matrix}\right\}$$

has volume $a^{n-1}/2$, but it does not contain any lattice points. But if we restrict S to have certain shape, then we will be more confident about this estimate. Intuitively, when S is convex and big enough, then the estimate should be reasonable. Especially, when S is a sphere, we have the following lemma.

Lemma 2.1.11. *Let $\mathcal{L}$ be a n dimensional lattice, and v_n be the volume of the closed unitary hyper ball S of dimension n in $span(\mathcal{L})$. For any $r > 0$, the number of lattice points $\mathbf{v} \in S \cup \mathcal{L}$ such that $\mathbf{v} \in \mathcal{L}$ is noted as $s_{\mathcal{L}}(r)$. Then,*

$$\lim_{r\to\infty} \frac{r^n v_n}{s_{\mathcal{L}}(r)} = vol(\mathcal{L})$$

Besides, another important prediction given by Gaussian heuristic is an estimate of $\lambda_1(\mathcal{L})$, where $\mathcal{L}$ is a random lattice. For the definition of random lattice, please refer to section 2.3.

Heuristic 2.1.12 (Gaussian heuristic 2)**.** *Let $\mathcal{L}$ be a n dimensional random lattice in $\mathbb{R}^n$, the dimension n is large, then $\lambda_1(\mathcal{L})$ is approximately $(vol(\mathcal{L})/v_n)^{1/n}$. We note $GH = (vol(\mathcal{L})/v_n)^{1/n}$ be the prediction of Gaussian heuristic for $\lambda_1(\mathcal{L})$.*

2.1.3 Sublattices and projected lattices

The group property and the geometric structure of lattices allow us to decompose lattices in different ways. When the dimension of a lattice is big, it is interesting to look at its sublattices and projected lattices to explore its hidden structures.

Definition 2.1.13 (Sublattice)**.** *Let $\mathcal{L}$ be a lattice in $\mathbb{R}^m$. A* sublattice *of $\mathcal{L}$ is a lattice $\mathcal{L}'$ included in $\mathcal{L}$.*

When $\mathcal{L}'$ is a sublattice of $\mathcal{L}$, then all its basis vectors $\mathbf{B}' \in \mathcal{L}'$ can be expressed by integer combinations of basis vectors of $\mathcal{L}$. This means there exist an integer matrix $\mathbf{T}$ such that $\mathbf{B}' = \mathbf{TB}$. On the other hand, for any d-by-n integer matrix $\mathbf{T}$, if $\mathbf{T}$ is of rank d, then $\mathbf{TB}$ is a set of d linearly independent vectors, thus $\mathcal{L}(\mathbf{TB})$ is clearly a sublattice of $\mathcal{L}(\mathbf{B})$. Especially, when the rank of $\mathcal{L}'$ is equivalent d and $vol(\mathcal{L}(\mathbf{TB})) = vol(\mathcal{L}(\mathbf{B})) \cdot \sqrt{\det(\mathbf{TT}^T)}$.

When $d = n$, then the sublattice is full-rank. Especially, if $\mathbf{T}$ is a unimodular matrix, then it degenerates into the trivial case of $\mathcal{L}' = \mathcal{L}$. When $d < n$, one particular interesting case is the primitive sublattice.

Definition 2.1.14 (Primitive sublattice). *A sublattice $\mathcal{L}'$ of $\mathcal{L}$ is called primitive if and only if for any basis $(\mathbf{b}_1, \ldots, \mathbf{b}_r)$ of $\mathcal{L}'$, there exist $(\mathbf{b}_{r+1}, \ldots, \mathbf{b}_n) \in \mathcal{L}$ such that $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is a basis of $\mathcal{L}$.*

For a lattice $\mathcal{L}(\mathbf{B})$ defined by the basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, we can naturally obtain a subset of its first r basis vectors $\mathbf{B}_r = (\mathbf{b}_1, \ldots, \mathbf{b}_r)$ with $r < n$, and $\mathcal{L}(\mathbf{B}_r)$ is a primitive sublattice of $\mathcal{L}(\mathbf{B})$. This is noted as $\mathcal{L}(\mathbf{B}_r)$. In general, for any primitive sublattice $\mathcal{L}'$ of $\mathcal{L}$, we have $\mathcal{L}' = \mathcal{L} \cap span(\mathcal{L}')$. Conversely, for any a subspace $E \subset \mathbf{R}^m$ and $\mathcal{L}$ generated by basis $\mathbf{B}$, if the set $\mathbf{B}' = \{\mathbf{b}_i \in \mathbf{B} : \mathbf{b}_i \in E\}$ is linearly independent, then $\mathcal{L}' = \mathcal{L}(\mathbf{B}) \cap E$ is a primitive sublattice generated by the basis $\mathbf{B}'$, which can be augmented to a basis of $\mathcal{L}$ by $\{\mathbf{b}_i \in \mathbf{B} : \mathbf{b}_i \notin E\}$.

Apart from sublattices, another way to lower the dimension of the lattice is by projection. The group property is always preserved by projection of a lattice, but there is no guarantee for the discreteness of the projected lattice. In fact, let $\mathbf{B} = \begin{pmatrix} 1 & 0 \\ \sqrt{2} & 1 \end{pmatrix}$ be the basis for lattice $\mathcal{L}$, then its projection over the first coordinate is not discrete, because the set $\mathbb{Z} + \sqrt{2}\mathbb{Z}$ is dense over $\mathbb{R}$.

In order to ensure discreteness, we have to restrict the subspace corresponding to the projection. In the previous example, the projection of the lattice is not discrete because both its basis vectors are projected into the subspace, and it falls into the unfortunate case where no integer combination of these two projected vectors can be zero. Intuitively, when we project lattice $\mathcal{L}$ into a subspace of $span(\mathcal{L})$ of smaller dimension, then its rank would lower down accordingly, and we hope that some of its basis vectors would be projected to zero. For example, this would be the case if we project orthogonally a lattice over the orthogonal supplement of a primitive sublattice.

Lemma 2.1.15 (Projected lattice). *Let $\mathcal{L}$ be a n-rank lattice in $\mathbb{R}^m$, and $\mathcal{L}'$ be a d-rank primitive sublattice of $\mathcal{L}$: $1 \leqslant d \leqslant n$. Let $\pi_{\mathcal{L}'}$ denote the orthogonal projection over $span(\mathcal{L}')^\perp$. Then $\pi_{\mathcal{L}'}(\mathcal{L})$ is a lattice of rank $n - r$.*

Proof. Suppose $\mathbf{B}_r = \{\mathbf{b}_1, \ldots, \mathbf{b}_r\}$ is a basis of $\mathcal{L}'$ which can be completed into a basis of $\mathcal{L}$ in the form of $\mathbf{B}_n = \mathbf{B}_r \cup \{\mathbf{b}_{r+1}, \ldots, \mathbf{b}_n\}$. Then $\pi_{\mathcal{L}'}(\mathcal{L})$ is the lattice generated by the basis $\{\pi_{\mathcal{L}'}(\mathbf{b}_{r+1}), \ldots, \pi_{\mathcal{L}'}(\mathbf{b}_n)\}$, where $\pi_{\mathcal{L}'}(\mathbf{b}_i, \forall r < i \leqslant n)$ are linearly independent vectors. □

Very often, when we define a lattice with a given basis $\mathbf{B}$, the order of the basis vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is also fixed, and this order is very important. Then its primitive sublattice $\mathcal{L}(\mathbf{B}_r) : r < n$, and the projected lattice $\pi_{\mathcal{L}(\mathbf{B}_r)}(\mathcal{L})$ decompose the lattice into two parts in orthogonal subspaces $span(\mathbf{B}_r)$ and $span(\mathbf{B}_r)^\perp$ respectively. Especially, for integers i, j such that $1 \leqslant i \leqslant j \leqslant n$, we can define the projection of the sublattice $\mathbf{B}_j$ over the orthogonal supplement of $span(\mathbf{B}_i)$, and as a result the projected sublattice $\pi_{\mathcal{L}(\mathbf{B}_i)}(\mathbf{B}_j)$ is used to disclose the "local structure" of a lattice much like slicing the lattice at indices i and j. This decomposition is very useful in many applications.

For convenience of notation, when the basis is clear in the context, we will use π_i to mean the projection $\pi_{\mathcal{L}(\mathbf{B}_{i-1})}$ for $1 < i \leqslant n$, and by convention we use π_1 to mean the identity. We will also use $\mathbf{B}_{[i,j]}$ to mean the basis $\{\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_j)\}$, therefore the previous projected sublattice $\pi_{\mathcal{L}(\mathbf{B}_i)}(\mathbf{B}_j), 1 \leqslant i \leqslant j \leqslant n$ can be denoted by $\mathcal{L}(\mathbf{B}_{[i,j]})$.

2.1.4 Gram-Schmidt orthogonalization

Unlike in the vector space $\mathbb{R}^n$ where we can always find orthonormal bases, for lattices, the typical situation is that there is no orthogonal basis. But we still have a Gram-Schmidt Orthogonalization process. Given a set of linearly independent vectors $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$, they produce a set of vectors $\{\mathbf{b}_1^\star, \dots, \mathbf{b}_n^\star\}$ which are orthogonal between each other. But with lattices, there are two main differences compared to vector spaces:

1. Apart from $\mathbf{b}_1^\star$, the vectors $\mathbf{b}_2^\star, \dots, \mathbf{b}_n^\star$ usually do not belong to the lattice.

2. Their norms $\|\mathbf{b}_1^\star\|, \dots, \|\mathbf{b}_n^\star\|$ are not unitary, and are not equal in general.

Definition 2.1.16 (Gram-Schmidt Orthogonalization)**.** *Let $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ be linearly independent vectors in $\mathbb{R}^m$. The Gram-Schmidt Orthogonalization (GSO) is the orthogonal family $(\mathbf{b}_1^\star, \dots, \mathbf{b}_n^\star)$ where $\mathbf{b}_i^\star$ is defined as $\pi_i(\mathbf{b}_i)$.*

The vectors in the orthogonal family can be computed recursively as follows:

$$\mathbf{b}_i^\star = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^\star, \quad \text{where } \mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^\star \rangle}{\|\mathbf{b}_j^\star\|^2} \text{ for all } 1 \leqslant j < i \leqslant n$$

Each $\mathbf{b}_j^\star$ is obtained by eliminating from $\mathbf{b}_j$ the component in $span(\mathbf{b}_1, \dots, \mathbf{b}_{j-1})$. Actually, this process can also be described as multiplication of an elementary matrix. Consequently, the orthogonalization process can be described in terms of decomposing the basis $\mathbf{B}$ into a product of two matrices:

$$\begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \vdots \\ \mathbf{b}_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ \mu_{2,1} & 1 & 0 & \dots & 0 \\ \mu_{3,1} & \mu_{3,2} & 1 & \dots & 0 \\ \vdots & & \vdots & \ddots & \\ \mu_{n,1} & \mu_{n,2} & \dots & \mu_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{b}_1^\star \\ \mathbf{b}_2^\star \\ \mathbf{b}_3^\star \\ \vdots \\ \mathbf{b}_n^\star \end{pmatrix} \tag{2.1}$$

Here, the matrix $\mathbf{M} = (\mu_{i,j})_{0<i\leqslant n, 0<i\leqslant n}$ is lower-triangular with 1 on its diagonal, and the row vectors in $(\mathbf{b}_1^\star, \dots, \mathbf{b}_n^\star)$ are orthogonal to each other. In fact, we can further normalize the vectors $\mathbf{b}_i^\star$. This operation can also be described as a product of two matrices. As a result, the basis is decomposed into a product of three matrices: the matrix $\mathbf{M}$, a diagonal matrix $\mathbf{D}$ and an $n \times m$ matrix $\mathbf{O}$.

$$\begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_n \end{pmatrix} = \mathbf{M} \cdot \begin{pmatrix} \|\mathbf{b}_1^\star\| & 0 & \dots & 0 \\ 0 & \|\mathbf{b}_2^\star\| & & \\ \vdots & & \ddots & \\ 0 & & & \|\mathbf{b}_n^\star\| \end{pmatrix} \begin{pmatrix} \mathbf{b}_1^\star / \|\mathbf{b}_1^\star\| \\ \mathbf{b}_2^\star / \|\mathbf{b}_2^\star\| \\ \vdots \\ \mathbf{b}_n^\star / \|\mathbf{b}_n^\star\| \end{pmatrix}$$

The matrix $\mathbf{D}$ is a diagonal matrix of rank n, so normally it suffices to store the vector $(\|\mathbf{b}_1^\star\|, \dots, \|\mathbf{b}_n^\star\|)$, which saves space.

The rows of the matrix $\mathbf{O}$ are orthonormal vectors, which means a rigid rotation from the canonical basis of $\mathbb{R}^m$. When $n < m$, this signifies throwing away the last $m - n$ axes after the rigid rotation. Therefore, up to a rotation, all lattices of rank n are congruent to some full-rank lattices of rank n. For example, $\mathcal{L}(\mathbf{B})$ is congruent to the lattice $\mathcal{L}(\mathbf{M} \cdot \mathbf{D})$.

In fact, it is easy to see that $(\mathbf{MD}) \cdot (\mathbf{O})$ is a QR-decomposition of $\mathbf{B}$, and when given a QR-decomposition $\mathbf{B} = \mathbf{QR}$, then we can compute the GSO decomposition by taking $\mathbf{O} = \mathbf{R}$, the diagonals of $\mathbf{D}$ to be the same as the diagonals of $\mathbf{Q}$, and each entry of $\mathbf{M}$ to be

$$\mu_{i,j} = \begin{cases} 0 & j > i \\ 1 & j = i \\ q_{i,j}/q_{j,j} & j < i \end{cases}$$

This triangular decomposition of $\mathbf{B}$ makes it easier to analyze the structure of the basis, especially, for the basis of the projected sublattice $\mathbf{B}_{[i,j]}$. In fact, $\mathbf{B}_{[i,j]}$ can be expressed as the product of truncations of $\mathbf{M}$, $\mathbf{D}$, and $\mathbf{O}$:

$$\mathbf{B}_{[i,j]} = \begin{pmatrix} \pi_i(\mathbf{b}_i) \\ \pi_i(\mathbf{b}_{i+1}) \\ \vdots \\ \pi_i(\mathbf{b}_j) \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \mu_{i+1,i} & 1 & \dots & 0 \\ \vdots & & \ddots & \\ \mu_{j,i} & \mu_{j,i+1} & \dots & 1 \end{pmatrix} \begin{pmatrix} \|\mathbf{b}_i^\star\| & 0 & \dots & 0 \\ 0 & \|\mathbf{b}_{i+1}^\star\| & \dots & \\ \vdots & & \ddots & \\ 0 & & & \|\mathbf{b}_j^\star\| \end{pmatrix} \begin{pmatrix} \mathbf{b}_i^\star/\|\mathbf{b}_i^\star\| \\ \mathbf{b}_{i+1}^\star/\|\mathbf{b}_{i+1}^\star\| \\ \vdots \\ \mathbf{b}_j^\star/\|\mathbf{b}_j^\star\| \end{pmatrix}.$$

Whereas the volume of $\mathbf{B}_{[i,j]}$ is

$$vol(\mathcal{L}(\mathbf{B}_{[i,j]})) = \prod_{k=i}^{j} \|\mathbf{b}_k^\star\|.$$

Therefore, with the GSO decomposition $\mathbf{M}$, $\mathbf{D}$ and $\mathbf{O}$, we also get the representation for all $\mathbf{B}_{[i,j]}$ simultaneously. More generally, when we have a set of vectors $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_l\} \in \mathbb{R}^m$ which generates a lattice $\mathcal{L}$ of rank n, $n \leqslant l \leqslant m$. We will have the following decomposition,

$$\mathbf{P} \cdot \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_l \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ \mu_{2,1} & 1 & 0 & \dots & 0 \\ \mu_{3,1} & \mu_{3,2} & 1 & \dots & 0 \\ \vdots & & \vdots & \ddots & \\ \mu_{n,1} & \mu_{n,2} & & \mu_{n,n-1} & 1 \\ & & & & \mu_{n+1,n} \\ & & \vdots & & \vdots \\ \mu_{l,1} & \mu_{l,2} & \dots & \mu_{l,n} & \mu_{l.n} \end{pmatrix} \cdot \begin{pmatrix} \|\mathbf{b}_1^\star\| & 0 & \dots & 0 \\ 0 & \|\mathbf{b}_2^\star\| & & \\ \vdots & & \ddots & \\ 0 & & & \|\mathbf{b}_n^\star\| \end{pmatrix} \begin{pmatrix} \mathbf{b}_1^\star/\|\mathbf{b}_1^\star\| \\ \mathbf{b}_2^\star/\|\mathbf{b}_2^\star\| \\ \vdots \\ \mathbf{b}_n^\star/\|\mathbf{b}_n^\star\| \end{pmatrix}$$

where $\mathbf{P}$ is a $l \times l$ permutation matrix. The first n elements of $\mathbf{P} \cdot \mathbf{B}$ is a basis of $\mathcal{L}$. This gives an algorithm that computes a basis from a generating set of $\mathcal{L}$.

In addition, this also shows that it is easy to distinguish if a vector $\mathbf{v}$ belongs to a lattice $\mathcal{L}$. If $\mathbf{v}$ is an integer combination of $\mathbf{B}$, which means that there exists an integer vector $\mathbf{x}$ such that $\mathbf{v} = \mathbf{xB}$, then the equation $\mathbf{v} = \mathbf{xMDO}$ must have an integer solution. Multiplying $\mathbf{O}^T \cdot \mathbf{D}^{-1}$ on both sides we get:

$$\mathbf{v} \cdot \mathbf{O}^T \cdot \mathbf{D}^{-1} = \mathbf{xM}.$$

Here, $\mathbf{O}^T$ is the transpose of $\mathbf{O}$ such that $\mathbf{O}\mathbf{O}^T = I_{n,n}$, where $I_{n,n}$ is an identity matrix of dimension $n \times n$, and the inverse of $\mathbf{D}$ is given by $diag(\|\mathbf{b}_1^\star\|^{-1}, \dots, \|\mathbf{b}_n^\star\|^{-1})$. And $\mathbf{M}$ is already a lower-triangular matrix, thus the solution of this system should be straight-forward. This procedure also solves the following problem: Given a vector basis $\mathbf{B}$ of rank n and a point $\mathbf{v}$ in the lattice $\mathcal{L}(\mathbf{B})$, determine the integer vector $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{Z}^n$ such that $\mathbf{v} = \mathbf{xB}$.

Algorithm 1 Determine membership and find the decomposition

```
Input: The basis of a lattice $\mathbf{B}$, and a vector $\mathbf{v}$
Output: Determine whether $\mathbf{v} \in \mathcal{L}(\mathbf{B})$. If so, compute $\mathbf{x} \in \mathbb{Z}^n$ such that $\mathbf{v} = \mathbf{xB}$ if $\mathbf{v} \in \mathcal{L}(\mathbf{B})$.
1: Calculate the GSO of the basis $[\mathbf{M}, \mathbf{D}, \mathbf{O}] \leftarrow$ decompose$(\mathbf{B})$.
2: Solve the linear system $\mathbf{v}\mathbf{O}^T\mathbf{D}^{-1} = \mathbf{xM}$.
3: if $\mathbf{v} = \mathbf{xB}$ and $\mathbf{x} \in \mathbb{Z}^n$ then
4:     return (yes,$\mathbf{x}$)
5: else
6:     return no
7: end if
```

2.2 Hard lattice problems

One of the great advantage of lattice-based cryptography is its hardness guarantee. This hardness lies in two facts. First of all, many lattice problems are known to be NP-hard, whereas for integer factoring and discrete log, we do not yet know their concrete complexity class. Secondly, thanks to a worst-case to average-case reduction, we are able to define a lattice problem over a class of hardest instances, such that its hardness guarantees the hardness of many lattice problems over random lattices. This means that for many lattice problems, if an algorithm is able to solve any instance with non-negligible probability, it also solves hardest instances with high probability.

Most of these hard problems are related with short vectors in the lattice. In particular, two problems, the shortest vector problem and the closest vector problem have a central position in all hard problems.

2.2.1 Shortest vectors

The norm of a shortest nonzero vector in the lattice is also called $\lambda_1(\mathcal{L})$. This is the minimal distance. If we consider lattices with fixed volume $vol(\mathcal{L}) = 1$, then $\lambda_1(\mathcal{L})$ can be arbitrarily close to 0, for example in the lattice generated by the basis $\begin{pmatrix} \epsilon & 0 \\ 0 & 1/\epsilon \end{pmatrix}$, there exists a vector of norm ϵ which can be arbitrarily small. Conversely, it turns out that there is a maximum value for $\lambda_1(\mathcal{L})$ among all lattices of fixed volume.

Definition 2.2.1 (Hermite's constant). *The supremum of $\lambda_1(\mathcal{L})^2/vol(\mathcal{L})^{2/n}$ over all $\mathcal{L}$ of dimension n is denoted by γ_n, and is called* Hermite's constant of dimension *n.*

It can be proved that this supremum is always reached by one or some lattices. The exact values of γ_n is very difficult to find, and they are only known for $1 \leqslant n \leqslant 8$, and for $n = 24$. For these dimensions the lattices reaching this Hermite's constant are also known.

For general n, there are already asymptotic bounds. For example, Minkowski gave an upper bound of γ_n with is linear in n.

$$\gamma_n < 1 + n/4 \tag{2.2}$$

This is a consequence of the following theorem.

Theorem 2.2.2. *(Minkowski's Convex Body Theorem) Let $\mathcal{L}$ be a full-rank lattice of $\mathbb{R}^n$. Let S be a measurable subset of $\mathbb{R}^n$, convex, symmetric with respect to 0, such that $vol(S) > 2^n \cdot vol(\mathcal{L})$. Then S contains at least one non-zero point of $\mathcal{L}$.*

Proof. The proof of this theorem uses Blichfeldt's lemma, which states that for a measurable set S with $vol(S) > vol(\mathcal{L})$, there are at least two different points $\mathbf{x}, \mathbf{y} \in S$ such that $\mathbf{x} - \mathbf{y} \in \mathcal{L}$. [59] Now, consider the lattice $2\mathcal{L} = \{2\mathbf{v} : \mathbf{v} \in \mathcal{L}\}$, it has volume $2^n \cdot vol(\mathcal{L})$, thus S should contain two distinct points $\mathbf{x}, \mathbf{y} \in S$ such that $\mathbf{x} - \mathbf{y} \in 2\mathcal{L}$. And by symmetry $-y$ also belongs to S, and as a result of convexity $(\mathbf{x} - \mathbf{y})/2 \in \mathcal{L}$ is a non-zero lattice point. □

Definition 2.2.3 (n-ball)**.** *A n-ball of radius r is the generalization of a closed ball in dimension n, defined as $B_n(r) = \{\sum_{i=1}^n x_i^2 \leqslant r^2\}$. Its volume is*

$$v_n(r) = \frac{\pi^{n/2}}{\Gamma(n/2+1)} \cdot r^n \approx \left(\frac{2\pi e}{n}\right)^{n/2} \cdot r^n.$$

Using Stirling's formula, we can establish the bounds:

$$\left(\frac{2\pi e}{n}\right)^{n/2} \cdot \frac{1}{2\pi n} \cdot r^n \leqslant v_n(r) \leqslant \left(\frac{2\pi e}{n}\right)^{n/2} \cdot \frac{\sqrt{n/2}}{e} \cdot r^n$$

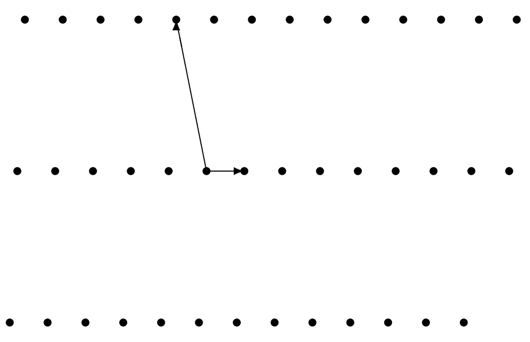
By default, we mean $r = 1$, which refers to a unit n-ball.

Let $\mathcal{L}$ be an arbitrary lattice dimension n, for any n-ball $B_n(r)$ such that $v_n(r) > 2^n \cdot vol(\mathcal{L})$, according to Minkowski's convex body theorem, $B_n(r)$ contains a non-zero vector of $\mathcal{L}$. This implies that $\lambda_1(\mathcal{L}) \leqslant r$. Since this holds for arbitrary $\mathcal{L}$ of dimension n, we obtain

$$\gamma_n \leqslant \frac{4}{v_n^{2/n}} \leqslant \frac{2n}{\pi e} \cdot (2\pi n)^{2/n} < 1 + n/4$$

The definition of $\lambda_1(\mathcal{L})$ is straightforward, it measures the minimal distance between two lattice vectors. On the other hand, this gives only information in one direction, see fig. 2.2. If we want to explore the next closest distances to other neighbors, we can similarly define $\lambda_2(\mathcal{L}), \ldots$. But $\lambda_2(\mathcal{L})$ is not simply defined as the second shortest vector in the lattice, because in some cases it will produce parallel vectors to a shortest vector, which is the illustrated case in fig. 2.2. In general, we hope that all vectors having norms $\leqslant \lambda_d$ could span $\mathbb{R}^d$. This is described in the following definition initially given by Minkowski.

Figure 2.2: A lattice in $\mathbb{R}^2$ which has big $\lambda_2(\mathcal{L})/\lambda_1(\mathcal{L})$ ratio



Definition 2.2.4 (Successive Minima)**.** *Let $\mathcal{L}$ be a lattice of dimension n. For $i = 1, \ldots, n$, the i-th successive minimum is defined as*

$$\lambda_i(\mathcal{L}) = \min_r\{B_n(r) \cap \mathcal{L} \text{ contains at least } i \text{ linear independent vectors}\}$$

Clearly, the minima are increasing: $\lambda_1(\mathcal{L}) \leqslant \lambda_2(\mathcal{L}) \leqslant \cdots \leqslant \lambda_n(\mathcal{L})$

Similarly to the upper bound for $\lambda_1(\mathcal{L})$ given by Minkowski, there exists an upper bound for successive minima.

Theorem 2.2.5 (Minkowski's Second Theorem). *Let $\mathcal{L}$ be a lattice of rank n, we have*

$$\left(\prod_{i=1}^{n}\lambda_i(\mathcal{L})\right)^{1/n} \leqslant \sqrt{\gamma_n}\cdot vol(\mathcal{L})^{1/n}. \tag{2.3}$$

The interested reader is referred to the book [85] for a proof of this theorem. One interesting remark is that for n-rank lattice $\mathcal{L}$, although we can find a set of linearly independent vectors reaching $\lambda_1(\mathcal{L}),\dots,\lambda_n(\mathcal{L})$ simultaneously, however, this set does not necessarily form a basis. In fact, as soon as $n = 5$, there is a counter-example. Consider the lattice generated by the following basis,

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

All of its maxima are 2, but it has no basis reaching all these maxima.

2.2.2 Shortest vector problem

In practical settings of cryptology, one often deals with integer lattices. Here is the formal definition of SVP which serves as the basis for many cryptographic primitives.

Definition 2.2.6 (Shortest Vector Problem, SVP). *Given a basis $\mathbf{B}$ of the lattice $\mathcal{L} \subset \mathbb{Z}^m$, find a nonzero vector $\mathbf{v}$ in $\mathcal{L}$ such that*

$$\|\mathbf{u}\| = \min_{\mathbf{v}\in\mathcal{L}(\mathbf{B}),\|\mathbf{v}\|\neq 0} \|\mathbf{v}\|.$$

This is also called the exact SVP problem, in comparison with several approximate versions of SVP. The problem does not state precisely what the input basis should be like. But obviously $\mathbf{B}$ should not contain short vectors that are already very close to the answer, for example, for full rank lattices, bases are usually given in HNF in this problem.

The exact SVP problem is known to be NP-hard under randomized reductions [4]. In many cryptographic applications, we are also interested in approximate versions of the problems. For example, we can relax the target vector by a factor of γ.

Definition 2.2.7 (Approx-SVP$_\gamma$, ASVP$_\gamma$). *Given a basis $\mathbf{B}$ of the n-dimension lattice $\mathcal{L} \subset \mathbb{Z}^m$ and a factor γ, find a non-zero vector $\mathbf{v}$ in $\mathcal{L}$ such that $\|\mathbf{v}\| \leqslant \gamma \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$.*

In practice however, we usually do not know the value of $\lambda_1(\mathcal{L})$ in advance, it is sometimes even difficult to judge if a vector is a solution or not, although we have asymptotic bounds on $\lambda_1(\mathcal{L})$. On the other hand, if the bound for the norm of the target vector only depends on the volume of the lattice $vol(\mathcal{L})$, it will be much easier to decide if a vector belongs to the solution set or not.

Definition 2.2.8 (Hermite-SVP$_\gamma$, HSVP$_\gamma$). *Given a basis $\mathbf{B}$ of the n-dimension lattice $\mathcal{L} \subset \mathbb{Z}^m$ and a factor γ, find a non-zero short vector $\mathbf{v}$ in $\mathcal{L}$ such that $\|\mathbf{v}\| \leqslant \gamma \cdot vol(\mathcal{L})^{1/n}$.*

In a lot of cryptographic applications, the lattices have a much shorter vector than the average $\lambda_1(\mathcal{L})$ for random lattices. It motivated to study if the problem of finding short vectors in such lattices would be much easier than the original SVP problem.

Definition 2.2.9 (Unique SVP_γ , uSVP_γ)**.** *Given a basis* $\mathbf{B}$ *of the n-dimension lattice* $\mathcal{L} \subset \mathbb{Z}^m$ *such that* $\lambda_2(\mathcal{L}) > \gamma\lambda_1(\mathcal{L})$*, find a shortest vector of* $\mathcal{L}(\mathbf{B})$*.*

The proof of the hardness results of these problems make use of the gap-version of the lattice problem [43]. In general, a gap-version GAPX_γ of an optimization problem $\mathbf{X}$ is a promise problem where the instance is guaranteed to either have a good optimum (the YES instances) or is far from it (the NO instances). The ratio between the optimum value in the YES and the NO cases is at least γ.

Definition 2.2.10 (GapSVP_γ)**.** *Given a basis* $\mathbf{B}$ *of the n-dimension lattice* $\mathcal{L}$ *and a factor* γ*.* GapSVP_γ *is a promise problem whose*
*- **Yes instances** are all lattices satisfying* $\lambda_1(\mathcal{L}) < d$*, and*
*- **No instances** are all lattices satisfying* $\lambda_1(\mathcal{L}) > \gamma \cdot d$*.*

2.2.3 Closest vector problem

Definition 2.2.11 (Closest Vector Problem, CVP)**.** *Given a basis* $\mathbf{B}$ *of the lattice* $\mathcal{L} \in \mathbb{Z}^m$ *of rank n, and a point* $\mathbf{x} \in \mathbb{Q}^m$*, find a lattice vector* $\mathbf{v}$ *which is closest to* $\mathbf{x}$*:*

$$\mathbf{v} = \arg \min_{\mathbf{w} \in \mathcal{L}(\mathbf{B})} \|\mathbf{w} - \mathbf{x}\|.$$

Similar to the case of SVP, there is an approximate version for CVP:

Definition 2.2.12 (Approx-CVP_γ, ACVP_γ)**.** *Given a basis* $\mathbf{B}$ *of an n-dimensional lattice* $\mathcal{L} \subset \mathbb{Z}^m$ *and a factor* γ*, and a point* $\mathbf{x} \in \mathbb{Q}^n$*, find a vector* $\mathbf{v} \in \mathcal{L}$ *such that* $\|\mathbf{v} - \mathbf{x}\| \leqslant \gamma \cdot dist(\mathcal{L}, \mathbf{x})$*.*

The CVP is reminiscent of a problem in coding theory, which serves as basis for the security of linear codes.

Definition 2.2.13 (General decoding problem for linear codes)**.** *Let* $\mathcal{C}$ *be a* (n, k) *linear code over a finite field* $\mathbb{F}$ *and* $\mathbf{y} \in \mathbb{F}^n$*, find* $\mathbf{x} \in \mathcal{C}$ *such that the distance between* $(\mathbf{x}, \mathbf{y})$ *is minimal. Here, a* (n, k) *linear code is a k-ranked linear subspace of the vector space* $\mathbb{F}^n$*.*

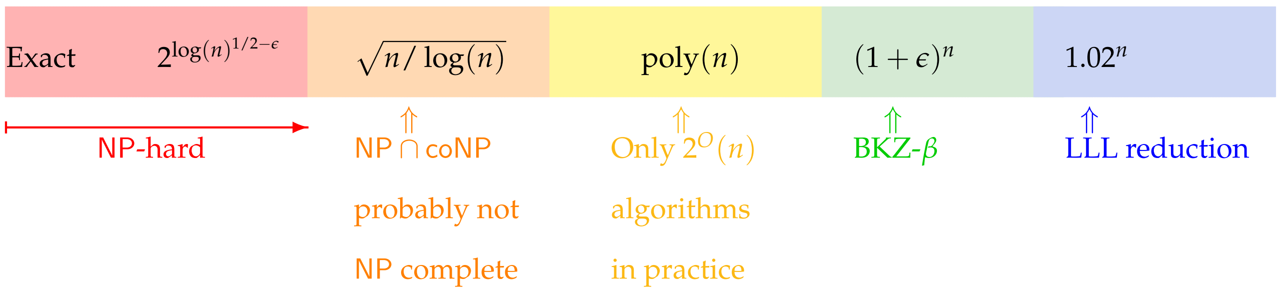
For many cryptographic applications, the lattice, usually served as the public key, is fixed and published for a relatively long period of time, only the point $\mathbf{x}$ which is the ciphertext varies as an input. The adversary is allowed to do any sort of preprocessing and store a polynomial amount of information of the lattice, and the question is whether in this scenario CVP is still hard to compute.

Definition 2.2.14 (CVP_γ with Preprocessing, CVPP_γ)**.** *Given a basis* $\mathbf{B}$ *of an n-dimensional lattice* $\mathcal{L} \subset \mathbb{Z}^m$ *and a factor* γ*, one is allowed to do arbitrary preprocessing and store polynomial amount of information. Given a point* $\mathbf{x} \in \mathbb{Z}^n$*, the computation problem consists of finding a vector* $\mathbf{v} \in \mathcal{L}$ *such that* $\|\mathbf{v}\| \leqslant \gamma \cdot dist(\mathcal{L}, \mathbf{x})$*.*

Both these problems have been proved to be NP-hard [90] [8] [21]. A lot of cryptosystems using these hardness results were proposed. For example, Goldreich-Goldwasser-Halevi (GGH) cryptosystem [32] is based on hardness of CVP. And McEliece cryptosystem is an application for the linear coding problem [53]. The GGH encryption challenges have been solved in [60], while the GGH signature is broken by Nguyen [63] .

According to the work of Arora, Babai, Stern, and Sweedyk [7], CVP is NP-hard to approximate within any constant factor, while SVP is hard to approximate within any constant factor less than $\sqrt{2}$, as shown by Micciancio [54]. In fact, Goldreich, Micciancio, Safra, and Seifert [33] gave a Turing reduction from SVP to CVP, showing that any hardness for CVP implies the same hardness for SVP (but not vice versa). As Regev [2] pointed out that there exist $c > 0$ such that $\mathsf{GapCVP}_{c\sqrt{n}}$ is in $\mathsf{NP} \cap \mathsf{coNP}$, accordingly, we have $\mathsf{GapSVP}_{c\sqrt{n}}$ is in $\mathsf{NP} \cap \mathsf{coNP}$ too. But CVP, SVP cannot be NP-hard to approximate under randomized reductions within $\sqrt{n/\log n}$ unless polynomial hierarchy collapses [2, 31, 36]. By definition, CVPP can not be harder than CVP. In fact, it is shown to be NP-hard to approximate within any constant factor less than $\sqrt{3}$ shown by Regev [71]. And any polynomial algorithm is not likely to be able to approximate within factor $(\log n)^{1/2-\epsilon}$ unless NP has quasi-polynomial time algorithms. The complexity of ASVP [22], [72] is summerized into graphic representation by figure 2.3.

Figure 2.3: ASVP_γ hardness bar


Exact
$2^{\log(n)^{1/2-\epsilon}}$
$\sqrt{n/\log(n)}$
$\mathrm{poly}(n)$
$(1+\epsilon)^n$
1.02^n
NP-hard
NP ∩ coNP probably not NP complete
Only $2^{O}(n)$ algorithms in practice
BKZ-β
LLL reduction

2.2.4 Learning with error

Another important lattice problem is the learning with errors (LWE) problem proposed by Regev [73] [57]. This problem is shown to have various applications in encryption, hash function and signature schemes.

Definition 2.2.15 (Learning with errors(LWE) problem)**.** *Given a modulus q, a rank n, and an "error" probability distribution χ on $\mathbb{Z}_q$, let $A_{\mathbf{s},\chi}$ on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ be defined as follows:*

$$\begin{pmatrix} \mathbf{a}_1, \mathbf{a}_1 \cdot \mathbf{s} + e_1 \\ \mathbf{a}_2, \mathbf{a}_2 \cdot \mathbf{s} + e_2 \\ \vdots \\ \mathbf{a}_m, \mathbf{a}_m \cdot \mathbf{s} + e_m \end{pmatrix},$$

where $\mathbf{a}_i \in \mathbb{Z}_q^n$ are vectors uniformly chosen at random, and $e_i \in \mathbb{Z}_q$ are chosen according to χ and additions are made in $\mathbb{Z}_q$. An algorithm solves LWE *with modulus q and modulus χ if, for any $\mathbf{s} \in \mathbb{Z}_q^n$, given an arbitrary number of independent samples from $A_{\mathbf{s},\chi}$, it outputs $\mathbf{s}$ with non-negligible probability.*

The LWE problem can be informally described as to recover a secret $\mathbf{s}$ given a sequence of "approximate" random linear equations on $\mathbf{s}$. In the scheme proposed by Gentry, Peikert and Vaikuntanathan [30], error is assumed to be a discrete Gaussian distribution $\mathbb{N}(0, \sigma^2)$, and when $\sigma > \sqrt{n}$ this problem is at least as hard as worst- case GapSVP with approximation factor $O(n^2)$ [6], regardless of the value of q [44]. A recent work by Micciancio and Peikert [56] also showed that LWE is hard for binary errors: Let n and m be integers, and $q \geqslant n^{O(1)}$ a sufficiently large polynomially bounded (prime) modulus. Then solving LWE with parameters n, m, q and independent uniformly random binary errors (i.e., in $\{0,1\}$) is at least as hard as approximating lattice problems in the worst case on $\Theta(n/\log n)$-dimensional lattices within a factor $\gamma = \tilde{O}(\sqrt{n} \cdot q)$.

The LWE problem has a dual problem known as the SIS problem presented below. It is first presented in the work of Ajtai [3].

Definition 2.2.16 (Short integer solution, SIS)**.** *Let n and q be integers, where n is the primary security parameter and usually $q = poly(n)$, and let $\beta > 0$. Given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, for some $m = poly(n)$, the problem asks to find a nonzero integer vector $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{Az} = \mathbf{0} \mod q$ and $\|z\| < \beta$.*

For any q that is at least some polynomial in n, solving the SIS problem implies a solution to standard worst-case lattice problems such as GapSVP [74]. The work by Micciancio and Peikert [56] showed hardness results for SIS with small parameters: Let $\beta > \beta_\infty > 1$ be reals, $Z = \{\mathbf{z} \in \mathbb{Z}^m : \|\mathbf{z}\| \leqslant \beta \text{ and } \|\mathbf{z}\|_\infty \leqslant \beta_\infty\}$, and let the $q \geqslant \beta \cdot n^\delta$, for some constant $\delta > 0$, then solving SIS with parameters n, m, q and solution set $Z\backslash\{\mathbf{0}\}$ is at least as hard as approximating lattice problems in the worst case on n-dimensional lattices to within a factor $\gamma = \max\{1, \beta \cdot \beta_\infty/q\}\cdot$.

A more detailed survey on LWE is available [74]. For a more comprehensive survey on complexity of lattice problems in general, we refer to [2,43,59] for detailed discussion and proofs.

2.2.5 Worst-case to average-case reduction

In 1996, Ajtai established the first worst to average reduction for lattice problems [3]. This groundbreaking work established a connection between the worst-case and average-case complexity of certain lattice approximation problems. In cryptographic settings, we require that a problem to be hard to solve on the average, so that when the key is randomly generated, the corresponding function is almost certainly hard to break. Ajtai's work pointed out a class of worst-case instances of lattices problems so that solving these instances with non-negligible probability implies solving the average instances for many lattice problems with overwhelming probability. This arouses substantial theoretical as well as practical interest.

Following this work, a lot of efforts have been paid for building new cryptographic primitives, or trying to establish weaker worst-case assumptions on the complexity of lattice problems, or improving the efficiency of of lattice-based functions in terms of key size or computation time. The topic of worst to average reduction is surveyed in detail by Micciancio [55].

2.3 Random lattices

One of the most important applications of lattice in cryptology is to provide computationally hard problems that can be used to create trap-door functions. However, when we say that a lattice related problem is hard, it is important to specify which category of lattices and what basis we are given. In fact, even for the problems which are considered to be difficult, there exist special lattices for which the problems are easy to solve.

In the most general definitions, we allow $\mathcal{L} \in \mathbb{R}^m$ which means that the coordinates of vectors can take any value in $\mathbb{R}^m$. As a matter of fact, so far it is still very difficult for computers to manage irrational numbers in general. There are usually two solutions. Either we can use floating point numbers for approximation, where we have to be careful to choose a working precision and be aware of errors. Or by restricting the basis vectors to be rational , we can actually multiply by a suitable constant to the basis matrix to make it an integer matrix where exact computations can be performed .

In this section, we restrict our discussion to full-rank lattices. We can do this because any $\mathcal{L} \in \mathbb{R}^m$ of rank n is congruent to a full-rank $\mathcal{L} \in \mathbb{R}^n$ with a proper rotation.

2.3.1 Random lattices in $\mathbb{R}^n$

Up to scaling, all lattices can be normalized to have volume 1. Thus, let $\mathcal{X}_n$ be the set of full-rank lattices of rank n modulo its volume:

$$\mathcal{X}_n = \{\mathcal{L} \in \mathbb{R}^n : vol(\mathcal{L}) = 1\}$$

Lemma 2.3.1. *$\mathcal{X}_n$ is isomorphic to $\mathsf{SL}_n(\mathbb{R})/\mathsf{SL}_n(\mathbb{Z})$.*

Proof. Let us denote $\mathcal{L}_0 = \mathcal{L}((\mathbf{e}_1, \ldots, \mathbf{e}_n)^T)$ the lattice in $\mathcal{X}_n$ generated by the canonical basis of $\mathbb{R}^n$. By the definition of $\mathcal{X}_n$, we have a transitive group action of $\mathsf{SL}_n(\mathbb{R})$ on $\mathcal{X}_n$, i.e. the map

$$\begin{aligned} \mathsf{SL}_n(\mathbb{R}) &\to \mathcal{X}_n \\ \mathbf{T} &\twoheadrightarrow \mathbf{T}.\mathcal{L}(:= \mathcal{L}(\mathbf{T} \cdot (\mathbf{e}_1, \ldots, \mathbf{e}_n)^T)). \end{aligned}$$

is surjective. Hence by the classical result in the theory of group action on topological spaces, we know that

$$\mathcal{X}_n \cong \mathsf{SL}_n(\mathbb{R})/\mathsf{Stabilizer}(\mathcal{L}_0).$$

It suffices to show that $\mathsf{Stabilizer}(\mathcal{L}_0) = \mathsf{SL}_n(\mathbb{Z})$. As was proved in lemma 2.1.8, $\mathsf{GL}_n(\mathbb{Z}) = \{\mathbf{T} : \mathcal{L}(\mathbf{TB}) = \mathcal{L}(\mathbf{B})\}$, which means that $\mathsf{GL}_n(\mathbb{Z})$ is the set of all possible matrices which transform one basis into other bases of the same lattice. Thus the set $\mathsf{SL}_n(\mathbb{Z}) = \mathsf{SL}_n(\mathbb{R}) \cap \mathsf{GL}_n(\mathbb{Z})$ is the stabilizer of the group action. This completes the proof. $\square$

In particular $\mathcal{X}_n$ is a homogeneous space of $\mathsf{SL}_n(\mathbb{R})$. Hence on $\mathcal{X}_n$, there is a translation invariant measure which is unique up to scaling, known as the Haar measure. In fact, the Haar measure is well defined for any locally compact groups. Then, since $\mathsf{SL}_n(\mathbb{Z})$ is a closed subset, the quotient group $\mathsf{SL}_n(\mathbb{R})/\mathsf{SL}_n(\mathbb{Z})$ also has a Haar measure which is also unique up to a scaler. It is proved the Haar measure on this quotient group is finite, and can be normalized to be a natural probability measure, which is called Haar probability measure [91] which we call μ. With this probability measure, we can define what a random lattice is. In theory a random lattice in $\mathcal{X}_n$ is a sample from $\mathcal{X}_n$ according to the measure μ.

2.3.2 Random integer lattices

For integer matrix, the set of all lattices with a fixed volume V: $\mathcal{X}_n(V) = \{\mathcal{L} \subset \mathbb{Z}^m : vol(\mathcal{L}) = V\}$ is finite, thus the notion of randomness is straight forward. It consists of choosing one lattice uniformly from the whole set $\mathcal{X}_n(V)$.

Meanwhile, for an integer lattice $\mathcal{L}$ of volume V of dimension n, when we normalize its basis vectors by a factor of $V^{-1/n}$, its entries become rational numbers, and when we take increasing value of V, we would hope that the behavior of uniformly random integer lattices will approach the uniform random lattices in $\mathcal{X}_n$. Indeed, this is given by following theorem proved in [34]:

Theorem 2.3.2. *$\mathcal{X}_n(V)$ is uniformly distributed in $\mathcal{X}_n$ in the following sense: Given any measurable subset $A \subset \mathcal{X}_n$ whose boundary has zero measure with respect to μ, the fraction of lattices in $\mathcal{X}_n(V)$ that lie in A tends to $\mu(A)$ as V tends to infinity.*

This theorem states that, when we increase the value of V, the uniform distribution over $\mathcal{X}_n(V)$ approaches the uniform distribution over the $\mathcal{X}_n$ set of all lattices modulo volume. This is similar to reducing quantization error in signal processing: we increase the digits in our representation to approach the real value.

However, in many applications, we need to be able to generate a random instance of the lattices with a given rank n. Although for a fixed $V \in \mathbb{Z}$, the size of $\mathcal{X}_n(V)$ is finite, yet it is not clear how to sample efficiently from $\mathcal{X}_n(V)$.

In order to know the elements in $\mathcal{X}_n(V)$, we present the lattice with basis in *Hermite normal form*. In fact, we shall see that an integer lattice has exact one unique basis of Hermite normal form.

Definition 2.3.3 (Hermite Normal Form, HNF)**.** *The basis matrix* **B** *of a full-rank integer lattice $\mathcal{L}$ is in* Hermite Normal Form *if*

1. **B** *is lower triangular,*

2. $\forall 1 \leqslant i \leqslant n, b_{i,i} > 0$*, and*

3. $\forall 1 \leqslant i < j \leqslant n, 0 \leqslant b_{j,i} < b_{i,i}$.

Theorem 2.3.4. *For an integer lattice $\mathcal{L}$, there exist one unique basis* **B** *in Hermite Normal Basis.*

Proof. To show the existence of such a basis we give a constructive proof. Given any integer basis **B**, first of all, we show that we can find an equivalent basis that is lower triangular, with positive elements on its diagonal. Then based on this basis we can find a basis in HNF.

To eliminate the zeros in the upper triangular, we start from the last column to the 2nd column, and for each column, we try to eliminate the non-zero element in the upper diagonal part one by one. Suppose at some point we arrive at the following situation:

$$\begin{pmatrix} \star & \star & 0 & 0 & \dots & 0 \\ \star & \star & \vdots & \vdots & & \vdots \\ \star & \star & 0 & & & \\ b_{j,1} & \dots & b_{j,i} & 0 & & \\ \vdots & & \vdots & \vdots & & \\ b_{i,1} & \dots & b_{i,i} & 0 & & \\ \vdots & & \vdots & \ddots & & \vdots \\ & & & & & 0 \\ b_{n,1} & & b_{n,i} & & & b_{n,n} \end{pmatrix}$$

where both $b_{i,i}$ and $b_{j,i}$ are non-zero. We will perform substitution and swap operations to eliminate the non-zero element in $b_{j,i}$. Notice that there must be some non-zero element on i-th column, otherwise the matrix is degenerated and the lattice is not full-rank. And in fact, by substituting $\mathbf{b}_j$ with $-\mathbf{b}_j$ or $\mathbf{b}_j + x\mathbf{b}_i, x \in \mathbb{Z}$, the represented lattice is not changed. Therefore we do the following algorithm:

Algorithm 2 Compute the Hermite normal form

```
Input: A basis $\mathbf{B}$ of a full-rank integer lattice.
Output: The HNF of this lattice.
1: for $i = n, n-1, \ldots, 2$ do
2:   if $b_{i,i} = 0$, find $k$ such that $b_{k,i} \neq 0$ and swap$(\mathbf{b}_i, \mathbf{b}_k)$.
3:   $\forall 1 \leqslant j \leqslant n$, if $b_{j,i} < 0$ then $\mathbf{b}_j \leftarrow -\mathbf{b}_j$.
4:   for $j = 1, \ldots, i-1$ do
5:     while $b_{j,i} > 0$ do
6:       if $b_{j,i} < b_{i,i}$, swap$(\mathbf{b}_i, \mathbf{b}_j)$.
7:       $\mathbf{b}_j \leftarrow \mathbf{b}_j - \lfloor b_{j,i}/b_{i,i} \rfloor \mathbf{b}_i$, where $\lfloor x \rfloor$ means the biggest integer in $(-\infty, x]$.
8:     end while
9:   end for
10:  for $j = i+1, \ldots, n$ do
11:    $\mathbf{b}_j \leftarrow \mathbf{b}_j - \lfloor b_{j,i}/b_{i,i} \rfloor \mathbf{b}_i$.
12:  end for
13: end for
```

One might notice that the while loop in line $5-8$ is very similar to the Euclidean algorithm for calculating the greatest common divisor. Therefore the proof of termination is obvious. In addition, each time we finish a while loop, we create one more zero entry in the matrix, therefore the while loop will be executed at most $O(n^2)$ times. Thus this algorithm terminates in polynomial time.

For the proof of uniqueness, we suppose that otherwise there exist two distinct basis of HNF $\mathbf{B}$ and $\mathbf{B}'$ for the same lattice $\mathcal{L}$. And let i be the smallest row index such that $\mathbf{b}_i \neq \mathbf{b}'_i$, and let j be the biggest column index such that $b_{i,j} \neq b'_{i,j}$, $j \leqslant i$. As $\mathbf{b}_i$ and $\mathbf{b}'_i$ belongs to $\mathcal{L}$, thus any integer combination of them belongs to $\mathcal{L}$ too, especially one of their combination $\tilde{\mathbf{b}}$ has value $d = gcd(b_{i,j}, b'_{i,j})$ on its j-th component, and this is $\tilde{\mathbf{b}}$'s last non-zero component by the choice of j. Therefore d must be some multiple of $b_{j,j}$, because of the lower-triangular structure of the basis matrix. This means that $b_{i,j} \geqslant d \geqslant b_{j,j}$, which violates the property of basis in HNF, therefore absurd. □

Remark. *The one-to-one correspondence relationship between integer lattice and basis in HNF facilitates the sampling of random integer matrices. In fact, it suffices to enumerate all possible basis in HNF to list all integer lattices. Besides, when presenting basis in HNF, it is easy to see that the volume of the lattice is the product of all the diagonal elements.*

The situation is even simpler when V is a prime p. In this case, the elements in $\mathcal{X}_n(V)$ takes one of the following form, according to different possible position of p in its diagonal:

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & p \end{pmatrix}, \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & p & \\ & & x_1 & 1 \end{pmatrix}, \ldots, \begin{pmatrix} p & & & \\ x_1 & \ddots & & \\ \vdots & & 1 & \\ x_{n-1} & & & 1 \end{pmatrix}$$

we say a basis $\mathbf{B}$ such that $\mathcal{L}(\mathbf{B}) \in \mathcal{X}_n(p)$ is of category i if its i-th diagonal element is p. Category 1 has p^{n-1} elements, while the total number of the rest categories sums up to $\frac{p^{n-1}-1}{p-1}$. When p is large enough, compared to category 1, other categories form a negligible minority. Indeed, the following theorem, attributed to [34] again, confirms that category 1 matrices alone is uniform distributed, as p grows to infinity.

Theorem 2.3.5. *Let p be a prime number, and $x_1, \ldots, x_n$ be numbers randomly and independently chosen from $\{1, \ldots, p\}$, then we define $\mathcal{Y}_n(p)$ the set of all lattices whose basis are of the following form:*

$$\begin{pmatrix} p & 0 & \ldots & \\ x_1 & 1 & \ldots & \\ \vdots & & \ddots & \\ x_{n-1} & 0 & \ldots & 1 \end{pmatrix},$$

then the uniform distribution over $\mathcal{Y}_n(p)$ is statistically close to to the uniform distribution over $\mathcal{X}_n$.

Remark. *This theorem directly yields an simple alternate algorithm of sampling integer lattices, which satisfies randomness guarantee, although it is also possible to generate the uniform distribution of integer lattices of given volume, using a different process. It suffices to generate a big prime number p along with $n-1$ random integers in the interval $[1, p-1]$. However, it is not clear whether the theorem still holds if we remove the condition that the volume is a prime p.*

2.4 Lattice reduction

Among all the bases of a lattice, some bases are superior to others in better revealing the structure of the lattice. Informally, lattice reduction is the process of searching such "superior" bases. In practice, lattice reduction plays a core role in many lattice problems. The efficiency of reduction algorithms is a key element to evaluate the security of many lattice-based crypto-systems.

Unlike bases in a vector space, a lattice usually does not have an orthogonal basis. However, we can attempt to get a basis that is close to being orthogonal. Since the lattice volume – the volume of the parallelepiped spanned by lattice basis is fixed, better orthogonality also means shorter basis.

From the introduction of successive minima, we know that even the vectors in the shortest possible basis could exceed the values of successive minima. In theory, the shortest basis can be defined by the leading element among all bases by sorting $(\|\mathbf{b}_1\|, \ldots, \|\mathbf{b}_n\|)$ in lexicographic order.

From an algorithmic point of view, the notion of Hermite-Korkine-Zolotarev(HKZ) reduction is more interesting. An HKZ-reduced basis is more relaxed than the shortest basis, and still, it approximates the successive minima. Besides it has local properties which allow inductive analysis using this properties. Besides, many other weaker notions and algorithms stem from this HKZ-reduction, which makes it very important.

2.4.1 Hermite-Korkine-Zolotarev reduction

Size-reduction is a much weaker reduction notion introduced by Hermite. Intuitively speaking, in a size-reduced basis, for each pair of basis vectors $\mathbf{b}_i$ and $\mathbf{b}_j$ with $i < j$, it assures that $\|\mathbf{b}_j\| = \min_{a \in \mathbb{Z}} \|\mathbf{b}_j + a\mathbf{b}_i\|$. In other words, $\mathbf{b}_j$ can not be made shorter by simply subtracting a multiple of $\mathbf{b}_i$.

Definition 2.4.1 (Size-reduced). *A basis $\mathbf{b} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of a lattice is* size-reduced *if its Gram-Schmidt orthogonalization satisfies, for all $1 \leqslant j < i \leqslant n$,*

$$|\mu_{i,j}| \leqslant \frac{1}{2}.$$

The following is a description of the size-reduction algorithm.

Algorithm 3 Size reduction algorithm

```
Input: A basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ of the lattice $\mathcal{L}$.
Output: A size-reduced basis $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$
1: Calculate the GSO of the basis $[\mathbf{M}, \mathbf{D}, \mathbf{O}] \leftarrow \text{decompose}(\mathbf{B})$.
2: for $i = 2$ to $n$ do
3:   for $j = i - 1$ downto 1 do
4:     $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lceil \mu_{i,j} \rfloor \mathbf{b}_j$
5:     for $k = 1$ to $j$ do
6:       $\mu_{i,k} \leftarrow \mu_{i,k} - \lceil \mu_{i,j} \rfloor \mu_{j,k}$
7:     end for
8:   end for
9: end for
```

The size reduction algorithm runs in polynomial time with respect to the lattice rank n. Computationally, size-reduction of a lattice basis is analogous to Gram-Schmidt orthogonalization of a basis in $\mathbb{R}^m$. Indeed, if we were working in a vector space, and by replacing the 4th line by $\mathbf{b}_i \leftarrow \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j$ we would obtain an orthogonal basis. In the size-reduction algorithm, we round $\mu_{i,j}$ so that $\mathbf{b}_i$ remains an integer combination of the basis and $\mathbf{b}_i \in \mathcal{L}$. This ensures that the projection of $\mathbf{b}_i$ onto the line in the direction of $\mathbf{b}_j^\star$ for all $j < i$ is smaller than $\frac{1}{2}\|\mathbf{b}_j^\star\|$, and putting it all together, we have the following inequality:

$$\|\mathbf{b}_i^\star\| \leqslant \|\mathbf{b}_i\| \leqslant \|\mathbf{b}_i^\star\|^2 + \frac{1}{4}\sum_{j=1}^{i-1} \|\mathbf{b}_j^\star\|^2 \tag{2.4}$$

The HKZ reduction notion is related to shortest vectors. Starting from $i = 1$, we can inductively assign $\mathbf{b}_i^\star$ to be the shortest vector of $\pi_i(\mathcal{L})$.

Definition 2.4.2 (HKZ-reduced)**.** *A basis $\mathbf{B} = \{\mathbf{b}_1, \dots \mathbf{b}_n\}$ of a lattice is HKZ-reduced if it is size-reduced and such that for all $1 \leqslant i \leqslant d$, $\|\mathbf{b}_i^\star\| = \lambda_1(\pi_i(\mathcal{L}))$.*

The algorithm which makes n calls to the SVP algorithm. Its description is given as follows.

Algorithm 4 HKZ reduction algorithm

```
Input: A basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ of the lattice $\mathcal{L}$, a factor $\epsilon$
Output: A HKZ-reduced basis $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$
1: for $i = 1$ to $n - 1$ do
2:   $\mathbf{b}_i' \leftarrow \mathsf{SVP}(\pi_i(\mathcal{L}(\mathbf{B})))$
3:   Compute $\mathbf{x}$ such that $\mathbf{b}_i = \mathbf{x} \cdot \pi_i(\mathbf{B})$ using algorithm 1.
4:   $\mathbf{c} \leftarrow \mathbf{x} \cdot (\mathbf{b}_i, \dots, \mathbf{b}_n)$.
5:   $\mathbf{B} \leftarrow$Find proper $(\mathbf{b}_{i+1}, \dots, \mathbf{b}_n)$ to complete $(\mathbf{b}_1, \dots, \mathbf{b}_{i-1}, \mathbf{c}, \mathbf{b}_{i+1}, \dots \mathbf{b}_n)$ into a basis of $\mathcal{L}$.
6:   $\mathbf{B} \leftarrow$Size-reduce($\mathbf{B}$)
7: end for
```

Line 5 is feasible because of the following lemma.

Lemma 2.4.3. *Let $\mathbf{v}$ be the shortest vector of the lattice $\mathcal{L}(\mathbf{B})$, then $\mathcal{L}(\{\mathbf{v}\})$ is a primitive sublattice of $\mathcal{L}(\mathcal{B})$.*

Proof. $\{\mathbf{v}\}$ is a basis for the lattice $\mathcal{L}(\mathbf{B}) \cap \mathit{span}(\mathbf{v})$. □

In the algorithm, $\pi_i(\mathbf{c})$ is the shortest vector of lattice $\pi_i(\mathcal{L}(\mathbf{B}))$, and therefore it can be completed to a basis of $\pi_i(\mathcal{L}(\mathbf{B}))$ in the form of $\pi_i(\mathbf{c}), \pi_i(\mathbf{b}_{i+1}), \ldots, \pi_i(\mathbf{b}_n))$, then $\{\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{c}, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_n$ is a basis of lattice $\mathcal{L}(\mathbf{B})$.

The length of HKZ-reduced basis vectors is a good approximation to the successive minima. Obtaining a HKZ basis is no easier than solving SVP, thus an HKZ reduced basis is unlikely to be obtained by polynomial-time algorithms.

2.4.2 LLL reduction

The first algorithm solving ASVP within bounded approximation factor in polynomial time is the LLL algorithm. The idea is generalized from 2 dimensional case.

Definition 2.4.4 (Lagrange-reduced). *Let $\mathcal{L}$ be a two-rank lattice of $\mathbb{R}^n$. A basis of $(\mathbf{b}_1, \mathbf{b}_2)$ of $\mathcal{L}$ is said to be Lagrange-reduced if and only if $\mathbf{b}_1$ and $\mathbf{b}_2$ satisfies the following two conditions:*

1. *$\|\mathbf{b}_1\| \leqslant \|\mathbf{b}_2\|$, and*

2. *$|\langle \mathbf{b}_1, \mathbf{b}_2 \rangle| \leqslant \|\mathbf{b}_1\|^2/2$.*

The geometrical representation of a pair of Lagrange-reduced basis vectors is given in figure 2.4, where the possible value of $\mathbf{b}_2$ with regard to $\mathbf{b}_1$ is the blue region. It is not hard to deduce that the pair of vectors satisfy the following relationship.

$$\|\mathbf{b}_1\| \geqslant \|\mathbf{b}_2\| \geqslant \|\mathbf{b}_2^\star\| \geqslant \frac{\sqrt{3}}{2}\|\mathbf{b}_1\|$$

Hence we have an upper bound for $\|\mathbf{b}_1\|$:

$$\|\mathbf{b}_1\| \leqslant \left(\frac{4}{3}\right)^{1/4} vol\left(\mathcal{L}(\{\mathbf{b}_1, \mathbf{b}_2\})\right)^{1/2}$$

In fact, this is the only natural reduction notion for vectors of dimension 2, in the sense that both $\mathbf{b}_1$ and $\mathbf{b}_2$ is already optimized:

Theorem 2.4.5. *Let $(\mathbf{b}_1, \mathbf{b}_2)$ be a basis of a two dimension lattice $\mathcal{L}$ of $\mathbb{R}^m$. The basis $(\mathbf{b}_1, vb_2)$ is Lagrange-reduced if and only if $\|\mathbf{b}_1\| = \lambda_1(\mathcal{L})$ and $\|\mathbf{b}_2\| = \lambda_2(\mathcal{L})$.*

Proof. It is easy to see the sufficiency of $\|\mathbf{b}_1\|$ and $\|\mathbf{b}_2\|$ being $\lambda_1(\mathcal{L})$ and $\lambda_2(\mathcal{L})$ respectively. It suffices to prove the necessity. In a pair of Lagrange-reduced basis, we can write $\mathbf{b}_2$ as $r\mathbf{b}_1 + \mathbf{b}_2^\star$ where r and $\mathbf{b}_2^\star$ satisfying $|r| \leqslant 1/2$, and $\|\mathbf{b}_2^\star\|^2 \geqslant (1-r^2)\|\mathbf{b}_1\|^2$. For any vector $\mathbf{v} \in \mathcal{L}$,there exist $x, y \in \mathbb{Z}$ such that $\mathbf{v} = x\mathbf{b}_1 + y\mathbf{b}_2$, and its norm is

$$\begin{aligned}\|\mathbf{v}\| &= \sqrt{(x+ry)^2\|\mathbf{b}_1\|^2 + y^2\|\mathbf{b}_2^\star\|^2} \\ &\geqslant \sqrt{x^2 + 2rxy + y^2} \cdot \|\mathbf{b}_1\|\end{aligned}$$

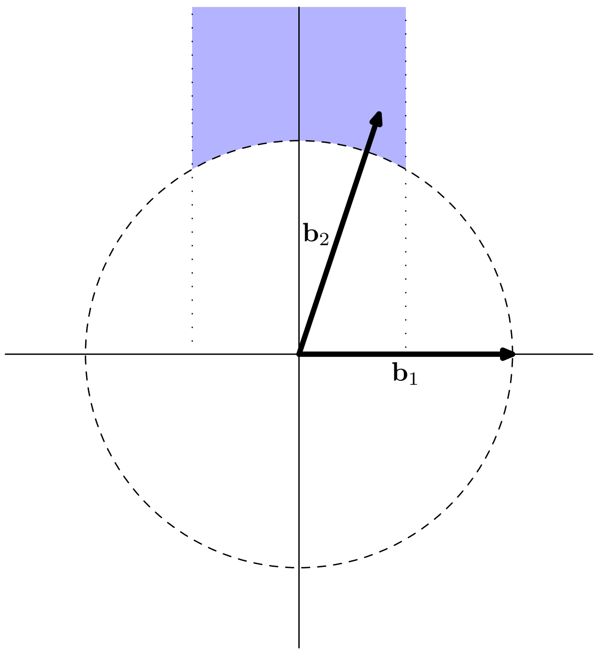
Since $|2rxy| < |xy| < \max(x^2, y^2)$, we have $\|\mathbf{v}\| \geqslant \min(x^2, y^2) \cdot \|\mathbf{b}_1\|$, thus any non-zero $\mathbf{v}$ can not be shorter than $\mathbf{b}_1$.

Because $\|\mathbf{b}_1\|^2 \leqslant \frac{1}{1-r^2}\|\mathbf{b}_1\|^2 \leqslant \frac{4}{3}\|\mathbf{b}_1\|^2$, we have

$$\begin{aligned}\|\mathbf{b}_2\|^2 &= \frac{1}{r^2}\|\mathbf{b}_1\| + \|\mathbf{b}_2^\star\|^2 \\ &\leqslant \frac{1}{4}\|\mathbf{b}_1\|^2 + \|\mathbf{b}_2^\star\|^2 \\ &\leqslant \frac{4}{3}\|\mathbf{b}_2^\star\|^2\end{aligned}$$

For any vector $\mathbf{v}$ that is linearly independent with $\mathbf{b}_1$, $\|\mathbf{v}\|^2 \geqslant y^2\|\mathbf{b}_2^\star\|^2 \geqslant \frac{3}{4}y^2\|\mathbf{b}_2\|^2$. Thus if $|y| > 1$, then $\|\mathbf{v}\| > \|\mathbf{b}_2\|$. With $|y| = 1$, $\|\mathbf{v}\|$ is minimized with $x = 0$, which means that $\mathbf{b}_2$ is the shortest among all vectors that are linearly independent with $\mathbf{b}_1$. □

Figure 2.4: A Lagrange-reduced basis



Given a basis $(\mathbf{b}_1, \mathbf{b}_2)$ of a lattice $\mathcal{L}$, a Lagrange-reduced basis can be obtained by following algorithm which is attributed to Lagrange:

Algorithm 5 Lagrange reduction algorithm

Input: A basis of 2 dimension lattice $\mathcal{L}$.
Output: A Lagrange-reduced basis of $\mathcal{L}$.

```
1: size reduce $(\mathbf{b}_1, \mathbf{b}_2)$.
2: while $\|\mathbf{b}_1\| > \|\mathbf{b}_2\|$ do
3:     swap $\mathbf{b}_1$ and $\mathbf{b}_2$.
4:     size reduce $(\mathbf{b}_1, \mathbf{b}_2)$.
5: end while
```

There are many ways to extend Lagrange reduction notion into arbitrary dimensions. Among them, LLL reduction proposed in 1982 by Lenstra et al. [46] serves is the first one which can be achieved in polynomial time.

Definition 2.4.6 (LLL-reduced)**.** *A basis* $\mathbf{B}$ *is LLL-reduced with factors* (η, δ) *such that* $\frac{1}{4} < \delta < 1$ *and* $1/2 < \eta < \sqrt{\delta}$*, if it is size-reduced in a relaxed sense,*

$$\mu_{i,j} < \eta, \forall i < j \tag{2.5}$$

and its Gram-Schmidt orthogonalization satisfies the Lovasz condition,

$$\|\mathbf{b}^{\star}_{i+1} + \mu_{i+1,i}\mathbf{b}^{\star}_{i}\|^2 \geqslant \delta\|\mathbf{b}^{\star}_{i}\|^2 \tag{2.6}$$

for $1 \leqslant i < n$*.*

In this definition, if the factor δ tends to 1, then condition 2.6 equals to condition 1 in Lagrange reduction. But δ is usually set to be smaller than 1 to guarantee polynomial running time. In the original literature, this factor is fixed to be $\frac{3}{4}$, for the simplicity of presentation. When factor η tends to 0.5, it would be exactly the condition for size reduction, and for many softwares such as NTL and magma, the default value of (η, δ) is set to be close to $(\frac{1}{2} + \epsilon, 1 - \epsilon)$, which improves stability, yields shorter $\|\mathbf{b}_1\|$ but costs longer termination time [83]. With absence of specifying (η, δ), we will follow this convention of softwares. The LLL reduction algorithm [46], which is described in algorithm 7 in the next chapter, ensures the output basis to be LLL-reduced upon termination.

For a LLL reduced basis, the neighboring basis vectors satisfies

$$\|\mathbf{b}^{\star}_{i}\| \leqslant (\delta - \eta^2)^{1/2}\|\mathbf{b}^{\star}_{i+1}\|, \qquad \forall 1 \leqslant i < n \tag{2.7}$$

Altogether, this amounts to the following bound on $\|\mathbf{b}_1\|$:

$$\|\mathbf{b}_1\| \leqslant \left(\delta - \eta^2\right)^{(n-1)/4} \cdot vol(\mathcal{L})^{1/n} \tag{2.8}$$

And the basis vectors are also approximations to $\lambda_i(\mathcal{L})$ within bounded factor:

$$\|\mathbf{b}_i\| \leqslant \left(\delta - \eta^2\right)^{(n-1)/2} \cdot \lambda_i(\mathcal{L}) \tag{2.9}$$

which implies that $\mathbf{b}_1$ provides a solution to $\mathsf{SVP}_{(\delta-\eta^2)^{(n-1)/2}}$. For (η, δ) close to $(0.5, 0.999)$, the approximation factor is upper bounded by 1.075^{n-1}. In practice, when using a random lattice, the average approximation factors is smaller than this upper bound. According to Gama and Nguyen's experiment [23] on numerous random lattice bases, on average we have $\|\mathbf{b}_1\| \approx 1.021^n vol(\mathcal{L})^{1/n}$ in high dimension.

For an input lattice $\mathcal{L} \subset \mathbb{Z}^m$ of dimension n, whose entry size is bounded by $B \in \mathbb{Z}$, the original LLL algorithm runs in time complexity of $O(n^5 m \log^3 B)$. This complexity is later improved to be $O(n^4 m \log B(d + \log B)$. The wide application make LLL reduction algorithm a popular field of study, and its algorithmic details are presented in section 2.5.

2.4.3 BKZ reduction

Block Korkine-Zolotarev(BKZ) reduction with a parameter $2 < \beta < n$ is stronger than LLL reduction. Upon termination, the output basis is guaranteed to be BKZ-reduced, with the following property.

Definition 2.4.7 (BKZ-reduced). *A basis B is BKZ-reduced with a block size β and with a factor (η, δ) such that $1/4 < \epsilon < 1$, if it is LLL-reduced with factor (η, δ), and each $\mathbf{b}_i^\star$ satisfies that $\|\mathbf{b}_i^\star\| = \lambda_1(\mathcal{L}(\mathbf{B}_{[i,k]}))$ for $1 \leqslant i < n$ and $k = \min(i + \beta - 1, n)$.*

This can be viewed as a generalization from LLL-reduction, for $\beta = 2$, the definition amounts to LLL-reduced basis.

To make a basis more reduced than LLL-reduced, the Blockwise-Korkine-Zolotarev (BKZ) reduction algorithm [79] is the best algorithm known in practice. It outputs a BKZ-reduced basis with blocksize $\beta \geq 2$, from an input basis $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ of a lattice $\mathcal{L}$. It starts by LLL-reducing the basis $\mathbf{B}$, then iteratively reduces each local block $B_{[j,min(j+\beta-1,n)]}$ for $j = 1$ to n, to make sure that the first vector of each such block is the shortest in the projected lattice. This gives rise to Algorithm 6, which proceeds in such a way that each block is already LLL-reduced before being enumerated: there is an index j, initially set to 1. At each iteration, BKZ performs an enumeration of the local projected lattice $L_{[j,k]}$ where $k = \min(j + \beta - 1, n)$ to find $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{Z}^n$ such that $\|\pi_j(\sum_{i=j}^k v_i \mathbf{b}_i)\| = \lambda_1(L_{[j,k]})$. We let $h = \min(k + 1, n)$ be the ending index of the new block in the next iteration:

- If $\|\mathbf{b}_j^\star\| > \lambda_1(L_{[j,k]})$, then $\mathbf{b}^{new} = \sum_{i=j}^k v_i \mathbf{b}_i$ is inserted between $\mathbf{b}_{j-1}$ and $\mathbf{b}_j$. This means that we no longer have a basis, so LLL is called on the generating set $(\mathbf{b}_1, \dots, \mathbf{b}_{j-1}, \mathbf{b}^{new}, \mathbf{b}_j, \dots, \mathbf{b}_h)$, to give rise to a new LLL-reduced basis $(\mathbf{b}_1, \dots, \mathbf{b}_h)$.
- Otherwise, LLL is called on the truncated basis $(\mathbf{b}_1, \dots, \mathbf{b}_h)$.

Thus, at the end of each iteration, the basis $B = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ is such that $(\mathbf{b}_1, \dots, \mathbf{b}_h)$ is LLL-reduced. When j reaches n, it is reset to 1, unless no enumeration was successful, in which case the algorithm terminates: the goal of z in Alg. 6 is to count the number of consecutive failed enumerations, to check termination.

Algorithm 6 The Block Korkine-Zolotarev (BKZ) algorithm

```
Input: A basis $B = (\mathbf{b}_1, \dots, \mathbf{b}_n)$, a blocksize $\beta \in \{2, \dots, n\}$, the Gram-Schmidt triangular matrix $\mathbf{U}$ and
    $\|\mathbf{b}_1^*\|^2, \dots, \|\mathbf{b}_n^*\|^2$.
Output: The basis $(\mathbf{b}_1, \dots, \mathbf{b}_n)$ is BKZ-$\beta$ reduced
 1: $z \leftarrow 0$; $j \leftarrow 0$; LLL($\mathbf{b}_1, \dots, \mathbf{b}_n, \mathbf{U}$);// LLL-reduce the basis, and update $\mu$
 2: while $z < n - 1$ do
 3:     $j \leftarrow (j \mod (n-1)) + 1$; $k \leftarrow \min(j + \beta - 1, n)$; $h \leftarrow \min(k + 1, n)$; // define the local block
 4:     $\mathbf{v} \leftarrow$Enum($\mu_{[j,k]}, \|\mathbf{b}_j^*\|^2, \dots, \|\mathbf{b}_k^*\|^2$); // find $\mathbf{v} = (v_j, \dots, v_k) \in \mathbb{Z}^{k-j+1} - \vec{0}$ s.t. $\|\pi_j(\sum_{i=j}^k v_i \mathbf{b}_i)\| =$
        $\lambda_1(L_{[j,k]})$
 5:     if $\mathbf{v} \neq (1, 0, \dots, 0)$ then
 6:         $z \leftarrow 0$; LLL($\mathbf{b}_1, \dots, \sum_{i=j}^k v_i \mathbf{b}_i, \mathbf{b}_j, \dots, \mathbf{b}_h, \mathbf{U}$) at stage $j$; //insert the new vector in the lattice at the start
            of the current block, then remove the dependency in the current block, update $\mu$.
 7:     else
 8:         $z \leftarrow z + 1$; LLL($\mathbf{b}_1, \dots, \mathbf{b}_h, \mathbf{U}$) at stage $h - 1$; // LLL-reduce the next block before enumeration.
 9:     end if
10: end while
```

This forms the main topic of this work and we will discuss in great detail in following chapters.

2.4.4 Characterizing reduced bases

A common way to check the quality of basis is to look at the sequence of Gram-Schmidt norms $\|\mathbf{b}_1^\star\|, \dots, \|\mathbf{b}_n^\star\|$. For random lattices, a good basis has a sequence where the decrease is slow. In practice, it has been observed that the GS norms for basis produced by reduction algorithms have a typical shape, such that the sequence $\|\mathbf{b}_i^\star\|$ form a geometric progression of ratio $\|\mathbf{b}_i^\star\| / \|\mathbf{b}_{i+1}^\star\| = q$, and Fig. 2.5 shows an example in reduced basis of dimension 100.

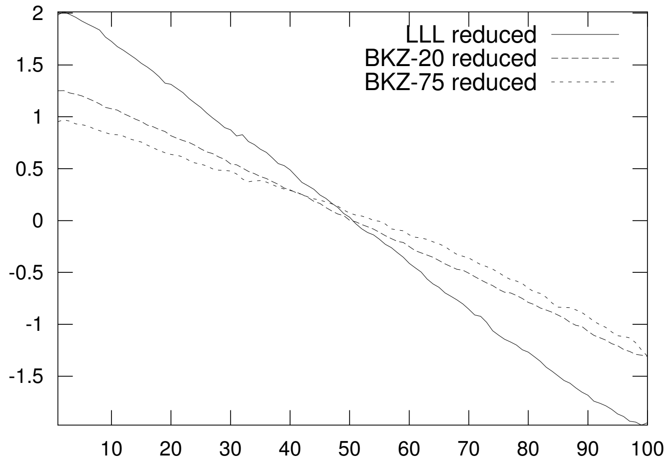


Figure 2.5: Example of Gram-Schmidt norms for a 100 dimensional lattice of unit volume, after different reductions. The y axis shows the value $\ln(\|\mathbf{b}_i^\star\|)$, which is almost linear in indices i.

Intuitively, this can be explained by the fact that many reduction algorithms such as LLL and BKZ reduction have an inductive property: when a basis $\mathbf{B}$ is reduced, so is $\pi_i(\mathbf{B})$ for the same parameter. Especially for LLL reduction, the reduction criteria is imposed only for neighboring two basis vectors.

Numerically, it is usually useful to look at the following numerical values which evaluate the quality of "being-reduced" for a basis.

Definition 2.4.8 (Root Hermite Factor)**.** *The root-Hermite-factor of a basis* $\mathbf{B}$ *is defined as*

$$\delta(\mathbf{B}) = \left(\|\mathbf{b}_1\| / vol(\mathcal{L})^{1/n}\right)^{1/n} \tag{2.10}$$

For random lattices, it has been observed that a typical LLL-reduced basis will have $\delta(\mathbf{B}) = 1.021$, and the δ value of a typical BKZ-reduced basis is related to β, for example, a typical BKZ-20 reduced basis have $\delta(\mathbf{B}) = 1.014$ and BKZ-30 reduced basis have $\delta(\mathbf{B}) = 1.0134$.

Table 2.1: Average root Hermite factor for different reduced basis of random lattices

	LLL	BKZ-20	BKZ-30	BKZ-50	BKZ-100
$\delta(\mathbf{B})$	1.021	1.014	1.0134	1.012	1.009

It is clear BKZ reduction with bigger β outputs much reduced basis, as root Hermite factor approaches to 1. But it gets increasingly harder to get closer to 1. This is illustrated in figure 2.6, which shows average root-Hermite factor for BKZ-β with β up to 250.

The prediction and significance of root Hermite factor of reduction is a central topic of this work and more discussions will be presented several times in later chapters.

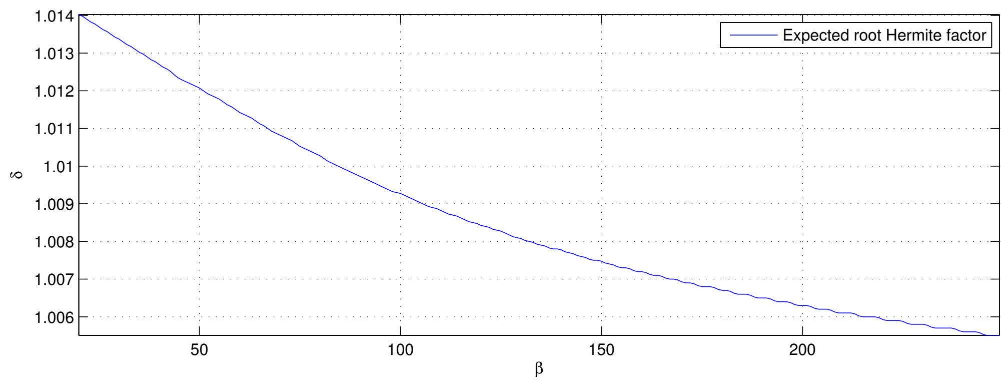


Figure 2.6: Expected root Hermite factor for BKZ-β reduced basis of random lattices

Definition 2.4.9 (Half Volume)**.** *The original definition of half volume applies for basis* $\mathbf{B}$ *with even dimension* n*,*

$$\eta(\mathbf{B}) = \frac{\prod_{i=1}^{n/2} \|\mathbf{b}_i^\star\|}{\prod_{i=n/2+1}^{n} \|\mathbf{b}_i^\star\|}. \tag{2.11}$$

We extend this definition to the case where n is odd by convention.

$$\eta(\mathbf{B}) = \frac{\prod_{i=1}^{\lfloor n/2 \rfloor} \|\mathbf{b}_i^\star\|}{\prod_{i=\lceil n/2 \rceil+1}^{n} \|\mathbf{b}_i^\star\|} \tag{2.12}$$

These two definitions measures the quality of the basis in two different aspects. The root-Hermite factor describes the distance between current $\mathbf{b}_1$ and $\lambda_1(\mathcal{L})$ or $vol(\mathcal{L})^{1/n}$, while the value of half volume is related to the cost of enumeration on the basis vectors, which is the time complexity of SVP. If we approximate $\|\mathbf{b}_i^\star\|$ with a geometric progression with ratio q, then we can deduce that

$$\eta(\mathbf{B}) \approx q^{\lfloor \frac{n}{2} \rfloor^2}, \tag{2.13}$$

and

$$\delta(\mathbf{B}) \approx \sqrt{q}. \tag{2.14}$$

The link between lattice reduction and SVP is two-fold. First of all, it usually provides an approximate solution to SVP. In case where n is small, it will provide a solution to SVP with non-negligible probability [23]. In addition, it is preferable to perform enumeration on a reduced basis to obtain even shorter vectors than current $\mathbf{b}_1$. We will explain the enumeration procedure in following chapters.

2.5 LLL Reduction

The invention of LLL reduction algorithm has lead to important breakthroughs in many fields including cryptology, integer programing, algorithmic number theory, computer algebra, and so on. Since its first publication in 1982, a lot of efforts were made to improve its performance, or study its behavior. Following the original algorithm using rational computation, many improvements were achieved by using floating point computation for approximation.

Let $\mathcal{L} \subset \mathbb{Z}^m$ be a lattice of dimension n, represented by a basis $\mathbf{B}$, such that the absolute value of its entries are bounded by integer B. The complexity of the original algorithm is $O(n^5 m \log^3 B)$, which has a total degree of 9. The first provable floating-point version is given by Schnorr [78], which reduce the complexity to a total degree 8. The original algorithm has a cubic complexity in the bit size of entries. This complexity is passed to quadratic [64], and recently it is reduced to quasi-linear [66].

Table 2.2: Complexity bounds of the original LLL and the floating-point LLL algorithms

LLL [46]	Schnorr [78]	L^2 [64]	$\tilde{L}^1$ [66]
$O(n^5 m \log^3 B)$	$O(n^3 m (n + \log B)^2 \log B)$	$O(n^4 m \log B (n + \log B))$	$O(n^{5+\epsilon} \log B + m^{\omega+1+\epsilon} \log B^{1+\epsilon})$

In practical side, several software implementation are available, such as NTL, fplll, magma, LiDIA, Maple. Among them, fplll is a dedicated LLL reduction program, which is the most efficient and is still frequently updated, while other programs are components of some larger computer algebraic libraries, so that LLL can be used together with other algorithms with convenience.

2.5.1 Algorithm description

The original LLL reduction is described in algorithm 7.

Algorithm 7 LLL reduction algorithm

Input: A basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of the lattice $\mathcal{L}$, factors (η, δ)
Output: A LLL-reduced basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$

```
1: i = 2;
2: while $i \leqslant n$ do
3:     size reduce($\mathbf{B}_i$) with $\eta$.
4:     if $\|\mathbf{b}_i^\star + \mu_{i,i-1}\mathbf{b}_{i-1}^\star\|^2 < \delta\|\mathbf{b}_{i-1}^\star\|^2$ then
5:         swap($\mathbf{b}_i, \mathbf{b}_{i-1}$);
6:         if $i > 2$ then $i \leftarrow i - 1$ end
7:     else
8:         $i \leftarrow i + 1$
9:     end if
10: end while
```

A typical choice of (η, δ) is $(0.501, 0.999)$, when the algorithm achieve almost best output quality.
The computation of GSO coefficients here uses the exact value of $\mu_{i,j}$ with algebraic representation.
During execution of the algorithm, it keeps the loop invariant that the partial basis $\mathbf{B}_i = (\mathbf{b}_1, \ldots, \mathbf{b}_i)$ is LLL-reduced. Therefore upon termination, the whole output basis is guaranteed to be LLL-reduced. However, the polynomial complexity of this algorithm is not evident from a glance. A common proof is to use a volume argument due to Lovasz.

2.5.2 Polynomial running time proof

In line 6 of the algorithm, the index i goes backwards, which is the main obstacle to show the polynomial running time of this algorithm. The sketch of the proof goes like following. We will first define an integer value D. By showing that each time line 6 is executed, D will decrease at least by a factor of $\delta < 1$, we show that this line will only be executed by at most polynomial time.

Define $d_i = \prod_{j=1}^{i} \|\mathbf{b}_i^\star\|^2$, and $D = \prod_{j=1}^{i} d_i$. Note that d_i is the volume of the lattice $\mathbf{B}_i$, therefore is an integer, thus so is D. Each time line 6 is executed, a swap takes place, and only $\mathbf{b}_{i-1}^\star$ and $\mathbf{b}_i^\star$ can be modified among all Gram-Schmidt vector norms. To avoid confusion, we denote $\mathbf{c}_i^\star$ and $\mathbf{c}_{i-1}^\star$ the renewed value after swap. We have

$$\|\mathbf{b}_{i-1}^\star\| \cdot \|\mathbf{b}_i^\star\| = \|\mathbf{c}_{i-1}^\star\| \cdot \|\mathbf{c}_i^\star\|$$

and as Lovasz' condition is not satisfied, we have $\|\mathbf{c}_{i-1}^\star\|^2 < \delta\|\mathbf{b}_{i-1}^\star\|^2$. Therefore, d_{i-1} decreases by at least a factor of δ, while all other $d_j, j \neq i-1$ is not modified. Consequently, D decreases by at least a factor of δ. It follows that the number of swaps is at most logarithmic in the initial value of D, which can be upper bounded by B^{2n}. Thus the while loop will be executed at most $O(n \log B)$ times before index i is augmented to n.

Finally we still have to bound the the size of $\mu_{i,j}$ during arithmetic operations in computation. Analysis in [46] provide a proof of bound for the size of $\mu_{i,j}$. This completes the proof and concludes to the following theorem.

Theorem 2.5.1. *For a lattice basis* $\mathbf{B}$ *with integer entries, algorithm 7 computes LLL-reduced basis in time polynomial in* B*.*

2.5.3 Floating point algorithms

The computation of GSO in the original LLL reduction uses exact arithmetic computation, which is expensive. A straight forward idea is to use floating point arithmetic to speed up. The first floating point algorithm, consists of replacing simply the value of $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$ and thus $\mu_{i,j}$. However, this idea suffers from a number of drawbacks, and require additional reparation routine to guarantee the correctness of output basis.

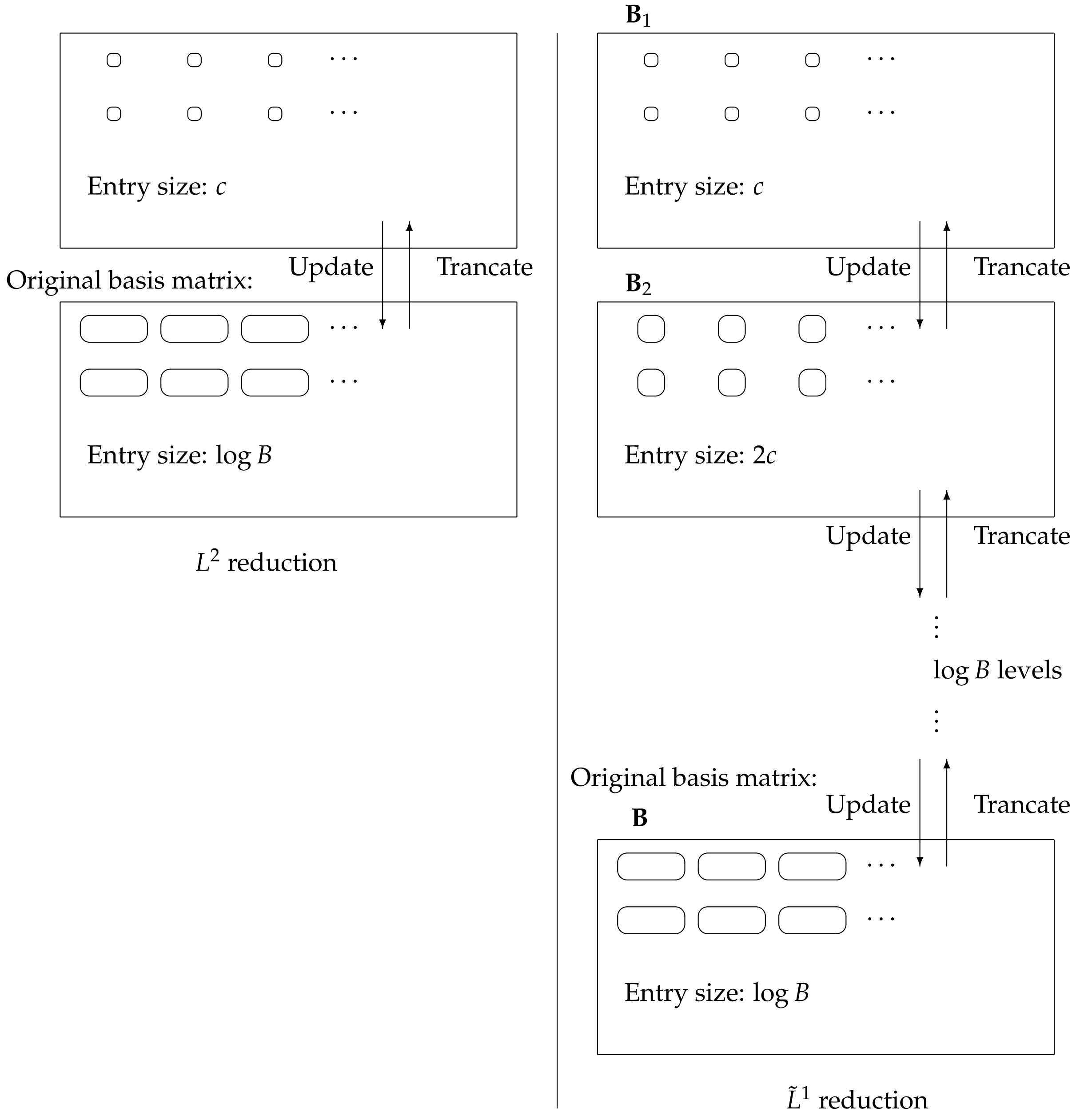
Schnorr reduction The first provable variant of LLL reduction is provided by Schorr [78], providing a complexity of a total degree of 7. The main idea consists of using a finite number of iterations of Schulz's method to maintain the accuracy of Newton's iteration for the inverse of a matrix. By taking the working precision to be $\ell = c_1 n + c_2 B$, where c_1 and c_2 are explicitly computable variables, Schnorr proved that this algorithm manage to terminate and returns a $(0.55, 0.95)$ LLL reduced basis.

L^2 **reduction** The L^2 reduction was introduced by Nguyen and Stehlé [64]. It has been proved to have quadratic complexity with regard to $\log B$. During computation, the Gram matrix of the initial basis matrix is computed both at the beginning of the algorithm and updated for each change of the basis vectors. Besides, It uses a lazy size-reduction procedure, which repeatedly does size reduction, updates the Gram matrix, until size reduction no longer alter the underlying matrix. The frequent renewal of Gram matrix and the lazy size reduction allows to set up a working precision much lower than the previous Schnorr reduction. The working precision is proved to be $c \approx 1.58d$, which is independent of $\log B$ compared to $\geq 12d + 7\log_2 B$ in Schorr's reduction. The number of times of applying transformation on our original basis matrix is $O(\log B)$, while the original basis has entry size $O(\log B)$. Consequently, the total complexity is only quadratic in $\log B$, which explains the name of the algorithm.

$\tilde{L}^1$ **reduction** This algorithm, proposed by Novocin et al. in 2010 [66], is the first reduction algorithm quasilinear in $\log B$. Like in L^2, the working precision required by the $\tilde{L}^1$ algorithm is also independent of $\log B$. But instead of applying our transformation directly to the original basis matrix as L^2 reduction does, in $\tilde{L}^1$ reduction the renewal of the underlying Gram matrix is done in a total of $O(\log \log B)$ different levels, $\mathbf{B}_1, \mathbf{B}_2, \ldots$. The entry of $\mathbf{B}_i$ is composed of first $2^{i-1} \cdot c$ bits of the original basis matrix. The transformation in level i is carried to level $i+1$, only when level i is fully reduced. This delayed transformation ensures that the renewal of the matrix in level i is only done for $O(2^{\log\log B - i})$ times. This gives the quasilinear complexity of the reduction algorithm.

Figure 2.7 is an illustration of the L^2 and the $\tilde{L}^1$ reduction algorithms.

Figure 2.7: Illustration of L^2 and $\tilde{L}^1$ reductions



2.6 Solving SVP

The two main algorithms for solving SVP and CVP and their approximate versions are the sieve and enumeration algorithm.

2.6.1 Sieve algorithm

We only present a brief introduction of the sieve algorithm, and more focus will be put on enumeration algorithm. Sieve algorithm was first discovered by Ajtai et al. [5] in 2011. This is a probabilistic algorithm which is asymptotically better than the deterministic enumeration algorithm. The algorithm starts with a basis that is already reduced.

The idea is to sample a lot of random lattice points in a ball $B_n(r) \cap \mathcal{L}$, and as the number of points increases, at some point there will be two points $\mathbf{v}_1$ and $\mathbf{v}_2$ such that $\|\mathbf{v}_1 - \mathbf{v}_2\| < r/2$. This is similar to finding collision in birthday attack, and the number of points need for collision can be analysed rigorously. At the end of sampling we expect to find many such pairs of points, and all their differences lie in $B_n(r/2) \cap \mathcal{L}$. Iteratively we reduce the value of r by a constant factor, and repeat the sampling procedure again. This is the sieving procedure which is the most expensive in the algorithm. At the end we will be able to find a short enough vector.

2.6.2 Enumeration algorithm

The enumeration algorithm is deterministic in its original appearance in the 1980s. It is the simplest method for solving SVP, by searching directly for a proper integer combination of the basis vector. It consists of exhaustively searching for the projections of any shortest vector in the projected lattices $\pi_i(\mathcal{L})$, from $i = n$ to 1. In search for a shortest vector $\mathbf{v}$ in a n dimensional lattice $\mathcal{L}$, where the length of $\mathbf{v}$ is known to be bounded by r, the algorithm tries to search for all vectors in $\pi_i(\mathcal{L})$ with length $< r$, and then lifts them to $\pi_{i-1}(\mathcal{L})$, and keeps those vectors in $\pi_{i-1}(\mathcal{L})$ with smaller length than r, and so on. Until $i = 1$, the shortest vector is picked out. The estimation can be obtained by Gaussian heuristic, or with a much conservative estimate using a known vector, for example $r = \|\mathbf{b}_1\|$, or $\sqrt{\gamma_n} vol(\mathcal{L})^{1/n}$.

The actual algorithm travels through the enumeration tree by depth first search, where the each node can be derived from its parents and its preceding siblings. Therefore the memory requirement besides GSO is linear with the enumeration tree depth, which is the dimension n. However, the time complexity of this algorithm is exponential. In fact, the total number of nodes visited in layer i is the number of vectors in $\pi_i(\mathcal{L})$ with length bounded by r, which can be evaluated by Gaussian heuristic.

The number of nodes in level i, i.e, the number of vectors in lattice $\pi_i(\mathcal{L})$ with norms smaller than r is approximately

$$N_i = \frac{r^{n-i+1} v_{n-i+1}}{vol(\pi_i(\mathcal{L}))}. \tag{2.15}$$

Because $\sqrt{\gamma_n} = \Theta(n)$, r is usually chosen to be $vol(\mathcal{L})^{1/n}\Theta(\sqrt{n})$, unless we are given other additional information. Now, consider the layer $i = \lfloor n/2 \rfloor$,

$$\begin{aligned} N_{\lfloor n/2 \rfloor} &= \frac{r^{n-\lfloor n/2 \rfloor+1} v_{n-\lfloor n/2 \rfloor+1}}{\sqrt{vol(\mathcal{L})/\eta(\mathbf{B})}} \\ &\approx \Theta(\eta(\mathbf{B})^{1/2} \cdot (e\pi)^{n/2}) \end{aligned}$$

According to the definition of $\eta(\mathbf{B})$ in equation 2.13, $N_{\lfloor n/2 \rfloor}$ is super-exponential in n. Therefore so is the complexity of enumeration algorithm.

Although sieving has a smaller asymptotic complexity in theory, in practice, enumeration algorithm has a small constant which makes it more competitive than the former algorithm.

2.7 Computational challenges proposed by Darmstadt University

A series of lattice problems are proposed by Darmstadt University to challenge solve lattice problems of different dimensions on a website [48]. This serves as an interface to test and compare efficiency of problem solving by different methods. Besides, it provides a reference value for the present limit dimensions of problems that are solvable in practice.

So far, three categories of problems are proposed. For each category it provides problems in different dimensions, where the problems can be either downloaded or generated by given source code. It provides a list of problem solvers ranged in the order of the dimension of the problem and the quality of the solution.

The challenge includes following three categories:

Finding short vectors in Ajtai lattice Following Ajtai's work [3], a special class of lattice basis is considered to be the hardest among all lattices. The solutions of all other lattices are implied in some of these hardest lattices. The challenge consists of finding short vectors in some of these Ajtai lattices of high dimension. The construction of these lattice bases and the proof of existence of short vectors in each of the corresponding lattices is presented in [13].

Challenge lattices are provided in different dimensions, where initially $\|\mathbf{b}_1\| = q$ with q being some known constant. One need to either find a shorter vector than $\mathbf{b}_1$ in a higher dimension of a challenge that is not solved before, or find a even shorter vector than the previous solution.

SVP **for random lattices** This challenge requires to find solutions to HSVP_γ for random integer lattices, where $\gamma = 1.05 \cdot \frac{\Gamma(n/2+1)^{1/n}}{\sqrt{\pi}}$. For each dimension, it is possible to generate different sample lattices with different integer seeds, and a solution to any of them is taken into account. The challengers are required to either find a solution to HSVP_γ for a dimension unsolved before, or find a solution with a better approximation factor for HSVP compared to the previous solution (it can be a solution of a different lattice of the same dimension).

SVP **for ideal lattices** While the previous challenge focus on SVP hardness on random integer lattices, this challenge tries to the test hardness of ideal lattices. The class of ideal lattices is a special class of lattice with some additional algebraic structures. It is used to build efficient primitives and homomorphic encryption scheme, which require less key length and faster operations thanks to its algebraic structure.

Ideal lattices are often defined in terms of polynomial rings. For a polynomial $v(x) = v_n x^n + \cdots + v_0$, its vector representation is $(v_n, \ldots, v_0)$. The definition of a ideal lattice is the following: Let $f(x)$ be monic polynomial, i.e. the coefficient of the largest exponent is 1. Then the set of all vector representations of the polynomials in the ring ring $\mathbb{Z}[X]/f(x)$ form a ideal lattice. Especially, for $f(x) = x^n - 1$, this lattice is cyclic. The most often used $f(x)$ is $f(x) = x^n + 1$ and $f(x) = x^n + x^{n-1} + \cdots + 1$.

It is unknown if SVP could be much easier on ideal lattice by using its ideal structure, which motivates to set up a separate challenge for SVP for ideal lattices [67]. The challenge is further separated into two different HSVP_γ challenges, with $\gamma_1 = 1.05 \cdot \frac{\Gamma(n/2+1)^{1/n}}{\sqrt{\pi}}$, and $\gamma_2 = n$.

2.8 Notations

To conclude this chapter, we list all the main notations in the following table for convenience of reference.

Notation	Interpretation
$\mathcal{L}$	A lattice in $\mathbb{R}^m$
n	Dimension of a lattice
$\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$	Basis of a lattice
$\mathbf{B}_i$	The sublattice generated by the first i vectors in $\mathbf{B}$
$\pi_i(\mathbf{B})$	The projected lattice of $\mathbf{B}$ on the orthogonal supplement of the linear span of $\mathbf{B}_i$
$\mathcal{L}(\mathbf{B}_{[j,k]})$	$\mathcal{L}(\pi_j(\mathbf{B}_k))$
$\mathrm{vol}(\cdot)$	Volume (of a lattice $\mathcal{L}$).
$\mathcal{B}_n$	A ball of dimension n with unit radius
v_n	The volume of $\mathcal{B}_n$
$\lVert\mathbf{b}_i^*\rVert$	Gram Schmidt norms of a basis
$\mu_{i,j}$	$\frac{\langle \mathbf{b}_i, \mathbf{b}_j^\star\rangle}{\lVert\mathbf{b}_j^\star\rVert^2}$
$\lambda_i(\mathcal{L})$	Norm of the i-th Minkowski's successive minima in lattice $\mathcal{L}$
$GH(\mathcal{L})$	Prediction of $\lambda_1(\mathcal{L})$ by Gaussian heuristic. $GH(\mathcal{L}) = (vol(\mathcal{L})/v_n)^{1/n}$
$\delta(\mathcal{L})$	Root Hermite factor of a basis. $\delta(\mathbf{B}) = \left(\lVert\mathbf{b}_1\rVert / vol(\mathcal{L})^{1/n}\right)^{1/n}$
$\eta(\mathcal{L})$	Half volume of a basis. $\eta(\mathbf{B}) = \frac{\prod_{i=1}^{\lfloor n/2\rfloor} \lVert\mathbf{b}_i^\star\rVert}{\prod_{i=\lceil n/2\rceil+1}^{n} \lVert\mathbf{b}_i^\star\rVert}$

Table 2.3: Principal notations used in this work

CHAPTER 3

Enumeration for SVP

Contents

Enumeration is the one of the most basic algorithms for solving many lattice problems including exact and approximate version of SVP and CVP. Besides being a stand-alone algorithm, it is also an essential sub-procedure for BKZ reduction. As a stand-alone, its performance is dependent on how much the basis is reduced. And as a sub-procedure for lattice reduction, its complexity influences the behavior of the reduction algorithm. Speeding up enumeration is the central topic of this chapter.

In this chapter we will focus on enumeration procedure for SVP, for the need of discussion for BKZ reduction. Though the enumeration procedure for CVP is very similar. We will begin by discussing the properties of short vectors in random lattices, then proceed by talking about the enumeration procedure and its analysis. The exhaustive search in full enumeration procedure can be greatly improved by abandoning branches which are less likely to lead to solutions. This is called *pruned enumeration*.

The pruned enumeration turns the sequential deterministic algorithm into a probabilistic algorithm which can be implemented with parallel computation. The pruned enumeration uses a pruning function to decide whether a branch should be rejected. This pruning function is precomputed prior to enumeration, and is a deciding factor of running time. With a given basis, it is possible to compute a quasi-optimal pruning function, which minimizes the expected time for computing a shortest vector.

In high dimension, searching for the solution of the exact SVP by enumeration is infeasible in practice. But we can still hope to solve the approximate SVP_γ problems over random lattices. In fact, for a random lattice of dimension n, there are approximately γ^n vectors within the radius of $\gamma \cdot \lambda_1(\mathcal{L})$. Taking the distribution of short vectors into consideration, we are able to find proper pruning functions which allow enumeration of basis higher than 200.

A pruning function of dimension n is a vector of dimension n. The pruning function is a solution of an optimization problem, which is specific for each basis. Initially, we should always precompute a pruning function whenever we are given a new basis, and this is done by searching the solution in a $n/2$ dimensional space. Regardless of the searching method and initialization of the search, the optimization problem always appears to converge to the global minima, which suggests that we can treat the problem as a convex optimization problem, and use faster algorithms than random search. In addition, the pruning function display very similar shape in general, which shows that they lie in a subspace whose dimension is much smaller than $n/2$. Extracting this subspace from abundant data in high dimension is a well studied subject in data mining, and is usually addressed by using principal component analysis(PCA) method. Our PCA shows that these data are distributed almost in a subspace of dimension 4, which means that it is possible to express it with a linear combination of 4 terms, in other words, it is enough to decide a bounding function with 4 parameters. Eventually, we present a way to effectively parameterize the bounding function so that it can be generated very simply and quickly.

Since the complexity is also affected by the reductions applied prior to enumeration. We will also dedicate a section for the discussion of the basis quality.

3.1 Distribution of short vectors

In this section, we investigate the distribution of the shortest vector in a random lattice, and in general, the joint distribution of the first N short vectors in a random lattices. The notion of random lattices is given in section 2.3, and is based on the definition of uniform distribution over all lattices of the unit volume.

According to the results of Södergren [86], the norms of N first shortest vectors define a stochastic process, which converges to a Poisson process when the dimension n tends to infinity. In particular, $\lambda_1(\mathcal{L})$ of a random lattice defines a variable which converges to an exponential distribution. Interestingly, the expectation of number of short vectors according to this distribution coincide with the prediction by Gaussian heuristic. Experiments show that this distribution converges very fast, and the predictions corresponds to the experiments as soon as $n > 10$.

3.1.1 Poisson process

Definition 3.1.1 (Exponential distribution)**.** *The exponential distribution with parameter λ is characterized by following probability density function (pdf):*

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geqslant 0 \\ 0, & x < 0. \end{cases}$$

The exponential distribution describes the intervals of events in a Poisson process, its has expectation $E(X) = 1/\lambda$, and variance $Var(X) = 1/\lambda^2$.

A Poisson process can be described with 3 collection of variables:

- The sequence of *interval times* $\mathbf{X} = (X_1, X_2, \dots)$, where X_i is the time interval between $i-1$-th and i-th event.
- The sequence of *arrival times* $\mathbf{T} = (T_1, T_2, \dots)$, where T_i is the time point at which i-th event takes place.
- The *counting process* $\mathbf{N} = (N_t, t \geqslant 0)$, where N_t denote the number of arrivals in $(0, t]$ for any non-negative t.

Clearly, T_i is the partial sum of the first i elements in X,

$$T_i = \sum_{i=1}^{i} X_i$$
$$X_i = T_i - T_{i-1}$$

$\mathbf{T}$ and $\mathbf{N}$ are inverses of one another:

$$T_n = \min\{t \geqslant 0 : N_t = n\}, n \in N$$
$$N_t = \max\{n \in N : T_n \leqslant t\}, t \in [0, \infty)$$

Definition 3.1.2 (Poisson process). *A Poisson process of intensity λ is a process whose interval times are i.i.d variables of exponential distribution with parameter λ.*

In Poisson process, events occur continuously and independently at a constant average rate. Poisson process has an important property of being memoryless, i.e., the number of arrivals occurring in any bounded interval of time after time t is independent of the number of arrivals occurring before time t.

3.1.2 Short vectors in unit volume random lattices

Let $\mathcal{X}_n$ be the space of the all unit volume lattices of dimension n. For a random lattice $\mathcal{L} \in \mathcal{X}_n$, we study the norms of non-zero vector pairs $\pm\mathbf{v}$.

Given a lattice $\mathcal{L} \in \mathcal{X}_n$, we order its non-zero vector norms as $0 < l_1 \leqslant l_2 \leqslant l_3 \leqslant \dots$ where we count the common length of the vectors $\mathbf{v}$ and $-\mathbf{v}$ only once. For $j \geqslant 1$, we define

$$u_i = v_n \cdot l_j^n, \tag{3.1}$$

where v_n is the volume of a n dimensional unitary ball , so u_i is the volume of the n dimensional ball of radius l_i.

Theorem 3.1.3 (Distribution of short vectors). *[86] For any fixed N, the N-dimensional random variable $(u_1, \dots, u_N)$ converges in distribution to the distribution of the first N points of a Poisson process on the positive real line with intensity $1/2$ as $n \to \infty$.*

Corollary 3.1.4. *For a lattice $\mathcal{L} \in \mathcal{X}_n$, the distribution of $v_n \cdot \lambda_1(\mathcal{L})^n$ converges in distribution to the exponential distribution with parameter $1/2$ as $n \to \infty$.*

The cumulative distribution function (cdf) of variable X with exponential distribution is given by:

$$Pr(X < x) = \begin{cases} 1 - e^{-x/2} & x \geqslant 0 \\ 0 & x < 0 \end{cases} \tag{3.2}$$

Then $\lambda_1(\mathcal{L})$ defines the variable Y to be $Y = (X/v_n)^{1/n}$. With a substitution of variable $x = v_n \cdot y^n$ we have:

$$Pr(Y < y) = \begin{cases} 1 - e^{-y^n \cdot v_n/2} & y \geqslant 0 \\ 0 & y < 0 \end{cases} \tag{3.3}$$

Then the pdf of variable Y is:

$$f(y) = \begin{cases} \frac{1}{2} n v_n y^{n-1} \cdot e^{-y^n \cdot v_n/2} & y \geqslant 0 \\ 0 & y < 0 \end{cases} \tag{3.4}$$

And the limit of expectation $E(Y)$ is:

$$\begin{aligned} E(Y) &= \left(\frac{2}{v_n}\right)^{1/n} \cdot \frac{\Gamma(1/n)}{n} \\ &= \left(\frac{2}{v_n}\right)^{1/n} \cdot \Gamma(1 + 1/n) \end{aligned}$$

Here, $\Gamma(x)$ function can be developed into Taylor series at $x = 1$:

$$\Gamma(1 + \epsilon) = 1 + \epsilon \cdot \gamma + o(\epsilon), \tag{3.5}$$

where $\gamma \approx 0.577$ is the Euler-Mascheroni constant.

Upon this we can summarize that $E(\lambda_1(\mathcal{L}))$ tends to

$$(1 - \gamma/n) \cdot \left(\frac{2}{v_n}\right)^{1/n} \tag{3.6}$$

as n tends to $+\infty$.

And by calculating the expectation of N_i, the number of arrivals in the Poisson process, we obtain the following corollary, which coincide with the prediction given by Gaussian heuristic.

Corollary 3.1.5. *In a random lattice $\mathcal{L}$ of dimension n, for $r > 1$, the total number of lattice vectors (including the symmetric vectors) inside the ball of radius r is expected to be*

$$v_n \cdot r^n, \tag{3.7}$$

and the ith shortest vector is expected to have the radius

$$\left(\frac{i}{v_n}\right)^{1/n} \tag{3.8}$$

for $n \to \infty$

3.1.3 Comparing with random instances in experiment

In practice, the lattices we deal with are mostly integer lattices. We presented in section 2.3 a simple algorithm to randomly sample from integer lattices of prime volume p. Once normalized with a factor of $1/p^{1/n}$, the uniform distribution over integer lattices converges towards the uniform distribution of random lattices of unit volume.

Concluding from previous theorem, we have following properties concerning the N short vectors in random lattices of volume V. As n tends to infinity, we have:

- The variable defined by $\lambda_1(\mathcal{L})^n \cdot v_n / vol(\mathcal{L})$ converges to the exponential distribution with parameter $1/2$.

- The expected value of $\lambda_1(\mathcal{L})$ converges to $(1 - \gamma/n) \cdot (2 \cdot vol(\mathcal{L})/v_n)^{1/n}$.

- The number of vectors with norms shorter than $r \cdot (vol(\mathcal{L})/v_n)^{1/n}$ is r^n, taking into count both $\pm\mathbf{v}$.

Comparing with the prediction given by Gaussian heuristic, which is $(vol(\mathcal{L})/v_n)^{1/n}$, this expectation of λ_1 is not exactly the same. There is a difference of a factor of $(1 - \gamma/n)2^{1/n}$, which is a slight difference when n is big. However, the expected number of points within given radius is the same as predictions of Gaussian heuristic.

We also compared this distribution to the experiment data to observe the convergence speed (Fig. 3.1 Fig. 3.2 and Fig. 3.3). We generate 1000 random integer lattices of dimension n, and compute their $\lambda_1(\mathcal{L}) \cdot (v_n/vol(\mathcal{L}))^{1/n}$ value respectively. Their cumulative histogram (solid line) is then compared to the cumulative distribution function of $x^{1/n}$, where x is a random variable of exponential distribution of parameter $1/2$ (dashed line). It is clear that for $n > 10$ these two curves are already very close to each other, which shows that the distribution of $\lambda_1(\mathcal{L})$ converges very quickly.

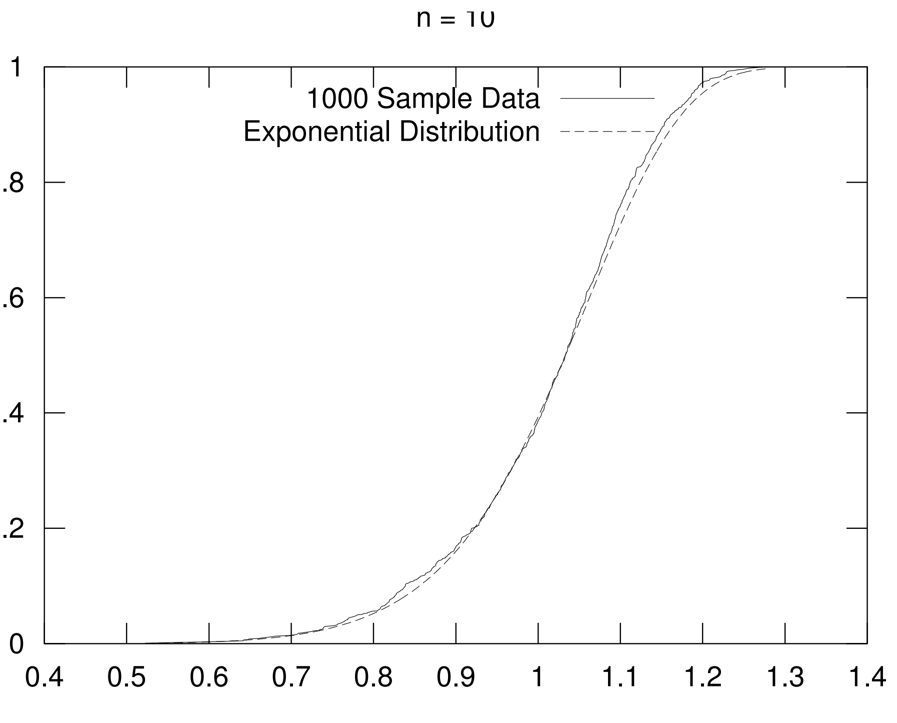


Figure 3.1: Lattice dimension $n = 10$

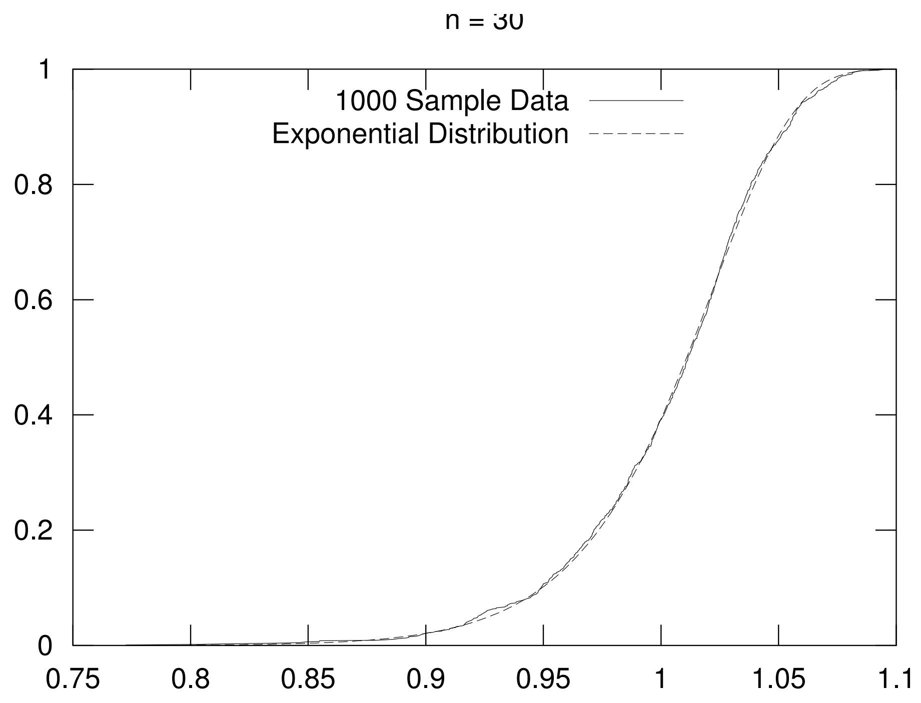


Figure 3.2: Lattice dimension $n = 30$

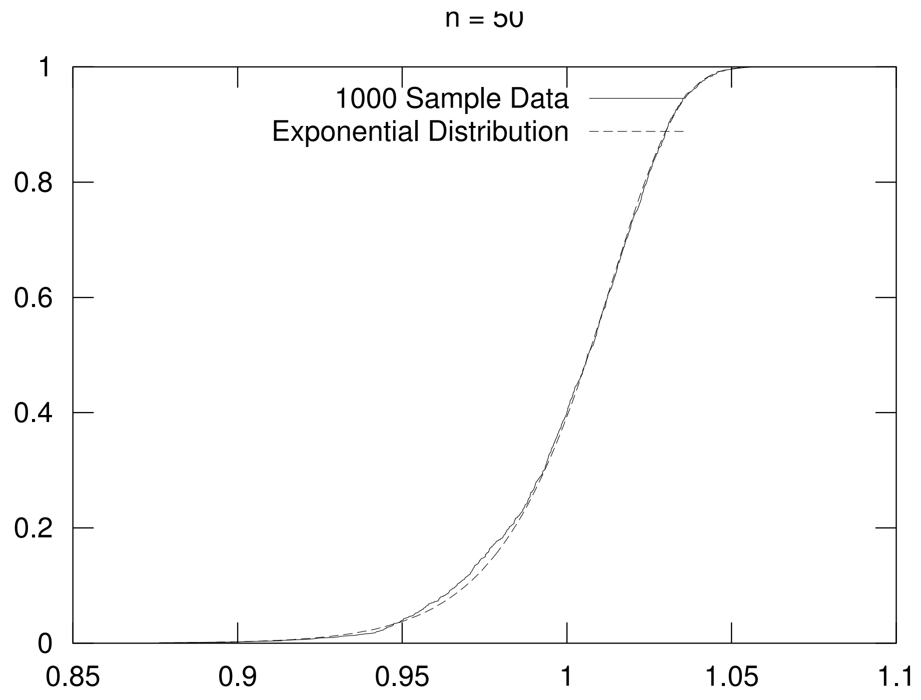


Figure 3.3: Lattice dimension $n = 50$

3.2 Enumeration procedure and analysis

In this section, we present the classical enumeration algorithm, and its complexity analysis in worst case as well as average case. For clarity we summarize the definitions of notations that we are going to use in the table 3.1.

Table 3.1: Extract of important notations for current chapter

Notation	Interpretation
N_i	The number of nodes in each layer
N	$\sum_{i=1}^{n} N_i$
$GH(\mathcal{L})$	Prediction of $\lambda_1(\mathcal{L})$ by Gaussian heuristic. $GH(\mathcal{L}) = (vol(\mathcal{L})/v_n)^{1/n}$
R	The enumeration bound
r	$R/GH(\mathcal{L})$
q	$\approx \|\mathbf{b}_i^\star\|/\|\mathbf{b}_{i+1}^\star\|$
For pruned enumeration	
$p_{\mathrm{s}}(R, \mathbf{R})$	Probability of finding a vector of known norm R
$p_{\mathrm{succ}}(R, \mathbf{R})$	Probability of finding any vector whose norm is smaller than R
$\mathbf{R}$	Bounding function $\mathbf{R} = (R_1^2, R_2^2, \dots)$
$\mathbf{r}$	normalized bounding function $\mathbf{r} = (\frac{R_1^2}{R_n^2}, \dots,)$

3.2.1 Algorithm description

Given a basis $\mathbf{B}$ of dimension n, The enumeration procedure tries to find an integer combinations of the basis $x_1\mathbf{b}_1 + \cdots + x_n\mathbf{b}_n$, by going through the enumeration tree. The algorithm terminates when it has scanned through all possibilities, and output the shortest vector that it finds. Alternatively, if we are aimed to find a short vector within a certain bound R, or we have an upper bound on $\lambda_1(\mathcal{L})$, then we can as well terminate as soon as we find the first vector satisfying this condition. Since we already have an estimation of the shortest vector in a random lattice, setting up an upper bound with 99% confidence is not difficult. In this way the asymptotic complexity does not change, but usually we will be able to save time for some constant factor, the price is that we might fail with certain probability, or we will end up with a second-to-shortest vector, unless we really know the exact value of $\lambda_1(\mathcal{L})$ a priori, which can happen in some crypto-systems like NTRU.

We adopt the algorithm description given by [24], where the program checks the enumeration tree with depth first search, and always output the first short vector with norm shorter than R. If ever one wants to search through all short vectors, it suffices to set $R = \|b_1\|$ and modify the termination condition at line 11.

The description in pseudo-code is given in algorithm 8.

Algorithm 8 Enumeration

Input: A basis **B** and a bound R of short vector.

Output: A vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v}\| < R$, or `failure` if no vector in $\mathcal{L}(\mathbf{B})$ is shorter than R.

```
1: $[\mathbf{M}, \mathbf{D}, \mathbf{O}] \leftarrow$ GramSchmidtOrthogonolization($\mathbf{B}$)
2: $(x_1, \ldots, x_n) \leftarrow (1, 0, \ldots, 0)$ //current combination
3: $(\rho_1, \ldots, \rho_n, \rho_{n+1}) \leftarrow (0, 0, \ldots, 0)$ //sentinel element $\rho_{n+1} = 0$ by convention
4: $(c_1, \ldots, c_n) \leftarrow (0, \ldots, 0)$ //$c_i = \sum_{j=i}^{n} x_j \mu_{j,i-1}$
5: $(w_1, \ldots, w_n) \leftarrow (0, \ldots, 0)$ //jumps from previous $c_i$
6: last_nonzero $\leftarrow 1$ //last $i$ such that $x_i \neq 0$
7: while true do
8:   $\rho_i \leftarrow \rho_{i+1} + (x_i + c_i)^2 \cdot \|\mathbf{b}_i^\star\|^2$
9:   if $\rho_i < R$ then
10:     if $i = 1$ then
11:       return $\sum_{j=1}^{n} x_j \mathbf{b}_j$;
12:     else
13:       $i \leftarrow i - 1$;
14:       $c_i \leftarrow \sum_{j=i}^{n} x_j \mu_{j,i-1}$
15:       $x_i \leftarrow \lceil -c_i \rfloor; w_i \leftarrow 1$
16:     end if
17:   else
18:     $k \leftarrow k + 1$
19:     if $k = n + 1$ then
20:       return failure;
21:     end if
22:     if $k \geqslant$ last_nonzero then
23:       last_nonzero $\leftarrow k$
24:       $x_k \leftarrow x_k + 1$ // only enumerate positive half
25:     else
26:       if $x_i > c_i$ then $x_i \leftarrow x_i w_i$ else $x_i \leftarrow x_i + x_i$
27:       $w_i \leftarrow w_i + 1$
28:     end if
29:   end if
30: end while
```

The enumeration tree consists of n levels. At level i , for $1 \leqslant i \leqslant n$, we are working in the basis $\pi_i(\mathbf{B})$ and we will look for nodes $(x_i, \ldots, x_n)$ such that the vector $\mathbf{v}_i \pi_i \left(\sum_{j=i}^{n} x_j \mathbf{b}_j\right)$ in the basis $\pi_i(\mathbf{B})$ has shorter norm than R. We use ρ_i to mean $\|\mathbf{v}_i\|^2$, and it satisfies

$$\rho_i = \|x_n \pi_i(\mathbf{b}_n) + \cdots + x_i \pi_i(\mathbf{b}_i)\|^2 < R^2. \tag{3.9}$$

We can decompose **B** and calculate the GSO of the basis $\mathbf{MDO} = \mathbf{B}$, Then Equation. 3.9 can be rewritten as

$$\rho_i = x_n^2 \|\mathbf{b}_n^\star\|^2 + (x_{n-1} + x_n \mu_{n,n-1})^2 \|\mathbf{b}_{n-1}^\star\|^2 + \cdots + \left(x_i + \sum_{j=i+1}^{n} x_j \mu_{j,i}\right)^2 \|\mathbf{b}_i^\star\|^2 < R^2 \tag{3.10}$$

when the node in level i being visited satisfies equation 3.10, we go down to level $i - 1$ to visit its children, by lifting the vector to $\pi_{i-1}(\mathbf{B})$. For each child, ρ_{i-1} is calculated from ρ_i by relation:

$$\rho_{i-1} = \rho_i + \left(x_{i-1} + \sum_{j=i}^{n} x_j \mu_{j,i-1} \right)^2 \|\mathbf{b}_{i-1}^\star\|^2 < R^2 \tag{3.11}$$

Clearly, $\rho_n \leqslant \rho_{n-1} \leqslant \cdots \leqslant \rho_1$ is an increasing sequence. When a node has several children, the visit is done in the increasing order of ρ_{i-1}. The increment from ρ_i to ρ_{i-1} is $\left(x_{i-1} + \sum_{j=i}^{n} x_j \mu_{j,i-1} \right)^2 \|\mathbf{b}_{i-1}^\star\|^2$. Its minimum value is reached when we take x_{i-1} to be $\lceil -\sum_{j=i}^{n} x_j \mu_{j,i-1} \rfloor$. We note $c_{i-1} = \sum_{j=i}^{n} x_j \mu_{j,i-1}$, and by symmetry we suppose $c_{i-1} > 0$, then our searching order is

$$x_{i-1}^{(1)} = \lceil -c_{i-1} \rfloor, \quad x_{i-1}^{(2)} = \lceil -c_{i-1} \rfloor + 1, \quad x_{i-1}^{(3)} = \lceil -c_{i-1} \rfloor - 1, \quad x_{i-1}^{(4)} = \lceil -c_{i-1} \rfloor + 2, \quad \ldots \text{ until } \rho_{i-1} \geqslant R^2. \tag{3.12}$$

This order ensures the monotonous increasing of ρ_{i-1} among all children of the same parent. Besides, $x_{i-1}^{(j+1)}$ can be computed directly from $x_{i-1}^{(j)}$ by

$$x_{i-1}^{(j+1)} = \begin{cases} x_{i-1}^{(j)} - j & \text{if } x_{i-1}^{(j)} > 0 \\ x_{i-1}^{(j)} + j & \text{if } x_{i-1}^{(j)} < 0 \end{cases} \tag{3.13}$$

Thus the value of ρ_{i-1} is completely computed from its father if it is the first child, or otherwise from its father and its proceeding sibling. In this way the memory we need is linear with the tree depth, which is $\Theta(n)$. There is one exception to the traversal of x_i: in order to avoid checking both $\mathbf{v}$ and $-\mathbf{v}$, we only check the nodes whose last non-zero component is positive. For this purpose, the index of the last non-zero component is flagged.

Initially, we start by visiting the node $(x_1, \ldots, x_n) = (1, 0, \ldots, 0)$. If we manage to visit a valid node in level $i = 1$, it means that we have found a vector whose norm is smaller than R, which implies the termination of the algorithm.

3.2.2 Complexity analysis

The running time T of this algorithm is proportional to the number of nodes in the enumeration tree N, and the time to process each node is almost a constant t_{node}. N is the sum of N_i, the number of nodes in each level of the enumeration tree.

$$T = N \cdot t_{node} = \sum_{i=1}^{n} N_i \cdot t_{node} \tag{3.14}$$

The number of nodes on level i is half of the number of vector point in the lattice $\pi_i(\mathbf{B})$ with smaller norm than R. Therefore this can be estimated by Gaussian heuristic [24]. The volume of the projected lattice is

$$vol(\pi_i(\mathbf{B})) = \prod_{j=i}^{n} \|\mathbf{b}_j^\star\| \tag{3.15}$$

We can use Gaussian heuristic to predict N_i, the number of nodes at level i,

$$N_i \approx \frac{1}{2} \cdot \frac{v_{n-i+1} \cdot R^{n-i+1}}{\prod_{j=i}^{n} \|\mathbf{b}_j^\star\|} \tag{3.16}$$

And the total number of nodes for the entire enumeration tree is $N = \sum_{i=1}^{n} N_i$.

For random lattices, after a typical reduction algorithm (LLL or BKZ for example), $\|\mathbf{b}_i^\star\|/\|\mathbf{b}_{i+1}^\star\| \approx q$ for some constant q, where the value of q depends on the algorithm. This approximation allows to simplify the previous estimation:

$$N_i = \frac{R^{n-i+1} v_{n-i+1} \|\mathbf{b}_1\|^{i-1}}{q^{(i-2)(i-1)/2} vol(\mathcal{L})} \tag{3.17}$$

Depending on the choice of R, this may yield different complexity:

- If one takes $R = \|\mathbf{b}_1\|$, then (3.17) becomes

$$\begin{aligned} N_i &\approx \frac{q^{n(n-1)/2} \cdot v_{n-i+1}}{q^{(i-2)(i-1)/2}} \\ &\approx q^{(n(n-1)-(i-2)(i-1))/2} \cdot v_{n-i+1} \end{aligned}$$

- If one takes $R = \sqrt{\gamma_n} \cdot vol(\mathcal{L})^{1/n}$, then $\sqrt{\gamma_n} = \Theta(\sqrt{n})$, thus (3.17) becomes

$$\begin{aligned} N_i &\approx \frac{q^{(i-1)(n-1)/2} \cdot 2^{O(d)}}{q^{(i-2)(i-1)/2}} \\ &\approx q^{(i-2)(n-i+1)/2} \cdot 2^{O(d)} \end{aligned}$$

- If one takes $R = GH \approx \sqrt{\frac{n}{2\pi e}} \cdot vol(\mathcal{L})^{1/n}$, then (3.17) becomes

$$\begin{aligned} N_i &\approx \frac{vol(\mathcal{L})^{n-i+1}}{\prod_{j=i}^n \|\mathbf{b}_j^\star\|} \cdot \left(\frac{n}{n-i+1}\right)^{(n-i+1)/2} \\ &\approx \frac{\prod_{j=1}^i \|\mathbf{b}_j^\star\|}{vol(\mathcal{L})^{i/n}} \cdot \left(\frac{n}{n-i+1}\right)^{(n-i+1)/2} \\ &\approx q^{(i-1)(n-i+2)/2} \cdot \left(\frac{n}{n-i+1}\right)^{(n-i+1)/2} \end{aligned}$$

In all these case, the maximal value is achieved for $i \approx n/2$, then (3.17) becomes

$$N_i \approx q^{n^2/8} 2^{O(n)}, \tag{3.18}$$

and the complexity of the algorithm is super-exponential in n.

3.3 Pruned enumeration for SVP

Pruned enumeration was first introduced by Schnorr and Euchner as a subroutine in their reduction algorithm [79]. However, the first correct analysis and efficient pruning enumeration is attributed to Gama et al. [24]. They provide an efficient way to compute the success probability and complexity of pruned enumeration, which leads to great speed-ups.

A closer look at the equation. 3.9 shows that not each nodes in the enumeration tree actually have the different probability of containing a solution in its branches. In enumeration, $\rho_n \leqslant \cdots \leqslant \rho_1$ is a monotonously increasing sequence, and if ρ_i is already very close to R when i is still big, then probably its branches have no child nodes within R bound, and this sub-branch ends up without finding any solution.

Consider a typical vector $\mathbf{v}$ of dimension n with norm R, define $\rho_i = \sum_{i=1}^{i} v_i^2$, then experiments confirmed that most probably the sequence ρ_i increases slowly at an almost constant speed until reaching $\rho_n = R^2$. On the other hand, if ρ_i sequence of a vector $\mathbf{v}$ increases much faster than $i/n \cdot R^2$, then the probability that $\|\mathbf{v}\| \leqslant R$ is small.

Following is a figure depicting the typical ρ_i sequence for a vector randomly sampled from the surface of hypersphere of radius R, dimension $n = 40$. We generate 1000 samples, draw their mean ρ_i value and give standard deviation as the error bar. (Fig. 3.4)

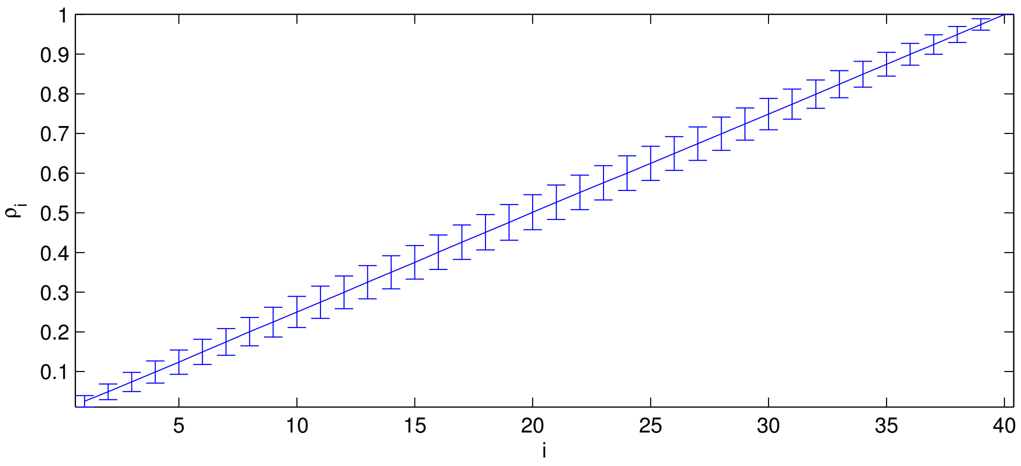


Figure 3.4: The mean value and standard deviation of ρ_i for points randomly distributed over the unit sphere of dimension 40.

The idea of pruning consists of discarding branches that are unlikely to produce a solution. This is to say, we predefine a sequence $\mathbf{R} = (R_1^2, \ldots, R_n^2)$ satisfying $R_1^2 \leqslant R_2^2 \leqslant \cdots \leqslant R_n^2 = R$ as our bounding function. At level i of the enumeration tree, we narrow our boundary by only checking the nodes within bound R_{n-i+1}. In this way, we are actually enumerating nodes $\pi_i(\sum_{j=i}^{n} x_j \mathbf{b}_j) \in \mathbb{R}^{n-i+1}$, which is inside following set:

$$C_{R_1^2,\ldots,R_{n-i+1}^2} = \left\{ (z_1, \ldots, z_{n-i+1}) \in \mathbf{B}^{n-i+1}, \quad \forall j \leqslant n-i+1, \ \sum_{l=1}^{j} z_l^2 < R_j^2 \right\}. \tag{3.19}$$

To introduce pruning into the algorithm, it suffices to change the condition (3.9) into

$$\rho_i = \|x_n \pi_i(\mathbf{b}_n) + \cdots + x_i \pi_i(\mathbf{b}_i)\|^2 < R_{n-i+1}^2. \tag{3.20}$$

and change line 9 of algorithm 8.

3.3.1 Running time analysis

We use Gaussian heuristic to predict the number of nodes in each level. We need to compute the volume of the convex set $C_{R_1^2,\ldots,R_i^2}$ for $i = 1, \ldots, n$, and it follows that

$$N_i \approx \frac{1}{2} \cdot \frac{vol(C_{R_1^2,\ldots,R_{n-i+1}^2})}{\prod_{j=i}^{n} \|\mathbf{b}_j^\star\|} \tag{3.21}$$

For doing so, a naive way would be Monte Carlo method. That is to say, we uniformly sample a lot of points from a hyperball of radius R in $\mathbb{R}^i$, and the probability that it falls into $C_{R_1^2,\ldots,R_i^2}$ is the proportion between the volume of the polytope $C_{R_1^2,\ldots,R_i^2}$ and $v_{n-i} \cdot R^i$, the volume of the hyperball.

A beautiful analysis by Gama et al. in [24] also presents a way to compute the exact value of volume of $c_{R_1^2,\ldots,R_i^2}$ under the constraint of $R_1^2 = R_2^2 \leqslant R_3^2 = R_4^2 \leqslant \cdots \leqslant R_{n-1}^2 = R_n^2$. We cite their analysis and proof in the following.

The distribution of the vector $(u_1^2 + u_2^2, u_3^2 + u_4^2, \ldots, u_{d-1}^2 + u_d^2)$ when $\mathbf{u}$ is uniformly chosen from $\mathrm{Ball}_n = \{\sum_{i=1}^n u_i^2 \leqslant 1\}$ is given by a Dirichlet distribution with parameters $(1,\ldots,1)$, which is simply a uniform distribution over the set of all vectors whose coordinates are non-negative and sum to at most 1 (see Page 593 of [20]). More precisely, there is:

Lemma 3.3.1. *Let $n = 2\ell$ be even. Let $R_1^2 = R_2^2, R_3^2 = R_4^2, \ldots, R_{n-1}^2 = R_n^2$ with $0 \leq R_1^2 \leq R_3^2 \leq \cdots R_{n-1}^2$. For any $(t_1,\ldots,t_\ell) \in \mathbb{R}_{\geq 0}^\ell$, denote by $\mathcal{P}_\ell(t_1,\ldots,t_\ell)$ the following polytope:*

$$\mathcal{P}_\ell(t_1,\ldots,t_\ell) = \{(x_1,\ldots,x_\ell) \in \mathbb{R}^\ell \text{ s.t. } \forall i \in \{1,\ldots,\ell\}\ x_i \geq 0 \text{ and } \sum_{j=1}^{i} x_j \leq t_i\}.$$

And let $(r_1, r_2, \ldots, r_n) = (\frac{R_1^2}{R_n^2}, \frac{R_2^2}{R_n^2}, \ldots, \frac{R_n^2}{R_n^2})$, Then:

$$\Pr_{\mathbf{u}\sim Ball_n}(\forall j \in [1,n], \sum_{i=1}^{j} u_i^2 \leq r_j) = \ell! \cdot vol\mathcal{P}_\ell(r_2, r_4, \ldots, r_{n-2}, r_n) \tag{3.22}$$

with:

$$vol\mathcal{P}_\ell(t_1,\ldots,t_\ell) = \int_{y_1=0}^{t_1}\int_{y_2=y_1}^{t_2}\int_{y_3=y_2}^{t_3}\cdots\int_{y_\ell=y_{\ell-1}}^{t_\ell} dy_\ell \ldots dy_3 dy_2 dy_1. \tag{3.23}$$

Furthermore, the integral of (3.23) can be computed numerically as follows:

1. *Let $t_1,\ldots,t_\ell \in \mathbb{R}$ be given.*
2. *Let $Q_0 = 1$.*
3. *For $i = \ell$ downto 1 do:*
   - *(a) Compute the unique polynomial $Q \in \mathbb{R}[X]$ such that $Q'(X) = Q_{\ell-i}(X)$ and $Q(0) = 0$.*
   - *(b) Let $Q_{\ell-i+1} \in \mathbb{R}[X]$ defined by $Q_{\ell-i+1}(X) = Q(t_i) - Q(X)$.*
4. *Return $Q_\ell(0)$ as $vol\mathcal{P}_\ell(t_1,\ldots,t_\ell)$.*

Proof. Recall that the distribution of the vector $(u_1^2 + u_2^2, u_3^2 + u_4^2, \ldots, u_{d-1}^2 + u_d^2)$ when $\mathbf{u}$ is chosen from Ball_d is is simply a uniform distribution over the set of all vectors whose coordinates are non-negative and sum to at most 1. This proves that:

$$\Pr_{\mathbf{u}\sim\mathrm{Ball}_n}\left(\forall j \in [1,n], \sum_{i=1}^{j} u_i^2 \leq r_j\right) = \frac{vol\mathcal{P}_{n/2}(r_2, r_4, \ldots, r_{n-2}, r_n)}{vol\mathcal{P}_{n/2}(1,1,\ldots,1)},$$

Notice that $vol\mathcal{P}_{n/2}(1,1,\ldots,1) = 1/\ell!$ because it is the volume of the standard simplex, which proves (3.22).

In general, computing the volume of a polytope is not easy, but our polytope has a special shape which makes it easy:

$$vol\mathcal{P}_\ell(t_1,\ldots,t_\ell) = \int_{x_1=0}^{t_1}\int_{x_2=0}^{t_2-x_1}\int_{x_3=0}^{t_3-x_1-x_2}\cdots\int_{x_\ell=0}^{t_\ell-\sum_{i=1}^{\ell-1}x_i} dx_\ell\ldots dx_3dx_2dx_1,$$

which by the change of variable $y_i = \sum_{j=1}^{i} x_i$ becomes:

$$vol\mathcal{P}_\ell(t_1,\ldots,t_\ell) = \int_{y_1=0}^{t_1}\int_{y_2=y_1}^{t_2}\int_{y_3=y_2}^{t_3}\cdots\int_{y_\ell=y_{\ell-1}}^{t_\ell} dy_\ell\ldots dy_3dy_2dy_1,$$

which proves (3.23).

Finally, we note that this integral can be evaluated easily, by successive integrations giving rise to a multivariate polynomial in $t_1,\ldots,t_\ell$. Numerically, it can be evaluated by the iterative process described in algorithm 9.

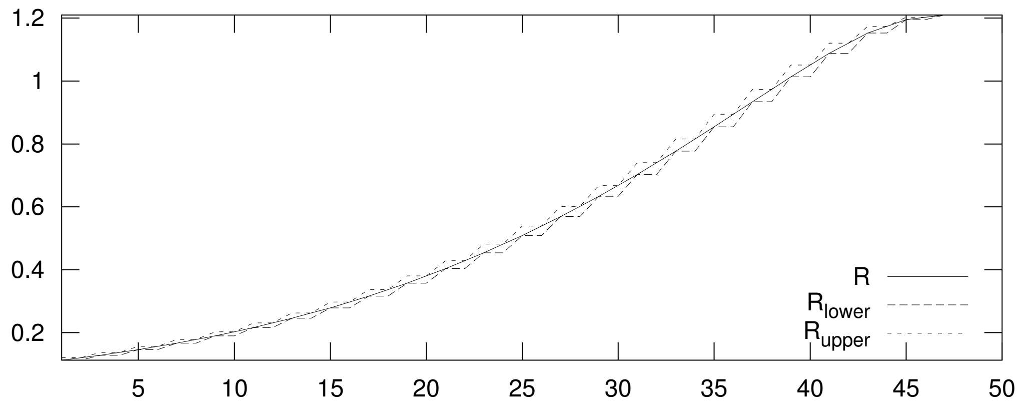
Algorithm 9 `VolumePolytope`

```
Input: $R_1^2,\ldots,R_n^2$
Output: The volume of the polytope defined in (3.23)
1: $C \leftarrow 1$; // $C \in \mathbb{R}[X]$ is a polynomial
2: for $i = n, n-1, \ldots, 1$ do
3:     $C \leftarrow \int_0^x C(t)dt$;
4:     $C \leftarrow C(R_i^2/R_n^2) - C(x)$;
5: end for
6: return $C(0)$
```

□

The cost for even layers in the enumeration tree can be estimated using the previous lemma, and the odd layers can be estimated by interpolation. For general $\mathbf{R} = (R_1^2, R_2^2, \ldots, R_n^2)$, the cost estimate of the bounding $\mathbf{R}_{upper} = (R_2^2, R_2^2, R_4^2, R_4^2, \ldots)$ serves as an upper estimate, and the bounding $\mathbf{R}_{lower} = (R_1^2, R_1^2, R_3^2, R_3^2, \ldots)$ serves as a lower estimate, as illustrated in figure 3.5.

Figure 3.5: Upper and lower estimates the cost of a given bounding function.



$$N_i(\mathbf{R}_{lower}) \leqslant N_i(\mathbf{R}) \leqslant N_i(\mathbf{R}_{upper}), \qquad i = 2, 4, \ldots. \tag{3.24}$$

Fig. 3.6 shows an example of estimating the number of nodes in even layers for enumeration with different pruning strategy. The number of nodes is shown in log base, and the pruning strategy correspond to full enumeration(no pruning), success probability 95% and success probability 25%. The colored zone of Fig. 3.6 is the distance between the upper estimate and the lower estimate for even layers. According the figure, these bounds are fairly close to each other, which suggests to estimate the total number of nodes by a loose and cheap interpolation, as done in Algorithm 10.

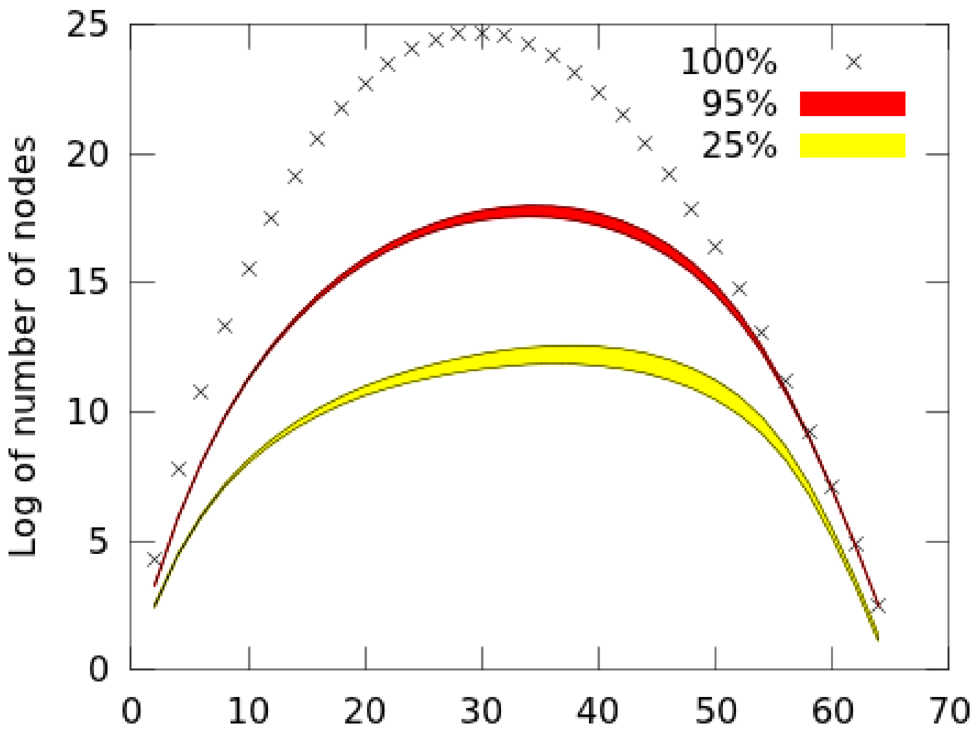


Figure 3.6: Accuracy of the estimate for the enumeration cost.

For odd layers, we approximate their N_i values by averaging the neighboring N_i for even layers, i.e.

$$\begin{aligned} N &= \sum_{i=1}^{n} N_i \\ &\approx \sum_{i=1}^{\lfloor n/2 \rfloor} N_{2i} + (N_{2i-1} + N_{2i+1})/2 \\ &\approx 2 \cdot \sum_{i=1}^{\lfloor n/2 \rfloor} N_{2i}. \end{aligned}$$

Here, the last approximations are due to neglecting the N_i terms for $i = 1, n$, and $i = n - 1$ if n is odd. For these layers, N_i is usually very small.

Thus we have algorithm 10 which computes efficiently an estimate of the enumeration cost for general $\mathbf{R}$, which is the average value of the upper and lower estimates. It is easy to write an analogue of this algorithm that computes only the upper or the lower estimate.

Algorithm 10 Estimation of the enumeration cost

Input: A bounding function $\mathbf{R} = (R_1^2 \leq \cdots \leq R_n^2)$, and the Gram-Schmidt squared norms $\|\mathbf{b}_1^\star\|^2, \ldots, \|\mathbf{b}_n^\star\|^2$ of the input basis for enumeration.

Output: Estimation of the total number of nodes t to be enumerated when using the Gaussian heuristic as the enumeration radius.

```
1: $N \leftarrow 0$;
2: for i=2,4,... do
3:     $f_i \leftarrow i \cdot R_i + \log(v_i) - \sum_{i=n-i+1}^{n} \log(\|\mathbf{b}_i^\star\|)$; // log of cost of full enumeration within radius $R_i$ (level i).
4:     $C_i \leftarrow \frac{1}{2}(\texttt{VolumePolytope}(R_1^2, R_3^2, \ldots, R_{i-1}^2) + \texttt{VolumePolytope}(R_2^2, R_4^2, \ldots, R_i^2))$
5:     $N_i \leftarrow (i/2)! \cdot C_i \cdot f_i$ // $C_i \cdot (\frac{i}{2})! = \Pr_{\mathbf{u} \sim Ball_i} (\forall t \in [1, i], \sum_{l=1}^{t} u_i^2 \leq R_t)$
6:     $N \leftarrow N + N_i$;
7: end for
8: return $2N$
```

3.3.2 Success probability for finding a single vector

We want to investigate the probability of successfully finding a vector with a given pruning strategy. In this subsection, we first study the probability of finding a single vector $\mathbf{v}$ with fixed norm. By taking into consideration the density of vectors within a ball of radius r, we will extend our computation to the probability of finding any of the vectors within range.

Let $p_s(\|\mathbf{v}\|, \mathbf{R})$ be the probability that a vector of norm $\|\mathbf{v}\|$ can be found by enumeration with pruning $\mathbf{R} = (R_1, \ldots, R_n)$. The analysis of the basic case where $\|\mathbf{v}\|^2 = R_n^2$ is first presented by Gama et al. [24]. And we naturally extend the computation to general $\|\mathbf{v}\|$.

To define such success probability, it is important to make clear our assumptions. The enumeration is a deterministic algorithm. When we fix a bounding function $\mathbf{R}$, the result of each independent launch is identical. The "probability" here refers to the fact that we have no information of $\mathbf{v}$ in advance, especially, we do not know the direction of $\mathbf{v}/\|\mathbf{v}\|$, although we are given $\|\mathbf{v}\|$. It looks to us as if it is a random vector uniformly distributed over the sphere of radius $\|\mathbf{v}\|$. In other words, the probability measures the event that such a random vector is found by our enumeration routine, assuming $\mathbf{v}$ is uniformly distributed over the sphere. Alternatively, if we are given many sufficiently random independent input bases, then the event that the program outputs the target vector is also measured by probability p_s.

To be exact, we need the following heuristic assumption on the input basis:

Heuristic 3.3.2. *The distribution of the coordinate of the target vector* $\mathbf{v}$*, when written in normalized Gram-Schmidt basis* $(\mathbf{b}_1^\star/\|\mathbf{b}_1^\star\|, \ldots, \mathbf{b}_n^\star/\|\mathbf{b}_n^\star\|)$ *of the input basis, looks like a uniformly distributed vector of norm* $\mathbf{v}$.

This heuristic says that in a random basis, the vector $\mathbf{v}$ is not oriented in any particular direction. For different basis $\mathbf{B}$ of a fixed lattice $\mathcal{L}$, if it is sufficiently randomized and it is not too strongly reduced, then it will satisfy the randomness requirement in the assumption. Because the number of vectors of norm $r \cdot (vol(\mathcal{L})/v_n)^{1/n}$ is expected to be r^n, there will be sufficiently many such vectors to choose from, as a result, the collection of these basis show similar behavior as random basis when we calculate p_s.

We first consider the basic case where $\|\mathbf{v}\|^2 = R_n^2$. Let $\mathbf{r}$ be the normalization of $\mathbf{R}$, i.e.,

$$\mathbf{r} = (r_1, \ldots, r_n) = \left(\frac{R_1^2}{R_n^2}, \frac{R_2^2}{R_n^2}, \ldots, \frac{R_n^2}{R_n^2} \right).$$

Obviously $p_s(R_n^2, \mathbf{R}) = p_s(1, \mathbf{r})$. We note $p_s(1, \mathbf{r})$ as $p_s(\mathbf{r})$ for short.

Let $\mathbf{v}$ be the normalized target vector, and let $\mathbf{x} = (x_1, \dots, x_n) \in \mathbf{R}^n$ be its coordinates in the orthonormal basis $(\mathbf{b}_1^\star/\|\mathbf{b}_1^\star\|, \dots, \mathbf{b}_n^\star/\|\mathbf{b}_n^\star\|)$. By definition, $\mathbf{v}$ belongs to the pruned tree if and only if for all $k = 1, \dots, n, \sum_{j=k}^{n} x_j^2 \leqslant r_{n-k+1}$. By Heuristic 3.3.2, $\mathbf{x}$ is uniformly distributed over the surface of the hypersphere of radius $\|\mathbf{x}\| = \|\mathbf{v}\|$. We use S^{n-1} to denote the unit hypersphere in n dimension, then, p_s can be estimated as

$$p_s(\mathbf{r}) = \Pr_{\mathbf{u}\sim S^{n-1}}(\forall k \in [1, n], \sum_{j=k}^{n} u_j^2 \leqslant r_{n-k+1}) \tag{3.25}$$

Monte Carlo estimation

As before, one can estimate this probability through Monte Carlo simulation. That is, we uniformly generate many points on the unit sphere, and count he number of points which satisfy the bounding function $\mathbf{r}$. This is described in algorithm 11.

A random point on the unit sphere can be estimated in the following way. First, its n coordinates $v_1, \dots, v_n$are sampled from independent identical Gaussian distribution, then each coordinate is normalized to have unit norm. We use the property that when X is a centered Gaussian variable with variance 1, then X^2 conforms to χ_1^2 distribution. This follows from the following theorem.

Theorem 3.3.3. *If $X_1, X_2, \dots, X_n$ are i.i.d Gaussian variables, then the sum of their squares has the chi-squared distribution with n degrees of freedom.*

$$X_1^2 + X_2^2 + \dots X_n^2 \sim \chi_n^2$$

Besides, chi-squared distribution is a special case of gamma distribution, so that $X \sim \chi_n^2$ implies that $X \sim \Gamma(k/2, 2)$. In practice, many computational software have integrated support for gamma distribution. Therefore we adopt the expression of gamma distribution in our algorithm description.

Algorithm 11 Estimate success probability using Monte Carlo

```
Input: A bounding function $\mathbf{r} = (r_1, \dots, r_n)$, and $r_1 \leqslant \dots \leqslant r_n$.
Output: An estimate of $p_s$ with Monte Carlo sampling over a sphere.
 1: $t \leftarrow 0$; // counter for successful samples
 2: for $i = 1, \dots,$NUM_SAMPLES do
 3:     $\mathbf{s} \leftarrow$(GenerateRandomGamma(0.5,2))$_{1\times\beta}$; // $\mathbf{s} = (s_1, \dots, s_n)$ is a vector of length n, where each coordinate follows an independent distribution Gamma$(0.5, 2)$.
 4:     $c_1 \leftarrow s_1$;
 5:     for $j = 2, \dots, n$ do
 6:         $c_j \leftarrow s_j + c_{j-1}$; // $c_j \leftarrow \sum_{k=1}^{j} s_k$
 7:     end for
 8:     if $\forall j \in \{1, \dots, n\}, c_n \cdot r_j > c_j$ then
 9:         $t \leftarrow t + 1$; // One more successful point
10:     end if
11: end for
12: return $t$/NUM_SAMPLES;
```

Exact computation and approximate bounds

In some cases we can compute $p_s(\mathbf{r})$ exactly. In fact, when a vector $\mathbf{u}$ is uniformly sampled from S^{n-1}, its first $n-2$ coordinates $(u_1, \dots, u_{n-2})$ is distributed uniformly over all vectors whose coordinates are non-negative and sum to at most 1, then $(u_1^2 + u_2^2, \ldots, u_{n-3}^2 + u_{n-2}^2)$ is the Dirichlet distribution of parameter $(1, \ldots, 1)$. Therefore, when dimension n is even, and the bounding function satisfies the property $r_1 = r_2, r_3 = r_4, \ldots, r_{n-1} = r_n = 1$, the precise value of success probability $p_{\mathrm{s}}(\mathbf{r})$ can be numerically computed using equation 3.22, which we restate here:

$$\Pr_{\mathbf{u} \sim \mathrm{Ball}_{n-2}} (\forall j \in [1, n-2], \sum_{i=1}^{j} u_i^2 \le r_j) = \left(\frac{n-2}{2}\right)! \cdot vol\mathcal{P}_{\frac{n-2}{2}}(r_2, r_4, \ldots, r_{n-2}) \tag{3.26}$$

And for a general bounding function $\mathbf{r}$, we can still give an upper estimations with $\mathbf{r}'' = (r_2, r_2, \ldots, r_n, r_n)$, such that

$$p_{\mathrm{s}}(\mathbf{r}) \leqslant p_{\mathrm{s}}(\mathbf{r}'') \tag{3.27}$$

However, $\mathbf{r}' = (r_1, r_1, \ldots, r_{n-1}, r_{n-1})$ does not provide a lower estimate for $r_{n-1} < 1$, because there are always $p_{\mathrm{s}}(\mathbf{r}') = 0$ in this case. When $r_{n-1} \approx 1$, we get a lower estimation with $\mathbf{r}' = (r_1, r_1, \ldots, r_{n-3}, r_{n-3}, 1, 1)$. When the dimension is odd, we approximate $p_{\mathrm{s}}(\mathbf{r})$ by $p_{\mathrm{s}}(r_1, r_2, \ldots, r_n, 1)$.

Algorithm 12 gives an upper estimate for a general bounding $p_{\mathrm{s}}(\mathbf{r})$.

Algorithm 12 Estimate success probability $p_{\mathrm{s}}(\mathbf{r})$

```
Input: A bounding function $\mathbf{r} = (r_1, \ldots, r_n)$ satisfying $r_1 \le \cdots \le r_n = 1$.
Output: An estimation of $p_{\mathrm{s}}(r)$.
1: $\mathbf{r}' \leftarrow (r_1, r_3 \ldots, r_{2\cdot\lceil n/2\rceil-3})$;
2: $\mathbf{r}'' \leftarrow (r_2, r_4 \ldots, r_{2\cdot\lceil n/2\rceil-2})$;
3: $p_{low} \leftarrow$ VolumePolytope($\mathbf{r}''$) $\cdot$ ($\lceil n/2\rceil - 1$)!;
4: $p_{high} \leftarrow$ VolumePolytope($\mathbf{r}'$) $\cdot$ ($\lceil n/2\rceil - 1$)!;
5: return $(p_{low} + p_{high})/2$
```

Extend analysis to general $\|\mathbf{v}\|$

For general $\|\mathbf{v}\|$, we have the following observation: when $\|\mathbf{v}\|^2 > R_n^2$, the pruned enumeration will never give the solution searched for, because the convex body defined by $\mathbf{R}$ is completely contained in the interior of $\|\mathbf{v}\|$ sphere. When $\|\mathbf{v}\|^2 < R_n^2$, then $\mathbf{v}$ can lie in the intersection of the ball of sphere $\|\mathbf{v}\|$ and the polytope defined $\mathbf{R}$. In other words, if we define bounding $\mathbf{R}' = \mathrm{trim}(\mathbf{R}, \|\mathbf{v}\|^2)$, where trim operation consists of substituting R_i by $\min(\|\mathbf{v}\|^2, R_i)$:

$$\mathrm{trim}(\mathbf{R}, x) = (\min(x, R_1), \min(x, R_2), \ldots, \min(x, R_n)), \tag{3.28}$$

then the probability that $\mathbf{v}$ is found by enumeration with bounding function $\mathbf{R}$ is the same with bounding function $\mathbf{R}'$, i.e. $p_{\mathrm{s}}(\|\mathbf{v}\|, \mathbf{R}) = p_{\mathrm{s}}(\|\mathbf{v}\|, \mathrm{trim}(\mathbf{R}, \|\mathbf{v}\|^2))$. Now $\mathbf{R}'$ satisfies $R'^2_n = \|\mathbf{v}\|$, and we are back in the basic case.

In conclusion, $p_{\mathrm{s}}(\|\mathbf{v}\|, \mathbf{R})$ can be reduced to $p_{\mathrm{s}}(\mathbf{r})$ for some $\mathbf{r}$:

$$p_{\mathrm{s}}(\|\mathbf{v}\|, \mathbf{R}) = \begin{cases} 0 & \text{if } \|\mathbf{v}\|^2 > R_n \\ p_{\mathrm{s}}(1, \mathbf{r}) & \text{if } \|\mathbf{v}\|^2 = R_n \\ p_{\mathrm{s}}(\|\mathbf{v}\|, \mathrm{trim}(\mathbf{R}, \|\mathbf{v}\|^2)) & \text{if } \|\mathbf{v}\|^2 < R_n \end{cases} \tag{3.29}$$

3.3.3 Success probability considering multiple vectors

In a more general case, we only know the distribution of short vectors for the lattice basis before enumeration. We are not aiming to find a certain vector of specific norm, but more generally, we are interested in computing the probability of finding any of them, which we note as $p_{\text{succ}}(R, \mathbf{R})$.

We approximate the result using finite element method, by discretizing R into s intervals. For each interval, we count the average number of vectors belonging to it, and compute their expected norms $\bar{k}(R, R + \Delta R)$. Then, all vectors in the interval were considered to be of norm $\bar{k}$, and they are assumed to be independently and uniformly distributed over the sphere of radius $\bar{k}$ for approximation. The probability to find any of them is $1 - \left(1 - p_s(\bar{k}, \mathbf{B})\right)^{\bar{k}}$. In the event that some shorter vector has already been found for some $R' < R$, we avoid repetition by only taking into account the previous vector, which adds a factor of $1 - p_{\text{succ}}(R)$. Finally, we sum up probability over all intervals. The detailed description is given as follows.

Finite element analysis

We predict the distribution of vectors with Gaussian heuristic. Note that the rigorous analysis of short vectors in a lattice using Poisson process give identical results. So within radius R, the total number of vectors is (excluding the symmetric half):

$$t(R) = \frac{1}{2} \cdot \frac{R^n \cdot v_n}{vol(\mathcal{L})} \tag{3.30}$$

And therefore between the interval $[R, R + \Delta R]$, the expected number of vectors is $t(R + \Delta R) - t(R)$, and their expected length is

$$\bar{k}(R, R + \Delta R) = \frac{n}{n+1} \cdot \frac{(R + \Delta R)t(R + \Delta R) - rt(R)}{t(R + \Delta R) - t(R)}. \tag{3.31}$$

Then the probability that any vector of norm between R and $R + \Delta R$ is found is approximately

$$1 - (1 - p_s(\bar{k}(R, R + \Delta R), \mathbf{R}))^{t(R)}$$

Let $p_{\text{succ}}(R, \mathbf{R})$ be the probability that enumeration with pruning $\mathbf{R}$ finds any vector shorter R. Then, we can compute the probability of $p_{\text{succ}}(R, \mathbf{R})$ inductively by increasing R by ΔR:

$$p_{\text{succ}}(R + \Delta R, \mathbf{R}) = p_{\text{succ}}(R, \mathbf{R}) + (1 - p_{\text{succ}}(R, \mathbf{R})) \left(1 - (1 - p_s(\bar{k}(R, R + \Delta R), \mathbf{R}))^{t(R)}\right) \tag{3.32}$$

Choosing intervals of R

What we do here is to approximate the length of each vector by the expectation of norm over all vectors in the interval $[R, R + \Delta R]$. We have to choose ΔR carefully to ensure that this is a good approximation. We can always make ΔR small enough to avoid precision issue, but we try to avoid computing over too many intervals, in order to speed up computation.

At first it might seem tempting to make the following choices for ΔR:

- The most naive way is to choose ΔR to be a small constant, but in this way, the number of vectors in the interval $[R, \Delta R]$ grows exponentially when R increases, then $\bar{k}(R, R + \Delta R)$ may not be representative enough for all vectors in that interval.

- On the other hand, choosing $\Delta R = O(R^{1/n})$ can ensure that there will be a constant number of vectors in each interval, but the value of ΔR may be too big when R is small, making $\bar{k}(R, R + \Delta R)$ unrepresentative again.

These two examples represent the two extreme cases that should be avoided. Following is a way to generalize the strategy of dividing the interval $[0, R]$. There is a parameter ν in this strategy. When $\nu = 1$, this amounts to choosing a constant ΔR. While for $\nu = n$, the strategy chooses $\Delta R = O(R^{1/n})$, which is the second example given. Setting $\nu \in (1, n)$ gives an intermediate choice.

Our method consists of dividing the interval $[0, R]$ into s parts by defining the sequence

$$s_i = \left(\frac{i}{s}\right)^{1/\nu} \cdot R, \qquad \forall 0 \leqslant i \leqslant s, \tag{3.33}$$

where $\nu \in [1, n]$ is a parameter.

Figure. 3.7 illustrates the s_i sequences for different choices of ν. The solid line shows $t(R), 0 < R < 1.05 \cdot GH(\mathcal{L})$, which is the expected number of vectors with regard to radius in a typical 70-dimensional lattice. The $\times$ points, $+$ points and o points signify $\nu = 1, 17.5, 70$ respectively. Notice that the majority of the vectors are expected to have radius in the interval $[1, 1.05]$. For $\nu = 1$, the interval $[1, 1.05]$ is underrepresented, while for $\nu = 70$, the interval $[0, 1]$ is underrepresented.

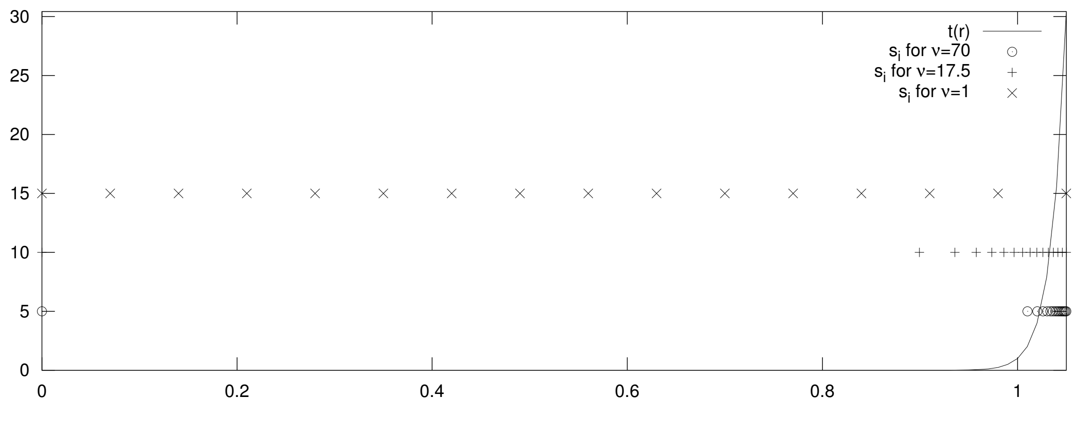


Figure 3.7: $t(R), 0 < R < 1.05 \cdot GH(\mathcal{L})$ for 70 dimensional lattice of unit volume (solid line). The dots show s_i sequence for $\nu = 1, 17.5, 70$ respectively.

In order to choose a good ν, we tested the speed of convergence of estimation by increasing s. After numerous experiments we found that $n/8 < \nu < n/2$ gives best performance in terms of convergence of estimation, therefore we decided to fix $\nu = n/4$ from now on. We also fix the total number of intervals to be $s = 1000$ in practice. Algorithm 13 is the pseudo-code for estimating $p_{\text{succ}}(R, \mathbf{R})$. By default we also note $p_{\text{succ}}(\mathbf{R})$ for $p_{\text{succ}}(R_n, \mathbf{R})$.

Algorithm 13 Evaluate $p_{\text{succ}}(\mathbf{R})$

```
Input: a pruning function $\mathbf{R} = R_1, \ldots, R_n$. The radius of enumeration $R$
Output: $p_{\text{succ}}(R, \mathbf{R})$ the probability that enumeration with pruning $\mathbf{R}$ finds a vector shorter than $R$.
    And $E_R$ the expected norm of the found vector.
1: $p \leftarrow 0, u \leftarrow 0$
2: for $i = 1, \ldots, 1000$ do
3:     $s_i \leftarrow (i/1000)^{4/n} \cdot R$
4:     $q \leftarrow (1-p)\left(1 - p_{\text{s}}(\bar{k}(s_{i-1}, s_i), \mathbf{R})^{t(s_i)-t(s_{i-1})}\right)$
5:     $p \leftarrow p + q$
6:     $u \leftarrow u + q \cdot \bar{k}(s_{i-1}, s_i)$
7: end for
8: $E_R \leftarrow u/p$
9: return $E_R, p$
```

3.4 Bounding function

In this section, we will show how to find an optimal bounding function $\mathbf{R}$ for enumeration. Our estimation of T_{enum} and p_{succ} is precise only when $R_{2i} = R_{2i+1}$, thus we will only search for function which satisfies this relationship. Define $R_{\text{odd}}(\mathbf{R}) = \{R_1, R_3, \ldots, R_{2\lceil n/2\rceil -1}\}$. Instead of searching for optimal bounding functions in $\mathcal{R}_n$, where $\mathcal{R}_n$ is the set of all possible bounding functions, we actually restrict our searching domain to $\mathcal{R}_{\lceil n/2\rceil} = \{R_{\text{odd}}(\mathbf{R})\}$. Recovering a solution from $\mathcal{R}_{\lceil n/2\rceil}$ is straightforward.

Besides, we are actually interested in the optimum bounding function modulo $GH(\mathcal{L})$, which describes the density of the given lattices. Therefore, hereafter we will be using $\mathbf{r} = R_{\text{odd}}\mathbf{R}/GH(\mathcal{L})$ to denote the bounding function modulo volume, and we search the optimal solution in space $\mathcal{R}_{\lceil n/2\rceil}$. And in our computation of cost and success probability, we use r in the world of modulo $GH(\mathcal{L})$ instead of R.

3.4.1 Optimization problem

The idea of enumeration using extreme pruning by Gama et al. [24] consists of taking a relatively small $p_{\text{succ}}(r, \mathbf{R})$ for each enumeration, but randomize the basis and run enumeration many times. For about $1/p_{\text{succ}}(r, \mathbf{R})$ enumerations, it is expected to find a solution vector within radius r. The problem of searching for best bounding function can be formulated as follows.

Given a lattice $\mathcal{L}(\mathbf{B})$, suppose that randomizing and preprocessing this basis to reduce it to similar qualities as $\mathbf{B}$ will take T_0 time in average, the expected running time for enumeration is $T_{enum}(\mathbf{R}, \mathbf{B}) = t_{node} \cdot N(\mathbf{R}, \mathbf{B})$, where $N(\mathbf{R}, \mathbf{B})$ is the number of nodes enumerated with bounding function $\mathbf{R}$. Here t_{node} is the average time for visiting a single node, which is approximately 10^{-7} second for double precision in C++ implementation. The total time expected for finding a solution is

$$T = \frac{T_0 + T_{enum}(\mathbf{R}, \mathbf{B})}{p_{\text{succ}}(r, \mathbf{R})} \tag{3.34}$$

An optimal bounding function enumeration with extreme pruning is a bounding function which minimizes the total running time T. This is to say, the optimal bounding function is the solution to the following optimization problem:

$$\begin{aligned} \underset{\mathbf{R}}{\text{minimize}} \quad & \frac{T_{\text{enum}}(\mathbf{R}, \mathbf{B}) + T_0}{p_{\text{succ}}(r, \mathbf{R})} \\ \text{subject to} \quad & R_{\text{odd}}(\mathbf{R}) \in \nabla_{\lceil n/2 \rceil} \end{aligned} \tag{3.35}$$

There are numerical ways to compute the solutions to this problem, which will be presented in following subsections.

The optimization problem is defined by three parameters: $(T_0, \mathbf{B}, r)$. Thus we note the solution to as $\mathbf{R}(T_0, \mathbf{B}, r)$. Figure. 3.8 shows some example bounding functions as numerical solutions to optimization problems. Here $n = 70$, and the solutions are in $\mathcal{R}_{\lceil n/2 \rceil}$ space.

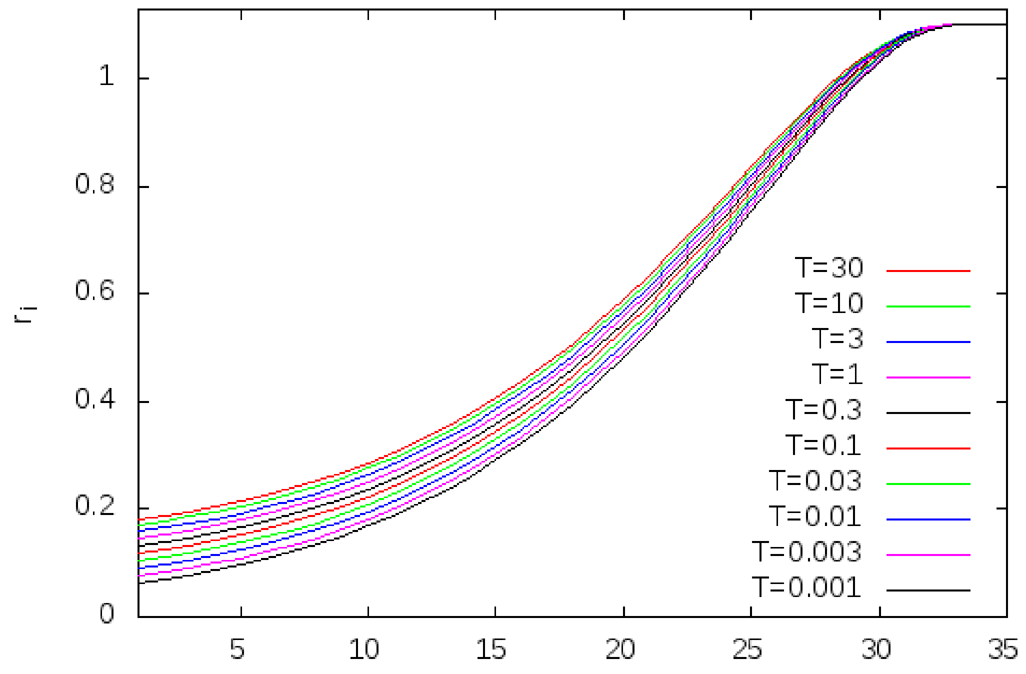


Figure 3.8: The numerical solutions $\mathbf{R}(T_0, \mathbf{B}, r)$ with a fixed $\mathbf{B}$ of $n = 70$, $r = 1.05$, and different T_0

3.4.2 Numerical solutions

Optimization problem of continuous variable is a classical topic. One possible method is to do a random search, using cross-entropy method for example. An advantage of random search is that it would finds the global optima with good chance. However, it suffers from the disadvantage of being very slow. Another classical way to search for optimal solution in optimization is to use Newton type methods, such as sub-gradient method or method of steepest descent. In contrast with the random search, this method is generally much more efficient, and has a good running time bound. However, there are two main disadvantages. First, they require the computation of first or second derivatives of the target function. This might make the computation numerically unstable, because we can only numerically compute the target function, then approximate derivatives by finite difference. Besides, it is possible that the method outputs a result which is a local optima, but not global optima. A similar method is golden section search. With this method, while we are not required to compute the first and second derivatives, we still risk of falling into local optima,

In our experiment we tried two different methods: random search using cross-entropy method, and golden section search.

Cross-entropy method

Cross-entropy method is attributed to Reuven Rubinstein [76], originally applied in domain of machine learning. Among all random search methods, one important trait of this method is that it tried to find random steps in an intelligent way, which is adaptive to the function values, which makes it more efficient.

To do so, it does not only store the current optima $\mathbf{R}$, but also maintain a list L of first `N` "good $\mathbf{R}'$s". It learns the variance of the elements in the list L. This variance is then used to generate new perturbations. Then we evaluate the perturbed $\mathbf{R}'$, $\mathbf{R}'$ is either updated to $\mathbf{R}$, or added to the list L, or discarded. The algorithm continues iteratively the previous procedure, until the distribution of good perturbed $\mathbf{R}'$ in L has too little variance, which probably indicate that we are numerically very close to the optima.

This method is described in pseudo-code in algorithm 14.

Algorithm 14 Cross entropy method for searching $\mathbf{R}(T_0, \mathbf{B}, r)$

```
Input: $\mathbf{B} = \left(\|\mathbf{b}_1^*\|^2, \ldots, \|\mathbf{b}_n^*\|^2\right)$, the Gram-Schmidt squared norms of a lattice;
    $T_0$ the pre-processing and randomization time.
    $r$ the upper bound of radius of the target vector.
Output: A bounding function $\mathbf{R} = (R_1, \ldots, R_{\lceil n/2 \rceil})$ which minimizes $\frac{T_{\text{enum}}(\mathbf{R},\mathbf{B},r)+T_0}{p_{\text{succ}}(\mathbf{R})}$
1: $\mathbf{R} \leftarrow \mathbf{R}_0$ //Initialize to any $\mathbf{R}_0$, for example, $\mathbf{R} \leftarrow (\frac{1}{\lceil n/2 \rceil}, \frac{2}{\lceil n/2 \rceil}, \ldots)$.
2: $L \leftarrow \emptyset$
3: $\Sigma \leftarrow 0.1 \cdot (0.1, 0.1, \ldots, 0.1)_{\lceil n/2 \rceil}$ // Initial variance
4: while $\|Sigma\| > \epsilon$ do
5:     $\Delta\mathbf{R} \leftarrow$ GenerateRandom $(\Sigma)$
6:     $\mathbf{R}' = \mathbf{R} + \Delta(\mathbf{R})$
7:     if $T_{total}(\mathbf{R}') > T_{total}(L$'s last element$)$ then
8:         Add $\mathbf{R}'$ to $L$.
9:         if $T_{total}(\mathbf{R}') > T_{total}(\mathbf{R})$ then
10:            $\mathbf{R} \leftarrow \mathbf{R}'$
11:        end if
12:    end if
13:    update $\Sigma$ according to $L$.
14: end while
```

Some details of the algorithm:

- Σ is initialized to a relatively big value, to allow a wide exploration range for random search. In our case, it is set to $0.1 \cdot (0.1, 0.1, \ldots, 0.1)_{\lceil n/2 \rceil}$. In each step of update, instead of taking Σ to be the standard deviation of elements in L, it is better to add a term σ_t. This σ_t varies with the number of iterations t, in the beginning iterations, σ_t is relatively big in order to encourage exploration, and gradually this value is diminished, to ensure convergence of the search.

- For efficiency of algorithm in practice, Σ is not necessarily updated for each change of L, it can be updated every N/10 updates of L, for example. Especially, the first update of Σ should take place after the list L is filled with N elements.

Golden section search

As opposed to cross entropy method, golden section search optimizes $\mathbf{R}$ index by index. This method is guaranteed to find out the optimum solution of unimodal functions. A function $f(\cdot)$ is unimodal on interval $[l, u]$ if it satisfies following property: if $l < x_0 < u$ is its minimal point, then $f(\cdot)$ is decreasing over $[l, x_0]$ and increasing over $[x_0, u]$. In general, if the target function is not convex, then it is possible that the method returns only a local optimum.

The description in pseudo code is given in algorithm. 15 and 16. Algorithm 16 is the normal golden section search for single variable target functions. Algorithm 15 makes calls to algorithm 16 for each index of $\mathbf{R}$ iteratively, until no updates happen.

Algorithm 15 Find optimized bounding functions for enumeration $\mathbf{R}(T_0, \mathbf{B}, r)$

```
Input: $\mathbf{B} = \left(\|\mathbf{b}_1^*\|^2, \ldots, \|\mathbf{b}_n^*\|^2\right)$, the Gram-Schmidt squared norms of a lattice;
    $T_0$ the pre-processing and randomization time.
    $r$ the upper bound of radius of the target vector.
Output: A bounding function $R_{\text{odd}}(\mathbf{R}) = (R_1, \ldots, R_{\lceil n/2 \rceil})$ which minimizes $\frac{T_{\text{enum}}(\mathbf{R},\mathbf{B},r)+T_0}{p_{\text{succ}}(\mathbf{R})}$
 1: $R_{\text{odd}}(\mathbf{R}) \leftarrow \mathbf{R}_0$ //Initialize to any $\mathbf{R}_0$, for example, $\mathbf{R} \leftarrow (\frac{1}{\lceil n/2 \rceil}, \frac{2}{\lceil n/2 \rceil}, \ldots)$.
 2: update ← true
 3: while update = true do
 4:   for $i = 1, \ldots, \lceil n/2 \rceil$ do
 5:     $\rho \leftarrow \underset{\rho \in [R_{i-1}, R_{i+1}]}{\text{minimize}} \quad T_{\text{total}}\left((R_1, \ldots, R_{i-1}, \rho, R_{i+1}, \ldots, R_{\lceil n/2 \rceil}), \mathbf{B}, T_0\right)$
        //Using GoldenSectionSearch (Alg. 16) to find optimal point of a single variable function
 6:     if $|\rho - r_i| > \epsilon$ then
 7:       $r_i \leftarrow \rho$
 8:       update ← true;
 9:     end if
10:   end for
11: end while
```

Following is a description of golden section search in optimizing a single variable function. In a nutshell, the idea is to tighten interested interval $[l, u]$ until the length of the interval is smaller than ϵ. In each iteration, we compare the value of $f(t)$ and $f(t')$ where t and t' are two symmetric points of golden section of the interval $[l, u]$. If $f(t) > f(t')$ then we tighten the interval $[l, u] \leftarrow [t, u]$, else we do the symmetric. [69]

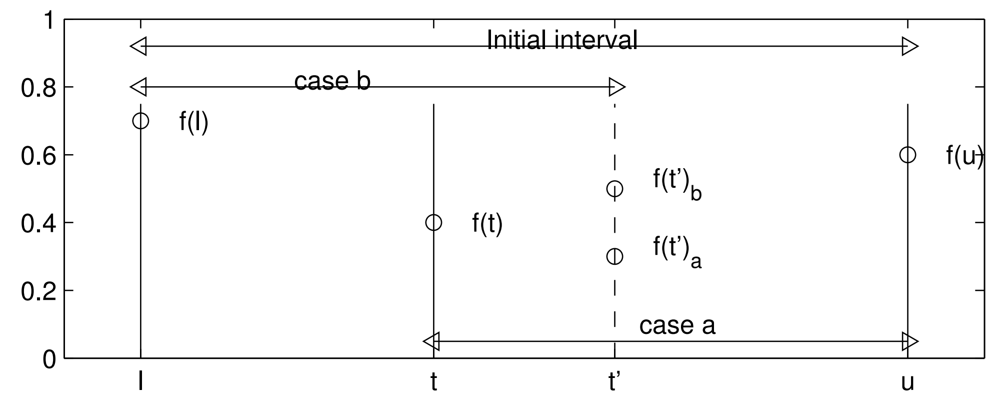


Figure 3.9: Illustration of golden section search algorithm: At each iteration, the initial interval is $[l, u]$. Depending on the comparison between $f(t)$ and $f(t')$ (case (a) or case (b)), the interested interval shrinks to by 0.618.

Algorithm 16 `GoldenSectionSearch`

Input: $f(\cdot)$ the target unimodal function to be optimized.
$[l, u]$ the interested interval
Output: t such that $f(t)$ minimizes $f(\cdot)$ over $[l, u]$

```
1: $\phi_1 \leftarrow \frac{\sqrt{5}-1}{2}, \phi_2 \leftarrow \frac{3-\sqrt{5}}{2}$
2: $t \leftarrow \phi_1 u + \phi_2 l$
3: while $u - l > \epsilon$ do
4:    if $u - t > t - l$ then
5:       $t' \leftarrow \phi_1 u + \phi_2 t$
6:    else
7:       $t' \leftarrow t$
8:       $t \leftarrow \phi_1 l + \phi_2 t'$
9:    end if
10:   if $f(t) > f(t')$ then
11:      $l \leftarrow t$
12:      $t \leftarrow t'$
13:   else
14:      $u \leftarrow t'$
15:   end if
16: end while
```

Using golden section search enabled us to find solution to the optimization problems faster than cross-entropy method. However, it is still quite slow. The time taken by finding a good bounding function is usually several hours on a machine of about 2GHz, in some case this is comparable to the time needed for enumeration itself. This is not satisfying enough, because we need to calculate a proper bounding function each time we want to perform a search of short vector for a new basis.

To overcome this problem, we prepared a large amount of bounding functions $\mathbf{R}(T_0, \mathbf{B}, r)$ for different dimension n, and different p_{succ}. For each enumeration we can choose to use the bounding function whose parameter is the closest to our need.

However, when we observed these numerical solutions of the optimization problem, we have discovered several interesting properties, which help us to solve it more efficiently.

Uniqueness of Solution We used several different optimization methods to find $\mathbf{R}$, such as cross-entropy method and golden section search. Regardless of the initial value of $\mathbf{R}$ and the optimization method we adopted, the result converges to the same optimal bounding. This strongly suggests that this problem has only one global optima. And the problem is very likely to be a convex optimization problem, although we cannot yet provide a proof of the convexity of the target function. In practice, we can apply all existing methods for convex optimization to this problem, like golden section search. We did not use Newton-type methods to avoid numerical stability issues.

Parameterization For various parameters $(T_0, \mathbf{B}, r)$, the bounding function $\mathbf{R}(T_0, \mathbf{B}, r)$ have very similar shapes, as shown in figure. 3.8. These bounding functions $\mathbf{R}(T_0, \mathbf{B}, r)$ dwell in a subspace inside $\mathcal{R}_{\lceil n/2 \rceil}$, which can be analyzed by principal component analysis (PCA). We would hope to explore following relationship for $\mathbf{R}(T_0, \mathbf{B}, r)$:

$$\mathbf{R}(T_0, \mathbf{B}, r) = \mathbf{R}_0 + T_0 \cdot \mathbf{R}_1 + r \cdot \mathbf{R}_2 \tag{3.36}$$

so that all $\mathbf{R}(T_0, \mathbf{B}, r)$ can be computed as linear combination of some known vectors. But this model is too simple to approximate the real case. In fact, the factors T_0, r, $\mathbf{B}$ are not necessarily linear in the previous equation, and they are not necessarily independent between one another. A more comprehensive model is

$$\mathbf{R}(T_0, \mathbf{B}, r) = \mathbf{R}_0 + f_1(T_0) \cdot \mathbf{R}_1 + f_2(r) \cdot \mathbf{R}_2 + f_3(\mathbf{B}) \cdot \mathbf{R}_3 + f_4(T_0, r) \cdot \mathbf{R4} + \dots \tag{3.37}$$

here $f_i(\cdot)$ are functions that we have to learn from the known bounding functions.

Our goal is to reproduce or approximate $\mathbf{R}(T_0, \mathbf{B}, r)$ using as few terms as possible. A good approximation will serve as a good initial value for optimization algorithm, and finally we hope to reproduce $\mathbf{R}$ with high precision, so that we can generate it immediately when we need it for enumeration.

Heuristic Cost The cost of the optimal bounding function $T_{\text{enum}}(\mathbf{R}(T_0, \mathbf{B}, r)) \approx T_0$. This heuristic result is not directly visible from the problem statement in (3.35). However in practice, $T_{\text{enum}}(\mathbf{R}, \mathbf{B})$ usually increase much faster than $1/p_{\text{succ}}(\mathbf{R})$, and it is often the case that $T_{\text{enum}}(\mathbf{R}(T_0, \mathbf{B}, r)) \approx T_0$. This heuristic is useful later when we want to compute $\mathbf{R}$ which make the enumeration run in some given time, for example during BKZ.

3.5 Optimize and parameterize bounding functions

The optimization problem of equation 3.35 finds a bounding function for enumeration with extreme pruning. This enumeration routine is used as a stand-alone algorithm to solve HSVP. However, we also want to apply the pruning on enumeration as subroutine of BKZ reduction. This problem is somehow different from previous one.

BKZ requires a subroutine to find a shortest vector in a local projected lattice $\mathcal{L}(\mathbf{B}_{[j,k]})$: given as input two integers j and k such that $1 \le j \le k \le n$, output $\mathbf{v} = (v_j, \dots, v_k) \in \mathbb{Z}^{k-j+1}$ such that $\|\pi_j(\sum_{i=j}^{k} v_i \mathbf{b}_i)\| = \lambda_1(\mathcal{L}(\mathbf{B}_{[j,k]}))$. In practice, as well as in the BKZ article [79], this is implemented by enumeration which is a variation of Alg. 8.

Similar analysis and pruning techniques can be applied to this enumeration procedure. Especially, when block size is large enough ($\beta > 50$), the behavior of the blocks in BKZ is very similar to that of random basis. However, there are two major difference between enumeration sub-procedure in BKZ and a stand alone procedure, namely, we have different constraint and optimization goal.

1. Instead of hoping enumeration to successfully find out a short vector for each block, we can tolerate more failure where certain blocks are left unchanged. In this case the optimization goal is much more complicated, because it is not evident how a single enumeration will effect the efficiency of BKZ reduction as a whole.

2. BKZ makes a lot of calls to enumeration routine, and each enumeration is usually in shorter blocksize than in a stand-alone enumeration routine. This means that we want to use yet less time to compute proper bounding functions for enumeration routine.

Considering these differences, we need bounding functions that are solutions to different optimization problems, defined as follows.

- One possible tentative is compute optimal $\mathbf{R}$ for a fixed r and $p_{\text{succ}}(r, \mathbf{R}) = p$. In this way the output quality of the enumeration can be anticipated.

$$\begin{array}{ll} \underset{R_{\text{odd}}(\mathbf{R}) \in \mathcal{R}_{\lceil n/2 \rceil}}{\text{minimize}} & T_{\text{enum}}(\mathbf{R}, \mathbf{B}) \\ \text{subject to} & p_{\text{succ}}(r, \mathbf{R}) = p \end{array} \tag{3.38}$$

Figure 3.10 shows examples of bounding functions in dimension 65, which are solutions to this optimization problem with $r = 1$, and various p, on the right is their enumeration cost for each level in the enumeration tree.

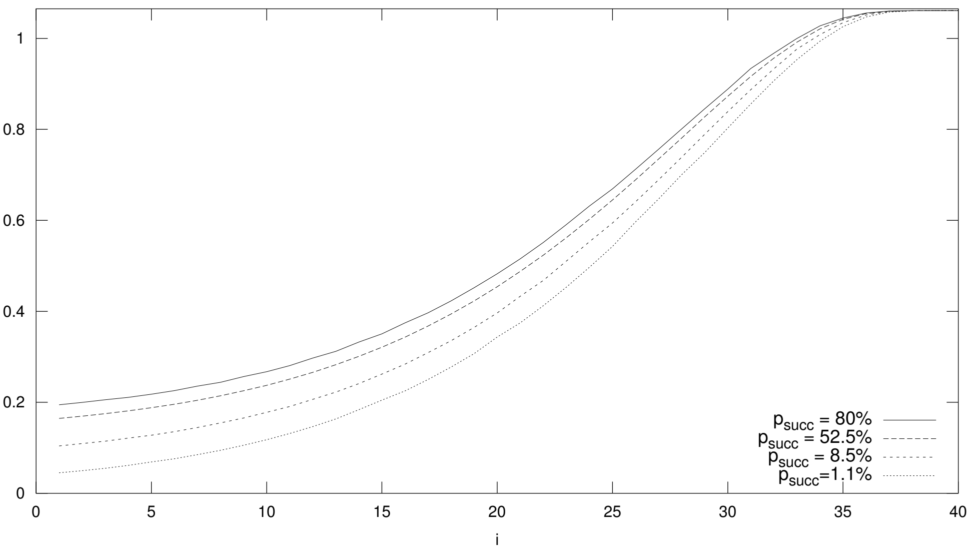


Figure 3.10: $\mathbf{R}(r, \mathbf{B}, p) \in \mathcal{R}_{n/2}$ with $n = 80$, $r = 1.03$, and different p.

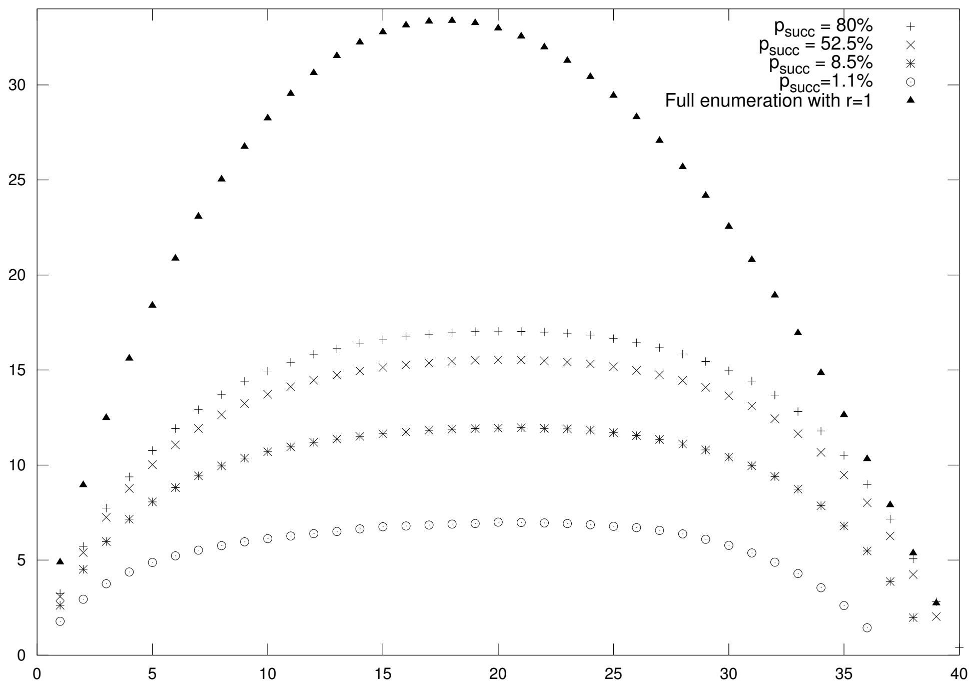


Figure 3.11: The number of nodes (N_i) in log basis for $\mathbf{R}(r, \mathbf{B}, p)$ with $r = 1.03$, and different p. For comparison, we put also the curve for naive enumeration with $r = 1$, which is considerable higher than any of the pruned enumeration.

- Another choice would be to compute optimal $\mathbf{R}$ for a fixed $T_{\text{enum}}(\mathbf{R}, \mathbf{B}) = T$ and r. In this way, the optimization problem becomes

$$\begin{aligned} \underset{R_{\text{odd}}(\mathbf{R}) \in \mathcal{R}_{\lceil n/2 \rceil}}{\text{maximize}} \quad & p_{\text{succ}}(r, \mathbf{R}) \\ \text{subject to} \quad & T_{\text{enum}}(\mathbf{R}, \mathbf{B}) = T \end{aligned} \tag{3.39}$$

  For both problems, we have to predefine p and r, or T and r, and it is not clear how to which values will lead to best efficiency of BKZ.

Similar to the optimization problem in (3.35), we name the solution to these problems $\mathbf{R}(r, \mathbf{B}, p)$ and $\mathbf{R}(r, \mathbf{B}, T)$ respectively.

3.5.1 Optimization problem with constraints

The optimization problems with bounds as in equation (3.38) and(3.39) are difficult to resolve in general, because we have to search in the subspace defined implicitly by equations $T_{\text{enum}}(\mathbf{R}, \mathbf{B}) = T$ or $p_{\text{succ}}(r, \mathbf{R}) = p$ respectively. We can use random search with relaxed constraint, in this way the search becomes even slower and we usually finish with a sub-optimal solution $\mathbf{R}$.

However, it is possible to find solutions to these problems by making several calls to the unbounded optimization problems of (3.35), by observing the following relationships between the unbounded optimization problems (equation 3.35), the optimization problem with constraint on r and p (equation 3.38), and the optimization problem with constraint on r and T (equation 3.39).

- If $\mathbf{R}(T_0, \mathbf{B}, r)$ is a solution to (3.35), then it is also the solution to (3.38) with r,$p = p_{\text{succ}}(\mathbf{R}(T_0, \mathbf{B}, r))$ and $\mathbf{B}$, and is also the solution to the problem (3.39) with r, $T = T_{\text{enum}}(\mathbf{R}(T_0, \mathbf{B}))$ and $\mathbf{B}$. This is to say, If we have an oracle to the solutions of optimization problem (3.35), then we also know the solution to problems (3.38) and (3.39) for certain T or p. By making several request for solutions to the problem (3.35), we hope that T or p will gradually approach our intended value.

- Moreover, as we mentioned before, since T_0 is a heuristic approximation to $T_{\text{enum}}(\mathbf{R}(T_0, \mathbf{B}, r))$, it is very likely that $\mathbf{R}(T_0, \mathbf{B}, r)$ is very close to the solution of problem (3.39) with $T = T_0$. This gives a good initialization value for searching.

We can use the following algorithm to solve problem (3.39) with an oracle to problem (3.35), as given in algorithm 17. It works in practice for most of the time, though there is no guarantee of termination in theory, and may diverge under certain cases.

Algorithm 17 Optimization for $\mathbf{R}(r, \mathbf{B}, T)$

```
Input: $(\mathbf{B}, T, r)$
Output: $\mathbf{R}$ with $T_{\text{enum}}(\mathbf{R}, \mathbf{B}) = T$ which maximizes $p_{\text{succ}}(\mathbf{R}, r)$.
1: $T' \leftarrow T$
2: $\mathbf{R} \leftarrow \mathbf{R}(T', \mathbf{B}, r)$
3: $\delta \leftarrow T_{\text{enum}}(\mathbf{R}, \mathbf{B}) - T$
4: while $\delta > \epsilon$ do
5:     $T' \leftarrow T' - \delta$
6:     $\mathbf{R} \leftarrow \mathbf{R}(T', \mathbf{B}, r)$
7:     $\delta \leftarrow T_{\text{enum}}(\mathbf{R}, \mathbf{B}) - T$
8: end while
```

3.5.2 Principal component analysis

In this subsection we introduce principal component analysis and apply it to the solutions to optimization problems. In fact, it is hard not to notice that the solutions to the optimization problems have very similar shapes. Although they all belong to $\mathcal{R}_{\lceil n/2 \rceil}$, we believe that they dwell in the some subspace of $\mathcal{R}_{\lceil n/2 \rceil}$ in with much smaller dimension than $\lceil n/2 \rceil$.

We will first find out the dimension of its subspace d, then try to build a model with d terms so that $\mathbf{R}$ can be expressed by linear combination of d known vectors.

Principal component analysis is a statistic tool which is commonly used in image processing for finding patterns in data of high dimension. Depending on the field of application, it has also a lot of other names. Here we borrow this concept because we are also in the situation where we want to identifying patterns in data, and highlight their differences and similarities. It is a non-parametric method, which can be viewed as an advantage since we do not have to provide any additional training information, or as an disadvantage, because once we obtain some observation and it is impossible to incorporate any assumption to the model. Therefore, PCA is only used here as a first step, to learn some important features of the data, such as the number of its main feature vectors, etc. Later we build our model with least square method.

PCA is defined as the process of orthogonal linear transformation which transforms the data $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ to a new coordinate system. In this new coordinate system, the first coordinate $\mathbf{w}_1$ (called the first principal component) represents the direction where $\mathbf{X}$ has greatest variance, and $\mathbf{w}_i$ is the direction where **X** has i-th greatest variance. The graphic representation of this process is illustrated in figure 3.12, which gives an example in 2 dimension.

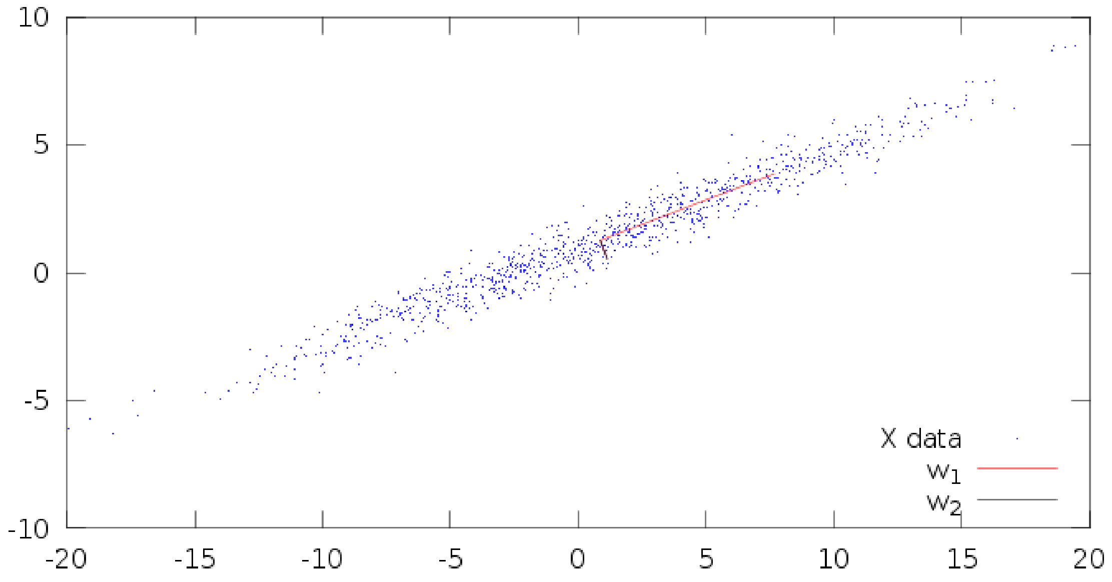


Figure 3.12: Illustration of PCA. Data points have greatest variation on direction $\mathbf{w}_1$, and second maximum on $\mathbf{w}_2$

More precisely, let **C** be the covariance matrix of data **X** of dimension m,

$$\mathbf{C} = cov(\mathbf{X})$$

and let $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$ be its eigenvector listed in the increasing order of corresponding eigenvalue. Then $\mathbf{w}_1$ corresponds to the first principal component, and $\mathbf{w}_i$ corresponds to the i-th principal component, so on.

The next step consists of selecting important principal components and leave out the unimportant ones. This is how dimension is compressed by PCA. Generally, the order of eigenvalue shows the order of significance of corresponding eigenvector, and one will pick up d vectors ($d < m$) which correspond to d largest eigenvalues. Later, all data points can be derived in this compressed d-dimension space:

$$\mathbf{x}_i \approx \mathbf{x}'_i = \bar{\mathbf{x}} + c_1\mathbf{w}_1 + c_2\mathbf{w}_2 + \cdots + c_d\mathbf{w}_d, \tag{3.40}$$

where $\bar{\mathbf{x}}$ is the mean over dataset X, and $c_i = \langle \mathbf{x}_i, \mathbf{w}_d \rangle$ is the length of the projection of $\mathbf{x}_i - \bar{\mathbf{x}}$ on direction $\mathbf{w}_i$.

The difference between $\mathbf{x}_i$ and $\mathbf{x}'_i$ is controlled in a statistical sense:

$$\mathbf{x}_i - \mathbf{x}'_i = c_{d+1}\mathbf{w}_{d+1} + c_{d+2}\mathbf{w}_{d+2} + \cdots + c_n\mathbf{w}_n, \tag{3.41}$$

where $c_{d+1}, \ldots, c_n$ are expected to be small, since $cov(X)$ has small eigenvalue for eigenvector $\mathbf{w}_{d+1}$.

3.5.3 Data acquisition and analysis

In order to apply PCA to our problem, we need abundant amount of solution bounding functions. This is done by solving a lot of instances of the optimization problem numerically using golden section search. Among the 3 parameters which determine $\mathbf{R}(T_0, \mathbf{B}, r)$, we start by fixing two of them and vary T_0.

We set $r = 1.05$ and $\mathbf{B} = \mathbf{B}_{70}$, where $\mathbf{B}_{70}$ is a typical BKZ-30 reduced basis of dimension 70, let T_0 be $\{30, 10, 3, 1, \ldots, 0.003, 0.001\}$. The illustration of the data is given in figure 3.8. Remember that the definition of $\mathbf{R}$ here is $R_{\text{odd}}(R_1^2, \ldots, R_n^2)$, here we are actually doing PCA and least square method over $\sqrt{\mathbf{R}(T_0, \mathbf{B}, r)}$. We call this data set $\mathcal{R}_{\mathbf{B}_{70},1.05}$ for short.

We compute the covariance matrix of the data set, and let $\mathbf{R}_1, \mathbf{R}_2, \ldots$ be the principal components listed in the order of importance. In addition, we note $\mathbf{R}_0$ to be the mean value over data set.

The 5 largest eigenvalues of the covariance matrix of $\mathcal{R}_{\mathbf{B}_{70},1.05}$ is

$$0.36, \qquad 1 \times 10^{-4}, \qquad 8 \times 10^{-8}, \qquad 7 \times 10^{-9}, \qquad 1 \times 10^{-9}.$$

After the second value, the following eigenvalues are almost negligible. Therefore we only select $\mathbf{R}_1$ and $\mathbf{R}_2$ as principal components.

Figure. 3.13 shows the principal components $\mathbf{R}_0, \mathbf{R}_1, \mathbf{R}_2$ for $\mathcal{R}_{\mathbf{B},1.05}$.

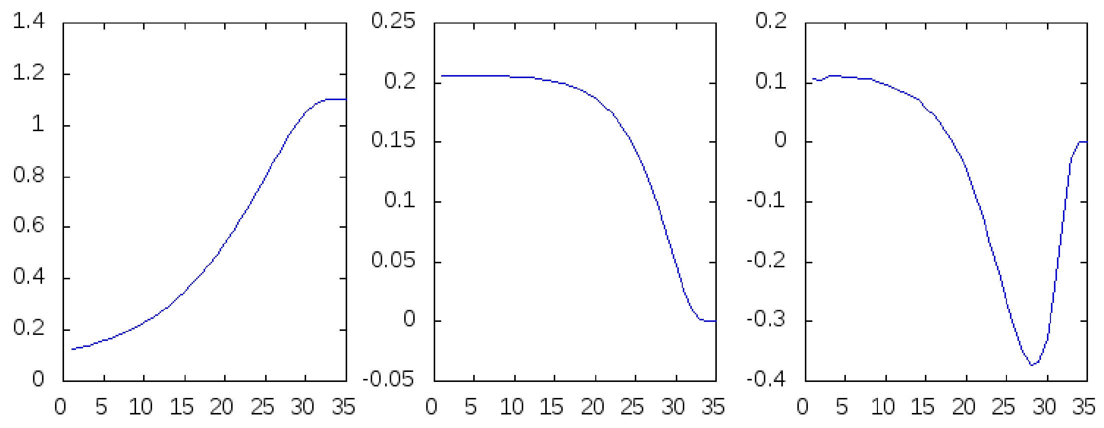


Figure 3.13: Principal components $\mathbf{R}_0, \mathbf{R}_1, \mathbf{R}_2$ for $\mathcal{R}_{\mathbf{B},1.05}$.

3.5.4 Parameterize bounding functions

PCA managed to extract from data the most important feature vectors. But it does not provide interpretations on the extracted data. Nor does it relate coefficient c the with the parameter T_0 for the solution. We compare the coefficients c_1 and c_2 of $\mathcal{R}_{\mathbf{B},1.05}$ with their corresponding $\ln(T_0)$ in figure 3.14 and figure 3.15.

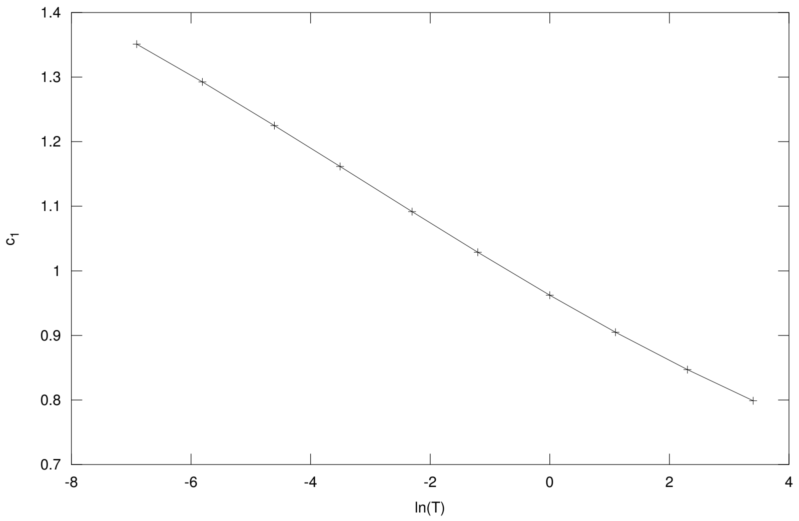


Figure 3.14: c_1 for Principal components $\mathbf{R}_0, \mathbf{R}_1, \mathbf{R}_2$ for $\mathcal{R}_{\mathbf{B},1.05}$.

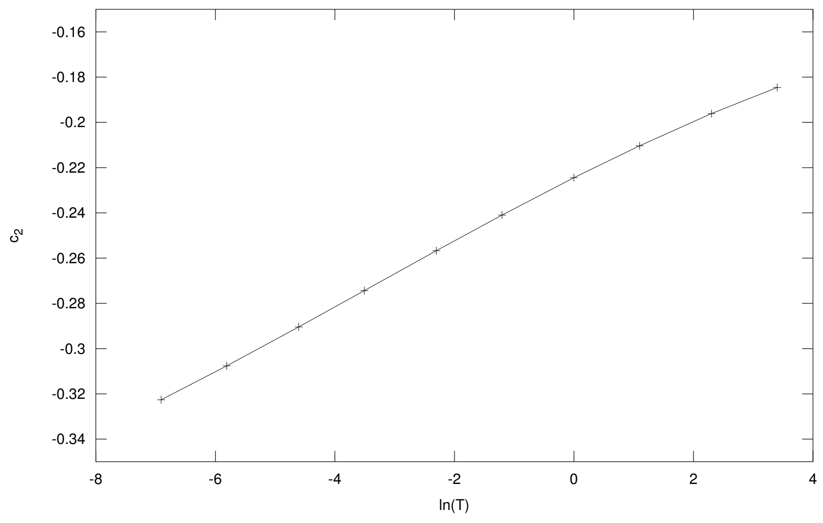


Figure 3.15: c_2 for Principal components $\mathbf{R}_0, \mathbf{R}_1, \mathbf{R}_2$ for $\mathcal{R}_{\mathbf{B},1.05}$.

Assume c_1 and c_2 to be functions of $\ln(T_0)$, then observation indicates that c_1 and c_2 are at most quadratic of $\ln(T_0)$, i.e. we expect that we can express vectors in $\mathcal{R}_{\mathbf{B},1.05}$ in the following form

$$\mathbf{R} = \mathbf{R}_0 + c_1(\ln(T_0)) \cdot \mathbf{R}_1 + c_2(\ln(T_0)) \cdot \mathbf{R}_2, \tag{3.42}$$

where $c_1(\cdot)$ and $c_2(\cdot)$ are polynomials of at most degree 2.

By collecting the linear and quadratic terms of $\ln(T_0)$, we can rewrite equation 3.42 as

$$\mathbf{R} = \mathbf{R}_0' + \ln(T_0) \cdot \mathbf{R}_1' + (\ln(T_0))^2 \cdot \mathbf{R}_2', \tag{3.43}$$

where $\mathbf{R}'_0$, $\mathbf{R}'_1$ and $\mathbf{R}'_2$ are linear combinations of $\mathbf{R}_0$, $\mathbf{R}_1$ and $\mathbf{R}_2$.

Notice that for equation 3.43, $\mathbf{R}'_0$ $\mathbf{R}'_1$ and $\mathbf{R}'_2$ can be computed by least square methods. In fact, equation 3.43 amounts to the following,

$$\begin{pmatrix} \mathbf{R}(T_1) \\ \mathbf{R}(T_2) \\ \vdots \\ \mathbf{R}(T_n) \end{pmatrix} = \begin{pmatrix} 1 & ln(T_1) & ln(T_1)^2 \\ 1 & ln(T_2) & ln(T_2)^2 \\ \vdots & & \\ 1 & ln(T_n) & ln(T_n)^2 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{R}_0 \\ \mathbf{R}_1 \\ \mathbf{R}_2 \end{pmatrix} \tag{3.44}$$

Therefore we have

$$\begin{pmatrix} \mathbf{R}_0 \\ \mathbf{R}_1 \\ \mathbf{R}_2 \end{pmatrix} = \begin{pmatrix} 1 & ln(T_1) & ln(T_1)^2 \\ 1 & ln(T_2) & ln(T_2)^2 \\ \vdots & & \\ 1 & ln(T_n) & ln(T_n)^2 \end{pmatrix}_{\text{left}}^{-1} \begin{pmatrix} \mathbf{R}(T_1) \\ \mathbf{R}(T_2) \\ \vdots \\ \mathbf{R}(T_n) \end{pmatrix} \tag{3.45}$$

where the A_{left}^{-1} is the generalized left inverse which is defined as $(A^T \cdot A)^{-1} A^T$.

3.5.5 Adding variable of r

We already know how to represent $\mathcal{R}_{\mathbf{B},1.05}$, and in this section we add the variable r into consideration. As before, we can first determine the number of principal component by PCA and then solve the model by least square method.

We still fix $\mathbf{B} = \mathbf{B}_{70}$, and let let T_0 be $\{30, 10, 3, 1, ..., 0.003, 0.001\}$, and r be $\{1, 1.01, 1.02, \ldots, 1.1\}$. Figure. 3.16 shows some examples of $\mathbf{R}$ for $T_0 = 1$ and different r.

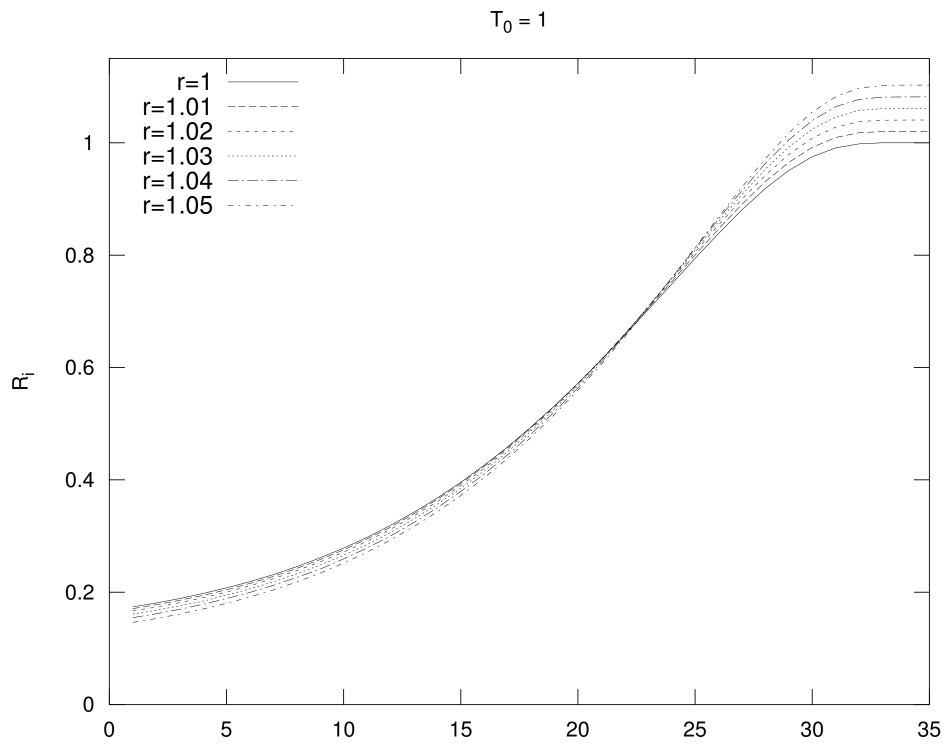


Figure 3.16: Example of $R_{\text{odd}}(\mathbf{R}(T_0, \mathbf{B}_{70}, r))$ with $T_0 = 1$ and different r.

For all solutions $\mathbf{R}(r)$, we have $R_n = r$. Therefore at least one component of $\mathbf{R}$ is linear in r. PCA shows that in this data set we have 3 significant components. Thus we should add a linear component $r \cdot \mathbf{R}_3$ into the extended model. As a result, we have the following model

$$\mathbf{R} = \mathbf{R}_0 + r \cdot \mathbf{R}_1 + \ln(T_0) \cdot \mathbf{R}_2 + \ln(T_0)^2 \cdot \mathbf{R}_3. \tag{3.46}$$

The parameters r is usually ≈ 1, therefore we substitute variable r with $(r-1)$.

$$\mathbf{R} = \mathbf{R}_0 + (r-1) \cdot \mathbf{R}_1 + \ln(T_0) \cdot \mathbf{R}_2 + \ln(T_0)^2 \cdot \mathbf{R}_3, \tag{3.47}$$

Then, $\mathbf{R}_0$, $\mathbf{R}_1$, $\mathbf{R}_2$ and $\mathbf{R}_3$ are computed from least square method. Their respective shapes are illustrated in figure 3.17. Notice that $\mathbf{R}_3$ curve is not as smooth as previous vectors. This may be because its small scale makes it especially vulnerable to quantification error.

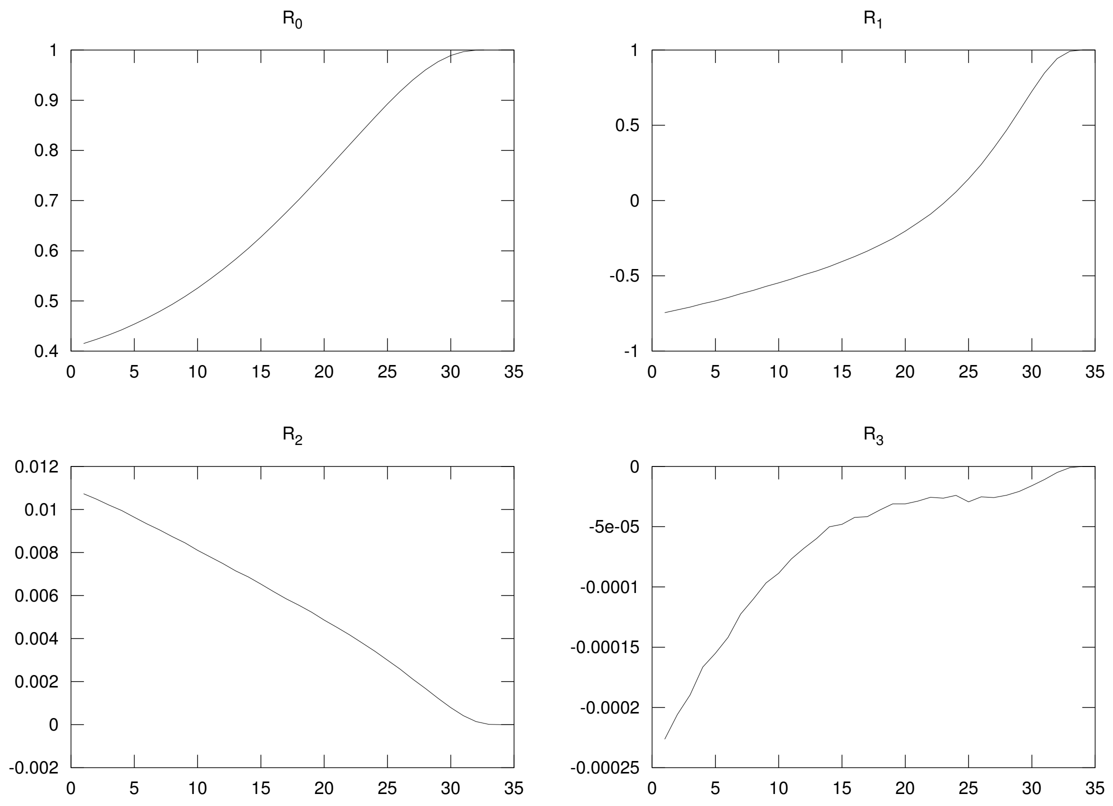


Figure 3.17: $\mathbf{R}_0$, $\mathbf{R}_1$, $\mathbf{R}_2$ and $\mathbf{R}_3$ for linearly reconstructing bounding functions in $\mathcal{R}_{\mathbf{B}_{70}}$

Finally, we are able to reconstruct $\mathbf{R}(T_0, \mathbf{B}_{70}, r)$ with this model. In theory, T_0 and r can take arbitrary meaningful value, but taking values in the interval $T_0 \in [0.001, 30]$ and $r \in [1, 1.1]$ has guaranteed stability.

Figure 3.18 shows a comparison between a reconstructed bounding function $\mathbf{R}(T_0, \mathbf{B}_{70}, r)$, and a bounding function given by numerically solving the optimization problem. Their difference is very small and hardly visible.

3.5.6 Consider different basis

Our discussion above have assumed a fixed basis $\mathbf{B}_{70}$, which is a typical BKZ-40 reduced basis of dimension 70. In reality, we may need to perform enumeration with different quality. And for different input basis, it is possible that the same bounding function will give different enumeration time. However, we would like to make following remarks:

- The bases may have different quality, which result in different enumeration cost. However, for these different basis, one bounding function provides similar proportion of speed-ups. This is because the enumeration tree has the biggest number of nodes around layer $\lfloor n/2 \rceil$, and a given bounding function always prunes $N_{\lfloor n/2 \rceil}$ by a fixed proportion.
- It is well known that the enumeration of a basis of bad quality can be made more efficient by performing a reasonable reduction prior to enumeration. This is discussed in a more rigorous sense

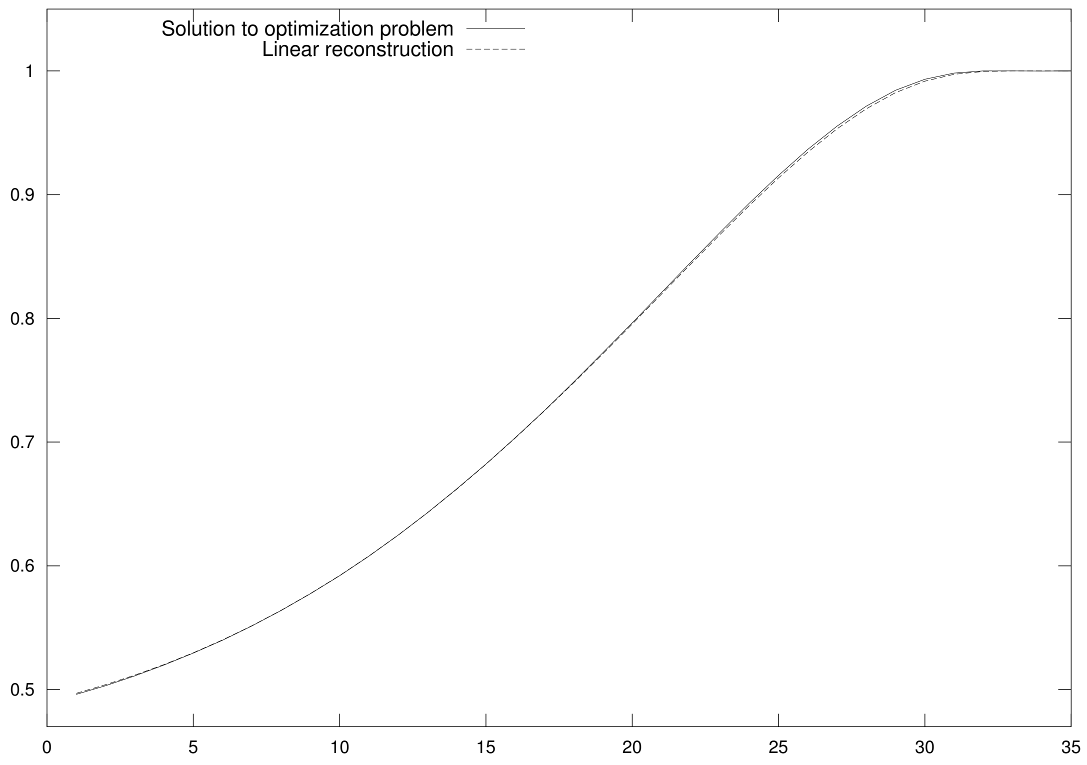


Figure 3.18: Comparison between optimization solution(solid line), and linear reconstruction(dashed line) of $\mathbf{R}(T_0 = 30, \mathbf{B}_{70}, r = 1)$

in section 5.1 for the discussion of *optimal reduction sequence*. The optimal strategy of reduction and enumeration will always assure that the enumeration with two of the three parameter given among r, p_{succ}, T, will accept a basis whose quality is not too bad, otherwise it would better to reduced a little more before enumeration.

Consequently, we do a change of variable to previous parameterization again. Because it is the final enumeration time T that is more interesting to us, not the "preprocessing" time T_0, we are motivated to replace $\ln(T_0)$ by a descriptor of log of pruned proportion of bounding function. This descriptor is defined as:

$$M = \max_{1 \leqslant i \leqslant n} \frac{N_i^{(full)}}{N_i^{(\mathbf{R})}}.$$

In this way, the pruned enumeration with $\mathbf{R}$ should be approximately 2^M times faster than the naive full enumeration with $r = 1$.

The modal is changed accordingly:

$$\mathbf{R} = \mathbf{R}_0 + (r-1) \cdot \mathbf{R}_1 + (M - n/4) \cdot \mathbf{R}_2 + (M - n/4)^2 \cdot \mathbf{R}_3, \tag{3.48}$$

here, we used the term $M - n/4$ instead of variable M because frequent choices of M are $\approx n/4$. Again $\mathbf{R}_0$, $\mathbf{R}_1$, $\mathbf{R}_2$ and $\mathbf{R}_3$ are computed from least square method. Their respective shapes are illustrated in figure **??**.

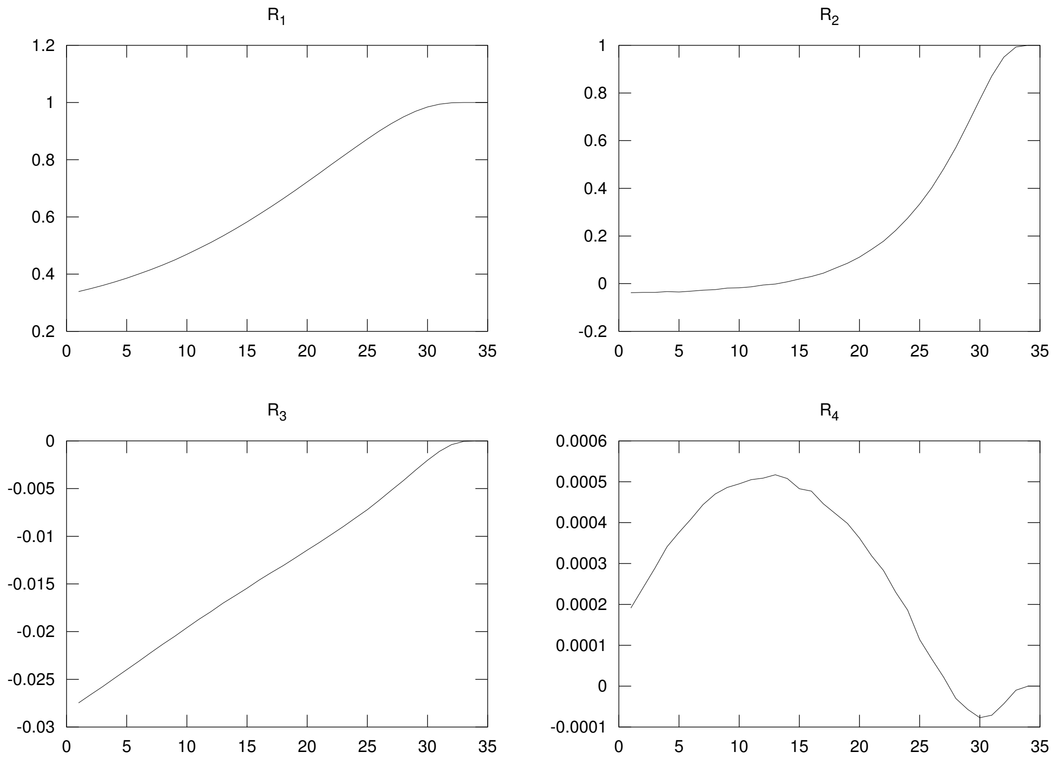


Figure 3.19: Comparison between optimization solution(solid line), and linear reconstruction(dashed line) of $\mathbf{R}(T_0 = 30, \mathbf{B}_{70}, r = 1)$

3.5.7 Extend model for different dimensions

Discussion in previous subsections applies for $\mathbf{R}$ with a fixed dimension 70. The same analysis is also applied to construct models for dimensions $n = 50, 60, 70, 80, 90, 110$. Using these models, we can conveniently compute an optimal bounding function in these dimensions.

For intermediate dimensions, we interpolate the models of the two neighboring dimension with known model. For example, we want to compute bounding function in dimension k, where $50 < k < 110$ is not a multiple of 10. But we have the model for dimension $k_1 = 10 \cdot \lceil k/10 \rceil$ and $k_2 = 10 \cdot \lfloor k/10 \rfloor$. We interpolate these two models to create a model for dimension k in the following way:

$$\mathbf{R}_i^{(k)} = \frac{k_1 - k}{k_1 - k_2} \cdot int(\mathbf{R}_i^{(k_2)}, k) + \frac{k - k_2}{k_1 - k_2} \cdot int(\mathbf{R}_i^{(k_1)}, k) \tag{3.49}$$

for $i = 1, 2, 3, 4$, and the funtion $int((R_1, \ldots, R_d), k)$ is defined as a vector of length k, and each of its i-th coordinate is

$$(\lceil j \rceil - j) R_{\lceil j-1 \rceil} + (j - \lceil j - 1 \rceil) R_{\lceil j \rceil}, \tag{3.50}$$

where $j = 1 + \frac{(d-1)(i-1)}{k-1}$.

CHAPTER 4

BKZ Reduction and Simulation

Contents

For many applications in cryptanalysis, BKZ algorithm is the most practical algorithm for lattice reduction in high dimension. It was proposed by Schnorr and Euchner in 1994 [79], in an application to solve subset sum problems. There was a minor improvement [81] with the notion of ”pruning”, but this pruning is not optimized, and there is no analysis of it. The first implementation is given as part of NTL library by Shoup [83,84]. However, its practical performance is little known.

In this chapter, we first give the basic algorithm description. The original BKZ reduction has a parameter β, namely, the blocksize. And it was considered that for $\beta > 25$ reduction would no longer be practical [23]. However, thanks to the idea of extreme pruning, we are able to go over this limit. We also present an analysis of BKZ, building upon the analysis of enumeration in the previous chapter. Based on this analysis, we propose a simulation algorithm which predicts the running time and quality of the output basis. When blocksize is big, say $\beta > 50$, the time spent in enumeration will dominate the time cost of BKZ algorithm, thus the running time can be approximately considered as the sum of enumeration time.

We also introduce many improvements to BKZ reduction, including the new enumeration routine introduced in chapter 3, applying preprocessing in each block prior to enumeration, and many others. The new BKZ 2.0 algorithm accepts blocksize as high as 90 or even more, making it the state-of-the-art [14]. The new algorithm needs more parameters for preprocessing and pruning. Building upon many experiments, we computed a comprehensive table consisting of optimal reduction parameters for blocksize < 90.

4.1 Original algorithm

4.1.1 Algorithm description

BKZ algorithm is an intermediate hierarchy between LLL and HKZ reduction in the following sense. It takes blocksize β as parameter. When $\beta = 2$, the output basis is LLL-reduced, whereas if $\beta = n$, the output basis would be HKZ-reduced. For all intermediate values, the bigger β is, the more reduced the output basis is, but also the longer time it takes.

One important characteristic of BKZ algorithm is that it runs by rounds. There is a sliding window of width β, which travels from beginning to the end. When it touches the end, this is the end of a round. It continues with a new round, until there is no more reduction in an entire round. The description in pseudo code is given in algorithm 18 and 19.

In `OneRoundBKZ`, the beginning and ending index of the sliding window j and k are increased in each iteration, until they reach the end n. Using enumeration we find the shortest vector in $\mathcal{L}(\mathcal{B}_{[j,k]})$. Let $\mathbf{v}$ be the integer coefficients of the basis vectors $(\mathbf{b}_j, \ldots, \mathbf{b}_k)$ such that $\pi_j(\mathbf{v} \cdot (\mathbf{b}_j, \ldots, \mathbf{b}_k))$ is the shortest vector in $\mathcal{L}(\mathbf{B}_{[j,k]})$. Let $\mathbf{b}_{new}$ denote the vector $\mathbf{v} \cdot (\mathbf{b}_j, \ldots, \mathbf{b}_k)$. Then, if $\mathbf{b}_{new} = \mathbf{b}_1$, then $\mathbf{b}_{new}$ is placed between $\mathbf{b}_{j-1}$ and $\mathbf{b}_j$ and a new basis is generated from this generating set, keeping $\mathbf{b}_{new}$ on the j-th place. Finally, LLL reduction is applied to the next block.

Upon termination of the algorithm, the output basis is guaranteed to LLL-β reduced, and satisfying the following condition:

$$\|\mathbf{b}_i^\star\| = \lambda_1(\mathcal{L}(\mathbf{B}_{[i,\min(i+\beta-1,n)]})) \tag{4.1}$$

4.1.2 Hermite factor of output basis

In this subsection we tried to establish an analysis based on average performance of BKZ on random basis.

Running time

No good upper bound on the complexity of BKZ is known. The best upper bound known for the number of calls (to the enumeration subroutine) is exponential (see [38]). However, as Hanrot, Pujol and Stehlé pointed out, after a polynomial number of calls to enumeration, the quality of the basis is already very close to the final output.

The cost of each enumeration is super-exponential in the blocksize β, therefore so is the BKZ algorithm.

Algorithm 18 Block Korkine-Zolotarev (BKZ) algorithm

Input: A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, a target Hermite factor δ_0 or target half volume η_0.
Output: The basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is BKZ-β reduced

```
1: $(\mathbf{B}, \mathbf{U}) \leftarrow$ LLL($\mathbf{b}_1, \ldots, \mathbf{b}_n, \mathbf{U}$);// LLL-reduce the basis, and obtain Gram-Schmidt Orthogonalisation $\mathbf{U}$
2: f$\leftarrow$ successful
3: while f=successful do
4:     $(\mathbf{B}, \mathbf{U}$,f$) \leftarrow$OneRoundBKZ$(\beta, \mathbf{B}, \mathbf{U})$
5: end while
```

Algorithm 19 `OneRoundBKZ`

```
Input: A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$,
    the Gram-Schmidt triangular matrix $\mathbf{U}$ and $\|\mathbf{b}_1^*\|^2, \ldots, \|\mathbf{b}_n^*\|^2$.
    a blocksize $\beta \in \{2, \ldots, n\}$, and enumeration radius $r$ and $T$.
Output: The basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ after one round of BKZ-$\beta$ reduction,
    and flag f marking if any enumeration had been successful during this round.
1: $j \leftarrow 0$;
2: f$\leftarrow$ unsuccessful;
3: while $j < n-1$ do
4:     $j \leftarrow (j \mod (n-1)) + 1$; $k \leftarrow \min(j+\beta-1, n)$; $h \leftarrow \min(k+1, n)$; // define the local block
5:     $\mathbf{v} \leftarrow$Enum($\mu_{[j,k]}, \|\mathbf{b}_j^*\|^2, \ldots, \|\mathbf{b}_k^*\|^2$); // find $\mathbf{v} = (v_j, \ldots, v_k) \in \mathbb{Z}^{k-j+1} - \mathbf{0}$ s.t. $\|\pi_j(\sum_{i=j}^k v_i \mathbf{b}_i)\| = \lambda_1(L_{[j,k]})$
6:     if $\mathbf{v} \neq (1, 0, \ldots, 0)$ then
7:         f$\leftarrow$ successful;
8:         LLL($\mathbf{b}_1, \ldots, \sum_{i=j}^k v_i \mathbf{b}_i, \mathbf{b}_j, \ldots, \mathbf{b}_h, \mathbf{U}$) at stage $j$; //insert the new vector in the lattice at the start of the current block, then remove the dependency in the current block, update $\mu$.
9:     else
10:        LLL($\mathbf{b}_1, \ldots, \mathbf{b}_h, \mathbf{U}$) at stage $h-1$; // LLL-reduce the next block before enumeration.
11:    end if
12: end while
```

Upper bound for output quality

As for the output quality of the basis, in the original work of Schnorr and Euchner [79], they proved that for BKZ-β reduced basis, $\|\mathbf{b}_1\|$ has an upper bound given by:

$$\|\mathbf{b}_1\| < \gamma_\beta^{\frac{n-1}{\beta-1}} \cdot vol(\mathcal{L})^{1/n}$$

The study by Gama and Nguyen also pointed out that with the same technique, we are able to prove an upper bound of

$$\|\mathbf{b}_1\| < \sqrt{\gamma_\beta}^{1+\frac{n-1}{\beta-1}} \cdot vol(\mathcal{L})^{1/n}$$

In practice, the quality of bases output by BKZ is better than the best theoretical worst-case bounds: according to [23], the Hermite factor for n-dimensional BKZ-β reduced lattices $\delta(\mathbf{B})$ seems to quickly converge as n grows to infinity, whereas theoretical upper bounds are $c'(\beta)$ is significantly larger than $c(\beta, n)$. For instance, $c(20, n) \approx 1.0128$ for large n.

By the end of this subsection, we will list all $\lim_{n\to+\infty} c(\beta, n)$ for $50 < \beta < 1000$. Especially, we will prove the following

$$\lim_{n\to+\infty} c(\beta, n) = \left(\frac{\beta}{2\pi e}(\pi\beta)^{\frac{1}{\beta}}\right)^{\frac{1}{2(\beta-1)}}. \tag{4.2}$$

Reduction blocks and random lattices

To predict the output quality of a BKZ-reduced basis, we need some assumption on each block of reduction.

Intuitively, the first minimum of most local blocks looks like that of a random lattice of dimension the blocksize: this phenomenon does not hold in small blocksize ≤ 30 (as noted by Gama and Nguyen [23]), but it becomes more and more true as the blocksize increases, as shown in Fig. 4.1, where we see that the expectation and the standard deviation of $\frac{\lambda_1(L)}{GH(L)}$ seem to converge to that of a random lattice. Intuitively,

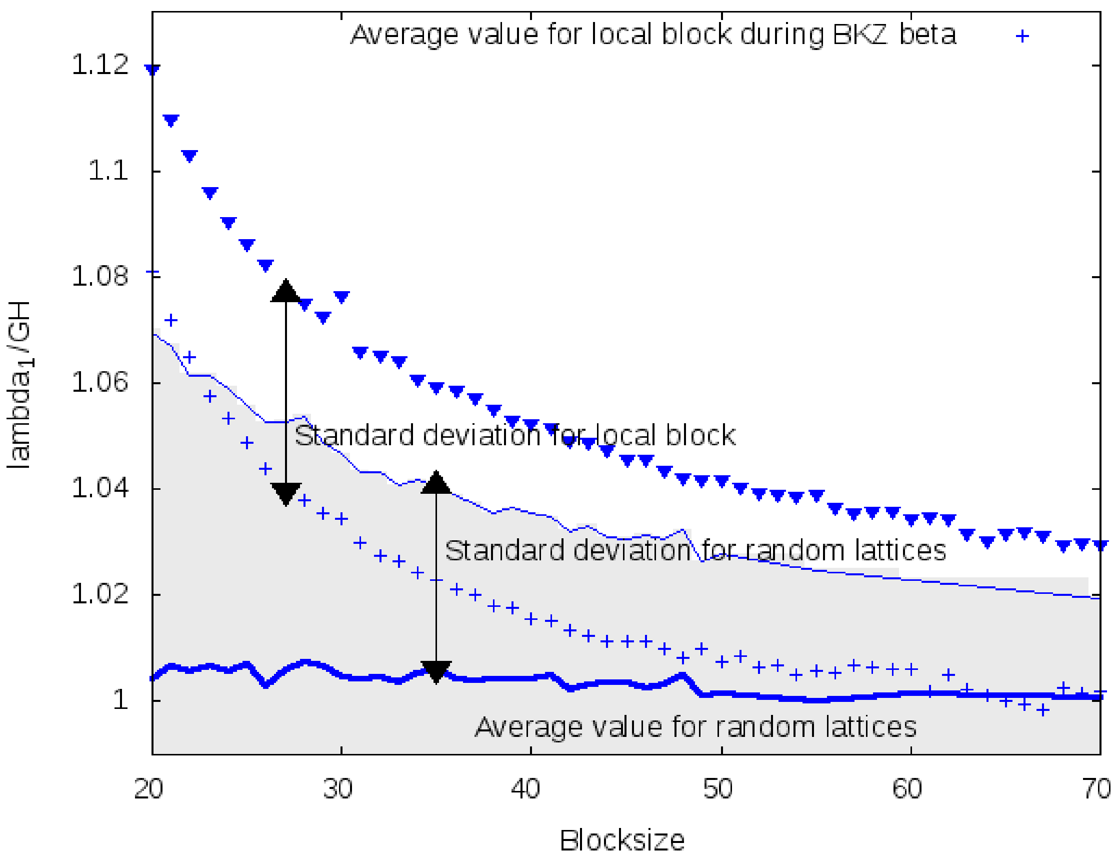


Figure 4.1: Comparing $\frac{\lambda_1(L)}{GH(L)}$ for a non-extreme local block during BKZ-β reduction, with a random lattice of dimension β. Expectations with and without standard deviation are given.

this may be explained by a concentration phenomenon: as the dimension increases, random lattices dominate in the set of lattices, so unless there is a strong reason why a given lattice cannot be random, we may heuristically assume that it behaves like a random lattice.

Inductive analysis on Hermite factor

Let $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be the output basis of BKZ algorithm, and $\{\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*\}$ be its Gram-Schmidt orthogonalized basis. We also note $l_i = \log(\|\mathbf{b}_i^*\|)$, $L_{[i,j]} = \log(vol(\mathcal{L}(\mathbf{B}_{[i,j]})))$ and $L_i = \log(vol(\mathcal{L}(\mathbf{B}_{[i,n]})))$ for simplicity.

For convenience we also define following values:

- **Local hermit factor** $h_i, \forall 1 \leqslant i \leqslant n$. It is the log of root Hermite factor for each block $\mathbf{B}_{[i,j]}$.

$$h_i = \log(\|\mathbf{b}_i^*\|) - \frac{1}{\min(\beta, n-i+1)} L_{[i,\min(i+\beta-1,n)]}.$$

- **Global hermit factor** $g_i, \forall 1 \leqslant i < n$. It is the log of root Hermite factor for each block $\mathbf{B}_{[i,n]}$.

$$
\begin{aligned}
g_i &= \frac{\beta-1}{n-i}\left(\log(\|\mathbf{b}_i^*\|) - \frac{1}{n-i+1}L_{[i,n]}\right) \\
&= \frac{\beta-1}{n-i+1}\cdot l_i - \frac{\beta-1}{(n-i)(n-i+1)}\cdot \sum_{j=i+1}^{n} l_j
\end{aligned}
$$

Local Hermit factor h_i is directly linked with our assumptions over enumerations, while g_1 the global Hermite factor is what we are attempting to analysis. Ultimately, the root Hermite factor of the whole basis is

$$
\delta(\mathbf{B}) = \left(\|\mathbf{b}_1\|/vol(L)^{1/n}\right)^{1/n} = \exp(g_1)^{\frac{n-1}{n(\beta-1)}}. \tag{4.3}
$$

In the following, we trying to build a relationship between g_1 and h_i.

Lemma 4.1.1. *We can calculate g_i from h_i from induction by the following relationship:*

$$
g_i = \begin{cases} \frac{\beta-1}{n-i}h_i & n-\beta+1 \leqslant i < n \\ \frac{\beta}{n-i+1}h_i + \frac{n-i-\beta+1}{n-i+1}\cdot\frac{\sum_{j=1}^{\beta-1} g_{i+j}}{\beta-1} & 1 \leqslant i \leqslant n-\beta \end{cases}
$$

Proof. BKZ reduced basis exhibit an inductive structure, such that if $\mathbf{B}$ is BKZ-reduced, then $\mathbf{B}_[i,n], \forall i$ are reduced too. Our analyses are based on this induction. For $n-\beta+1 \leqslant i \leqslant n$, the blocks span between indices i and n, therefore $g_i = (\beta-1)/(n-i)\cdot h_i$. For $1 \leqslant i \leqslant n-\beta$, the blocks span between i and $i+\beta$, and g_i have to be induced from h_i and $g_{i+1}, \ldots, g_{i+\beta-1}$.

Notice h_i and $g_i, g_{i+1}, \ldots, g_{i+\beta-1}$ satisfy following relationship:

$$
\begin{pmatrix} h_i \\ g_i \\ g_{i+1} \\ \vdots \\ g_{i+\beta-1} \end{pmatrix} = \begin{pmatrix} \frac{\beta-1}{\beta} & -\frac{1}{\beta} & \cdots & -\frac{1}{\beta} & 0 & \cdots \\ \frac{\beta-1}{n-i+1} & -\frac{\beta-1}{(n-i)(n-i+1)} & -\frac{\beta-1}{(n-i)(n-i+1)} & & \cdots & -\frac{\beta-1}{(n-i)(n-i+1)} \\ & \frac{\beta-1}{n-i} & -\frac{\beta-1}{(n-i-1)(n-i)} & & \cdots & -\frac{\beta-1}{(n-i-1)(n-i)} \\ & & \ddots & & & \\ & & & \frac{\beta-1}{n-i-\beta+1} & & -\frac{\beta-1}{(n-i-\beta)(n-i-\beta+1)} \end{pmatrix} \begin{pmatrix} l_i \\ l_{i+1} \\ \vdots \\ l_{i+\beta-1} \\ \vdots \\ l_{i+n} \end{pmatrix} \tag{4.4}
$$

Assume $c_0, c_{i+1}, \ldots, c_{i+\beta-1}$ to be the coefficients of induction so that the induction is of the following form:

$$
g_i = c_0 \cdot h_i + \sum_{j=1}^{\beta-1} c_{i+j}\cdot g_{i+j}
$$

which is equivalent to

$$
g_i = \begin{pmatrix} c_0 & c_{i+1} & \ldots, c_{i+\beta-1} \end{pmatrix} \cdot \begin{pmatrix} h_i \\ g_{i+1} \\ \vdots \\ g_{i+\beta-1} \end{pmatrix}
$$

We now compute $c_0, c_{i+1}, \ldots, c_{i+\beta-1}$. They are the solutions of the following system

$$\begin{pmatrix}\frac{\beta-1}{n-i+1} & -\frac{\beta-1}{(n-i)(n-i+1)} & -\frac{\beta-1}{(n-i)(n-i+1)} & \cdots & -\frac{\beta-1}{(n-i)(n-i+1)}\end{pmatrix}$$
$$= \begin{pmatrix}c_0 & c_{i+1} & \dots & c_{i+\beta-1}\end{pmatrix}\begin{pmatrix}\frac{\beta-1}{\beta} & -\frac{1}{\beta} & \cdots & & -\frac{1}{\beta} \\ & \frac{\beta-1}{n-i} & -\frac{\beta-1}{(n-i-1)(n-i)} & \cdots & -\frac{\beta-1}{(n-i-1)(n-i)} \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \frac{\beta-1}{n-i-\beta+1}\end{pmatrix}$$

This is a triangular system, which can be solved by forward substitution. Upon some computation, we obtain the solution as follows.

$$\begin{pmatrix}c_0 \\ c_{i+1} \\ c_{i+2} \\ \vdots \\ c_{i+\beta-1}\end{pmatrix} = \begin{pmatrix}\frac{\beta}{n-i+1} \\ \frac{n-i-\beta+1}{(\beta-1)(n-i+1)} \\ \frac{n-i-\beta+1}{(\beta-1)(n-i+1)} \\ \vdots \\ \frac{n-i-\beta+1}{(\beta-1)(n-i+1)}\end{pmatrix} \tag{4.5}$$

Which proves the induction formula in the lemma. □

By developing this induction and collecting the terms of h_i, we can write g_1 as linear combination of h_i's,

$$g_1 = \sum_{i=1}^{n-1} k_i h_i \tag{4.6}$$

And we note the sum of the coefficients k_i to be K.

$$K = \sum_{i=1}^{n-1} k_i. \tag{4.7}$$

In the following lemma, we study the properties of k_i, which describes the contribution of h_i to g_1.

Lemma 4.1.2. *We have the following property for k_i.*

1. $k_1 = \beta/n$.

2. $\forall i > 1, k_i < e \cdot \frac{\max(\beta, n-\beta)}{n(n-1)}$. *Particularly,* $k_i < e/n$.

3. $1 \leqslant K < 1 + \frac{\beta^2/4}{n^2}$. *The lower bound is reached when $n = \beta$. In addition, there exists a constant $c \approx 5.96$ such that* $\lim_{n\to\infty} \frac{K-1}{1/n^2} \to c$.

Proof. By developing the induction, we discuss k_i in two cases:

- If $2\beta \leqslant n$, k_i is given by following induction,

$$
\begin{aligned}
k_1 &= \beta/n \\
k_2 &= \frac{\beta(n-\beta)}{n(\beta-1)(n-1)} \\
&\vdots \\
\forall i, 2 < i \leqslant \beta, \quad k_i &= k_{i-1} \cdot \frac{\beta}{\beta-1} \\
&\vdots \\
\forall i, \beta+1 \leqslant i \leqslant n-\beta+1, \quad k_i &= k_{i-1} \cdot \frac{\beta}{\beta-1} - k_{i-\beta} \cdot \frac{1}{\beta-1} \\
&\vdots \\
\forall i, n-\beta+2 \leqslant i \leqslant n-1, \quad k_i &= k_{i-1} \cdot \frac{n-i}{n-i+1} - k_{i-\beta} \cdot \frac{n-i}{\beta(n-i+1)}
\end{aligned}
$$

- If $2\beta > n$, k_i is given by following induction,

$$
\begin{aligned}
k_1 &= \beta/n \\
k_2 &= \frac{\beta(n-\beta)}{n(\beta-1)(n-1)} \\
&\vdots \\
\forall i, 2 < i \leqslant n-\beta+1, \quad k_i &= k_{i-1} \cdot \frac{\beta}{\beta-1} \\
&\vdots \\
\forall i, n-\beta+2 \leqslant i \leqslant \beta, \quad k_i &= k_{i-1} \cdot \frac{n-i+1}{n-i} \\
&\vdots \\
\forall i, \beta+1 \leqslant i \leqslant n-1, \quad k_i &= k_{i-1} \cdot \frac{n-i}{n-i+1} - k_{i-\beta} \cdot \frac{n-i}{\beta(n-i+1)}
\end{aligned}
$$

Here, $\max\limits_{i>1} k_i$ is reached by k_β: when $n \leqslant 2\beta$,

$$
\begin{aligned}
k_\beta &= \left(\frac{\beta}{\beta-1}\right)^{\beta-1} \cdot \frac{n-\beta}{n(n-1)} \\
&= \left(1 + \frac{1}{\beta-1}\right)^{\beta-1} \cdot \frac{n-\beta}{n(n-1)} \\
&< e \cdot \frac{n-\beta}{n(n-1)}
\end{aligned}
$$

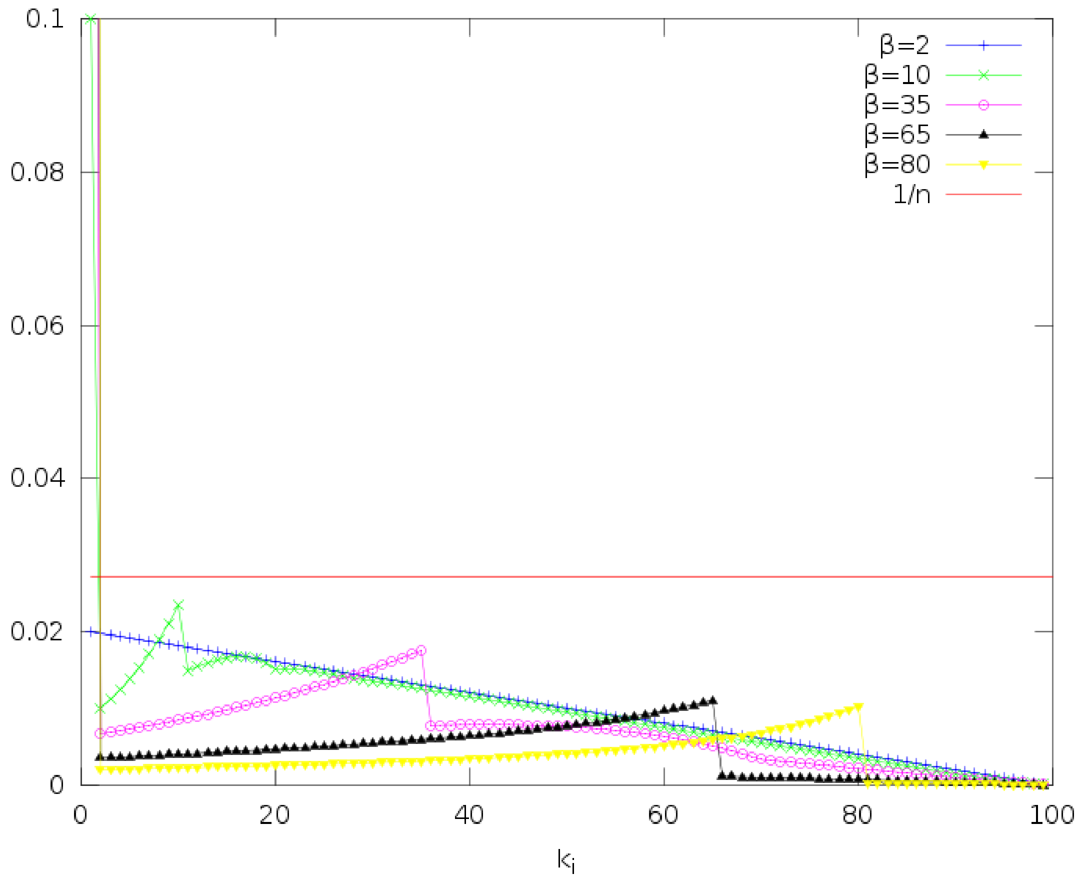
when $n < 2\beta$,

$$\begin{aligned} k_\beta &= \left(\frac{\beta}{\beta-1}\right)^{n-\beta} \cdot \frac{\beta}{n(n-1)} \\ &= \left(1+\frac{1}{\beta-1}\right)^{n-\beta} \cdot \frac{\beta}{n(n-1)} \\ &\leqslant \left(1+\frac{1}{\beta-1}\right)^{\beta-1} \cdot \frac{\beta}{n(n-1)} \\ &< e \cdot \frac{\beta}{n(n-1)} \end{aligned}$$

To conclude, $k_i < e \cdot \frac{\max(\beta, n-\beta)}{n(n-1)} \leqslant e/n$. □

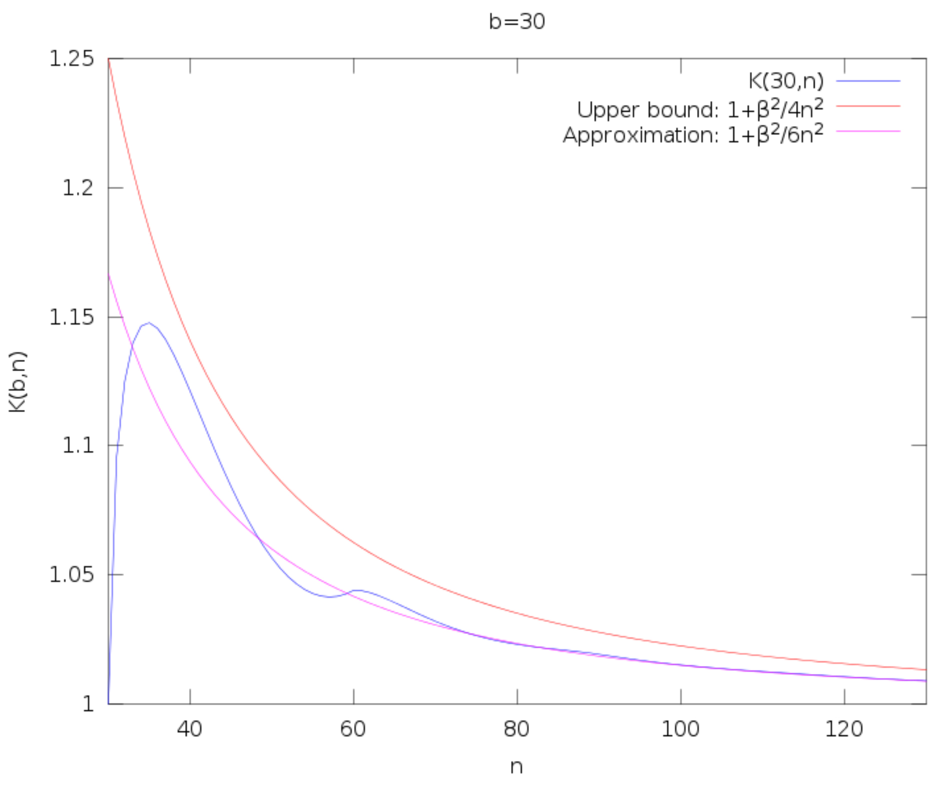
Following illustrates $k_i, i > 1$ for fixed n and varying β, $(n > 2\beta)$. The sequence is upper bounded by $\frac{n-i}{n(n-1)}$.

Figure 4.2: $k_i, i > 1$ for $n = 100$ and $\beta = 2, 10, 35, 65, 80$



And we have $\lim_{n\to\infty} K = 1$. More precisely, we have $K = 1 + O(\beta^2/n^2)$, and following figure illustrates the speed of convergence of K with fixed $\beta = 30$ and increasing n.

Figure 4.3: $K(\beta, n)$ for $\beta = 30$ and n from 30 to 130



Corollary 4.1.3. *If for $1 < i < n-1$, h_i's are i.i.d variables with $E(h_i) = \bar{h}$ and $Var(h_i) = \sigma^2$, then $E(g_1) = \frac{\beta}{n} \cdot h_1 + (K - \frac{\beta}{n})\bar{h}$, and $Var(g_1) < \frac{e^2}{n}\sigma^2$.*

Remark. *Here is some remarks about this corollary:*

- *h_1 has a much greater influence on g_1 than any other h_i:*
- *$E(g_1) = K\bar{h}$.*
- *Assuming Gaussian heuristic, $\bar{h} \approx v_\beta^{-1/\beta}$, using Stirling approximation, we have*

$$\lim_{n\to\infty} E(g_1) = \bar{h} = v_n^{-1/n} \approx \sqrt{\frac{\beta}{2\pi e}} \cdot (\pi\beta)^{\frac{1}{2\beta}}.$$

In other words, for a BKZ-β reduced basis of random lattices, we have

$$\lim_{n\to\infty} \delta(\mathbf{B}) = \lim_{n\to\infty} \left(\frac{\|\mathbf{b}_1\|}{vol(\mathcal{L})^{1/n}}\right)^{1/n} = \left(\frac{\beta}{2\pi e} \cdot (\pi\beta)^{\frac{1}{\beta}}\right)^{\frac{1}{2(\beta-1)}}. \tag{4.8}$$

- *If for each enumeration, the returned vector is the solution to* HSVP$_\gamma$*, which means $\bar{h} \approx \gamma \cdot v_\beta^{-1/\beta}$, then similarly we have*

$$\lim_{n\to\infty} \delta(\mathbf{B}) = \left(\frac{\beta\gamma^2}{2\pi e} \cdot (\pi\beta)^{\frac{1}{\beta}}\right)^{\frac{1}{2(\beta-1)}}. \tag{4.9}$$

This limit root Hermite factor for β between 50 and 1000 is illustrated in table 4.1 and figure 4.4.

Figure 4.4: Root Hermite factor for typical BKZ-β reduced basis ($50 \leqslant \beta \leqslant 1000$)

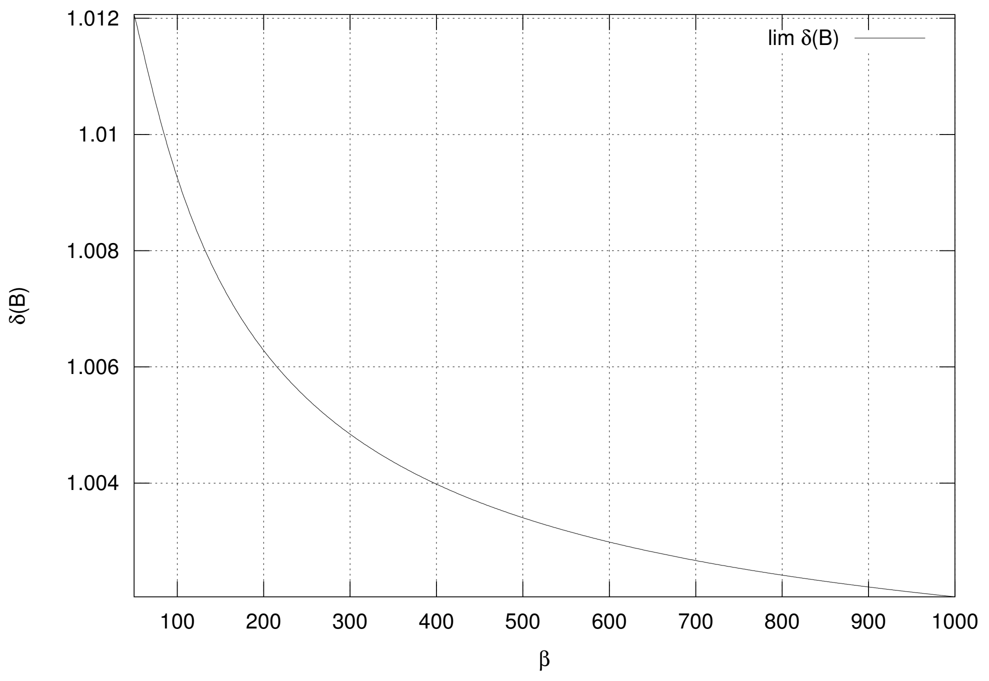


Table 4.1: Root Hermite factor for typical BKZ-β reduced basis ($50 \leqslant \beta \leqslant 1000$)

β	50	60	70	80	90	100	110
$\eta(\mathbf{B})$	1.0121	1.0115	1.0108	1.0103	1.0097	1.0093	1.0088
β	120	130	140	150	160	170	180
$\eta(\mathbf{B})$	1.0084	1.0081	1.0078	1.0075	1.0072	1.0067	1.0065
β	190	200	210	220	230	240	250
$\eta(\mathbf{B})$	1.0065	1.0063	1.0061	1.0059	1.0058	1.0056	1.0055
β	300	400	500	600	700	800	1000
$\eta(\mathbf{B})$	1.0048	1.0040	1.0034	1.0030	1.0027	1.0024	1.0020

4.2 Simulate reduction procedure

The worst-case analysis of running time is given by Hanrot et al [38], which also gives an asymptotic on the speed of convergence of the output quality of the basis. In contrast, this section is devoted to the analysis of reduction procedure on average.

BKZ reduction is naturally separated into several rounds. The goal of our simulation algorithm is to predict the Gram-Schmidt sequence $(\|\mathbf{b}_1^\star\|, \|\mathbf{b}_2^\star\|, \ldots, \|\mathbf{b}_n^\star\|)$ during the execution of BKZ, more precisely at the beginning of every round. One round of BKZ costs essentially $n-1$ enumeration calls.

We assume that the input basis is a "random" reduced basis, without special property. Once we can

```
Algorithm 20 SimulateBKZ1
Input: The Gram-Schmidt norms, given as $\ell_i = \log(\|\mathbf{b}_i^\star\|)$, for $i = 1, \ldots, n$,
    a blocksize $\beta \in \{50, \ldots, n\}$.
Output: A prediction for the Gram-Schmidt norms $\ell'_i = \log(\|\mathbf{b}_i^\star\|), i = 1, \ldots, n$, after one round of BKZ
    reduction.
```

predict the value of $\lambda_1(L_{[j,k]})$ for each local block, we know that this will be the new value of $\|\mathbf{b}_j^\star\|$ by definition of the enumeration subroutine, with the only exception if the first vector is already shorter than $\|\mathbf{b}_j^\star\|$. This could happen when, for example, the value of β in simulation is not big enough, and the basis already had higher quality than can be provided by BKZ-β. Knowing the value of new $\|\mathbf{b}_j^\star\|$ allows to deduce the volume of the next local block, and therefore iterate the process until the end of the round. This gives rise to our simulation algorithm (see Alg. 20).

```
Algorithm 21 SimulateOneRoundBKZ1
Input: The Gram-Schmidt norms, given as $\ell_i = \log(\|\mathbf{b}_i^\star\|)$, for $i = 1, \ldots, n$,
    a blocksize $\beta \in \{50, \ldots, n\}$.
Output: A prediction for the Gram-Schmidt norms $\ell'_i = \log(\|\mathbf{b}_i^\star\|), i = 1, \ldots, n$.
1: for $k = 1$ to $n$ do
2:     $f \leftarrow \min(k + \beta, n)$ //End index of local block
3:     $\log V \leftarrow \sum_{i=1}^{f} \ell_i - \sum_{i=1}^{k-1} \ell'_i$
4:     $\ell'_k \leftarrow$ prediction of $\lambda_1(\mathbf{B}_{[k,f]})$, based on $\log(V)$.
5: end for
```

To start with, we make the assumptions that $\beta \geqslant 50$. This allows to approximate each block as random basis.

We predict this first minimum $\lambda_1(L_{[j,k]})$ as follows:

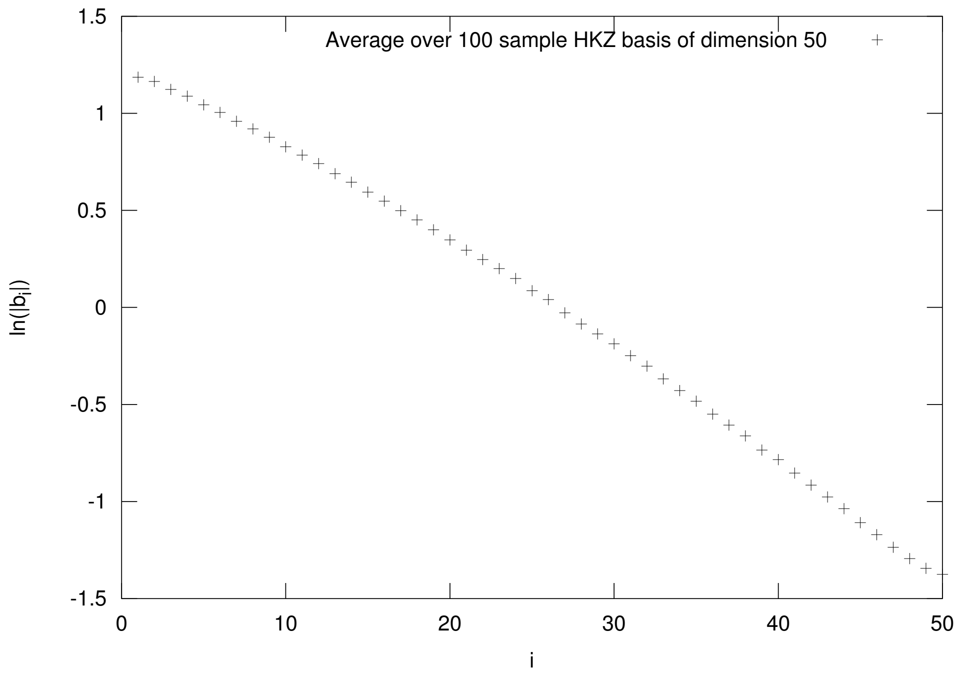
- For most indexes j, we choose $GH(\mathbf{B}_{[j,k]})$, unless $\|\mathbf{b}_j^\star\|$ was already better and $\|\mathbf{b}_1^\star\|, \ldots, \|\mathbf{b}_{j-1}^\star\|$ had not been changed in the current round.

- However, for the last indices j, namely those inside the last 50-dimensional block $L_{[n-49,n]}$, we do something different: since this last block will be HKZ-reduced at the end of the round, we assume that it behaves like an HKZ-reduced basis of a random lattice of the same volume. To simplify, we determine the last 50 Gram-Schmidt norms from the average Gram-Schmidt norms of an HKZ-reduced basis of a random 50-dimensional lattice of unit volume: these average norms were computed experimentally and are displayed in Fig. 4.5.

Here, if we use the prediction of Gaussian heuristic of λ_1 value for each block, we would need the assumption that the basis is not yet BKZ-β reduced. Because for each block, we suppose that its first vector is longer than the prediction by Gaussian heuristic and thus will always be updated by the enumeration. But this assumption is not always true. Especially, if the input basis is already much better reduced, this simulation will output a worse basis, while in actual reduction, this never happens.

In practice, we already know the length of the first vector of each block $\mathbf{b}_j^\star$. This may affect our prediction using Gaussian heuristic. For example, consider the case where $\|\mathbf{b}_j^\star\|$ is already shorter than the prediction given by Gaussian heuristic. Especially, we want to distinguish the case if a shorter vector than $\mathbf{b}_j^\star$ is found and the case if $\mathbf{b}_j^\star$ is unchanged. In the latter case, the rest of the basis is unchanged after this operation. Therefore we set our prediction of the shortest length to be $\min(\|\mathbf{b}_j^\star\|, GH)$. If for

Figure 4.5: Typical 50-dimensional HKZ reduced basis with unitary volume



enumeration of blocks 1 to $j-1$, $\mathbf{b}_1^\star, \ldots, \mathbf{b}_{j-1}^\star$ is not changed, then we know that $\mathbf{b}_j^\star$ is not modified since the beginning of the round, and if $\mathbf{b}_j^\star < GH$, the enumeration will return $\mathbf{b}_j^\star$ as the shortest vector, keeping the rest of the basis still unchanged. We use the flag ϕ to mark if $\mathbf{b}_1^\star, \ldots, \mathbf{b}_{j-1}^\star$ is unchanged in the current round. When ϕ is true, the comparison between $\|\mathbf{b}_j^\star\|$ and GH is used to predict λ_1 of the new block and ϕ is modified only when $\|\mathbf{b}_j^\star\| < GH$. The simulation is described in algorithm 22.

4.3 BKZ with pruned enumeration

An important improvement to BKZ comes from the improvement in enumeration. As presented in chapter 3, the enumeration process can be greatly sped up by using a pruning. Besides, instead of insisting on successfully finding λ_1, in BKZ we are not obliged to succeed in finding $\mathbf{b}_j^\star = \lambda_1(\mathbf{B}_{[j,\min n,j+\beta]})$. As long as any vector is renewed in a round, the basis is improved and the next round will take place to further improve the basis. This is to say, we are free to choose p, the probability of success in enumerating short vector of each block. Finally, we are interested in finding any vector that is shorter than current $\mathbf{b}_j^\star$, which means that instead of setting the goal vector to be of length $GH(\mathbf{B}_{[j,\min(j+\beta,n)]})$, there can be a relaxing factor of γ to the goal of enumeration $\gamma \cdot GH(\mathbf{B}_{[j,\min(j+\beta,n)]})$.

This new BKZ algorithm takes parameters β, p and γ. It needs a slightly different enumeration procedure, because we need to call many pruned enumeration for one block. In addition, the simulation of this reduction is not easy, because the enumeration is not probabilistic.

4.3.1 Probabilistic algorithm with parallel enumeration

The enumeration can be conveniently described as a main process calling for many children enumeration process. Each child process randomizes the basis independently and does an enumeration with

Algorithm 22 `SimulateOneRoundBKZ2`

```
Input: The Gram-Schmidt norms, given as $\ell_i = \log(\|\mathbf{b}_i^\star\|)$, for $i = 1, \dots, n$,
    a blocksize $\beta \in \{50, \dots, n\}$.
Output: A prediction for the Gram-Schmidt norms $\ell_i' = \log(\|\mathbf{b}_i^\star\|), i = 1, \dots, n$.
 1: $(r_1, \dots, r_{50}) \leftarrow$ average $\log(\|\mathbf{b}_1^\star\|, \dots, \|\mathbf{b}_{50}^\star\|)$ of an HKZ-reduced random unit-volume 50-dim lattice
 2: $\phi \leftarrow$ true //flag to mark whether $L_{[k,n]}$ has changed
 3: for $k = 1$ to $n - 50$ do
 4:   $f \leftarrow \min(k + \beta, n)$ //End index of local block
 5:   $d \leftarrow f - k + 1$ // Dimension of local block
 6:   $\log V \leftarrow \sum_{i=1}^{f} \ell_i - \sum_{i=1}^{k-1} \ell_i'$
 7:   if $\phi$ = true then
 8:     if $\frac{\log V - \log(v_d)}{d} < \ell_k$ then
 9:       $\ell_k' \leftarrow \frac{\log V - \log(v_d)}{d}$
10:       $\phi \leftarrow$ false
11:     else
12:       $\ell_k' \leftarrow \ell_k$
13:     end if
14:   else
15:     $\ell_k' \leftarrow \frac{\log V - \log(v_d)}{d}$
16:   end if
17: end for
18: $\log V \leftarrow \sum_{i=1}^{n} \ell_i - \sum_{i=1}^{n-50} \ell_i'$
19: for $k = n - 49$ to $n$ do
20:   $\ell_k' \leftarrow \frac{\log V}{50} + r_{k+50-n}$
21: end for
```

success probability p_0. The main process collects the output from children, and picks up the shortest vector among them. In fact, if a total of M randomized enumerations of success probability p_0 is launched, the success probability if then $p \approx p_0 \cdot M$. The child process and the parent process is described in algorithm 23 and 24 respectively.

Algorithm 23 `ChildEnum`

```
Input: The block $\mathbf{B}_{[j,k]}$, and the bounding function $\mathbf{R}$.
Output: $\mathbf{b}_{new} = \mathbf{v} \cdot \mathbf{B}_{[i,k]}$ the shortest vector in enumeration with bounding $\mathbf{R}$.
 1: $\mathbf{B}' \leftarrow$Randomize($\mathbf{B}_{[j,k]}$);
 2: $\mathbf{B}' \leftarrow$LLL($\mathbf{B}'$)
 3: $\mathbf{b}_{new} \leftarrow$ Enumeration($\mathbf{B}', \mathbf{R}$)
```

4.3.2 Simulation

The BKZ reduction with pruned enumeration differs from original BKZ as it introduces two more parameters, p and γ. For each block, the first vector is only expected to be renewed with probability p, so we no longer have a deterministic expectation. This complicates the simulation problem. We are obliged to make some restrictions in order to give predictions.

Algorithm 24 `OneRoundBKZ2` (Main process)

```
Input: B the input basis, $\beta$ the blocksize, $M$ the number of child process, and R the bounding function
    for enumeration.
Output: B after one round of BKZ-$(\beta, p, \gamma)$ reduction. $(p = M \cdot p_{\text{succ}}(\mathbf{B}), \gamma = R_n)$,
    and f marking if any enumeration had been successful during this round.
 1: f$\leftarrow$ unsuccessful;
 2: for $j = 1, \ldots, n-1$ do
 3:     $k \leftarrow \min(j + \beta - 1, m)$
 4:     for $i \leftarrow 1, \ldots, M$ do
 5:         $(\mathbf{b}_{new}^{(i)}, \mathbf{v}^{(i)}) \leftarrow$ChildEnum$(\mathbf{B}_{[j,k]}, \mathbf{R})$.
 6:     end for
 7:     $m \leftarrow \arg\min_{1 \leqslant i \leqslant M} \|\mathbf{b}_{new}^{(i)}\|$
 8:     if $\|\mathbf{b}_{new}^{(m)}\| < \|\mathbf{b}_j^\star\|$ then
 9:         f$\leftarrow$ successful;
10:         $\mathbf{B} \leftarrow \text{LLL}(\mathbf{b}_1, \ldots, \mathbf{b}_{j-1}, \sum_{i=j}^{k} v_i^{(m)} \cdot \mathbf{b}_i, \mathbf{b}_j, \ldots)$
11:     end if
12: end for
```

An ideal simulation process in the most general form, will take as input the following input and output for each round the shape of the basis being reduced. It is described in algorithm 25.

Algorithm 25 `Ideal_SimulateOneRoundBKZ`

```
Input: The Gram-Schmidt norms, given as $\ell_i = \log(\|\mathbf{b}_i^\star\|)$, for $i = 1, \ldots, n$,
    a blocksize $\beta \in \{2, \ldots, n\}$, a relax factor $\gamma$, success proba $p$.
Output: A prediction for the Gram-Schmidt norms $\ell_i' = \log(\|\mathbf{b}_i^\star\|), i = 1, \ldots, n$, after one round of BKZ
    reduction.
```

In practice, by making following restrictions on BKZ parameters, we still have a deterministic simulation algorithm.

1. $p \approx 1$, then the reduction process can be predicted with a deterministic simulation process.
2. $\beta \geqslant 50$, so that each block behave like random basis.

With the restriction that $p \approx 1$, we are expected to renew the first vector of each block with a new vector of length $\gamma \cdot GH(\mathbf{B}_{[j,k]})$. This is to say, we can substitute the quantity $\frac{\log V - \log(v_d)}{d}$ by $\gamma \cdot \frac{\log V - \log(v_d)}{d}$ in algorithm 22.

One tempting possibility to include p into the simulation is to launch several Monte Carlo simulations. The average output and a variance can be computed from these simulations. However, this simulation does not fit much in the experiment data. The problem is that we only consider $\mathbf{b}_j^\star$ to be generated by enumeration, but in fact, it can be modified during LLL reduction when we find much shorter vector than $\mathbf{b}_j^\star$. When enumeration shortens certain $\mathbf{b}_j^\star$ but not others, then swap between the vectors can happen during LLL reduction to re-order the vectors. We do not know how to incorporate the mechanism of swap and LLL into our simulation yet.

We still have following open problems to solve in order to find an ideal simulation procedure.

- Simulate efficiently while taking different $p < 1$ into consideration.

- Simulate BKZ with $\beta < 50$. It is reported in [23] that the blocks in BKZ differs considerably with random basis for small β. Thus, we do not know how to predict an average BKZ-β reduction for smaller β.

An interesting question would be to compare the efficiency of reduction with β, p and γ. For instances, with pruning option p and relax parameter γ, we are able to do a weak enumeration in large blocksizes using the same time as a strong enumeration in small blocksizes. With a correct simulation we can find out which strategy will improve the basis the best.

4.3.3 Simulation compared to experiment data

We compared the experimental results with the prediction produced by simulation algorithm. Under the restriction of $\beta \geqslant 50$ and $p \approx 1$, the simulation results are good approximation to the experiment data, We ran simulation both on random lattices and Darmstadt's lattice challenges. Figure 4.7 compares the sequence of $\|\mathbf{b}_i^\star\|$ produced by experimental data (in red) and prediction by simulation (in black). It turns out that the simulation matches well with the experimental data.

Meanwhile, the prediction of $\eta(\mathbf{B})$ during reduction procedure approximates experimental data as well. This prediction is an application of corollary 4.1.3. Figure 4.6 shows the evolution of $\delta(\mathbf{B})$ during BKZ-90. The improvement of $\delta(\mathbf{B})$ is mostly done in the beginning few rounds, which complements the theoretical results of [38].

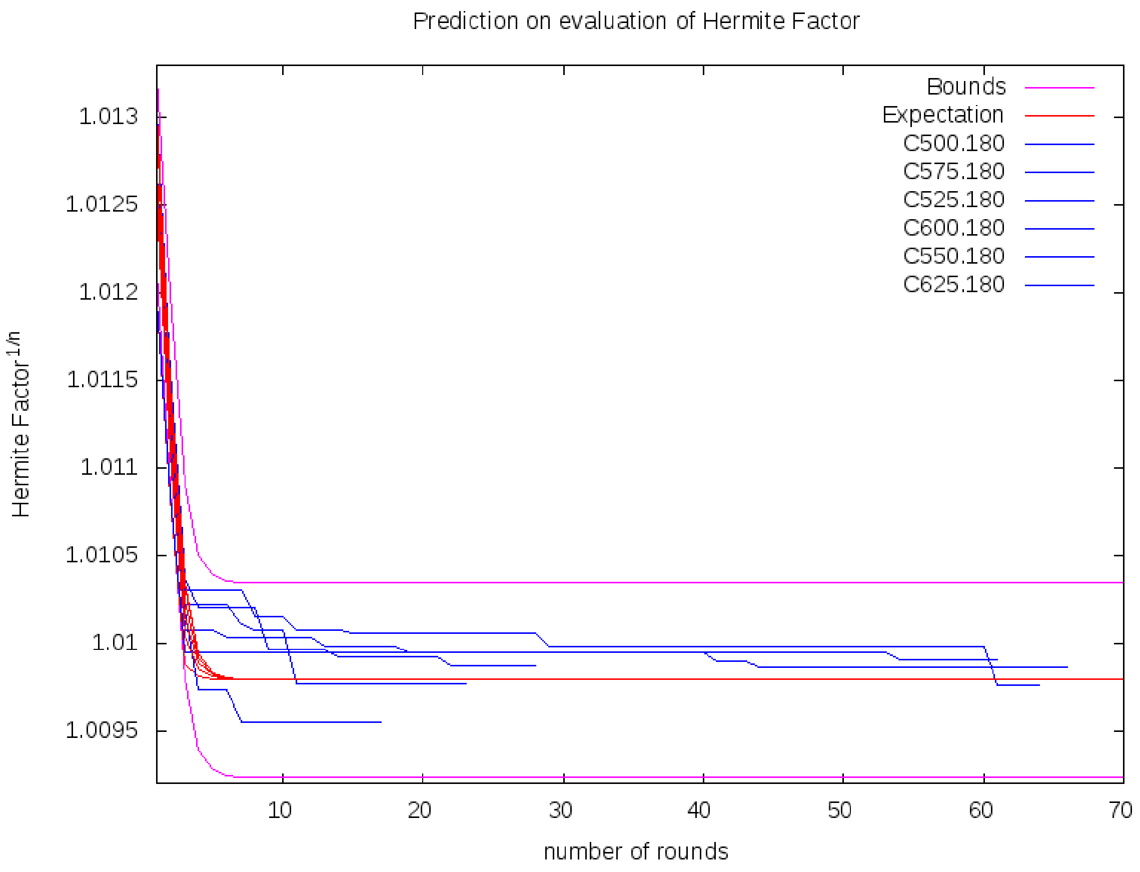
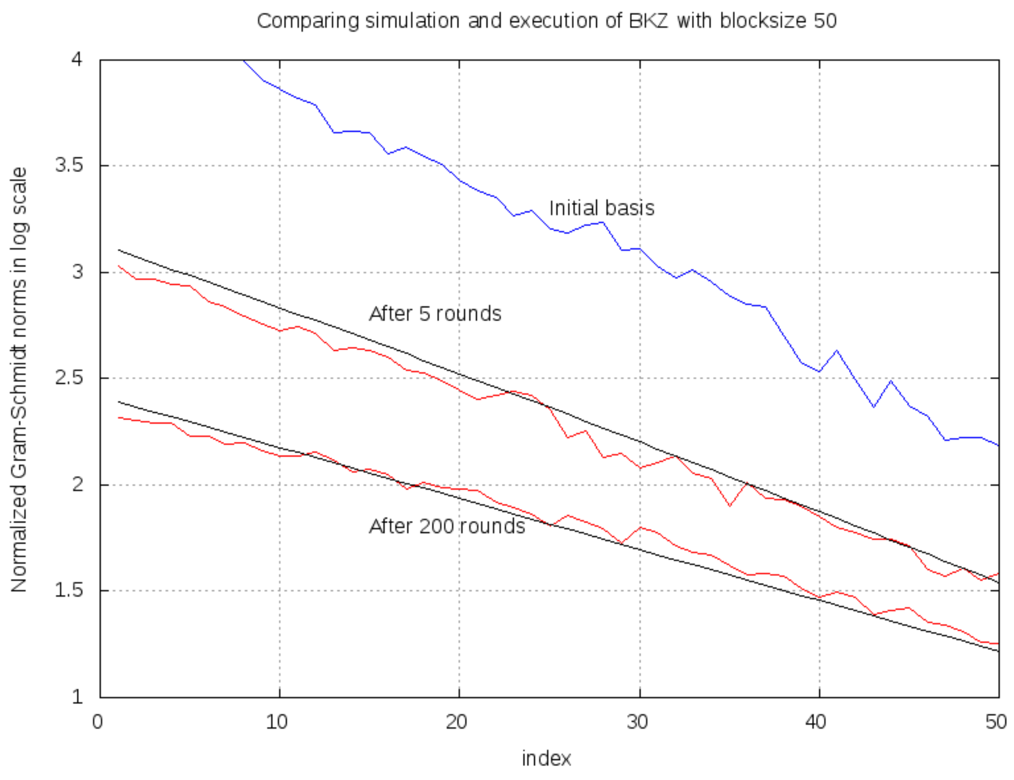
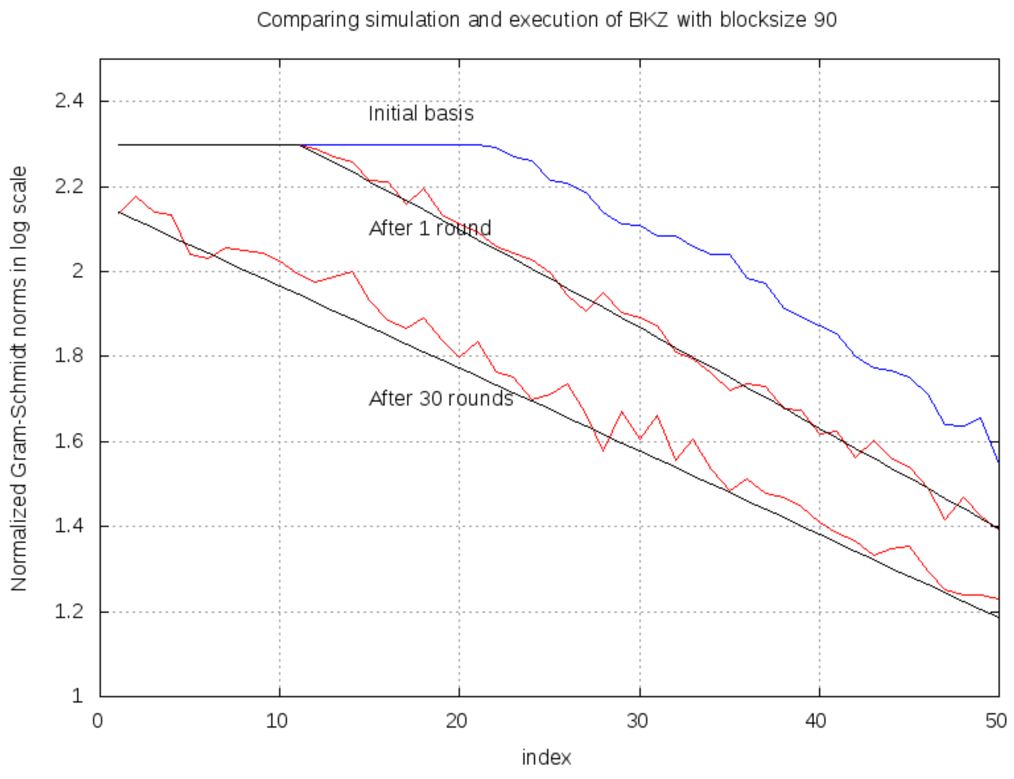


Figure 4.6: Evolution and prediction of $\delta(\mathbf{B})$ during BKZ-90 reduction in dim 180 for Darmstadt's lattice challenges 500-625

Figure 4.7: Comparing simulation with experiment data. The figures show a zoom-in view of the first 50 basis vectors





4.4 BKZ 2.0: improved reduction

We implemented an improved version of BKZ, which incorporates several modifications to the original algorithm. Besides pruned enumeration, another important improvement is to pre-process each block before enumeration takes place. It is folklore that the enumeration time is dependent on the quality of the basis given, and a reduction of the basis will shorten the enumeration time. But this reduction is never performed on BKZ blocks before. Especially, it is important to find out the optimal parameter for the preprocessing-reduction: this parameter is dependent on the current quality of the block, and it is dependent on β, p and γ.

4.4.1 Abort after a few rounds

The most significant improvements of BKZ reduction only occurs in the first few rounds. This is both proved by [38] and predicted by our simulation algorithm. Hence, instead of launch the reduction round and round again, it is more efficient to stop the reduction after K rounds, knowing that following rounds are only going to make minor contributions to the improvement of the basis quality. This adds another parameter K to BKZ reduction.

4.4.2 Preprocessing with a smaller reduction

Without any preprocessing, the quality of the basis of block $\mathbf{B}_{[j,k]}$ quickly becomes only LLL-reduced. This is because:

- For each enumeration, the local basis is only guaranteed to be LLL-reduced, even though the whole basis may be more than LLL-reduced. As enumeration finds vectors and the basis get renewed, the rest of the basis is modified in the LLL reduction following this successful enumeration.
- In high blocksizes, most enumerations are successful: they find a shorter vector than the first block vector. This implies that a local LLL-reduction will be performed to get a basis from a generating set: At the next iteration, the enumeration will proceed on a typical LLL-reduced basis, and not something likely to be better reduced.

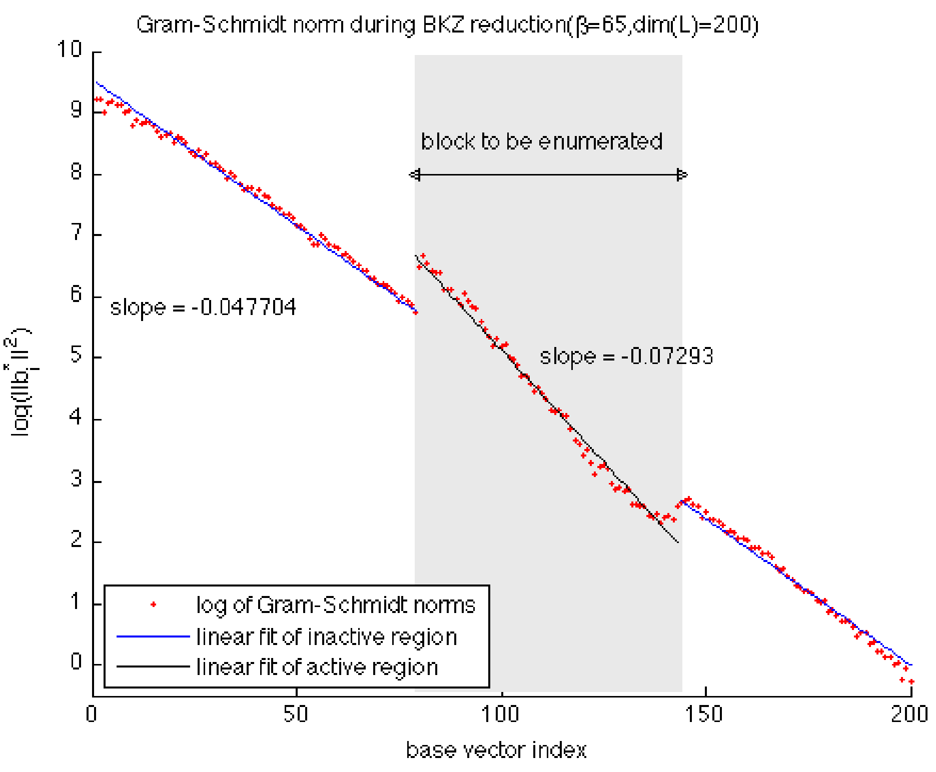


Figure 4.8: Quality of a local block in in original BKZ

Observation from experiments confirm the intuition. Figure 4.8 shows the quality of the whole basis during a typical BKZ iteration; one clearly sees that before enumeration, the local basis is only LLL-reduced, while other local bases (outside the window formed by the block) are more reduced.

Therefore, we apply to the block a BKZ-β_0 which aborts after Q rounds. Obviously β_0 is much smaller than β, otherwise the reduction cost is much greater than the original enumeration cost. The optimal β_0 and Q are dependent on β, γ and p, and we searched for this parameter by comparing different β_0 and Q for the reduction performance and choose the best among all parameters.

4.4.3 Recursive reduction

Our preprocessing reduction during BKZ uses a blocksize of β_0, and when this β_0 gets big, one wonders if it is worth doing another preprocessing for each block of size β_0.

In general, we can have a recursive BKZ reduction. BKZ-β_0 can make calls to BKZ-β_1, which again calls BKZ-β_2, and so on. Until level L, when β_L gets small enough, so that each enumeration in blocksize β_L can be performed directly without any prior reduction. At level i of reduction, the set of parameter is $\beta_i, p_i, \gamma_i, Q_i$, and is fixed at the beginning of the reduction. The total number of parameter is multipled by L, the depth of reduction, in comparison to the non-recursive BKZ algorithm.

An algorithmic description of this is given as follows (algorithm 27 and 26):

Algorithm 26 `RecEnumeration`

```
Input: B the input basis,
    $\beta_0, \beta_1, \dots, \beta_L$ the blocksizes,
    M the number of child process,
    $\mathbf{R}_0, \mathbf{R}_1, \dots, \mathbf{R}_L$ the bounding functions for enumeration,
    $Q_0, Q_1, \dots, Q_L$ the number of rounds before abort,
    $j, k$ the beginning and ending index of the block.
Output: B after one round of BKZ-$(\beta_0, p, \gamma)$ reduction. $(p = M \cdot p_{\text{succ}}(\mathbf{R}_0), \gamma = R_n)$,
    and f marking if any enumeration had been successful during this round.
1:  $J_0 \leftarrow j; K_0 \leftarrow k.$
2:  $l \leftarrow 1$
3:  $J_1 \leftarrow J_0 - 1$
4:  while $(q_1 < Q_1)$ do
5:      $J_l \leftarrow ((J_l - J_{l-1} + 1) \mod (K_{l-1} - J_{l-1})) + J_{l-1}$ //next J
6:      $K_l = \min(J_l + \beta_l - 1, K_{l-1})$; //next K
7:      while $(l < L)$ and $(K_l - J_l > \beta_{l-1})$ do
8:          $l \leftarrow l + 1$; //go down to small enums if current block is big
9:          $J_l \leftarrow J_{l-1}$;
10:         $K_l \leftarrow \min(J_l + \beta_l - 1, K_{l+1})$;
11:         $q_l \leftarrow 0$;
12:     end while
13:     $(\mathbf{b}_{new}, \mathbf{v}) \leftarrow$Enumerate$(\mathbf{B}_{[J_l,K_l]}, \mathbf{R}_i)$
14:     $\mathbf{B} \leftarrow$LLL$(\mathbf{b}_1, \dots, \mathbf{b}_{J_l-1}, \sum_{i=J_l,\dots,K_l} v_i\mathbf{b}_i, \mathbf{b}_{J_l}, \dots)$
15:     if $J_l = K_{l-1} - 1$ then
16:         $q_l \leftarrow q_l + 1$
17:     end if
18:     while $(q_l > Q_l)$ and $(l > 0)$ do
19:         $l \leftarrow l - 1$
20:         $\mathbf{b}_{new} \leftarrow$Enumerate$(\mathbf{B}_{[J_l,K_l]}, \mathbf{R}_i)$
21:         $\mathbf{B} \leftarrow$LLL$(\mathbf{b}_1, \dots, \mathbf{b}_{J_l-1}, \mathbf{b}_{new}, \mathbf{b}_{J_l}, \dots)$
22:         $\mathbf{B} \leftarrow$LLL$(\mathbf{b}_1, \dots, \mathbf{b}_{J_l-1}, \sum_{i=J_l,\dots,K_l} v_i\mathbf{b}_i, \mathbf{b}_{J_l}, \dots)$
23:         if $J_l = K_{l-1} - 1$ then
24:             $q_l \leftarrow q_l + 1$
25:         end if
26:     end while
27: end while
```

Algorithm 27 `OneRoundBKZ3` (Recursive)

Input: $\mathbf{B}$ the input basis, $(\beta_0, \beta_1, \ldots, \beta_L)$ the blocksizes,
M the number of child process,
$\mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_L$ the bounding functions for enumeration,
$Q_0, Q_1, \ldots, Q_L$ the number of rounds before abort,

Output: $\mathbf{B}$ after one round of BKZ-(β_0, p, γ) reduction. $(p = M \cdot p_{\text{succ}}(\mathbf{R}_0), \gamma = R_n)$,
and `f` marking if any enumeration had been successful during this round.

```
1:  f ← unsuccessful;
2:  for $j = 1, \ldots, n-1$ do
3:      $k \leftarrow \min(j + \beta - 1, m)$
4:      for $i \leftarrow 1, \ldots, M$ do
5:          $(\mathbf{b}_{new}^{(i)}, \mathbf{v}^{(i)}) \leftarrow$RecEnumeration$(\mathbf{B}, j, k)$.
6:      end for
7:      $m \leftarrow \arg\min_{1 \leqslant i \leqslant M} \|\mathbf{b}_{new}^{(i)}\|$
8:      if $\|\mathbf{b}_{new}^{(m)}\| < \|\mathbf{b}_j^\star\|$ then
9:          f ← successful;
10:         $\mathbf{B} \leftarrow$ LLL$(\mathbf{b}_1, \ldots, \mathbf{b}_{j-1}, \sum_{i=j}^{k} v_i^{(m)} \cdot \mathbf{b}_i, \mathbf{b}_j, \ldots)$
11:     end if
12: end for
```

4.4.4 Optimal parameters for BKZ

In the recursive version of BKZ, the number of parameters are multiplied by the total level L. As a matter of fact, for a given β, p, γ, there is an optimal preprocessing parameter $(\beta_0, p_0, Q_0), (\beta_1, p_1, Q_1), \ldots$.

In our experiment, we have actually fixed γ to be $\|\mathbf{b}_j^\star\|/GH(\mathbf{B}_{[j,k]})$ for simplicity. For each β_0 and p_0, we try to exhaustively try all possible preprocessing parameters, for $\beta_1 < \beta_0$ and $p \in \{5\%, 10\%, \ldots, 100\%\}$, unless it takes too long to finish, and record the time it takes to reduce a LLL-reduced 100 dimensional basis for one round. Then the best time of reduction with preprocessing is again compared with enumeration without any preprocessing, to draw the final conclusion for the best parameter.

This search tries to exhaust all possible parameters, but the experiment order is carefully chosen in order to avoid totally vain efforts and arrive at optimal parameters faster: For example, while we search for the best preprocessing parameter for $\beta_0 = 35, p_0 = 10\%$, if with $\beta_1 = 20$ the preprocessing the reduction is not faster than reduction without preprocessing, then there is no need to try for larger β_1.

In addition, there is a slight recursive pattern in the parameter, For example, with $\beta = 25$ and $\beta = 30$, our experiments show that it is always better not to apply any preprocessing during reduction. This means that when we use a preprocessing $\beta_1 < 30$, we would not need any recursive preprocessing with β_2 inside the block of size β_1. To be on the safe side, when $\beta_1 > 30$, we did not rely completely on the recursive pattern. Even when (β_2, p_2) is the recommended preprocessing parameter for BKZ-β_1, we always try to compare some parameters around β_2, p_2 and pick the best one, because the basis quality in BKZ blocks with preprocessing, can be better than LLL-reduced basis.

Our implementation is based on the NTL library, and it is known that its LLL reduction is not optimal. For example, it is slower than the implementation in fplll library. In addition, the running time of the reduction has much too do with the computation precision. While we used double precision in our experiment, it is possible that for input lattices of larger dimension or special structure, the computation precision needs to be increased. For both these reasons, we do not claim our recommended parameters to be optimal under all circumstances. It is probably close to optimal anyway. And the timing figures we provided are only used for finding out the reduction parameter, and should be interpreted with care.

In the tables that follow, we listed our recommended parameter for $\beta_0 = 25, 30, \ldots, 90$. For larger β_0, we do not recommend parameters for every p_0, because when $\beta_0 > 50$, large p_0 are inefficient and rarely used in practice. Instead, it would make more sense to use larger block size β_0 and smaller p_0 in order to obtain the similar output reduction quality.

The second line in the table shows the total time for the first round of reduction, while the third line shows the time spent on enumeration.

From the figures, we see that the enumeration in dimension 25 and 30 is fast enough even without preprocessing.

Table 4.2: Executing for $\beta_0 = 25$

$\beta_0 = 25$	5%	10 %	15%	20%	25%	30%	35%	40%	45%	50%
Total time (s)	0.16	0.17	0.17	0.17	0.18	0.18	0.18	0.19	0.20	0.19
Enum time (s)	0.03	0.04	0.03	0.03	0.03	0.03	0.04	0.03	0.03	0.03
$\beta_0 = 25$	55%	60 %	65%	70%	75%	80%	85%	90%	95%	100%
Total time (s)	0.20	0.19	0.19	0.20	0.20	0.20	0.20	0.21	0.22	0.26
Enum time (s)	0.03	0.04	0.03	0.04	0.04	0.04	0.04	0.04	0.05	0.08

Table 4.3: Executing for $\beta_0 = 30$

$\beta_0 = 30$	5%	10 %	15%	20%	25%	30%	35%	40%	45%	50%
Total time (s)	0.21	0.21	0.22	0.22	0.23	0.24	0.24	0.24	0.25	0.25
Enum time (s)	0.04	0.04	0.02	0.04	0.04	0.04	0.04	0.04	0.04	0.04
$\beta_0 = 30$	55%	60 %	65%	70%	75%	80%	85%	90%	95%	100%
Total time (s)	0.26	0.26	0.26	0.28	0.28	0.29	0.30	0.30	0.32	0.66
Enum time (s)	0.04	0.05	0.05	0.06	0.06	0.06	0.07	0.08	0.08	0.42

For $\beta_0 = 35$, if we use pruned enumeration, the preprocessing does not have any advantage either, except for full enumeration, where the preprocessing with chosen parameter is better than no preprocessing. For full enumeration in BKZ-35, we recommend a preprocessing of BKZ-30, $p = 5\%$ and round $Q = 1$.

Table 4.4: Executing for $\beta_0 = 35$

$\beta_0 = 35$	5%	10 %	15%	20%	25%	30%	35%	40%	45%	50%
Total time (s)	0.11	0.15	0.16	0.19	0.20	0.22	0.24	0.25	0.26	0.28
Enum time (s)	0.04	0.04	0.04	0.04	0.05	0.05	0.06	0.06	0.07	0.08
$\beta_0 = 35$	55%	60 %	65%	70%	75%	80%	85%	90%	95%	100%
Optimal β_0										30
Probability										5%
Rounds										1
Total time (s)	0.29	0.31	0.33	0.34	0.37	0.41	0.44	0.50	0.58	2.63
Enum time (s)	0.08	0.08	0.10	0.11	0.14	0.16	0.19	0.26	0.32	1.35

Table 4.5: Executing for $\beta_0 = 40$

$\beta_0 = 40$	5%	10 %	15%	20%	25%	30%	35%	40%	45%	50%
Total time (s)	0.15	0.20	0.22	0.28	0.32	0.35	0.39	0.42	0.47	0.55
Enum time (s)	0.04	0.06	0.07	0.07	0.1	0.13	0.15	0.18	0.22	0.27
$\beta_0 = 40$	55%	60 %	65%	70%	75%	80%	85%	90%	95%	100%
Optimal β_0									25	35
Probability									30%	75%
Rounds									1	1
Total time (s)	0.58	0.63	0.80	0.83	0.98	1.17	1.56	1.93	2.21	11.22
Enum time (s)	0.30	0.33	0.51	0.54	0.68	0.86	1.25	1.61	0.61	9.05

Table 4.6: Executing for $\beta_0 = 45$

$\beta_0 = 45$	5%	10 %	15%	20%	25%	30%	35%	40%	45%	50%
Optimal β_1										25
Probability										30%
Rounds										1
Total time (s)	0.23	0.38	0.49	0.60	0.91	1.06	1.17	1.49	1.65	2.12
Enum time (s)	0.10	0.19	0.29	0.37	0.65	0.77	0.87	1.18	1.33	0.43
$\beta_0 = 45$	55%	60 %	65%	70%	75%	80%	85%	90%	95%	100%
Optimal β_1	25	25	25	25	25	25	25	40	40	40
Probability	10%	20%	5%	20%	45%	65%	60%	10%	20%	70%
Rounds	1	1	1	1	1	1	1	1	1	1
Total time (s)	2.21	2.32	2.47	2.62	2.84	3.07	3.39	3.91	5.11	77.55
Enum time (s)	0.52	0.55	0.77	0.88	0.95	1.09	1.42	1.75	2.77	74.72

Starting from BKZ-50, preprocessing is obviously improving the performance of BKZ.

Table 4.7: Executing for $\beta_0 = 50$

$\beta_0 = 50$	5%	10 %	15%	20%	25%	30%	35%	40%	45%	50%
Optimal β_1		25	25	25	25	25	25	25	25	35
Probability		5%	15%	15%	30%	30%	55%	60%	55%	30%
Rounds		1	1	1	1	1	1	1	1	1
Total time (s)	0.95	1.93	2.15	2.30	2.58	2.86	3.11	3.34	3.73	4.10
Enum time (s)	0.77	0.30	0.39	0.53	0.71	0.92	1.03	1.21	1.59	1.53

$\beta_0 = 50$	55%	60 %	65%	70%	75%	80%	85%	90%	95%	100%
Optimal β_1	45	45	45	45	45	45	40	45	40	
Probability	5%	10%	15%	10%	15%	15%	30%	20%	50%	
Rounds	1	1	1	1	1	1	1	1	1	
Total time (s)	4.49	4.92	5.37	5.57	6.44	7.38	8.91	10.46	14.10	
Enum time (s)	2.21	1.94	2.49	2.77	3.57	4.44	5.76	7.41	10.70	

Table 4.8: Executing for $\beta_0 = 55$

$\beta_0 = 55$	5%	10 %	15%	20%	25%	30%	35%	40%	45%	50%
Optimal β_1	25	25	25	25	35	40	45	45	45	45
Probability	10%	55%	55%	65%	20%	15%	10%	10%	15%	20%
Rounds	1	1	1	1	1	1	1	1	1	1
Total time (s)	0.37	0.83	1.13	1.70	2.00	2.37	3.29	3.54	4.65	5.31
Enum time (s)	2.09	2.90	3.26	4.00	4.82	5.55	6.63	6.98	8.21	8.97

$\beta_0 = 55$	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
Optimal β_1	45	45	45	45	45	45	40	45	40	
Probability	20%	20%	25%	30%	35%	45%	45%	45%	45%	
Rounds	1	1	1	1	1	1	1	1	1	
Total time (s)	7.26	7.85	10.45	13.60	15.94	21.76	25.91	42.51	61.61	
Enum time (s)	10.91	11.58	14.33	17.57	20.02	25.95	30.16	46.77	69.02	

Table 4.9: Executing for $\beta_0 = 60$

$\beta_0 = 60$	5%	10 %	15%	20%	25%	30%	35%	40%	45%	50%
Optimal β_1	35	35	40	45	45	45	45	45	45	45
Probability	15%	25%	25%	15%	15%	20%	25%	20%	40%(*)	35%
Rounds	1	1	1	1	1	1	1	1	1	1
Total time (s)	1.36	2.65	4.05	5.35	7.99	10.02	11.84	17.39	13.32	16.88
Enum time (s)	4.07	5.79	7.93	9.49	12.06	14.36	16.39	21.78	16.52	20.04

$\beta_0 = 60$	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
Optimal β_1	45	45	50	50	50	50	50	50		
Probability	35%	35%	25%	30%	30%	25%	40%	50%		
Rounds	1	1	1	1	1	1	1	1		
Total time (s)	22.11	28.32	37.19	41.71	62.07	68.76	88.18	178.88		
Enum time (s)	25.24	31.50	40.33	44.98	65.30	71.95	91.53	182.37		

In the table, from 40%(*) on, we switched the program to a machine which is 1.6 times faster.

For larger β_0, some of the recommended parameter has recursion level $L > 1$, as was the case for $\beta_0 = 65, p_0 \leqslant 50\%$. In this case, we represented the parameters by simply listing $(\beta_2, p_2, Q_2, \beta_1, p_1, Q_1)$.

Table 4.10: Executing for $\beta_0 = 65$

$\beta_0 = 65$	5%	10 %	15%	20%	25%	30%	35%	40%	45%
Optimal β_0	45	45	50	50	50	50	50	50	50
Probability	15%	15%	10%	15%	20%	15%	20%	25%	25%
Rounds	1	1	1	1	1	1	1	1	1
Total time (s)	3.24	6.96	12.91	18.56	28.95	37.31	44.02	60.50	76.75
Enum time (s)	6.07	9.96	16.12	21.98	32.48	40.73	47.60	64.22	80.44

$\beta_0 = 65$	Optimal preprocessing parameter	Total time (s)
50%	25 0.1 1 50 0.3 1	97.47
55%	25 0.05 1 55 0.2 1	116.85
60%	25 0.05 1 55 0.25 1	136.13
65%	25 0.05 1 55 0.25 1	180.93
70%	25 0.05 1 55 0.3 1	231.70
75%	25 0.05 1 55 0.3 1	243.75
80%	25 0.05 1 55 0.3 1	387.41

Table 4.11: Executing for $\beta_0 = 70$

$\beta_0 = 70$	Optimal preprocessing parameter	Total time (s)
5%	50 0.05 1	85.35
10%	25 0.05 1 55 0.05 1	140.75
15%	25 0.05 1 55 0.05 1	191.15
20%	25 0.05 1 55 0.15 1	265.00

Table 4.12: Executing for $\beta_0 = 75$

$\beta_0 = 75$	Optimal preprocessing parameter	Total time (s)
5%	35 0.15 1 60 0.05 1	204.85
10%	25 0.15 1 55 0.15 1	464.9
15%	25 0.05 1 55 0.15 1	808.35
20%	25 0.05 1 55 0.15 1	1310.25

Table 4.13: Executing for $\beta_0 = 80$

$\beta_0 = 80$	Optimal preprocessing parameter	Total time (s)
5%	40 0.15 2 60 0.05 1	879.35
10%	35 0.15 1 60 0.05 1	3715.15

Table 4.14: Executing for $\beta_0 = 90$

$\beta_0 = 90$	Optimal preprocessing parameter	Total time (s)
0.5%	50 0.15 2 65 0.2 2	2443.35

For larger blocksizes, the enumeration can be predicted with Gaussian heuristic, and its performance can be predicted by simulation. Please refer to the next chapter for discussions about optimal parameter for enumerations.

CHAPTER 5

New Reduction Procedure

Contents

The simulation of BKZ reduction allows to predict the running time and output quality of each round. Hence, given an input basis, we are able to compare among possible β and other parameters, and decide for the best parameter for reduction. This is again an optimization problem. In the first time, we start by only looking at the different value of β.

The output basis after this round of reduction has improved quality than the input basis, therefore for the new round, we will conclude different parameters for reduction, and probably the optimal β of the next round is different from previous round. This is gives a different reduction algorithm than the previous BKZ reduction, where we fix β in advance. Instead, β can be automatically chosen at each round, depending on the current basis. It is adjusted (increased) each round until the basis is reduced to target quality. This process can be fully predicted using the simulation procedure, and the resulting basis sequence is what we call an optimal reduction sequence. It is called optimal because each round, the chosen β guarantees predicted by simulation.

The optimal reduction sequence also arguably answers the question of optimal enumeration. In chapter 3, we discussed how to choose a pruning function with given basis, but we do not know how to choose a good reduction in order to minimize running time. Now the optimal reduction sequence provides an answer for this. It suffices to compute for each basis in this sequence, the best possible enumeration time, and adding this to the reduction time. In most of the cases, this gives an almost optimal answer.

Following the same vain, it is natural to ask whether it is possible to automatically choose a good preprocessing strategy as well. It turns out that this is more complicated than The main idea is similar to that of optimal enumeration, since the goal of preprocessing is to prepare a block of basis for efficient enumeration as well. However, the goal of this enumeration procedure in BKZ is implicit. In BKZ enumeration, we are actually solving ASVP_γ with some probability p_0, where γ and p is not clear. Contrary to the common case of standalone SVP, where we are given an approximation factor, and we aim to solve it with 100% confidence with $1/p$ number of randomized enumerations. Here we are not hoping to be all-time successful for BKZ enumerations, and we do not have an explicit demand for the approximate factor γ. This is to say, our optimization goal is unclear. While this problem is still open, we provide some sub-optimal ways set up the preprocessing parameters automatically.

In practice, the new BKZ reduction algorithm allow us to solve the SVP challenge up to dimension 126.

5.1 Optimal reduction sequence

In BKZ reduction, we are able to predict for each round the output basis quality. This gives us the choice to use different reduction parameters for different round, depending on the character of the existing basis. In fact, we are able to compare for different parameters, the efficiency of reduction, thus choose the optimal reduction parameter for the basis given. And once we optimize the reduction parameter for each round iteratively, we obtain the optimal sequence of basis by using different parameters for each round respectively.

Algorithm 28 Block Korkin-Zolotarev (BKZ) algorithm

```
Input: A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, a target Hermite factor $\delta_0$ or target half volume $\eta_0$.
Output: The basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is BKZ-$\beta$ reduced
1: $(\mathbf{B}, \mathbf{U}) \leftarrow$ LLL($\mathbf{b}_1, \ldots, \mathbf{b}_n, \mathbf{U}$);// LLL-reduce the basis, and obtain Gram-Schmidt Orthogonalisation $\mathbf{U}$
2: while $\delta(\mathbf{B}) < \delta_0$ or $\eta(\mathbf{B}) < \eta_0$ do
3:     $(\beta, r, T) \leftarrow$ BestReductionParameter($\mathbf{B}$)
4:     $(\mathbf{B}, \mathbf{U}) \leftarrow$OneRoundBKZ$_{(\beta,r,T)}(\mathbf{B}, \mathbf{U})$
5: end while
```

The `BestReductionParameter` procedure will give the optimal parameter (β, r, T) which most efficiently reduces the given basis $\mathbf{B}$. This is done by solving an optimization problem again. Now the `OneRoundBKZ` is described as follows,

BKZ-β algorithm makes calls to enumerations of dimension at most β.

5.1.1 The optimization problem

We turn the problem of parameter choosing into an optimization problem thanks to simulation. By trying different parameters, and comparing their running time and output basis quality, we can choose the best among them for the next round of reduction. This again, can be view as an optimization problem, where the variable to be optimized is the reduction parameters, and the optimization goal is the quality of basis with regard to running time.

The parameters to be optimized include: β, γ, and p. β is the size of the enumeration block, γ and p the probability of are parameters for enumeration on the block $\mathbf{B}_{[j,k]}$. The enumeration is supposed to find a short vector of each block within radius $\gamma \cdot GH$ with total probability p.

We still need to well define an optimization goal which integrates both basis quality and running time. The quality of the basis can be described by several values, including half volume $\eta(\mathbf{B})$ and root Hermite factor $\delta(\mathbf{B})$. The running time T can be roughly estimated by adding enumeration time of each block. This gives us several choices of optimization goal functions to minimize. We note $\mathbf{B}$ and $\mathbf{B}'$ the basis before and after reduction respectively, then possible choices of goal functions include

- $f(\beta, \gamma, p) = (\eta(\mathbf{B}') - \eta(\mathbf{B}))/T$

- $f(\beta, \gamma, p) = (\delta(\mathbf{B}') - \delta(\mathbf{B}))/T$
- $f(\beta, \gamma, p) = (\eta(\mathbf{B}') - \eta(\mathbf{B}))/\log T$
- $f(\beta, \gamma, p) = (\delta(\mathbf{B}') - \delta(\mathbf{B}))/\log T$

The difference between these functions are subtle: The Hermite factor $\delta(\mathbf{B})$ describes the length of $\|\mathbf{b}_1\|$ with regard to the whole basis, whereas the half volume $\eta(\mathbf{B})$ dominates the cost of enumeration of the basis. If the reduction purpose is to prepare the basis for enumeration, then it would be logic to use $\eta(\mathbf{B})$, otherwise if one only wants a shorter basis, especially if one needs a short vector, the natural choice is $\eta(\mathbf{B})$.

Setting T as denominator is intuitive, as we want to improve as much quality as possible in unit time. As a matter of fact, this choice may favor very small β with minor improvements, which increases the number of rounds to reach the target quality. One can use $\log T$ to encourage larger steps of improvement.

The optimization problem can be formalized as:

$$\begin{aligned} \underset{\beta,\gamma,p}{\text{minimize}} \quad & f(\beta, \gamma, p) \\ \text{subject to} \quad & \gamma > 1, 2 \leqslant \beta \leqslant n, 0 < p < 1 \end{aligned} \tag{5.1}$$

5.1.2 Estimating reduction running time

In high dimension, enumeration time is usually dominant compared to the time spent on computing GS and LLL reduction. Thus our estimation of the running time is

$$T_{BKZ}(\mathbf{B}, \beta, \gamma, p) = \sum_{j=1,\ldots,n-1}^{k=min(j+\beta,n)} T_{enum}(\mathbf{B}_{[j,k]}, \gamma, p) + O(1), \tag{5.2}$$

We have discussed the estimation of running time for enumeration in chapter 3, when we search for a vector within $\gamma \cdot GH$ on basis $\mathbf{B}$. Here there is one more variable p, because we no longer expect to successfully find a short vector all the time. The estimated running time is simply

$$T_{enum}(\mathbf{B}_{[j,k]}, \gamma, p) = p \cdot T_{enum}(\mathbf{B}_{[j,k]}, \gamma) \tag{5.3}$$

In fact, estimation of enumeration is not definitive yet, because it will depend on the analysis of reduction. We will discuss how this optimal reduction sequence help to improve the enumeration performance in section 5.2. If we try to optimize every thing at the same time, it would be tremendously complicated. Instead in the first time, we use the enumeration.

Ultimately we wish to create a look-up table for optimal enumeration time. For a typically reduced basis $\mathbf{B}$, its dimension n and half volume $\eta(\mathbf{B})$ is enough to describe its relative character for estimating enumeration time, and searching for a short vector within approximate factor γ will take $T_{enum}(n, \eta(\mathbf{B}), \gamma)$ time. The base cases are established by record data by experiments, such as data from chapter 4. The construction of this table is discussed in the subsection 5.2.1 Once we have this look-up table, we can do the following estimation:

$$T_{enum}(\mathbf{B}_{[j,k]}, \gamma) = T_{enum}(k - j + 1, \eta(\mathbf{B}_{[j,k]}), \gamma) \tag{5.4}$$

5.1.3 Optimize β

For simplicity, we first consider a single parameter β, and supposing $\gamma \approx 1$, and $p \approx 1$. The algorithm of searching for optimal β is described in algorithm 29.

Algorithm 29 SearchOptimalβ

```
Input: The log Gram-Schmidt norm vectors of the basis to be reduced $\mathbf{B}^\star = (\mathbf{b}_1^\star, \ldots, \mathbf{b}_n^\star)$
    and the expected running time of the round $T_0$.
Output: The suggested optimal reduction blocksize $\beta$
 1: $\beta' \leftarrow \beta_0$; //$\beta_0$ is an initial value, which can be set to 20 for example.
 2: $\tilde{\mathbf{B}}^\star \leftarrow$SimulateOneRoundBKZ($\mathbf{B}^\star, \beta'$)
 3: $s \leftarrow (\eta(\tilde{\mathbf{B}}^\star) - \eta(\mathbf{B}^\star))/T_{BKZ}(\mathbf{B}^\star, \beta)$
 4: while $\beta' < n$ do
 5:     $\beta' \leftarrow \beta' + 1$;
 6:     $\tilde{\mathbf{B}}^\star \leftarrow$SimulateOneRoundBKZ($\mathbf{B}^\star, \beta'$)
 7:     $s' \leftarrow (\eta(\tilde{\mathbf{B}}^\star) - \eta(\mathbf{B}^\star))/T_{BKZ}(\mathbf{B}^\star, \beta)$
 8:     if $s' < s$ then
 9:         $s \leftarrow s'$
10:     else
11:         $\beta \leftarrow \beta'$; return;
12:     end if
13: end while
```

5.1.4 Adding other parameters

With multiple parameters, it is hard to see how to optimize several parameters except for exhaustive search in the whole parameter space. The exhaustive search algorithm for multiple parameters is similar to the single parameter version, besides there will be one more loop in search for an optimal γ. We do not know how to add p to the search yet, due to the restriction of our simulation algorithm. But in theory, once we have a good simulation which accepts p, adding p into optimization parameter is just adding yet another loop within the loop for γ.

However, more assumptions to the reduction will simplify the problem. For example, if we restrict the time of reduction for each round to be T_0, then on average, we will have about T_0/n time for each block enumeration. Suppose the block size is β, then the constraint $T_{BKZ}(\mathbf{B}^\star, \beta, \gamma) = T_0/n$ implies a unique value of γ. This value can be computed using binary search, since T_{BKZ} is monotonous increasing with respect to γ. This yields the algorithm 30.

5.1.5 Optimal sequence

For each round we are able to compute a set of optimal parameters. Doing this optimal reduction iteratively yields an optimal sequence of basis, which can be viewed as the fastest possible path a basis can be reduced. We define the optimal sequence by following algorithm.

Algorithm 30 `SearchOptimal`β`And`γ

```
Input: The log Gram-Schmidt norm vectors of the basis to be reduced $\mathbf{B}^\star = (\mathbf{b}_1^\star, \dots, \mathbf{b}_n^\star)$,
    The expected time for a reduction round $T_0$.
Output: The suggested optimal reduction blocksize $\beta$ and factor $\gamma$.
1: $\beta' \leftarrow \beta_0$; //$\beta_0$ is an initial value, which can be set to 20 for example.
2: $\gamma \leftarrow$Solve $(T_{BKZ}(\mathbf{B}_{\beta'}, \gamma) = T_0)$ //Binary search on $\gamma$
3: $\tilde{\mathbf{B}}^\star \leftarrow$SimulateOneRoundBKZ($\mathbf{B}^\star, \beta', \gamma$)
4: $s \leftarrow (\eta(\tilde{\mathbf{B}}^\star) - \eta(\mathbf{B}^\star))/T_{BKZ}(\mathbf{B}^\star, \beta)$
5: while $\beta' < n$ do
6:     $\beta' \leftarrow \beta' + 1$;
7:     $\gamma \leftarrow$Solve $(T_{BKZ}(\mathbf{B}_{\beta'}, \gamma) = T_0)$//Binary search on $\gamma$
8:     $\tilde{\mathbf{B}}^\star \leftarrow$SimulateOneRoundBKZ($\mathbf{B}^\star, \beta', \gamma$)
9:     $s' \leftarrow (\eta(\tilde{\mathbf{B}}^\star) - \eta(\mathbf{B}^\star))/T_{BKZ}(\mathbf{B}^\star, \beta)$
10:    if $s' < s$ then
11:        $s \leftarrow s'$
12:    else
13:        $\beta \leftarrow \beta'$; return;
14:    end if
15: end while
```

Algorithm 31 `OptimalSequence`

```
Input: The log Gram-Schmidt norm vectors of the basis to be reduced: $\mathbf{B}^\star = (\|\mathbf{b}_1^\star\|, \dots, \|\mathbf{b}_n^\star\|)$.
Output: A sequence of log Gram-Schmidt norm vectors $\mathbf{B}_1^\star, \mathbf{B}_2^\star, \dots$ produced by optimal reduction,
    and the accumulated reduction time $T_1, T_2, \dots$.
1: $\mathbf{B}^\star \leftarrow \mathbf{B}^\star$; $T_1 \leftarrow 0$
2: for $i = 2, 3, \dots$ do
3:     $(\beta, [\gamma]) \leftarrow$SearchOptimal$\beta$[And$\gamma$]($\ell_{i-1}$);
4:     $\mathbf{B}_i^{star} \leftarrow$SimulateOneRoundBKZ($\mathbf{B}_{i-1}^{star}, \beta[, \gamma]$);
5:     $T_i \leftarrow T_{BKZ}(\mathbf{B}^\star, \beta[, \gamma]) + T_{i-1}$
6: end for
```

5.2 Optimal enumeration

5.2.1 Putting together reduction and enumeration

In chapter 3, we discussed how to choose a pruning function with given basis, assuming that the basis is reduced within a constant time. However, we did not discuss how to choose a good reduction. A natural solution is to select a reduced basis from the optimal reduction sequence. The algorithm to optimize enumeration by choosing proper reduction is described in algorithm 32.

```
Algorithm 32 OptimalEnum
Input: The log Gram-Schmidt norm vectors of the basis to be reduced: $\ell = \{\log(\|\mathbf{b}_1\|), \ldots, \log(\|\mathbf{b}_n\|)\}$,
    The approximation factor $\gamma$.
Output: Best enumeration parameter $\mathbf{B}_1, \ldots, \mathbf{B}_n$
 1: $[(\mathbf{B}_1^\star, \mathbf{B}_2^\star, \ldots), (T_1, T_2, \ldots)] \leftarrow$ OptimalSequence($\mathbf{B}^\star$);
 2: $t \leftarrow +\infty$;
 3: for $i = 2, 3, \ldots$ do
 4:   $t' \leftarrow \min_{\mathbf{R} \in \mathbb{R}} \frac{T_{enum}(\mathbf{B}_i^\star, \gamma, \mathbf{R}) + T_i}{p\mathrm{succ}(\mathbf{R})}$
 5:   if t'<t then
 6:     $t \leftarrow t'$
 7:   else
 8:     return $(\ell_i, T_i)$
 9:   end if
10: end for
```

We searched for the optimal enumeration strategy for dimension < 250, assuming LLL-reduced quality for all lattices in the beginning. Due to the limit of our simulation procedure, we limit our computation for BKZ-β with $\beta \geqslant 50$.

In the following table, we present the optimal reduction for dimension $70 \leqslant \beta \leqslant 250$. The last column shows the number of operations in $\log_2$. Today on a typical CPU one can perform $\approx 10^7$ operations per second.

Table 5.1: Cost of enumeration

β	p	Optimal preprocessing parameter	Operations in $\log_2$
70	5%	$\beta_1 = 50, p_1 = 0.05, Q_1 = 1$	28.86
75	5%	$\beta_2 = 35, p_2 = 0.15, Q_2 = 1; \beta_1 = 60, p_1 = 0.05, Q_1 = 1$	30.35
80	5%	$\beta_2 = 40, p_2 = 0.15, Q_2 = 2; \beta_1 = 60, p_1 = 0.05, Q_1 = 1$	32.71
90	2%	$\beta_2 = 50, p_2 = 0.15, Q_2 = 2; \beta_1 = 65, p_1 = 0.2, Q_1 = 2$	35.17

For dimension $\geqslant 100$, we simplied our search by only considering BKZ-β and $p = 1$. So this can only be considered as an conservative estimate. A sequence of β is given, where each β is following by a number in paranthesis, which indicates the number of rounds of BKZ-β reduction to perform. For example, given a LLL-reduced random basis of dimension 100, it is recommended to do 5 rounds of BKZ-50, followed by two rounds of BKZ-60, before performing an extreme pruning with 0.13% probability for each enumeration process.

Table 5.2: Cost of enumeration

n	p	Optimal preprocessing parameter	Operations in $\log_2$
100	0.13%	50(5) 60(2)	39.21
110	6.8×10^{-6}	50(5) 60(3) 70(2) 80(1)	44.23
120	6.3×10^{-7}	50(6) 60(4) 70(3) 80(2) 90(1)	49.00
130	2.6×10^{-9}	50(7) 60(4) 70(3) 80(2) 90(2) 100(1)	54.21
140	1.2×10^{-5}	50(8) 60(5) 70(3) 80(2) 90(2) 100(1)	59.91
150	6×10^{-6}	50(9) 60(5) 70(4) 80(2) 90(3) 100(2) 110(1)	65.77
160	2.5×10^{-6}	50(10) 60(6) 70(4) 80(3) 90(3) 100(3) 110(2) 120(1)	71.74
170	1.3×10^{-6}	50(11) 60(7) 70(4) 80(3) 90(3) 100(3) 110(2) 120(2) 130(1)	77.89
180	7.8×10^{-7}	50(12) 60(7) 70(5) 80(3) 90(3) 100(3) 110(3) 120(2) 130(2) 140(1)	84.31
190	3.4×10^{-10}	50(14) 60(8) 70(5) 80(3) 90(4) 100(4) 110(3) 120(3) 130(3) 140(1)	95.76
200	1.5×10^{-7}	50(15) 60(9) 70(6) 80(3) 90(4) 100(4) 110(3) 120(3) 130(3) 140(2) 150(2) 160(2)	99.32
210	1.7×10^{-7}	50(16) 60(10) 70(6) 80(4) 90(4) 100(4) 110(4) 120(3) 130(3) 140(3) 150(2) 160(2) 170(1)	104.45
220	1.2×10^{-7}	50(17) 60(11) 70(7) 80(4) 90(5) 100(5) 110(4) 120(3) 130(3) 140(3) 150(3) 160(2) 170(2) 180(1)	111.47
230	7.4×10^{-12}	50(19) 60(11) 70(7) 80(4) 90(5) 100(5) 110(4) 120(4) 130(3) 140(3) 150(3) 160(3) 170(1)	119.90
240	1.8×10^{-7}	50(20) 60(12) 70(8) 80(4) 90(6) 100(5) 110(4) 120(4) 130(4) 140(3) 150(3) 160(3) 170(3) 180(3) 190(1) 200(2)	126.92
250	2.5×10^{-8}	50(22) 60(13) 70(8) 80(5) 90(6) 100(6) 110(5) 120(4) 130(4) 140(4) 150(3) 160(3) 170(3) 180(4) 190(2) 200(2) 210(1)	133.55

The estimation of enumeration time depends on the optimal reduction sequence, while the estimation of reduction complexity depends on estimates of enumeration time. These two problems should be considered together.

In fact, the enumeration of dimension n will only make calls to reduction of block size $< n$, and similarly the enumeration of block size β will only make calls to enumeration with dimension $\leqslant \beta$. Therefore, we can regard this problem as a dynamic programing problem, whereas the enumeration cost table can be gradually filled in, from low dimension to higher dimension.

5.3 Experiments and challenges

5.3.1 Darmstadt SVP challenge

The Darmstadt SVP challenge provide a platform to test our algorithm of optimal enumeration. It is enough to find a shorter vector than $1.05 \cdot GH(\mathbf{B})$ for each dimension. We apply reductions to the lattices with increasing sequence of β. Finally an enumeration is applied on this basis.

For example for dimension 124, we consequtively applied BKZ-60, BKZ-80, BKZ-102 and BKZ-118 to its basis, before finally performing a pruned enumeration on the reduced basis.

The challenge provide multiple lattice basis for the same dimension. Therefore, instead of randomizing the basis and redoing the reduction and enumeration, as the algorithm assumption goes, we apply the same procedure on different basis. The success probability for a single enumeration is about 0.02 in dimension 124, hence we launched reduction for about 54 instances simutaneously.

Following list shows a few examples of SVP challenges we solved:

Table 5.3: New solutions for Darmstadt's SVP challenges [48]

Dim(Lattice)	Solution Norm	Previous norm	$\|\mathbf{b}_1\|/GH(\mathbf{B})$
126	2969	Unsolved	1.0436
124	2884	2936	1.0217
124	2936	Unsolved	1.0427
122	2913	Unsolved	1.0444

5.3.2 Darmstadt lattice challenge

Table 5.4: New Solutions for Darmstadt's lattice challenge [48]

Dim(lattice)	Dim(sublattice)	New norm	Previous norm	Ratio	Hermite factor
800	230	120.054	Unsolved		1.00978^{230}
775	230	112.539	Unsolved		1.00994^{230}
750	220	95.995	Unsolved		1.0976^{220}
725	210	85.726	100.90	0.85	1.00978^{210}
700	200	78.537	86.02	0.91	1.00993^{200}
675	190	72.243	74.78	0.97	1.00997^{190}
650	190	61.935	66.72	0.93	1.00993^{190}
625	180	53.953	59.41	0.91	1.00987^{180}
600	180	45.420	52.01	0.87	1.00976^{180}
575	180	39.153	42.71	0.92	1.00977^{180}
550	180	32.481	38.29	0.85	1.00955^{180}
525	180	29.866	30.74	0.97	1.00990^{180}

Darmstadt's lattice challenge is based on Ajtai's construction of hard lattice instances. For each dimension, the challenge is to find a vector of norm $< q$ in an Ajtai lattice [3], where q depends on the dimension; and try to minimize the norm. Before year 2010, the highest challenge solved was 725: the first solutions to all challenges in dimension 575 to 725 were found by Gama and Nguyen in 2008, using NTL's implementation of BKZ with SH pruning. All solutions were found by reducing appropriate sublattices of much smaller dimension (typically around 150-200), whose existence follows from the structure of Ajtai lattices: we followed the same strategy.

We used an relaxed enumeration on reduced basis to find the first ever solution up to dimension 800, and significantly shorter vectors in all challenges 525 to 725, as summarized in Table 5.4: the first column is the dimension of the challenge, the second one is the dimension of the sublattice we used to find the solution, the third one is the best norm found by BKZ 2.0, the fourth one is the previous best norm found by former algorithms, the fifth one is the ratio between norms, and the sixth one is the Hermite factor of the reduced basis of the sublattice, which turns out to be slightly below 1.01^{dim}. The

Table 5.5: Number of BKZ rounds required to break NTRUSign-157, as predicted by Alg. 20, starting from a BKZ-20 reduced basis

Blocksize β	106	110	115	120
Number of rounds	13	6	5	4

factor 1.01^{dim} was considered to be the state-of-the-art limit in 2008 by Gama and Nguyen [23], which shows the improvement.

5.4 Revising security estimate of NTRU lattices

In the NTRU cryptosystem [41], recovering the secret key from the public key amounts to finding a shortest vector in high-dimensional lattices of special structure. Because NTRU security estimates are based on benchmarks with BKZ, it is interesting to see the limits of this methodology.

In the original article [41], the smallest parameter set NTRU-107 corresponds to lattices of dimension 214, and it was estimated that key recovery would cost at least 2^{50} elementary operations. The best experimental result to recover the secret key for NTRU-107 by direct lattice reduction (without ad-hoc techniques like [23,50,52] which exploit the special structure of NTRU lattices) is due to May in 1999 [50], who reported one successful experiment using BKZ with SH pruning [80], after 663 hours on a 200-MHz processor, that is $2^{48.76}$ clock cycles. We performed experiments with BKZ 2.0 on 10 random NTRU-107 lattices: We applied LLL and BKZ-20, which takes a few minutes at most; We applied BKZ -65 with 5%-pruning, and checked every 5 minutes if the first basis vector was the shortest vector corresponding to the secret key, in which case we aborted. BKZ 2.0 was successful for each lattice, and the aborted BKZ-65 reduction took less than 2000s on the average, on a 2.83Mhz single core. So the overall running time is less than 40 minutes, that is $2^{42.62}$ clock cycles, which gives a speedup of at least 70, compared to May's experiment, and is significantly lower than 2^{50} elementary operations. Hence, there is an order of magnitude between the initial security estimate of 2^{50} and the actual security level, which is approximately at most 40-bit.

NTRUSign

Now, we revisit recent parameters for NTRUSign. In the recent article by Hoffstein *et al.* [40], a summary of the latest parameters for NTRU encryption and signature is given. In particular, the smallest parameter for NTRUsign is $(N, q) = (157, 256)$, which is claimed to provide 80-bit security against all attacks knowns, and 93-bit security against key-recovery lattice attacks. Similarly to [23], we estimate that finding the secret key is essentially as hard as recovering a vector of norm $< q$ in a lattice of dimension $2N = 314$ and volume q^N, which corresponds to a Hermite factor of 1.00886^{2N}. We ran our simulation algorithm for these parameters to guess how many rounds would be required, depending on the blocksize, starting from a BKZ-20 reduced basis (whose cost is negligible here): the results are summarized in Table 5.5. We deduce that six rounds of BKZ-110 should be sufficient to break NTRUSign-157, which corresponds to roughly 2^{11} enumerations. And according to Table 5.2, extreme pruning enumeration in blocksize 110 can be done by searching through at most 2^{47} nodes, which corresponds to roughly 2^{54} clock cycles on a typical processor. This suggests that the security level of the smallest NTRUSign parameter against state-of-the-art lattice attacks is at most 65-bit, rather than 93-bit, which is a significant gap.

CHAPTER 6

Approximate GCD Algorithm

Contents

6.1 Introduction

Following Gentry's breakthrough work [26], there is currently great interest on fully-homomorphic encryption (FHE), which allows to compute arbitrary functions on encrypted data. Among the few FHE schemes known [11, 18, 26, 28, 89], the simplest one is arguably the one of van Dijk, Gentry, Halevi and Vaikuntanathan [89] (vDGHV), published at EUROCRYPT '10. The security of the vDGHV scheme is based on the hardness of *approximate integer common divisors problems* introduced in 2001 by Howgrave-Graham [42]. In the general version of this problem (GACD), the goal is to recover a secret number p (typically a large prime number), given polynomially many near-multiples $x_0, \ldots, x_m$ of p, that is, each integer x_i is of the hidden form $x_i = pq_i + r_i$ where each q_i is a very large integer and each r_i is a very small integer. In the partial version of this problem (PACD), the setting is exactly the same, except that x_0 is chosen as an exact multiple of p, namely $x_0 = pq_0$ where q_0 is a very large integer chosen such that no non-trivial factor of x_0 can be found efficiently: for instance, [18] selects q_0 as a rough number, *i.e.* without any small prime factor.

By definition, PACD cannot be harder than GACD, and intuitively, it seems that it should be easier than GACD. However, van Dijk *et al.* [89] mention that there is currently no PACD algorithm that does not work for GACD. And the usefulness of PACD is demonstrated by the recent construction [18], where Coron, Mandal, Naccache and Tibouchi built a much more efficient variant of the FHE scheme by van Dijk *et al.* [89], whose security relies on PACD rather than GACD. Thus, it is very important to know if PACD is actually easier than GACD.

The hardness of PACD and GACD depends on how the q_i's and the r_i's are exactly generated. For the generation of [89] and [18], the noise r_i is extremely small, and the best attack known is simply gcd exhaustive search: for GACD, this means trying every noise (r_0, r_1) and check whether $\gcd(x_0 - r_0, x_1 - r_1)$ is sufficiently large and allows to recover the secret key; for PACD, this means trying every noise r_1 and check whether $\gcd(x_0, x_1 - r_1)$ is sufficiently large and allows to recover the secret key. In other words, if ρ is the bit-size of the noise r_i, then breaking GACD (resp. PACD) requires $2^{2\rho}$ (resp. 2^{ρ}) polynomial-time operations, for the parameters of [18, 89].

Our results

We present new algorithms to solve PACD and GACD, which are exponentially faster in theory and practice than the best algorithms considered in [18, 89]. More precisely, the running time of our new PACD algorithm is $2^{\rho/2}$ polynomial-time operations, which is essentially the "square root" of that of gcd exhaustive search. This directly leads to a new GACD algorithm running in $2^{3\rho/2}$ polynomial-time operations, which is essentially the $3/4$-th root of that of gcd exhaustive search. Our PACD algorithm relies on classical algorithms to evaluate univariate polynomials at many points, whose space requirements are not negligible. We therefore present additional tricks, some of which reduce the space requirements, while still providing substantial speedups. This allows us to experimentally break the FHE challenges proposed by Coron *et al.* in [18], which were assumed to have comparable security to the FHE challenges proposed by Gentry and Halevi in [27]: the latter GH-FHE-challenges are based on hard problems with ideal lattices; according to Chen and Nguyen [14], their security level are respectively 52-bit (Toy), 61-bit (Small), 72-bit (Medium) and 100-bit (Large). Table 6.1 gives benchmarks for our attack on the FHE challenges, and deduces speedups compared to gcd exhaustive search. We can conclude that the FHE challenges of [18] have a much lower security level than those of Gentry and Halevi [29].

Table 6.1: Time required to break the FHE challenges by Coron *et al.* [18]. Size in bits, running time in seconds for a single 2.27GHz-core with 72Gb of RAM. Timings are extrapolated for RAM > 72 Gb.

Name	Toy	Small	Medium		Large	
Size(public key)	0.95Mb	9.6Mb	89Mb		802Mb	
Size(modulus)	1.6×10^5	0.86×10^6	4.2×10^6		19×10^6	
Size(noise)	17	25	33		40	
Expected security level	≥ 42	≥ 52	≥ 62		≥ 72	
Running time of gcd-search	2420	8.3×10^6	1.96×10^{10}		1.8×10^{13}	
	40 mins	96 days	623 years		569193 years	
Concrete security level	≈ 42	≈ 54	≈ 65		≈ 75	
Running time of the	99	25665	1.635×10^7	6.6×10^6	6.79×10^{10}	2.9×10^8
new attack implemented	1.6 min	7.1 hours	190 days	76 days	2153 years	9 years
Parameters	$d = 2^8$	$d = 2^{12}$	$d = 2^{13}$	$d = 2^{15}$	$d = 2^{10}$	$d = 2^{19}$
Memory	≤ 130 Mb	≤ 15 Gb	≤ 72 Gb	≈ 240 Gb	≤ 72 Gb	≈ 25 Tb
Speedup	24	324	1202	2997	264	62543
New security level	≤ 37.7	≤ 45.7	≤ 55	≤ 54	≤ 67	≤ 59

Interestingly, we can also apply our technique to different settings, such as noisy factoring and attacking low-exponent RSA encryption. A typical example of noisy factoring is the following: assume that p is a divisor of a public modulus N, and that one is given a noisy version p' of p, which differs from p by at most k bits at unknown positions, can one recover p from (p', N) faster than exhaustive search? This may have applications in side-channel attacks. Like in the PACD setting, we obtain a square-root attack: for a 1024-bit modulus, the speedup can be as high as 1200 in practice. Similarly, we speed up several exhaustive search attacks on low-exponent RSA encryption.

Related work

Multipoint evaluation of univariate polynomials has been used in public-key cryptanalysis before. For instance, it is used in factoring (such as in the Pollard-Strassen factorization algorithm [68, 88] or in ECM speedup [58]), in the folklore square-root attack on RSA with small CRT exponents (mentioned by Boneh and Durfee [9], and described in [62, 70]), as well as in the recent square-root attack [17] by Coron, Joux, Mandal, Naccache and Tibouchi on Groth's RSA Subgroup Assumption [35]. But this does not imply that our attack is trivial, especially since the authors of [18] form a subset of the authors of [17]. In fact, in most cryptanalytic applications (including [17]) of multipoint evaluation, one is actually interested in the following problem: given two lists $\{a_i\}_i$ and $\{b_j\}_j$ of numbers modulo N, find a pair (a_i, b_j) such that $\gcd(a_i - b_j, N)$ is non-trivial. Instead, we use multipoint evaluation differently, as a way to compute certain products of m elements modulo N in $\tilde{O}(\sqrt{m})$ polynomial-time operations, where $\tilde{O}()$ is the usual notation hiding poy-logarithmic terms. More precisely, it applies to products $\prod_{i=1}^{m} x_i \bmod N$ which can be rewritten under the form $\prod_{j=1}^{m_1}\prod_{k=1}^{m_2}(y_j + z_k) \bmod N$ where both m_1 and m_2 are $O(\sqrt{m})$. The Pollard-Strassen factorization algorithm [68, 88] can be viewed as a special case of this technique: it computes $m! \bmod N$ to factor N.

In 2011, Cohn and Heninger [15] announced an attack on PACD and GACD based on Coppersmith's small root technique. This attack is interesting from a theoretical point of view, but from a practical point of view, we show in that for the FHE challenges of [18], it is expected to be slower than gcd exhaustive search, and therefore much slower than our attack.

In section 6.2, we describe our square-root algorithm for PACD, and apply it to GACD. In section 6.3, we discuss implementation issues, present several tricks to speed up the PACD algorithm in practice, and we discuss the impact of our algorithm on the fully-homomorphic challenges of Coron *et al.* [18]. Finally, we apply our main technique to different settings: noisy factoring (section 6.4) and attacking low-exponent RSA (section 6.5).

6.2 A Square-Root Algorithm for Partial Approximate Common Divisors

In this section, we describe our new square-root algorithm for the PACD problem, which is based on evaluating univariate polynomials at many points. In the last subsection, we apply it to GACD.

6.2.1 Overview

Consider an instance of PACD: $x_0 = pq_0$ and $x_i = pq_i + r_i$ where $0 \le r_i < 2^\rho, 1 \le i \le m$. We start with the following basic observation due to Nguyen (as reported in [18, section6.1]):

$$p = \gcd\left(x_0, \prod_{i=0}^{2^\rho - 1}(x_1 - i) \pmod{x_0}\right) \tag{6.1}$$

At first sight, this observation only allows to replace 2^ρ gcd computations (with numbers of size $\approx \gamma$ bits) with essentially 2^ρ modular multiplications (where the modulus has $\approx \gamma$ bits): the benchmarks of [18] report a speedup of ≈ 5 for the FHE challenges, which is insufficient to impact security estimates.

However, we observe that (6.1) can be exploited in a much more powerful way as follows. We define the polynomial $f_j(x)$ of degree j, with coefficients modulo x_0:

$$f_j(x) = \prod_{i=0}^{j-1} (x_1 - (x+i)) \pmod{x_0} \tag{6.2}$$

Letting $\rho' = \lfloor \rho/2 \rfloor$, we notice that:

$$\prod_{i=0}^{2^\rho - 1} (x_1 - i) \equiv \prod_{k=0}^{2^{\rho'+(\rho \bmod 2)}-1} f_{2^{\rho'}}(2^{\rho'}k) \pmod{x_0}.$$

We can thus rewrite (6.1) as:

$$p = \gcd\left(x_0, \prod_{k=0}^{2^{\rho'+(\rho \bmod 2)}-1} f_{2^{\rho'}}(2^{\rho'}k) \pmod{x_0}\right) \tag{6.3}$$

Clearly, (6.3) allows to solve PACD using one gcd, $2^{\rho'+(\rho \bmod 2)} - 1$ modular multiplications, and the multi-evaluation of a polynomial (with coefficients modulo x_0) of degree $2^{\rho'}$ at $2^{\rho'+(\rho \bmod 2)}$ points, where $\rho' + (\rho \bmod 2) = \rho - \rho'$. We claim that this costs at most $\tilde{O}(2^{\rho'}) = \tilde{O}(\sqrt{2^\rho})$ operations modulo x_0, which is essentially the square root of gcd exhaustive search. This is obvious for the single gcd and the modular multiplications. For the multi-evaluation part, it suffices to use classical algorithms (see [49,92]) which evaluate a polynomial of degree d at d points, using at most $\tilde{O}(d)$ operations in the coefficient ring. Here, we also need to compute the polynomial $f_{2^{\rho'}}(x)$ explicitly, which can fortunately also be done using $\tilde{O}(\sqrt{2^\rho})$ operations modulo x_0. We give a detailed description of the algorithms in the next subsection.

6.2.2 Description

We first recall our algorithm to solve PACD, given as algorithm 33, and which was implicitly presented in the overview.

Algorithm 33 Solving PACD by multipoint evaluation of univariate polynomials

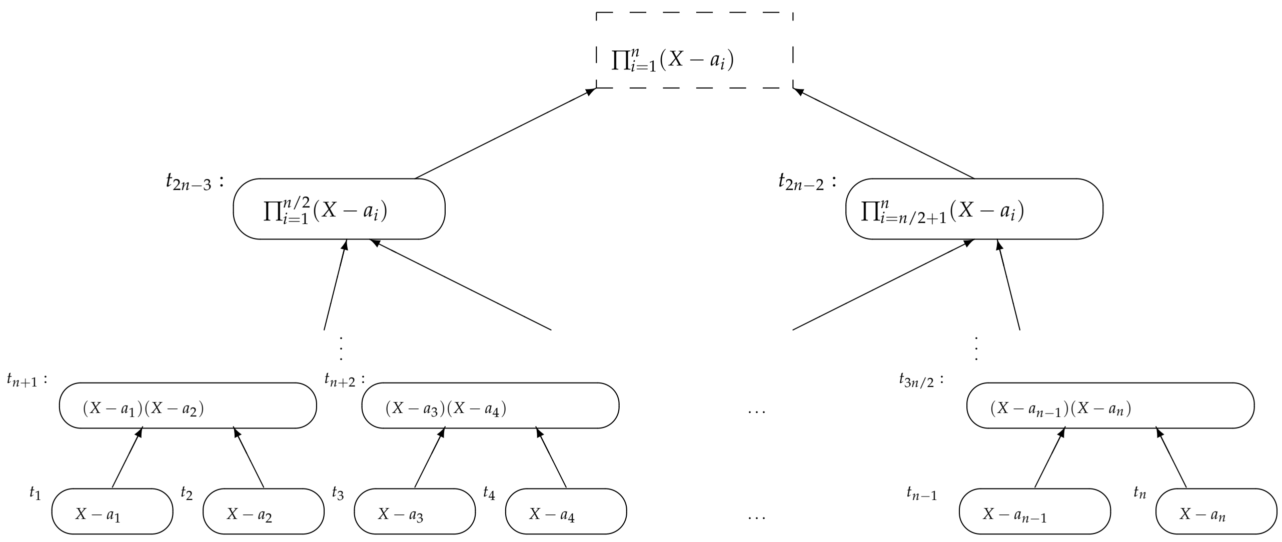
```
Input: An instance $(x_0, x_1)$ of the PACD problem with noise size $\rho$.
Output: The secret number $p$ such that $x_0 = pq_0$ and $x_1 = pq_1 + r_1$ with appropriate sizes.
1: Set $\rho' \leftarrow \lfloor \rho/2 \rfloor$.
2: Compute the polynomial $f_{2^{\rho'}}(x)$ defined by (6.2), using algorithm 34.
3: Compute the evaluation of $f_{2^{\rho'}}(x)$ at the $2^{\rho'+(\rho \bmod 2)}$ points $0, 2^{\rho'}, \ldots, 2^{\rho'}(2^{\rho'+(\rho \bmod 2)} - 1)$, using
   $2^{\rho \bmod 2}$ times algorithm 35 with $2^{\rho'}$ points. Each application of algorithm 35 requires the computa-
   tion of a product tree, using algorithm 34.
```

algorithm 33 relies on two classical subroutines (see [49,92]):

- a subroutine to (efficiently) compute a polynomial given as a product of n terms, where n is a power of two: algorithm 34 does this in $\tilde{O}(n)$ ring operations, provided that quasi-linear multiplication of polynomials is available, which can be achieved in our case using Fast Fourier techniques. This subroutine is used in Step 2. The efficiency of algorithm 34 comes from the fact that when the algorithm requires a multiplication, it only multiplies polynomials of similar degree.

- a subroutine to (efficiently) evaluate a univariate degree-n polynomial at n points, where n is a power of two: algorithm 35 does this in $\tilde{O}(n)$ ring operations, provided that quasi-linear polynomial remainder is available, which can be achieved in our case using Fast Fourier techniques. This subroutine is used in Step 3, and requires the computation of a tree product, which is achieved by algorithm 34. algorithm 35 is based on the well-known fact that the evaluation of a univariate polynomial at a point α is the same as its remainder modulo $X - \alpha$, which allows to factor computations using a tree.

Figure 6.1: Polynomial product tree $T = \{t_1, \ldots, t_{2n}\}$ for $\{a_1, \ldots, a_n\}$.



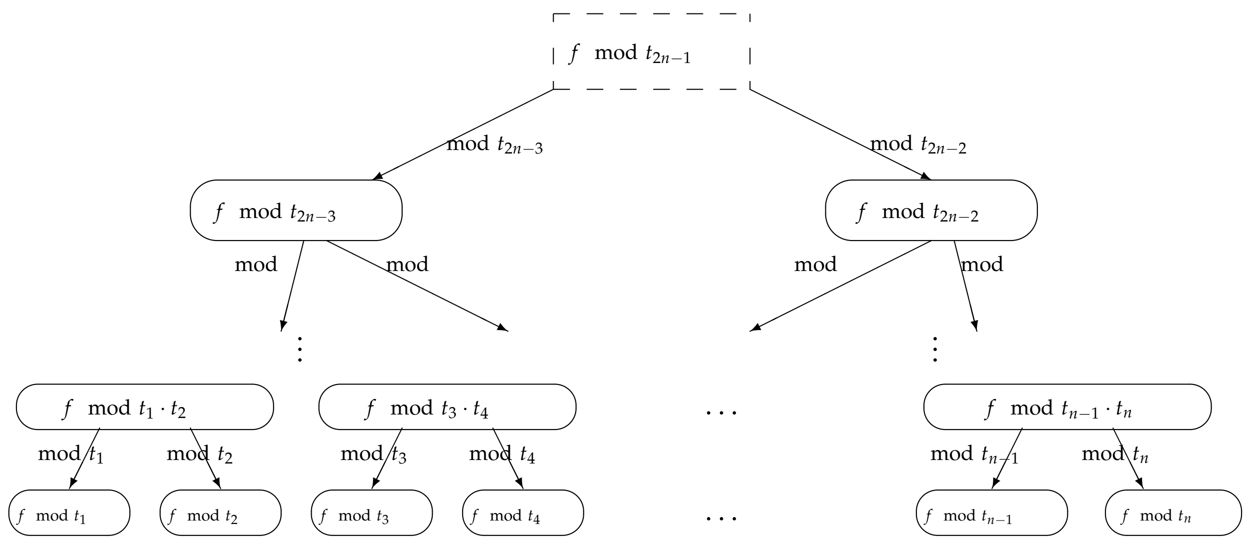
Algorithm 34 $[T, D] \leftarrow$`TreeProduct`(A)

Input: A set of $n = 2^l$ numbers $\{a_1, \ldots, a_n\}$.

Output: The polynomial product tree $T = \{t_1, \ldots, t_{2n-1}\}$, corresponding to the evaluation of points $A = \{a_1, \ldots, a_n\}$ as shown in Figure 6.1.
$D = [d_1, \ldots, d_{2n-1}]$ descendant indices for non-leaf nodes or 0 for leaf node.

```
1: for $i = 1 \ldots n$ do
2:     $t_i \leftarrow X - a_i$ {Initializing leaf nodes}
3:     $d_j \leftarrow 0$
4: end for
5: $i \leftarrow 1$ {Index of lower level}
6: $j \leftarrow n + 1$ {Index of upper level}
7: while $j \leqslant 2n - 1$ do
8:     $t_j \leftarrow t_i \cdot t_{i+1}$
9:     $d_j \leftarrow i$
10:    $i \leftarrow i + 2$
11:    $j \leftarrow j + 1$
12: end while
```

Figure 6.2: Evaluation on the polynomial tree $T = \{t_1, \ldots, t_{2n-1}\}$ for $\{a_1, \ldots, a_n\}$.



Algorithm 35 $V \leftarrow$ `RecursiveEvaluation`(f, t_i, D)

```
Input: A polynomial $f$ of degree $n$.
    A polynomial product tree rooted at $t_i$, and whose leaves are $\{X - a_k, \ldots, X - a_m\}$
    An array $D = [d_1, \ldots, d_{2n-1}]$ descendant indices for non-leaf nodes or 0 for leaf node.
Output: $V = \{f(a_k), \ldots, f(a_m)\}$
1: if $d_i = 0$ then
2:     return $\{f(a_i)\}$ {When $t_i$ is a leaf, we apply an evaluation directly.}
3: else
4:     $g_1 \leftarrow f \mod t_{d_i}$ {left subtree}
5:     $V_1 \leftarrow$ RecursiveEvaluation$(g_1, t_{d_i}, D)$
6:     $g_2 \leftarrow f \mod t_{d_i+1}$ {right subtree}
7:     $V_2 \leftarrow$ RecursiveEvaluation$(g_2, t_{d_i+1}, D)$
8:     return $V_1 \cup V_2$
9: end if
```

It follows that the running time of algorithm 33 is $\tilde{O}(2^{\rho'}) = \tilde{O}(\sqrt{2^\rho})$ operations modulo x_0, which is essentially the "square root" of gcd exhaustive search. But the space requirement is $\tilde{O}(2^{\rho'}) = \tilde{O}(\sqrt{2^\rho})$ polynomially many bits: thus, algorithm 33 can be viewed as a time/memory trade-off, compared to gcd exhaustive search.

6.2.3 Logarithmic speedup

In the previous analysis, the time complexity $\tilde{O}(n)$ actually stands for $O(n \log^2(n))$ ring multiplications. Interestingly, Bostan, Gaudry and Schost showed in [10] that when the structure of the factors are very regular, there is an algorithm which speeds up the theoretical complexity by a logarithmic term $\log(n)$. This BGS algorithm is tailored for the case where we want to estimate a function f on a set of points with what we call a hypercubic structure. An important subprocedure is `ShiftPoly` which, given as input a polynomial f of degree at most 2^d, and the evaluations of f on a set of 2^d points with hypercubic structure, outputs the evaluation of f on a shifted set of 2^d points, using $O(2^d)$ ring operations. More precisely:

Theorem 6.2.1. *(see Th. 5 of [10]) Let α, β be in ring $\mathbb{P}$ and d be in $\mathbb{N}$ such that $\mathbf{d}(\alpha,\beta,d)$ is invertible, with $\mathbf{d}(\alpha,\beta,d) = \beta \cdot 2 \ldots d \cdot (\alpha - d\beta) \ldots (\alpha + d\beta)$. And suppose also that the inverse of $\mathbf{d}(\alpha,\beta,d)$ is known. Let $F(\cdot) \in \mathbb{P}[X]$ of degree at most d and $x_0 \in \mathbb{P}$. There exists an algorithm `ShiftPoly` which, given as input $F(x_0)$, $F(x_0+\beta)$, ..., $F(x_0+d\beta)$, outputs $F(x_0+\alpha)$, $F(x_0+\alpha+\beta)$, ..., $F(x_0+\alpha+d\beta)$ in time $2M(d)+O(d)$ time and space $O(d)$. Here, $M(d)$ is the time of multiplying two polynomial of degree at most d.*

We note $E(k_1,\ldots,k_j)$ for $\left\{\sum_{i=1}^{j} p_{k_i} 2^{k_i}\right\}$ with each p_{k_i} ranging over $\{0,1\}$. This is the set enumerating all possibilities of bits $\{k_1,\ldots,k_j\}$. Given a set A and an element and p, $A+p$ is defined as$\{a+p \mid \forall a \in A\}$. Then we have

$$E(k_1,\ldots,k_{j+1}) = E(k_1,\ldots,k_j) \cup \left(E(k_1,\ldots,k_j) + 2^{k_{j+1}}\right).$$

This is what we call a set with hypercubic structure.

Given a linear polynomial $f(x)$ and a set with hypercubic structure of 2^ρ points, the proposed algorithm iteratively calls algorithm36 which uses `ShiftPoly`, and calculates the evaluation of $F_i(X) = \prod_{Y \in E(k_1,\ldots,k_i)} f(X+Y)$ on $E(b_{k-i},\ldots,k_\rho)$ until $i = \lfloor n/2 \rfloor$. The i-th iteration costs $O(2^i)$ ring operations, thus the total complexity amounts to $O(2^{\rho/2})$ ring operations.

Algorithm 36 i-th iteration of the evaluation of $F_i(X)$

```
Input: For $i = 1,\ldots,\lfloor \rho/2 \rfloor$, the evaluation of $F_i(X)$ on points $X \in E(k_{\rho-i+1},\ldots,k_\rho)$
Output: the evaluation of $F_{i+1}(X)$ on points $X \in E(k_{\rho-i},\ldots,k_\rho)$
1: $F_i(X)$ for $X \in E(k_{\rho-i+1},\ldots,k_\rho) + 2^{k_{\rho-i}} \leftarrow$ ShiftPoly($F_i(X), X \in E(k_{\rho-i+1},\ldots,k_\rho)$)
2: $F_i(X)$ for $X \in E(k_{\rho-i},\ldots,k_\rho) + 2^{k_{i+1}} \leftarrow$ ShiftPoly($F_i(X), X \in E(k_{\rho-i},\ldots,k_\rho)$)
3: $F_{i+1}(X) = F_i(X) \cdot F_i(X + 2^{k_{i+1}})$, for all $X \in E(k_{\rho-i},\ldots,k_\rho)$
```

6.2.4 Application to GACD

Any PACD algorithm can be used to solve GACD, using the trivial reduction from GACD to PACD based on exhaustive search over the noise r_0. More precisely, for an arbitrary instance of GACD:

$$x_i = pq_i + r_i \text{ where } 0 \leq r_i < 2^\rho, 0 \leq i \leq m$$

we apply our PACD algorithm for all pairs $(x_0 - r_0, x_1)$ where r_0 ranges over $\{0,\ldots,2^\rho - 1\}$.

It follows that GACD can be solved in $\tilde{O}(2^{3\rho/2})$ operations modulo x_0, using $\tilde{O}(2^{\rho/2})$ polynomially many bits. This is exponentially faster than the best attack of [89], namely gcd exhaustive search, which required $2^{2\rho}$ gcd operations. Note that in [89], another hybrid attack was described, where one performs exhaustive search over r_0 and factor the resulting number using ECM, but because of the large size of the prime factors (namely, a bit-length $\geq \rho^2$), this attack is not faster: it also requires at least $2^{2\rho}$ operations.

Following our work, it is noted with [19] that one can heuristically beat the GACD bound $\tilde{O}(2^{3\rho/2})$ using more samples of x_i, by removing the "smooth part" of $\gcd(y_1,\ldots,y_s)$ where $y_i = \prod_{j=0}^{2^\rho-1}(x_i - j)$ and s is large enough. The choice of s actually gives different time/memory trade-offs. For instances, if $s = \Theta(\rho)$, the running time is heuristically $\tilde{O}(2^\rho)$ poly-time operations and similar memory. From a practical point of view however, our attack is arguably more useful, due to memory requirements and better $\tilde{O}()$ constants.

6.3 Implementation of the Square-Root PACD Algorithm

We implemented both algorithm 33 and the logarithmic speedup using the NTL library [83]. In this section, we describe various tricks that we used to implement efficiently algorithm33. The implementation was not straightforward due to the size of the FHE challenges.

6.3.1 Obstructions

The main obstruction when implementing algorithm 33 is memory. Consider the Large FHE-challenge from [18]: there, $\rho = 40$, so the optimal parameter is $\rho' = 20$, which implies that $f_{2^{\rho'}}$ is a polynomial of degree 2^{20} with coefficients of size 19×10^6 bits. In other words, simply storing $f_{2^{\rho'}}$ already requires $2^{20} \times 19 \times 10^6$ bits, which is more than 2Tb, while we also need to perform various computations. This means that in practice, we will have to settle for suboptimal parameters.

More precisely, assume that we select an additional parameter d, which is a power of two less than $2^{\rho'}$. We rewrite (6.3) as:

$$p = \gcd\left(x_0, \prod_{k=0}^{2^\rho/d-1} f_d(dk) \pmod{x_0}\right) \tag{6.4}$$

This gives rise to a constrained version of algorithm 33, called algorithm 37.

Algorithm 37 Solving PACD by multipoint evaluation of univariate polynomials, using fixed memory

```
Input: An instance $(x_0, x_1)$ of the PACD problem with noise size $\rho$, and a polynomial degree $d$ (which
    must be a power of two).
Output: The secret number $p$ such that $x_0 = pq_0$ and $x_1 = pq_1 + r_1$ with appropriate sizes.
1: Compute the polynomial $f_d(x)$ defined by (6.2), using algorithm 34.
2: Compute the evaluation of $f_d(x)$ at the $2^\rho/d$ points $0, d, 2d, \dots, d(2^\rho/d - 1)$, using $2^\rho/d^2$ times algo-
    rithm 35 with $d$ points. Each application of algorithm 35 requires the computation of a product tree,
    using algorithm 34.
```

The running time of algorithm 37 is $\frac{2^\rho \tilde{O}(d)}{d^2}$ elementary operations modulo x_0, and the space requirement is $\tilde{O}(d)$ polynomially many bits. Note that each of the $2^\rho/d^2$ times applications of algorithm 35 can be done in parallel.

6.3.2 Tricks

The use of algorithm 37 allows several tricks, which we now present.

Minimizing the Product Tree

Each application of algorithm 35 requires the computation of a product tree, using algorithm 34. But this product tree requires to store $2n - 1$ polynomials. Fortunately, these polynomials have coefficients which are in some sense much smaller than the modulus x_0: this is because we evaluate the polynomial $f_d(x)$ at points in $\{0, \dots, 2^\rho - 1\}$, which is very small compared to the modulus x_0. However, a naive implementation would not exploit this. For instance, consider the polynomial $(X - a_1)(X - a_2) = X^2 - (a_1 + a_2)X + a_1a_2$, which belongs to the product tree. In a typical library for polynomial computations, the polynomial coefficients would be represented as positive residues modulo x_0. But if $a_1 + a_2$ is small, then $-(a_1 + a_2) + x_0$ is actually big. This means that many coefficients of the product tree polynomials will actually be as big as x_0, if they are represented as positive residues modulo x_0, which drastically reduces the choice of the degree d.

To avoid this problem, we instead slightly modify the polynomial $f_d(X)$, in order to evaluate at small negative numbers inside $\{0, \ldots, 1-2^\rho\}$, so that each polynomial of the product tree has "small" positive coefficients. This drastically reduces the storage of the product tree. More precisely, we rewrite (6.4) as:

$$p = \gcd\left(x_0, \prod_{k=0}^{2^\rho/d^2-1} \prod_{\ell=0}^{d-1} f'_{d,k}(-\ell d) \pmod{x_0}\right) \tag{6.5}$$

where

$$f'_{d,k}(x) = \prod_{i=0}^{d-1}(x_1 - 2^\rho - x + dk - i) \pmod{x_0} \tag{6.6}$$

Each product $\prod_{\ell=0}^{d-1} f'_{d,k}(-\ell) \pmod{x_0}$ is computed by applying algorithm 35 once, using the d points $0, -d, -2d, \ldots, -d(d-1)$.

Powers of Two

We need to compute the polynomial $f'_{d,k}(x)$ defined by (6.6) before each application of algorithm 35, using a simplified version of algorithm 34, which only computes the root rather than the whole product tree. However, notice that the degree of each polynomial of the product tree is exactly a power of two, which is the worst case for the polynomial multiplication implemented in the NTL library [83]. For instance, in NTL, multiplying two 512-degree polynomials with Medium-FHE coefficients takes 50% more time than multiplying two 511-degree polynomials with Medium-FHE coefficients.

To circumvent threshold phenomenons, we notice that each polynomial of the product tree is a monic polynomial, except the leaves (for which the leading coefficient is -1). But the product of two monic polynomials whose degree is a power of two can be derived efficiently from the product of two polynomials with degree strictly less than the power of two, using:

$$(X^n + P(X)) \times (X^n + Q(X)) = X^{2n} + X^n(P(X) + Q(X)) + P(X)Q(X).$$

We apply this trick to speed up the computation of the polynomial $f'_{d,k}(x)$.

Precomputations

Now that we use (6.5), we change several times the polynomial $f'_{d,k}(x)$, but we keep the same evaluation points 0, $-d$, $-2d$, $\ldots$, $-d(d-1)$, and therefore the same product tree. This allows to perform precomputations to speed up algorithm 35. Indeed, the main operation of algorithm 35 is computing the remainder of a polynomial with one of the product tree polynomials, and it is well-known that this can be sped up using precomputations depending on the modulus polynomial. One classical way to do this is to use Newton's method for remainder (algorithm 38). This algorithm requires the following notation: for any polynomial f of degree n and for any integer $m \geqslant n$, we define the m-degree polynomial $\text{rev}(f, m)$ as $\text{rev}(f, m) = f(1/X) \cdot X^m$. In algorithm 38, Line 1 is independent of f. Therefore, whenever one needs to compute many remainders with respect to the same modulus g, it is more efficient to precompute and store h, so that Line 1 does not need to be reexecuted. Hence, in an offline phase, we precompute and store (on a hard disk) the polynomial $\bar{g}$ of Line 1 for each product tree polynomial. And for each remainder required by algorithm 35, we execute the last two lines of algorithm 38.

Algorithm 38 Remainder using Newton's method (see [49, section7.2])

```
Input: Polynomials $f \in \mathbb{R}[X]$ of degree $2n-1$, $g \in \mathbb{R}[X]$ of degree $n$.
Output: The polynomial $h = f \mod g$
1: $\bar{g} \leftarrow \mathrm{Inverse}(\mathrm{rev}(g,n)) \mod X^n$
2: $s \leftarrow \mathrm{rev}(f, 2n-1) \cdot \bar{g} \mod X^n$
3: $h \leftarrow f - \mathrm{rev}(s, n-1) \cdot g$
```

It follows that each remainder operation of algorithm 35 is reduced to two polynomial multiplications.

The NTL library also contains routines for doing remainders with precomputations, but algorithm 38 turns out to be more efficient for our setting. This is because many factors impact the performance of polynomial arithmetic, such as the size of the modulus and the degree.

6.3.3 Logarithmic Speedup and Further Tricks

We also implemented the BGS algorithm described in section 6.2.3, which offers an asymptotical logarithmic speedup, but our implementation was not optimized due to lack of time: a good implementation would require the so-called middle product [10], which we instantiated by a normal product. On the FHE challenges, our implementation turned out to be twice as slow as algorithm 33 for Medium and Large, and marginally slower (resp. faster) for Toy (resp. Small).

Since memory is the main obstruction for choosing d, it is very important to minimize RAM requirements. Since algorithm 35 can be reduced to multiplications using precomputations, one may consider the use of special multiplication algorithms which require less memory than standard algorithms, such as in-place algorithms. We note that there has been recent work [39,75] in this direction, but we did not implement these algorithms. This suggests that our implementation is unlikely to be optimal, and that there is room for improvement.

6.3.4 New Security Estimates for the FHE Challenges

Table 6.1 reports benchmarks for our implementation on the fully-homomorphic-encryption challenges of Coron *et al.* [18], which come in four flavours: Toy, Small, Medium and Large. The security level ℓ is defined in [18] is defined as follows: the best attack should require at least 2^{ℓ} clock cycles on a standard single core. The row "Expected security level" is extracted from [18].

Our timings refer to a single 2.27GHz-core with 72Gb of RAM. First, we assessed the cost of gcd exhaustive search, by measuring the running time of the (quasi-linear) gcd routine of the widespread gmp library, which is used in NTL [83]: timings were measured for each modulus size of the four FHE-challenges. This gives the "concrete security level" row, which is slightly higher than the expected security level of [18].

We also report timings for our implementation of our square-root PACD algorithm: these timings are below the expected security level, which breaks all four FHE-challenges of [18]. For the Toy and Small challenges, the parameter d was optimal, and we did not require much memory: the speedup is respectively 24 and 324, compared to gcd exhaustive search. For the Medium and Large challenges, we had to use a suboptimal parameter d, due to RAM constraints: we used $d = 2^{13}$ (resp. $d = 2^{10}$) for Medium (resp. Large), instead of the optimal $d = 2^{16}$ (resp. $d = 2^{20}$). But the speedups are already significant: 1202 for Medium, and 264 for Large. The timings are obtained by suitably multiplying the running time of a single execution of algorithm 35 and algorithm 34: for instance, in the Large case, this online phase took between 64727s to 65139.4s, for 5 executions, and the precomputation storage was 21Gb.

Table 6.1 also provides extrapolated figures if the RAM was $\geq$ 72 Gb, which allows larger values of d: today, one can already buy servers with 4-Tb RAM. For the Large challenge, the potential speedup is over 60,000. Using a more optimized implementation, we believe it is possible to obtain larger speedups, so the New security level row should only be interpreted as an upper bound. But our implementation is already sufficient to show that the FHE-challenges of [18] fall short of the expected security level.

Hence, one needs to increase the parameters of the FHE scheme of [18], which makes it less competitive with the FHE implementation of [29]. It can be noted that the new security levels of the challenges of [18] are much lower than those given by [14] on the challenges of Gentry and Halevi [29], namely 52-bit (Toy), 61-bit (Small), 72-bit (Medium) and 100-bit (Large).

6.4 Applications to Noisy Factoring

Consider a typical "balanced" RSA modulus $N = pq$ where $p, q \leq 2\sqrt{N}$. A celebrated lattice-based cryptanalysis result of Coppersmith [16] states that if one is given half of the bits of p, either in the most significant positions, or the least significant positions, then one can recover p and q in polynomial time. Although this attack has been extended in several works (see [51] for a survey), all these lattice-based results require that the unknown bits are consecutive, or spread across extremely few blocks. This decreases its potential applications to side-channel attacks where errors are likely to be spread unevenly.

This suggests the following setting, which we call noisy factoring. Assume that one is given a noisy version p' of the prime factor p, which differs from p by at most k bits, not necessarily consecutive, under either of the following two cases:

- If the k positions of the noisy bits are known, we can recover p (and therefore q) by exhaustive search using at most 2^k polynomial-time operations: we stress that in this case, we assume that we do not know if each of the k bits has been flipped, otherwise no search would be necessary.
- If instead, none of the positions is known, but we know that exactly k bits have been modified, we can recover p by exhaustive search using at most $\binom{n}{k}$ polynomial-time operations, where n is the bit-length of p. If we only know an upper bound on the number of modified bits, we can simply repeat the attack with decreasing values of k.

These running times do not require that p and q are balanced.

In this section, we show that our previous technique for PACD can be adapted to noisy factoring, yielding new attacks whose running time is essentially the "square root" of exhaustive search, that is, $\tilde{O}(2^{k/2})$ or $\tilde{O}(\sqrt{\binom{n}{k}})$ polynomial-time operations, depending on the case. Finally, we also extend our method to fault attacks of RSA signature scheme.

6.4.1 Known positions

We assume that the prime number p has n bits, so that: $p = \sum_{i=0}^{n-1} p_i 2^i$, where $p_i \in \{0, 1\}$ for $0 \leqslant i \leqslant n-1$.

In this subsection, we assume that all the bits p_i are known, except possibly at k positions $b_1, \dots, b_k$, which we sort, so that: $0 \leq b_1 \leqslant \cdots \leqslant b_k < n$. Denote by $p^{(1)}, \dots, p^{(2^k)}$ the 2^k possibilities for p, when

$(p_{b_1}, \dots, p_{b_k})$ ranges over $\{0,1\}^k$. With high probability, all the $p^{(i)}$'s are coprime with N, except one, which would imply that:

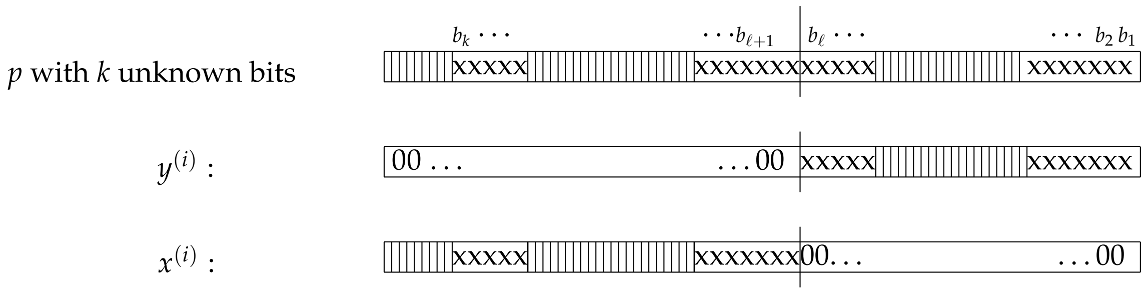
$$p = \gcd\left(N, \prod_{i=1}^{2^k} p^{(i)} (\text{mod } N)\right) \tag{6.7}$$

A naive evaluation of (6.7) costs 2^k modular multiplications, and one single gcd. We now show that this evaluation can be performed more efficiently using $\tilde{O}(2^{k/2})$ arithmetic operations with numbers with the same size as N.

The unknown bits $p_{b_1}, \dots, p_{b_k}$ can be regrouped into two sets $\{p_{b_1}, \dots, p_{b_\ell}\}$, and $\{p_{b_{\ell+1}}, \dots, p_k\}$ of roughly the same size $\ell = \lfloor k/2 \rfloor$, as illustrated in Figure 6.3:

- For $1 \leqslant i \leq 2^\ell$, let $y^{(i)} = \sum_{j=0}^{n-1} y_j^{(i)} 2^j$, where $y_j^{(i)} = \begin{cases} 0 & \text{if } j > b_\ell \\ t\text{-th bit of } i & \text{if } \exists t \leqslant \ell, j = b_t \\ p_j & \text{otherwise} \end{cases}$,
- For $1 \leqslant i \leq 2^{k-\ell}$, let $x^{(i)} = \sum_{j=0}^{n-1} x_j 2^j$, where $x_j^{(i)} = \begin{cases} 0 & \text{if } j \leqslant b_l \\ t\text{-th bit of } i & \text{if } \exists t > l, j = b_t \\ p_j & \text{otherwise} \end{cases}$,

Figure 6.3: Splitting the Unknown Bits in Two



Hence, by definition of $x^{(i)}$ and $y^{(i)}$, we have:

$$\prod_{i=1}^{2^k} p^{(i)} \equiv \prod_{i=1}^{2^\ell} \prod_{j=1}^{2^{k-\ell}} (x^{(j)} + y^{(i)}) \text{ (mod } N) \tag{6.8}$$

which gives rise to a square-root algorithm (algorithm 39) to solve the noisy factorization problem with known positions.

Algorithm 39 Noisy Factorization With Known Positions

Input: An RSA modulus $N = pq$ and the bits $p_0, \dots, p_{n-1}$ of p, except the k bits $p_{b_1}, \dots, p_{b_k}$, where the bit positions $b_1 \leq b_2 \leq \dots \leq b_k$ are known.

Output: The secret factor $p = \sum_{i=0}^{n-1} p_i 2^i$ of N.

```
1: Compute the polynomial $f(X) = \prod_{i=1}^{2^\ell} \left(X + y^{(i)}\right)$ mod $N$ of degree $2^\ell$, with coefficients modulo $N$, using algorithm 34.
2: Compute the evaluation of $f(X)$ at the points $\{x^{(1)}, \dots, x^{(2^{k-\ell})}\}$, using $1 + (k \bmod 2)$ times algorithm 35 with $2^\ell$ points.
3: return $p \leftarrow \gcd\left(N, \prod_{i=1}^{2^{k-\ell}} \left(f(x^{(i)})\right) \bmod N\right)$
```

Similary to section 6.2, the cost of algorithm 39 is $\tilde{O}(2^{k/2})$ polynomial-time operations. This is an exponential improvement over naive exhaustive search, but algorithm 39 requires exponential space. In practice, the improvement is substantial. Using our previous implementation, algorithm 39 gives a speedup of about 1200 over exhaustive division to factor a 1024-bit modulus, given a 512-bit noisy factor with 46 unknown bits at known positions.

Furthermore, in this setting, the points to be enumerated happen to satisfy the hypercubic property, thus we may apply the logarithmic speedup described in section 6.2.3.

Remember that the factor p can be calculated with formula (6.8). Now we can restate it as

$$\prod_{i=1}^{2^k} p^{(i)} \equiv \prod_{y\in E(b_{\ell+1},\dots,b_k)} \prod_{x\in E(b_1,\dots,b_\ell)} (x+y+M_p) \pmod N, \tag{6.9}$$

here $M_p = \sum_{i\notin\{b_1,\dots,b_k\}} p_i 2^i$ is the known bits of p . We define $F_i(X) = \prod_{y\in E(b_1,\dots,b_i)}(X+y+M_p)$.

Algorithm 40 Improved Noisy Factorization With Known Positions

Input: An RSA modulus $N = pq$ and a number p' differing from p by exactly k bits of unknown position.

Output: The secret factor p.

```
1: $F_0(0) \leftarrow M_p$
2: for $i = 1,\dots,\lfloor k/2\rfloor$ do
3:     Call algorithm 36 to calculate the evaluation of $F_i(X)$ on $E(b_{k-i}, b_k)$ given the evaluation of $F_{i-1}(X)$
       on $E(b_{k-i+1}, b_k)$
4: end for
5: if $k$ is odd then
6:     The evaluation $F_{\lfloor k/2\rfloor}(X)$ for $X \in E(b_{\lfloor k/2\rfloor+2},\dots,b_k) + 2^{b_{\lfloor k/2\rfloor+1}}$
         $\leftarrow$ShiftPoly$(F_{\lfloor k/2\rfloor}(X), X \in E(b_{\lfloor k/2\rfloor+2},\dots,b_k), 2^{b_{\lfloor k/2\rfloor+1}})$
7: end if
8: $p'' = \gcd\left(N, \prod_{X\in E(b_{\lfloor k/2\rfloor+1},\dots,b_k)} \left(F_{\lfloor k/2\rfloor}(X)\right)\right)$
9: return $p''$
```

As discussed in section 6.2, the cost of algorithm 40 is faster than algorithm 39 by a factor of $O(k)$ on a theoretical basis.

6.4.2 Unknown positions

In this subsection, we assume that p' differs from p by exactly k bits at unknown positions, and that p' has bit-length n. Our attack is somewhat reminiscent of Coppersmith's baby-step/giant-step attack on low-Hamming-weight discrete logarithm [87], but that attack uses sorting, not multipoint evaluation. To simplify the description, we assume that both k and n are even, but the attack can easily be adapted to the general case.

Pick a random subset S of $\{0,\dots,n-1\}$ containing exactly $n/2$ elements. The probability that S contains the indices of exactly $k/2$ flipped bits is: $\binom{n/2}{k/2}^2 / \binom{n}{k} \approx \frac{1}{\sqrt{k}}$. We now assume that this event holds, and let $\ell = \binom{n/2}{k/2}$. Similarly to the previous subsection, we define:

- Let $x^{(i)}$ for $1 \leqslant i \leq \ell$ be the numbers obtained by copying the bits of p' at all the positions inside S, and flipping exactly $k/2$ bits: all the other bits are set to zero.
- Let $y^{(i)}$ for $1 \leqslant i \leq \ell$ be the numbers obtained by copying the bits of p' at all the positions outside S, and flipping exactly $k/2$ bits: all the other bits are set to zero.

Now, with high probability over the choice of (p, q), we may write:

$$p = \gcd(N, \prod_{i=1}^{\ell}\prod_{j=1}^{\ell}(x^{(j)} + y^{(i)}) \pmod N) \tag{6.10}$$

which gives rise to a square-root algorithm (algorithm 41) to solve the noisy factorization problem with unknown positions.

Algorithm 41 Noisy Factorization With Unknown Positions

Input: An RSA modulus $N = pq$ and a number p' differing from p by exactly k bits of unknown position.
Output: The secret factor p.

```
1: repeat
2:     Pick a random subset $S$ of $\{0, \dots, n-1\}$ containing exactly $n/2$ elements.
3:     Compute the integers $x^{(i)}$ and $y^{(i)}$ for $1 \leqslant i \leq \ell = \binom{n/2}{k/2}$.
4:     Compute the polynomial $f(X) = \prod_{j=1}^{\ell}(X + y^{(j)}) \mod N$.
5:     Compute the evaluation of $f(X)$ at the $\ell$ points $\{x^{(1)}, \dots, x^{(\ell)}\}$.
6:     $p'' \leftarrow \gcd\left(N, \prod_{i=1}^{\ell}\left(f(x^{(i)})\right) \mod N\right)$
7: until $p'' > 1$
8: return $p''$
```

Similary to section 6.2, the expected cost of algorithm 41 is $\tilde{O}(\ell\sqrt{k})$ polynomial-time operations, where $\ell = \binom{n/2}{k/2}$ is roughly $\sqrt{\binom{n}{k}}$. This is an exponential improvement over naive exhaustive search, but algorithm 41 requires exponential space.

algorithm 41 is randomized, but like Coppersmith's baby-step/giant-step attack on low-Hamming-weight discrete logarithm [87], it can easily be derandomized using splitting systems. Deterministic versions are slightly less efficient, by a small polynomial factor: see [87].

6.4.3 Application to fault attacks on CRT-RSA signatures

Consider an RSA signature $s = m^d \bmod N$, where $N = pq$. In practice, this calculation is often accelerated using the Chinese remainder theorem. More precisely, s is derived from s_p and s_q, where $s_p = m^d \bmod p$ and $s_q = m^d \bmod q$. It is well-known that if a fault occurs during the computation of s_p (but not s_q), then the output s' will satisfy $s' \not\equiv m^d \bmod p$ and $s' \equiv m^d \bmod q$. Then the factorization of N is disclosed by

$$p = \gcd(s'^e - m \bmod N, N)$$

However, the message m may have already gone through some padding, so that only part of its bits are known. The positions of the unknown bits might be known, or not, which corresponds to the two situations presented in the previous discussion. The same technique to speed up the enumeration can be applied, by substituting (6.7) with

$$p = \gcd\left(N, \prod_{i=1}^{2^k}(s'^e - m^{(i)}) (\bmod N)\right). \tag{6.11}$$

6.5 Applications to Low-Exponent RSA

In this section, we show that our previous algorithms for noisy factoring can be adapted to attacks on both low-exponent RSA encryption. Consider an RSA ciphertext $c = m^e \bmod N$, where the public exponent e is very small. Assume that one knows a noisy version m' of the plaintext m, which differs from m by at most k bits, not necessarily consecutive, under either of the following two cases:

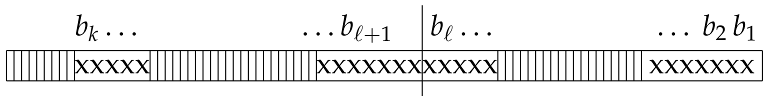
- If the k positions of the noisy bits are known, we can recover m by exhaustive search using at most 2^k polynomial-time operations: we stress that in this case, we assume that we do not know if each of the k bits has been flipped, otherwise no search would be necessary.
- If instead, none of the positions is known, but we know that exactly k bits have been modified, we can recover m by exhaustive search using at most $\binom{n}{k}$ polynomial-time operations, where n is the bit-length of m. If we only know an upper bound on the number of modified bits, we can simply repeat the attack with decreasing values of k.

This setting is usually called stereotyped RSA encryption [16]: there are well-known lattice attacks [16, 51] against stereotyped RSA, but they require that the unknown bits are consecutive, or split across extremely few blocks.

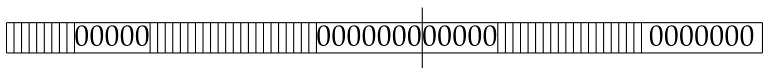
6.5.1 Known Positions

Assume that m is a plaintext of n bits, among which only k bits are unknown, whose (arbitrary) positions are $b_1, \ldots, b_k$. Let $c = m^e \mod N$ be the raw RSA ciphertext of m. If e is small (say, constant), we can "square root" the time of exhaustive search, using multipoint polynomial evaluation.

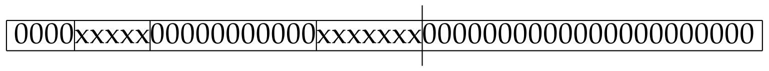
Let $\ell = \lceil (k - \log_2 e)/2 \rceil$, and assume that $k > 0$.

m :


Let m_0 be derived from m by keeping all the known $n - k$ bits, and setting all the k unknown bits to 0.

m_0 :


For $1 \leqslant i \leqslant 2^{k-\ell}$, let the x_i's enumerate all the integers when $(b_{\ell+1}, \ldots, b_k)$ ranges over $\{0,1\}^{k-\ell}$.

x_i :


Similarly, for $1 \leqslant j \leqslant 2^\ell$, let the y_j's enumerate all the integers when $(b_1, \ldots, b_\ell)$ ranges over $\{0,1\}^\ell$.

$$y_i : \quad \boxed{0000000000000000000000000000 \,|\, \text{xxxxx} \,|\, 0000000000 \,|\, \text{xxxxxxx}}$$

Thus, by construction, there is a unique pair (i, j) such that:

$$c = (m_0 + x_i + y_j)^e \bmod N.$$

Now, we define the polynomial $f(X) = \prod_{i=1}^{2^\ell} ((m_0 + y_i + X)^e - c) \bmod N$, which is of degree $e2^\ell$. If x_t corresponds to the correct guess for the bits $b_{\ell+1}, \ldots, b_k$, then $f(x_t) = 0$. Hence, if we evaluate $f(X)$ at $x_1, \ldots, x_{2^{k-\ell}}$, we would be able to derive the $k - \ell$ higher bits $b_{\ell+1}, \ldots b_k$, which gives rise to algorithm 42.

Algorithm 42 Decrypting Low-Exponent RSA With Known Positions

Input: An RSA modulus $N = pq$ and a ciphertext $c = m^e \bmod N$, where all the bits of m are known, except at k positions $b_1, \ldots, b_k$.
Output: The plaintext m.

```
1: Compute the polynomial $f(X) = \prod_{i=1}^{2^\ell} ((m_0 + y_i + X)^e - c) \bmod N$ of degree $e2^\ell$, with coefficients modulo $N$, using algorithm 34.
2: Compute the evaluation of $f(X)$ at the points $x^{(1)}, \ldots, x^{(2^{k-\ell})}$, using sufficiently many times algorithm 35.
3: Find the unique $i$ such that $f(x^{(i)}) = 0$.
4: Deduce from $x^{(i)}$ the bits $b_{\ell+1}, \ldots, b_k$.
5: Find the remaining bits $b_1, \ldots, b_\ell$ by exhaustive search.
```

By definition of ℓ, we have: $\sqrt{2^k/e} \leq 2^\ell \leq 2 \times \sqrt{2^k/e}$ and $sqrte2^k/2 \leq 2^{k-\ell} \leq 2 \times \sqrt{e2^k}$.

It follows that the overall complexity of algorithm 42. is $\tilde{O}(\sqrt{e2^k})$ polynomial-time operations, which is the "square root" of exhaustive search if e is constant.

6.5.2 Unknown Positions

In the previous section, we showed how to adapt our noisy factoring algorithm with known positions (algorithm 39) to the RSA case. Similarly, our noisy factoring algorithm with unknown positions (algorithm 41) can also be adapted. If the plaintext m is known except for exactly k unknown bit positions, then one can recover m using on the average $\tilde{O}(\ell\sqrt{ke})$ polynomial-time operations, where $\ell = \begin{pmatrix} n/2 \\ k/2 \end{pmatrix}$ is roughly $\sqrt{\begin{pmatrix} n \\ k \end{pmatrix}}$.

6.5.3 Variants

Our technique was presented to decrypt stereotyped low-exponent RSA ciphertexts, but the same technique clearly applies to a slightly more general setting, where the RSA equation is replaced by an arbitrary univariate low-degree polynomial equation. More precisely, instead of $c = m^e \bmod N$, we may assume that $P(m) \equiv 0 \pmod N$ where P is a univariate integer polynomial of degree e. This allows to adapt various attacks [16] on low-exponent RSA, such as randomized padding across several blocks.

Bibliography

[1] Divesh Aggarwal, Ueli Maurer, Igor Shparlinski, et al. The equivalence of strong rsa and factoring in the generic ring model of computation. In *WCC 2011-Workshop on coding and cryptography*, pages 17–26, 2011.

[2] Dorit Aharonov and Oded Regev. Lattice problems in np $\cap$ conp. *Journal of the ACM (JACM)*, 52(5):749–765, 2005.

[3] Miklós Ajtai. Generating hard instances of lattice problems. In *Proc. STOC '96*, pages 99–108. ACM, 1996.

[4] Miklós Ajtai. The shortest vector problem in l_2 is np-hard for randomized reductions. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 10–19. ACM, 1998.

[5] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proc. 33rd ACM Symp. on Theory of Computing (STOC)*, pages 601–610, 2001.

[6] Martin Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, Ludovic Perret, et al. On the complexity of the arora-ge algorithm against lwe. In *SCC–Third international conference on Symbolic Computation and Cryptography*, 2012.

[7] Sanjeev Arora, László Babai, Jacques Stern, and Z Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 724–733. IEEE, 1993.

[8] Elwyn Berlekamp, Robert McEliece, and Henk van Tilborg. On the inherent intractability of certain coding problems (corresp.). *Information Theory, IEEE Transactions on*, 24(3):384–386, 1978.

[9] Dan Boneh and Glenn Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):1339, 2000.

[10] Alin Bostan, Pierrick Gaudry, and Eric Schost. Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator. *SIAM Journal on Computing*, 36(6):1777–1806, 2007.

[11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. Cryptology ePrint Archive, Report 2011/344, 2011. `http://eprint.iacr.org/`.

[12] Daniel RL Brown. Breaking rsa may be as difficult as factoring. *Cryptology e-Print Archive: Report*, 380, 2005.

[13] Johannes Buchmann, Richard Lindner, and Markus Rückert. Explicit hard instances of the shortest vector problem. In *Post-Quantum Cryptography*, pages 79–94. Springer, 2008.

[14] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Proc. ASIACRYPT '11*, Lecture Notes in Comput. Sci. Springer, 2011.

[15] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. Cryptology ePrint Archive, Report 2011/437, 2011.

[16] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.

[17] Jean-Sébastien Coron, Antoine Joux, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Cryptanalysis of the rsa subgroup assumption from TCC 2005. In *Public Key Cryptography - Proc. PKC 2011*, volume 6571 of *Lecture Notes in Computer Science*, pages 147–155. Springer, 2011.

[18] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public-keys. In *Advances in Cryptology - Proc. CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[19] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 446–464, 2012.

[20] Luc Devroye. Non-uniform random variate generation, 1986. Available from `http://cg.scs.carleton.ca/~luc/rnbookindex.html`.

[21] Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *Information Theory, IEEE Transactions on*, 49(1):22–37, 2003.

[22] Nicolas Gama. *Géométrie Des Nombres Et Cryptanalyse de NTRU*. PhD thesis, Tesis Doctoral, 2008.

[23] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *Proc. EUROCRYPT '08*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008.

[24] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *Proc. EUROCRYPT '10*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010.

[25] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[26] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. STOC '09*, pages 169–178. ACM, 2009.

[27] Craig Gentry and Shai Halevi. Public challenges for fully-homomorphic encryption. Available at `https://researcher.ibm.com/researcher/view_project.php?id=1548`, 2010.

[28] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. Cryptology ePrint Archive, Report 2011/279, 2011. `http://eprint.iacr.org/`.

[29] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology - Proc. EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.

[30] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

[31] Oded Goldreich and Shafi Goldwasser. On the limits of non-approximability of lattice problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 1–9. ACM, 1998.

[32] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *Advances in Cryptology£CRYPTO'97*, pages 112–131. Springer, 1997.

[33] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and J-P Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999.

[34] Daniel Goldstein and Andrew Mayer. On the equidistribution of hecke points. In *Forum Mathematicum*, volume 15, pages 165–190. Berlin; New York: De Gruyter, c1989-, 2003.

[35] Jens Groth. Cryptography in subgroups of $\mathbb{Z}_n$. In *Proc. TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2005.

[36] Venkatesan Guruswami, Daniele Micciancio, and Oded Regev. The complexity of the covering radius problem. *Computational Complexity*, 14(2):90–121, 2005.

[37] Shai Halevi. Helib. A software library that implements homomorphic encryption (HE), April 2013.

[38] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 447–464. Springer, 2011.

[39] David Harvey and Daniel S. Roche. An in-place truncated fourier transform and applications to polynomial multiplication. In *Proc. ISSAC '10*, pages 325–329. ACM, 2010.

[40] Jeff Hoffstein, Nick Howgrave-Graham, Jill Pipher, and William Whyte. Practical lattice-based cryptography: NTRUEncrypt and NTRUSign. 2010. In [65].

[41] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory – Proc. ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[42] Nick Howgrave-Graham. Approximate integer common divisors. In *Proc. CaLC '01*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001.

[43] Subhash Khot. Inapproximability results for computational problems on lattices. In *The LLL Algorithm*, pages 453–473. Springer, 2010.

[44] Adeline Langlois and Damien Stehlé. Hardness of decision (r) lwe for any modulus. *IACR Cryptology ePrint Archive*, 2012:91, 2012.

[45] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *IACR Cryptology ePrint Archive*, 2012:90, 2012.

[46] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.

[47] Robert Lewand. *Monoalphabetic substitution ciphers*, chapter 1. MAA, 2000.

[48] Richard Lindner and Markus Rückert. TU Darmstadt lattice challenge. Available at `http://www.latticechallenge.org/`, 2010.

[49] Todd Mateer. *Fast Fourier Transform Algorithms with Applications*. PhD thesis, Clemson University, 2008.

[50] A. May. Cryptanalysis of NTRU–107. Draft of 1999, available on May's webpage.

[51] Alexander May. Using lll-reduction for solving rsa and factorization problems. In *The LLL algorithm*, pages 315–348. Springer, 2010. In [65].

[52] Alexander May and Joseph H. Silverman. Dimension reduction methods for convolution modular lattices. In *Proc. CaLC*, volume 2146 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2001.

[53] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42(44):114–116, 1978.

[54] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM journal on Computing*, 30(6):2008–2035, 2001.

[55] Daniele Micciancio. Cryptographic functions from worst-case complexity assumptions. In *The LLL Algorithm*, pages 427–452. Springer, 2010.

[56] Daniele Micciancio and Chris Peikert. Hardness of sis and lwe with small parameters. *IACR Cryptology ePrint Archive*, 2013:69, 2013.

[57] Adela Mihata and Emil Simion. New trends in lattice-based cryptography. *Acta Universitatis Apulensis*, 29:53–64, 2012.

[58] P. L. Montgomery. *An FFT Extension of the Elliptic Curve Method of Factorization*. PhD thesis, University of California Los Angeles, 1992.

[59] P. Q. Nguyen. Hermite's constant and lattice algorithms. 2010. In [65].

[60] Phong Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto '97. In *Proc. of Crypto '99*, volume 1666 of *LNCS*, pages 288–304. Springer, 1999.

[61] Phong Q. Nguyen. *Public-Key Cryptanalysis*, volume 477 of *Contemporary Mathematics*, chapter 4. AMS–RSME, 2009.

[62] Phong Q. Nguyen. Public-key cryptanalysis. In I. Luengo, editor, *Recent Trends in Cryptography*, volume 477 of *Contemporary Mathematics*. AMS–RSME, 2009.

[63] Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures. In *Advances in Cryptology-EUROCRYPT 2006*, pages 271–288. Springer, 2006.

[64] Phong Q. Nguyen and Damien Stehlé. Floating-point lll revisited. In *Advances in cryptology–Eurocrypt 2005*, pages 215–233. Springer, 2005.

[65] Phong Q. Nguyen and Brigette Vallée, editors. *The LLL Algorithm: Survey and Applications*. Information Security and Cryptography. Springer, 2010.

[66] Andrew Novocin, Damien Stehlé, and Gilles Villard. An lll-reduction algorithm with quasi-linear time complexity. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 403–412. ACM, 2011.

[67] Thomas Plantard and Michael Schneider. Creating a challenge for ideal lattices. *IACR Cryptology ePrint Archive*, 2013:39, 2013.

[68] John M. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521–528, 1974.

[69] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Golden Section Search in One Dimension*, chapter 10.1. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[70] Guopei Qiao and Kwok-Yan Lam. RSA signature algorithm for microcontroller implementation. In *Proc. CARDIS '98*, volume 1820 of *Lecture Notes in Computer Science*, pages 353–356. Springer, 2000.

[71] Oded Regev. Improved inapproximability of lattice and coding problems with preprocessing. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 363–370. IEEE, 2003.

[72] Oded Regev. Lattice-based cryptography. In *Advances in Cryptology-CRYPTO 2006*, pages 131–141. Springer, 2006.

[73] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[74] Oded Regev. The learning with errors problem. *Invited survey in CCC*, 2010.

[75] Daniel S. Roche. Space- and time-efficient polynomial multiplication. In *Proc. ISSAC '09*, pages 295–302. ACM, 2009.

[76] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer, 2004.

[77] Oliver Schirokauer, Damian Weber, and Thomas Denny. Discrete logarithms: The effectiveness of the index calculus method. In Henri Cohen, editor, *Algorithmic Number Theory*, volume 1122 of *Lecture Notes in Computer Science*, pages 337–361. Springer Berlin Heidelberg, 1996.

[78] Claus-Peter Schnorr. A more efficient algorithm for lattice basis reduction. *Journal of algorithms*, 9(1):47–62, 1988.

[79] 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.

[80] Claus-Peter Schnorr and H. H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *Proc. of Eurocrypt '95*, volume 921 of *LNCS*, pages 1–12. IACR, Springer-Verlag, 1995.

[81] Claus-Peter Schnorr and Horst Helmut Hörner. Attacking the chor-rivest cryptosystem by improved lattice reduction. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT*, volume 921 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1995.

[82] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[83] Victor Shoup. Number Theory C++ Library (NTL) version 5.4.1. Available at `http://www.shoup.net/ntl/`.

[84] Victor Shoup. A new polynomial factorization algorithm and its implementation. *J. Symb. Comput.*, 20(4):363–397, 1995.

[85] Carl Ludwig Siegel. *Lectures on the Geometry of Numbers*. Springer, 1989.

[86] Anders Södergren. On the poisson distribution of lengths of lattice vectors in a random lattice. *Mathematische Zeitschrift*, 269(3-4):945–954, 2011.

[87] Douglas R. Stinson. Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. *Math. Comput.*, 71(237):379–391, 2002.

[88] Volker Strassen. Einige Resultate über Berechnungskomplexität. *Jber. Deutsch. Math.-Verein.*, 78(1):1–8, 1976/77.

[89] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - Proc. EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[90] Peter van Emde-Boas. *Another NP-complete partition problem and the complexity of computing short vectors in a lattice*. Department, Univ., 1981.

[91] TN Venkataramana. Lattices in lie groups. In *Workshop on Geometric Group Theory*, 2010.

[92] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra (2nd ed.)*. Cambridge University Press, 2003.