RUHR-UNIVERSITÄT BOCHUM

RUB

RUHR-UNIVERSITÄT BOCHUM

Solving Subset Sum Problem using Lattice Reduction

Vukašin Karadžić

Master's Thesis – October 17, 2019.
Chair for Cryptography and IT Security.

Supervisor: Prof. Dr. Alexander May
Advisor: Dr. Claire Delaplace

Abstract

Both lattices and subset sum problem are well studied principles and problems in computer science. Using lattice reduction algorithms for solving the subset sum problem is also a studied approach which was first mentioned in 1983 in a paper by Ernest Brickell. Since then there were many improvements in this particular subfield, although the currently best algorithm for solving subset sum problems does not use lattice reduction, but is based on combinatorial approach. Lattices are gaining more and more attention because of their application to the post-quantum cryptography, and it would be interesting to see some breakthrough in the field of lattices, as that would certainly lead to a better algorithm for subset sum problems.

This thesis covers the principle of solving subset sum problem with lattice reduction algorithms, it provides more conclusive experimental results than currently available with the help of the latest state of the art lattice reduction library and explores the idea of using the combination of lattice reduction and combinatorial strategy in order to get better results.

Eidesstattliche Erklärung

Ich erkläre, dass ich keine Arbeit in gleicher oder ähnlicher Fassung bereits für eine andere Prüfung an der Ruhr-Universität Bochum oder einer anderen Hochschule eingereicht habe.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen, die anderen Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen kenntlich gemacht. Dies gilt sinngemäß auch für verwendete Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Ich erkläre mich damit einverstanden, dass die digitale Version dieser Arbeit zwecks Plagiatsprüfung verwendet wird.

Official Declaration

Hereby I declare, that I have not submitted this thesis in this or similar form to any other examination at the Ruhr-Universität Bochum or any other Institution of High School.

I officially ensure, that this paper has been written solely on my own. I herewith officially ensure, that I have not used any other sources but those stated by me. Any and every parts of the text which constitute quotes in original wording or in its essence have been explicitly referred by me by using official marking and proper quotation. This is also valid for used drafts, pictures and similar formats.

I also officially ensure, that the printed version as submitted by me fully confirms with my digital version. I agree that the digital version will be used to subject the paper to plagiarism examination.

Not this English translation, but only the official version in German is legally binding.

Date	author

Acknowledgements

First of all, I would like to thank my supervisors Prof. Dr. Alexandar May and Dr. Claire Delaplace. Thank you Prof. May for suggesting this beautiful topic to me and giving me starting pointers and clues about it. I owe my deepest gratitude to Dr. Delaplace, as she was there to answer with lot of patience any stupid question I had, and she unselfishly shared her knowledge and clever ideas with me.

I would like to thank Ruhr-University Bochum and all the teaching staff that I had in the past two years, as I learned a lot about cryptography and IT security in general. I must say thank you to group of my 12 colleagues that made me feel like home and helped me a lot with my life in Germany and integration.

Last but not least, I thank my parents, brother Rastko, sister Milica, dog Nera and girlfriend Marina for all the love and support they gave me while I was far away from them.

Contents

1 Introduction

The security of most modern cryptographic schemes is based on mathematical and computational problems. For instance, RSA cryptosystem is based on the hardness of factoring the product of two big different prime numbers, and code-based cryptosystems rely on the hardness of problems such as the syndrome decoding problem. If we could efficiently solve the underlying problems, we could easily break the corresponding cryptographic schemes.

In this thesis we look at lattice reduction algorithms, that are primarily used for attacking lattice-based cryptosystems, and use them to solve the **subset sum problem**. The search version of the problem can be defined as follows.

Definition 1.0.1 (`SubsetSumProblem`)**.** *Given* $a_1, a_2, ..., a_n, s \in \mathbb{Z}$*, find*

$$x_1, x_2, ..., x_n \in \{0, 1\} : \sum_{i=1}^{n} a_i x_i = s.$$

In the following sections we explain our motivation, give an brief overview of thesis organization and present the notation we use.

1.1 Motivation

In the past the solving of subset sum problem was investigated by the research community. Aside from lattice reduction attacks, there exists another class of algorithms used to solve subset sum problem: combinatorial algorithms.

Speaking about lattice reduction approach, there has been work on solving specific instances of the problem (i.e. [23, 11]), but not many papers working with the instances where a_i's are random were published. We wanted to execute broad experiments (using the new lattice reduction library published in Eurocrypt 2019) on random problems of different dimensions, check the current state of lattice reduction approach and review its performance in comparison with the combinatorial (meet-in-the-middle-like) algorithms. Furthermore, we came up with the idea of investigating how to mix the combinatorial and lattice reduction approach to improve the algorithms for the subset sum problem in general.

1.2 Organization of the Thesis

In Chapter 2 we take a broad look into the lattice theory, including the classical lattice problems and most important lattice reduction algorithms.

In Chapter 3 we take a look at the previous work regarding the subset sum problem, which includes the combinatorial algorithms and algorithms based on lattice reduction.

In Chapter 4 we describe the lattice reduction library that we used for our experiments, then test the parts of the aforementioned library in order to derive the algorithm to solve the problem. Finally, we test our algorithm on both random and modular subset sum problem instances that we previously generated, report the results and discuss them.

In Chapter 5 the idea of mixing combinatorial approach and lattice-reduction-based algorithms is explored.

Finally, in Chapter 6 closing conclusions are drawn and possible future work noted.

1.3 Notation

With bold letters $\mathbf{a}, \mathbf{b}, \mathbf{c}, ...$ we represent vectors. Vector $\mathbf{b}$ is a shorter representation for $(b_1, b_2, ..., b_m)$, where b_i are vector coordinates. With $b_{i,j}$ we represent the j-th coordinate of vector $\mathbf{b}_i$. We use $\lceil x \rfloor$ to represent the rounding of number x to the closest integer. We denote by B_r the set of first r vectors in lattice basis B. With $\|\mathbf{b}\|$ we denote the length of vector $\mathbf{b}$. With B^T we represent the transpose of matrix B. The distance between vector $\mathbf{v}$ and a set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, ...\}$ is represented by $\text{dist}(\mathbf{v}, \{\mathbf{x}_1, \mathbf{x}_2, ...\})$. In the thesis with 'subset sum problem' we sometimes mean 'instance of subset sum problem'. With $GSO(B)$ we denote the output of Gram-Schmidt orthogonalization process given an input B. Multinomial coefficient is presented and calculated with

$$\binom{n}{n_1, ..., n_k} = \frac{n!}{n_1! \cdots n_k!}.$$

We denote by $s \in_R S$ that element s is sampled uniformly at random from set S. Finally, by $poly(n)$ we denote a polynomial in n.

2 Lattices

Lattices are interesting for cryptography, as various cryptosystems can be constructed whose security is based on hardness of lattice problems. Their popularity is increasing, because the cryptosystems based on lattices are believed to be secure against attacker with quantum computational power. So, what is a lattice? Intuitively speaking, a lattice can be seen as a set of evenly distributed points in a space (i.e. $\mathbb{R}^n$). In the sections that follow, we cover the theory around lattices.

2.1 Basic Definitions

We start with the formal definition of a lattice.

Definition 2.1.1 (Lattice)**.** *Let* $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m \in \mathbb{R}^n$ *be linearly independent vectors. Lattice* $\mathcal{L}$ *is defined as*

$$\mathcal{L}(\mathbf{b}_1, ..., \mathbf{b}_m) = \{c_1\mathbf{b}_1 + \cdots + c_m\mathbf{b}_m \mid c_i \in \mathbb{Z}\}.$$

Set of vectors $\{\mathbf{b}_1, ..., \mathbf{b}_m\}$ is called *a* lattice *basis*. Lattices have more than one basis, and we call those bases *equivalent* to one another. The number of vectors in lattice basis m is also called a *rank* or *dimension* of the lattice. If $m = n$, it is said that lattice is *full-rank*. When we use the term vector length we are always talking about Euclidian norm meaning $\|(x_1, ..., x_n)\| = \sqrt{\sum_{i=1}^{n} x_i^2}$.

We can define a lattice in another way. We call this definition of a lattice an algebraic definition.

Definition 2.1.2 (Lattice (algebraic definition))**.** *Lattice* $\mathcal{L}$ *of* $\mathbb{R}^m$ *is a discrete nonempty subset of* $\mathbb{R}^m$ *closed under subtraction.*

Discrete in the Definition 2.1.2 means that distance between every two elements from $\mathcal{L}$ is at least $\delta > 0$. It can easily be showed that the aforementioned definition also satisfies the four group axioms (closure, associativity, identity element existence, inverse element existence), thus another definition of a lattice could be that a lattice is a discrete additive subgroup of group $(\mathbb{R}^m, +)$.

In order to get better intuition what a lattice is, we graphically present very simple lattice in Figure 2.1.

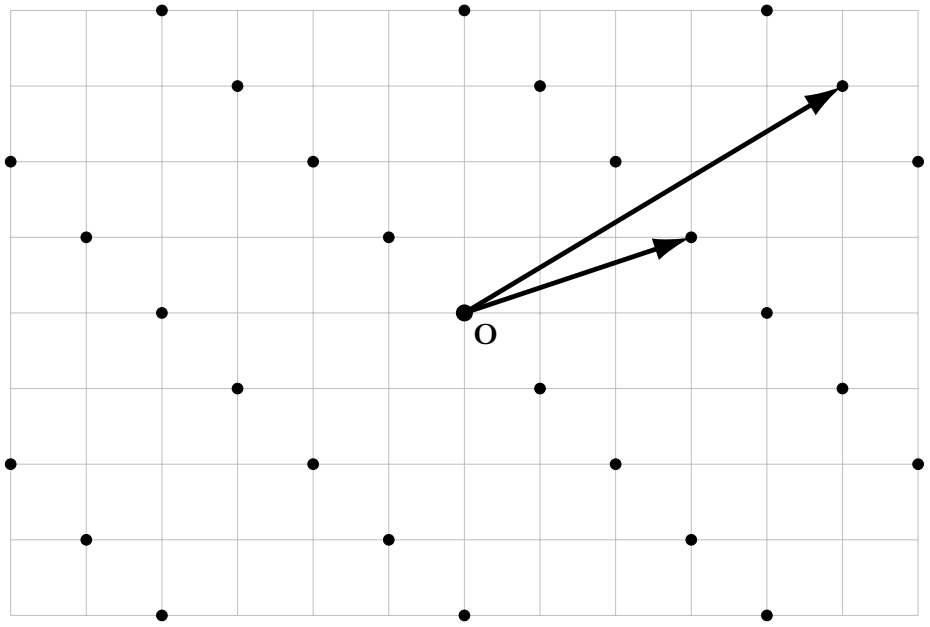


Figure 2.1: Example of a lattice in $\mathbb{R}^2$ with basis $\mathbf{b}_1 = (3, 1)$ and $\mathbf{b}_2 = (5, 3)$.

Another important notion in lattice world is the vector space $\text{Span}(B) = \{\sum_{i=1}^{n} r_i \mathbf{b}_i \mid r_i \in \mathbb{R}\}$ generated by basis B. The length of the shortest vector in the lattice is an important notion and as such we define it below.

Definition 2.1.3 ($\lambda_1(\mathcal{L})$)**.** *The length of the shortest nonzero vector in the lattice $\mathcal{L}$ is marked with $\lambda_1(\mathcal{L})$ and it is defined as*

$$\lambda_1(\mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L}(B), \mathbf{v} \neq \mathbf{0}} \|\mathbf{v}\| .$$

In thesis thesis we work with integer lattices, meaning that the basis vectors contain only integer coordinates, that is, $\mathbf{b}_i \in \mathbb{Z}^n$.

The *dot product* or *inner product* of vectors $\mathbf{v} = (v_1, ..., v_n)$ and $\mathbf{u} = (u_1, ..., u_n)$ is defined as $\langle \mathbf{v}, \mathbf{u} \rangle = \sum_{i=1}^{n} v_i u_i$. For two vectors $\mathbf{v}$ and $\mathbf{u}$ we say that they are *orthogonal* if $\langle \mathbf{v}, \mathbf{u} \rangle = 0$.

One of the most fundamental process when talking about lattices is the procedure of obtaining a set of orthogonal vectors (i.e. any two vectors from the set are orthogonal) given a set of linearly independent vectors. The classical algorithm for obtaining such set of orthogonal vectors is called *Gram-Schmidt Orthogonalization* (*GSO*).

Definition 2.1.4 (Gram-Schmidt Orthogonalization)**.** *Given a set of linearly independent*

vectors $\mathbf{b}_1, ..., \mathbf{b}_n$ *calculate the vectors* $\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n$ *iteratively as*

$$\tilde{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\tilde{\mathbf{b}}_j, \text{ where } \mu_{i,j} = \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle}.$$

Now we can define the volume of the lattice as $\text{Vol}(\mathcal{L}(B)) = \prod_{i=1}^{n} \left\| \tilde{\mathbf{b}}_i \right\|$ or "geometrically" as in Definition 2.1.5.

Definition 2.1.5 (Lattice Volume)**.** *Given a lattice basis* B *and lattice* $\mathcal{L}$ *generated by* B*, the volume of* $\mathcal{L}$ *is defined as*

$$\text{Vol}(\mathcal{L}) = \sqrt{\det(BB^T)}.$$

The volume of the lattice measures the average density of a lattice, in a sense that lattice point occupies on average $\text{Vol}(\mathcal{L})$ space in $\text{Span}(B)$ [9]. And geometrically we can think of $\text{Vol}(\mathcal{L})$ as the volume of parallelepiped spanned by B.

2.1.1 Sublattices and Projections

When we have a lattice of big dimension, it may be helpful to decompose them into smaller parts which are then easier to comprehend [9]. We present two ways of decomposing lattices into "smaller" structures: *sublattices* and *lattice projections.*

Definition 2.1.6 (Sublattice)**.** *Set* $\mathcal{L}'$ *is a sublattice of lattice* $\mathcal{L}$ *if*

$$\mathcal{L}' \subseteq \mathcal{L} \text{ and } \mathcal{L}' \text{ is a lattice itself.}$$

A more complicated way of lowering the dimension of the lattice is using projections, as we mentioned above. We now give two definitions, first we define what is vector projection π_i and then we define what is lattice projection.

Definition 2.1.7 (Vector Projection π_i)**.** *Given a basis* $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ *and corresponding Gram-Schmidt orthogonalization vectors* $\tilde{\mathbf{b}}_j$*, the projection of vector* $\mathbf{x}$ *on the orthogonal supplement of* $\text{Span}(\mathbf{b}_1, ..., \mathbf{b}_{i-1})$ *is denoted by* $\pi_i(\mathbf{x})$ *and calculated as*

$$\pi_i(\mathbf{x}) = \sum_{j=i}^{n} \frac{\langle \mathbf{x}, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle} \tilde{\mathbf{b}}_j.$$

Definition 2.1.8 (Lattice Projection)**.** *With* $\mathcal{L}_{[i,j]} = \mathcal{L}(B_{[i,j]})$*,* $1 \leq i < j \leq n$*, we call the lattice generated by the basis*

$$B_{[i,j]} = (\pi(\mathbf{b}_i), ..., \pi(\mathbf{b}_j)),$$

a projected lattice, where $\pi(\mathbf{b}_k)$ *is the projection of* $\mathbf{b}_k$ *on the orthogonal supplement of* $\text{Span}(\mathbf{b}_1, ..., \mathbf{b}_{i-1})$.

2.2 Lattice Problems

There are a couple of hard-to-solve problems in lattice field, on which the security of lattice-based cryptosystems is based. Two main problems are the Shortest Vector Problem (SVP) and Closest Vector Problem (CVP) which require us to find a very short lattice vector or a lattice vector close to target vector in $\mathbb{R}^n$. Other, still very important, problem is for instance the Bounded Distance Decoding (BDD) problem. The formal definitions of search version of SVP and CVP problems follow.

Definition 2.2.1 (SVP)**.** *Given lattice basis* $B = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$*, find a vector* $\mathbf{v} \in \mathcal{L}(B)$ *such that*

$$\|\mathbf{v}\| = \lambda_1(\mathcal{L}(B)).$$

Note that in the shortest vector problem our goal is to find shortest *non-zero* vector. Decisions problems associated with SVP and CVP are **NP**-hard.

Definition 2.2.2 (CVP)**.** *Given lattice basis* $B = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ *and a target vector* $\mathbf{t}$*, find a vector* $\mathbf{v} \in \mathcal{L}(B)$ *such that*

$$\|\mathbf{t} - \mathbf{v}\| \leq \|\mathbf{t} - \mathbf{u}\|$$

holds for every $\mathbf{u} \in \mathcal{L}(B)$*.*

In some cases we do not need to find exact shortest or closes vector, but enough short or enough close. If that is the case, we are then talking about *approximate* versions of shortest and closest vector problems. Usually the parameter in those approximate versions is noted with γ. The formal definitions follow.

Definition 2.2.3 (SVP_γ)**.** *Given lattice basis* $B = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$*, find a vector* $\mathbf{v} \in \mathcal{L}(B)$ *such that* $\mathbf{v} \neq \mathbf{0}$ *and*

$$\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\mathcal{L}).$$

Definition 2.2.4 (CVP_γ)**.** *Given lattice basis* $B = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ *and a target vector* $\mathbf{t}$*, find a vector* $\mathbf{v} \in \mathcal{L}(B)$ *such that*

$$\|\mathbf{t} - \mathbf{v}\| \leq \gamma \cdot \|\mathbf{t} - \mathbf{u}\|$$

holds for every $\mathbf{u} \in \mathcal{L}(B)$*.*

We must note that approximation problems for "small" γ are still very hard to solve. One more lattice problem is interesting to us, very similar to the CVP problem, and that is the Bounded Distance Decoding problem[1].

Definition 2.2.5 (BDD_γ)**.** *Given lattice basis* $B = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ *and a target vector* $\mathbf{t}$ *with the guarantee that for some* $\mathbf{u} \in \mathcal{L}(B) : \|\mathbf{t} - \mathbf{u}\| \leq \gamma \cdot \lambda_1(\mathcal{L}(B))$*, find a vector* $\mathbf{v} \in \mathcal{L}(B)$ *such that*

$$\|\mathbf{t} - \mathbf{v}\| \leq \|\mathbf{t} - \mathbf{u}\|$$

holds for every $\mathbf{u} \in \mathcal{L}(B)$*.*

[1] Definition taken from [4].

We can see that the difference between CVP and BDD is that in BDD we know that there exists a lattice vector inside a ball of some radius with center in target vector.

2.2.1 Solving SVP

We are interested in solving the SVP, because that is essential part of lattice reduction algorithm we use. As we mentioned before, finding a shortest vector in lattices of large dimensions is hard, but still there are methods used to solve this problem. Those methods are divided in two groups, *enumeration* and *sieving*. There also exist heuristical versions of enumeration and sieving algorithm that speed up the process, but they find a shortest vector with some probability less than 1. We will now describe both approaches more thoroughly.

2.2.1.1 Enumeration

Enumeration is technique for finding short(est) vectors in a lattice by enumerating all vectors shorter than a given bound. We could represent a general idea of lattice enumeration with the following procedure:

- Start by finding a non-zero vector shorter than a given bound L in a projected lattice $\mathcal{L}_{[n,n]}$.
- Expand the projected lattice with the next basis vector ($\mathcal{L}_{[n-1,n]}$) and look for combinations of vectors in the search space shorter than L, if such vectors are found, add them to the search space.
- Reiterate the step above until all basis vectors have been included.
- Finally, output the shortest vector found in the search space.

Two earliest works on the lattice enumeration were published more than 30 years ago [13, 21]. There have not been many research on enumeration topic long time after the earliest work, but in the last 15 years the enumeration got more popular again. As an example of further work we mention the paper from Gamma-Nguyen-Regev published in 2010 [16] and work in Chen's PhD thesis [9].

The algorithm runs through *enumeration tree* formed by all vector of a norm at most L in the projected lattices $\mathcal{L}_{[n,n]}, \mathcal{L}_{[n-1,n]}, ..., \mathcal{L}_{[1,n]}$. Enumeration tree is a tree of depth n, and for each $d \in \{0, ..., n\}$, the nodes at depth d are all the vectors from the rank-d projected lattice $\mathcal{L}_{[n-k,n]}$ [16]. The leaves in this enumeration tree are shorter than L.

There exist improvements of the algorithm called *pruned* enumeration. These types of enumeration algorithms eliminate certain branches in the enumeration tree. Pruning was introduced by Schnorr and Euchner [29]. Not all branches are equally likely to produce the solution and then, for the sake of algorithm speedup, during the execution some branches are eliminated and thus not visited. Still, we must note that in this case the solution returned may not be the shortest vector. The concept of *extreme* pruning was

introduced in [16]. Basic idea of extreme pruning is simple, and we present it in the form of algorithm in Algorithm 1.

Algorithm 1 Extreme pruning

Input: Basis B, bound L.
Output: Vector $\mathbf{v}$ shorter than L.

1: **while** vector $\mathbf{v}$ such that $\|\mathbf{v}\| < L$ is not found **do**
2: Randomize input basis B and perform basic reduction.
3: Run pruned enumeration that has small success probability.
4: **end while**

With success probability is meant the "probability" that the target (shortest) vector is still in the tree after branch elimination [16].

Let us mention that enumeration algorithms have super-exponential running time, but on the other side they are using just polynomial amount of memory.

2.2.1.2 Sieving

Lattice sieving follows the concept of general sieving principle in mathematics (one of the most known example is Sieve of Eratosthenes[2]): Start with a big set of objects and remove the unsuitable objects (while applying some elimination criterion) from the starting set until the given final condition is not fulfilled.
We could represent a general idea of lattice sieving with the following procedure:

- Sample a large number of long vectors from the lattice (starting sieve).
- Try combinations of the vectors currently in the sieve and if the resulting vector is shorter than a given bound L, include it into the next sieve.
- Reiterate the step above on the next sieve.
- Finally, output the shortest vector in the (last) sieve.

Lattice sieving was mentioned as early as in 2001 in a work by Ajtai, Kumar and Sivakumar [1]. Since then better algorithms appeared, out of which we mention the Nguyen-Vidick sieve [27] and Gauss sieve [25, 6]. Both of those algorithms, together with the Triple sieve algorithm [3] and simplified version of Becker-Gama-Joux sieve [7], are implemented in the library G6K [12] we later use and mention in Chapter 4. Let us mention that all lattice sieving algorithms have exponential running time and exponential memory requirements [6]. Next we detail a bit more on GaussSieve, and TripleSieve which we used for our experiments.

[2]Sieve of Eratosthenes is used for finding prime number up to some given limit.

GaussSieve

As stated in the beginning of the subsubsection 2.2.1.2, there are multiple versions of the GaussSieve, but we will present the original one published by Micciancio and Voulgaris in 2010 [25]. The GaussSieve simplified pseudocode is presented in Algorithm 2. In order to completely explain the algorithm, we must clarify what the other procedures in the pseudocode do and what are the variables introduced there. `SampleGaussian` procedure is sampling random lattice vectors with a distribution that is statistically close to a gaussian distribution [25, 27]. S is a data structure used to store temporarily removed vectors from the list L. `GaussReduce` procedure first reduces the vector $\mathbf{v}_{\text{new}}$ with list L meaning as long as there is some $\mathbf{v}' \in L$ such that

$$\left\|\mathbf{v}'\right\| \leq \mathbf{v}_{\text{new}} \text{ and } \left\|\mathbf{v}_{\text{new}} - \mathbf{v}'\right\| \leq \mathbf{v}_{\text{new}},$$

set $\mathbf{v}_{\text{new}} \leftarrow \mathbf{v}_{\text{new}} - \mathbf{v}'$. Then it analogously checks if any vector in list L can be reduced with updated $\mathbf{v}_{\text{new}}$ and every such vector is temporary removed from L and inserted to S. Finally the `GaussReduce` returns the updated $\mathbf{v}_{\text{new}}$. If the newly calculated $\mathbf{v}_{\text{new}}$ is a zero vector, that means we have a collision in the list. If the data structure S is empty, we know that all vectors in L are pairwise reduced and in that case a new vector is sampled. Finally, the number of samples needed to find the shortest vector (with high probability) cannot be bounded [25], and as a result the authors terminate the algorithm after a heuristically determined number of collisions in the list L had happened.

Algorithm 2 GaussSieve

Input: Basis B.
Output: Lattice vector $\mathbf{v}$ with $\mathbf{v} \leq \lambda_1(B)$.

$L \leftarrow \{\mathbf{0}\}, S \leftarrow \{\}, K \leftarrow 0$
while $K < c$ **do** # c heuristically determined
  if S is not empty **then**
    $\mathbf{v}_{\text{new}} \leftarrow S$.pop
  else
    $\mathbf{v}_{\text{new}} \leftarrow$ `SampleGaussian`(B)
  end if
  $\mathbf{v}_{\text{new}} \leftarrow$ `GaussReduce`$(\mathbf{v}_{\text{new}}, L, S)$
  if $\mathbf{v}_{\text{new}} = \mathbf{0}$ **then**
    $K \leftarrow K + 1$
  else
    $L \leftarrow L \cup \{\mathbf{v}_{\text{new}}\}$
  end if
end while

Authors showed that the algorithm in practice runs in time mostly $2^{0.48n}$ and uses at most $2^{0.21n}$ memory.

TripleSieve

In the lattice reduction library we used for our experiments, the TripleSieve algorithm is enabled by default in the subprocedure that we utilized. Because of that we wanted to shortly describe the TripleSieve. The basic sieving procedure of TripleSieve algorithm is presented in Algorithm 3.

Algorithm 3 TripleSieve

Input: Set of vectors L.
Output: Set of vectors L'.

$L' \leftarrow \{\}$
for all $\mathbf{u}, \mathbf{v}, \mathbf{w} \in L$ **do**
  if $\|\mathbf{u} \pm \mathbf{v} \pm \mathbf{w}\| \leq \gamma \cdot R$ **then** # $\gamma < 1$
    $L' \leftarrow L' \cup \{\mathbf{u} \pm \mathbf{v} \pm \mathbf{w}\}$
  end if
end for

Described with words, for every 3-tuple of vectors $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ we test if one of the combinations in $\mathbf{u} \pm \mathbf{v} \pm \mathbf{w}$ (we take *all* combinations into account) is shorter than $\gamma \cdot R$, where γ is smaller than 1 and R is a length limit. If so, this resulting vector is included into the next sieve. Having γ smaller than 1 ensures that the vectors "surviving" the sieve will be smaller with each iteration. The authors determined heuristically that algorithm runs in time $2^{0.5662n+o(n)}$ and uses $2^{0.1887n+o(n)}$ memory.

2.2.2 Solving CVP

Recall that in CVP problem instance we are given basis B and a target vector $\mathbf{t}$. We are then asked to find lattice vector $\mathbf{v}$ closest to $\mathbf{t}$. One popular algorithm used in finding the closest lattice vector to a given target vector is Babai Nearest Plane algorithm and we present it in Algorithm 4.

Algorithm 4 Babai Nearest Plane algorithm

Input: Basis $B = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$, $\mathbf{b}_i \in \mathbb{Z}^n$, target vector $\mathbf{t} \in \mathbb{Z}^n$.
Output: Vector $\mathbf{v} \in \mathcal{L}(B)$ s.t. $\|\mathbf{v} - \mathbf{t}\| \leq 2^{\frac{n}{2}} \operatorname{dist}(\mathbf{t}, \mathcal{L}(B))$.

δ-LLL reduce B with $\delta = \frac{3}{4}$
$\mathbf{v} \leftarrow \mathbf{t}$
for $i = n$ to 1 **do**
  $c_i \leftarrow \lceil \langle \mathbf{b}, \tilde{\mathbf{b}}_i \rangle / \langle \tilde{\mathbf{b}}_i, \tilde{\mathbf{b}}_i \rangle \rfloor$
  $\mathbf{v} \leftarrow c_i \mathbf{b}_i$
end for
$\mathbf{v} \leftarrow \mathbf{t} - \mathbf{v}$

As we will show later, LLL algorithm runs in polynomial time and together with the for loop we can conclude the Babai Nearest Plane algorithm runs in polynomial time as well.

2.3 Lattice Reduction Algorithms

Lattice reduction is a process of transforming the lattice basis into a "better" basis. The *reduced* basis generates the same lattice as the original one. It is helpful to find better bases, as that makes solving lattice problems easier. In most cases, and ours as well, better basis means basis having shorter and more orthogonal vectors. In Figure 2.2 two basis of the same lattice are presented, where the vectors marked in green represent the "good" basis and it can clearly be seen, that those vectors are shorter than their red counterparts.

One more concrete way of estimating the basis quality is with *root Hermite factor*, which we define in Definition 2.3.2.

Definition 2.3.1 (Root Hermite Factor)**.** *Root Hermite factor of basis B is defined as*

$$\delta(B) = \left(\frac{\|\mathbf{b}_1\|}{\mathrm{Vol}(\mathcal{L})^{\frac{1}{n}}} \right)^{\frac{1}{n}}.$$

Another important notion is *Hermite's constant.*

Definition 2.3.2 (Hermite's Constant)**.** *The supremum of*

$$\gamma_n = \frac{\lambda_1(\mathcal{L})^2}{\mathrm{Vol}(\mathcal{L})^{\frac{2}{n}}}$$

over all lattices $\mathcal{L}$ of dimension n is called Hermite's constant of dimension n.

Most used and famous algorithms for lattice reduction are LLL algorithm [24], named by its authors Lenstra, Lenstra and Lovász, the BKZ algorithm [29] introduced by Schnorr and Euchner and BKZ2.0 [10], the improvement of BKZ algorithm, presented by Chen and Nguyen.

In the rest of this chapter we elaborate on the three aforementioned lattice reduction algorithms and we will start with the oldest one, LLL.

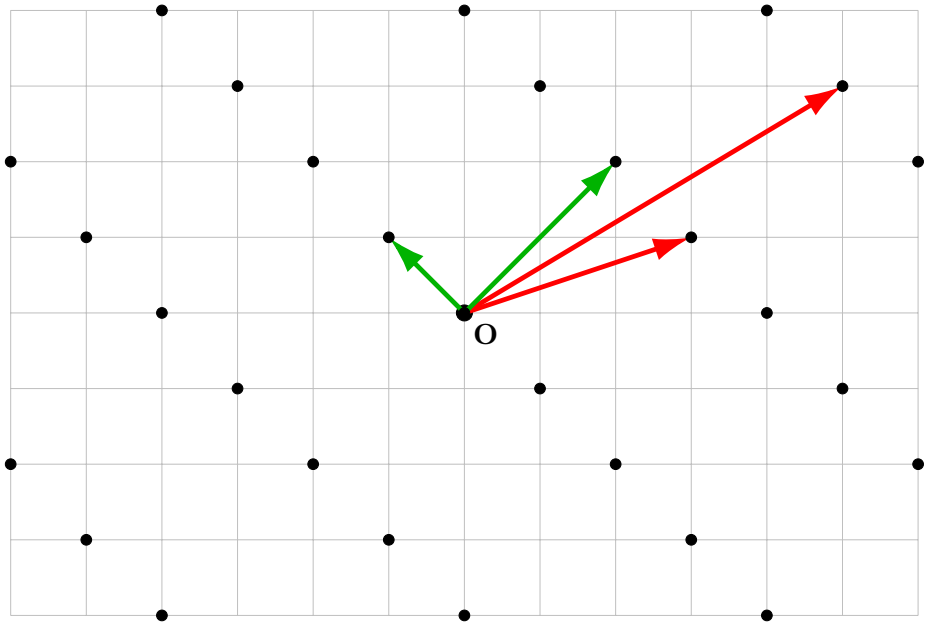


Figure 2.2: Example of "good" and "bad" lattice basis of a lattice example in $\mathbb{R}^2$. Bad basis is $\{(3,1),(5,3)\}$ and good basis is $\{(-1,1),(2,2)\}$.

2.3.1 LLL

When we run a lattice basis through LLL algorithm, we get an *LLL-reduced* basis as an output.

Definition 2.3.3 (LLL-reduced basis)**.** *Basis $B = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n)$ is called a δ-LLL-reduced basis if:*

1. *For all $1 \leq i \leq n$, $j < i$: $|\mu_{i,j}| \leq \frac{1}{2}$.*
2. *For all $1 \leq i < n$: $\delta \left\| \tilde{\boldsymbol{b}}_i \right\|^2 \leq \left\| \mu_{i+1,i} \tilde{\boldsymbol{b}}_i + \tilde{\boldsymbol{b}}_{i+1} \right\|^2$.*

We can rewrite second requirement as

$$\begin{aligned} \delta \left\| \tilde{\mathbf{b}}_i \right\|^2 \leq \left\| \mu_{i+1,i} \tilde{\mathbf{b}}_i + \tilde{\mathbf{b}}_{i+1} \right\|^2 &= \sum_{j=1}^{m} (\mu_{i+1,i} b_{i,j} + b_{i+1,j})^2 \\ &= \sum_{j=1}^{m} (\mu_{i+1,i} b_{i,j})^2 + 2 \sum_{j=1}^{m} \mu_{i+1,i} b_{i,j} b_{i+1,j} + \sum_{j=1}^{m} b_{i+1,j}^2 \\ &= \mu_{i+1,i}^2 \left\| \tilde{\mathbf{b}}_i \right\|^2 + \left\| \tilde{\mathbf{b}}_{i+1} \right\|^2, \end{aligned}$$

where the summand $2 \sum_{j=1}^{m} \mu_{i+1,i} b_{i,j} b_{i+1,j}$ was cancelled because $\tilde{\mathbf{b}}_i$ and $\tilde{\mathbf{b}}_{i+1}$ are orthogonal. Finally we get

$$\left\| \tilde{\mathbf{b}}_{i+1} \right\|^2 \geq (\delta - \frac{1}{4}) \left\| \tilde{\mathbf{b}}_i \right\|^2.$$

Here we can conclude that parameter δ must be greater than $\frac{1}{4}$ and it also must be smaller than 1. For δ-LLL-reduced basis certain properties hold (i.e. that the length of the first vector in the reduced basis is bounded by $c \cdot \lambda_1(\mathcal{L})$). In the following theorem we present one such fact.

Theorem 2.3.4. *Let $(\mathbf{b}_1, ..., \mathbf{b}_n)$ be a δ-LLL-reduced basis and let $\mathcal{L}$ be a lattice generated by this basis. Then it holds*

$$\|\mathbf{b}_1\| \leq \left(\frac{2}{\sqrt{4\delta - 1}}\right)^{n-1} \lambda_1(\mathcal{L}).$$

The proof of the theorem can be found in [28]. LLL algorithm is presented in Algorithm 5.

Algorithm 5 Lenstra-Lenstra-Lovász algorithm

Input: Basis $B = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$, $\mathbf{b}_i \in \mathbb{Z}^n$, parameter δ.
Output: δ-LLL-reduced basis $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$.

1: $\left\{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\right\} \leftarrow GSO(B)$.
2: **for** $i = 2$ to n **do**
3: **for** $j = i - 1$ to 1 **do**
4: $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lceil \langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle / \langle \tilde{\mathbf{b}}_j, \tilde{\mathbf{b}}_j \rangle \rfloor \mathbf{b}_j$
5: **end for**
6: **end for**
7: **if** for any $1 \leq i \leq n$: $\delta \left\|\tilde{\mathbf{b}}_i\right\|^2 > \left\|\mu_{i+1,i}\tilde{\mathbf{b}}_i + \tilde{\mathbf{b}}_{i+1}\right\|^2$ **then**
8: swap$(\mathbf{b}_i, \mathbf{b}_{i+1})$
9: goto 1
10: **end if**

It is obvious that the reduced basis still generates the same lattice, as only subtractions of other basis vectors multiplied by factor were done. The running time of LLL algorithm depends on the number of iteration (jumping from step 9. to step 1.) and how much steps does each iteration take. Both of the values are polynomial in M, where

$$M = \max\{n, l\} \text{ and } l = \log \max_i \|b_i\|,$$

thus the total running of LLL algorithm is polynomial as well. What about the output quality of LLL-reduced basis? It has been found that for random lattices a normal LLL-reduced basis will have $\delta(B) = 1.021$ [9].

2.3.2 BKZ

In this subsection we present the Block Korkin-Zolotareff algorithm. Before we go into details let us first define a *BKZ-reduced basis*.

Definition 2.3.5 (BKZ-reduced basis)**.** *Basis $B = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n)$ is called a δ-BKZ-reduced basis with a blocksize β if:*

1. *It is δ-LLL-reduced.*
2. *For all $1 \leq i \leq n$: $\left\|\tilde{\boldsymbol{b}}_i\right\| = \lambda_1(\mathcal{L}(B_{[i,k]})),\ k = \min(i + \beta - 1, n)$.*

As we can see, BKZ-reduced basis fulfills a stronger requirement than LLL-reduced basis, thus giving a better reduced basis. To present the algorithm itself, we use the interpretation presented in Chen's PhD thesis [9]. Given a basis B to be reduced, the BKZ algorithm iterates the subroutine `OneRoundBKZ` presented in Algorithm 6, until the subroutine passes through without having a single successful enumeration.

Algorithm 6 OneRoundBKZ subroutine

Input: Basis $B = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$, $\mathbf{b}_i \in \mathbb{Z}^n$, blocksize β.
Output: one round (partially) β-BKZ-reduced basis $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$, bool `enum` marking whether the enumeration was successful.

```
1: enum ← false
2: j ← 0
3: while j < n − 1 do
4:    j ← j + 1 (mod n − 1)
5:    k ← min(j + β − 1, n)
6:    h ← min(k + 1, n)
7:    Enumerate local block L_[j,k]
8:    if found shorter vector v than one available in basis B_[j,k] then
9:       enum ← true
10:      insert v into the basis in front of b_j
11:   end if
12:   LLL reduce the basis b_1, ..., b_h
13: end while
```

One downside of BKZ algorithm is that no good upper bound on its complexity is known [9]. It is known that the upper bound for the number of calls to the enumeration subroutine is exponential (although Hanrot, Pujol and Stehlé showed in [17] that already for the polynomial many calls to the enumeration subroutine the quality of the output basis is very close to the quality of the final output basis). It is also known that the cost of each enumeration is super-exponential in the blocksize β. Thus, the best bound currently available for whole BKZ is super-exponential as well.

We can model the output quality of β-BKZ-reduced basis with root Hermite factor value as well. For $\beta = 20$ the root Hermite factor of 20-BKZ-reduced basis is observed to be 1.013. For $\beta = 30$ that values decreases to 1.012. BKZ reduction with greater β outputs better reduced basis, as one would expect, meaning that the root Hermite value approaches 1 as β is increasing [9]. There is one more interesting relation between β,

Hermite's constant γ_β and $\|\mathbf{b}_1\|$, where $\mathbf{b}_1$ is from the reduced basis [9]. It is known that

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

holds.

2.3.3 BKZ2.0

BKZ2.0 is the improvement of BKZ algorithm, that will still output BKZ-reduced basis. The improvements are connected to the performance speedup. BKZ2.0 consists of following 4 enhancements over BKZ algorithm:

1. *Early-abort* - A limit on the number of enumeration oracle calls is set. As we know the number of oracle calls in BKZ seems exponential, with early-abort technique already exponential speed-up is achieved.
2. *Sound pruning* - Better pruning technique is introduced, and that is namely the mix of "normal" pruning and extreme pruning.
3. *Preprocessing of local bases* - Extension ensuring the local basis is much more reduced than LLL-reduced before invoking enumeration is added. The authors iteratively aborted-BKZ reduced the local basis, and the parameters for the local reduction were chosen heuristically based on β.
4. *Optimizing the enumeration bound* - Authors note that, as everything else with enumeration, the choice of the enumeration bound L influences the enumeration cost and depends on practical parameters. With the help of tests, they establish specific enumeration bound L, that leads to experimentally better results.

3 Solving the Subset Sum Problem

Currently there are multiple approaches regarding the solving of subset sum problem. In this thesis we focus only on one of them, and that is solving it with the help of lattice reduction algorithms. We do mention another type of the attacks and those are combinatorial attacks.

Let us first recall what is a subset sum problem.

Definition 3.0.1 (`SubsetSumProblem`)**.** *Given* $a_1, a_2, ..., a_n, s \in \mathbb{Z}$*, find*

$$x_1, x_2, ..., x_n \in \{0, 1\} : \sum_{i=1}^{n} a_i x_i = s.$$

To put it simply, we are given a set of n integers and a target sum s, and we are asked to find which of those integers contribute to the target sum s. The *solution vector* is the vector $(x_1, ..., x_n)$. We focus on instances where

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

and the following two kinds of problems:

- *Random subset sum problems*, where $a_i \in_R [1, 2^n]$.
- *Modular subset sum problems*, where $s = \sum_{i=1}^{n} a_i x_i \pmod{2^n}$.

Let us first explain why do we focus on problems where exactly half of the a_i's contribute to the sum. In most of the subset sum problem related papers the case where $\sum_{i=1}^{n} x_i = \frac{n}{2}$ is analyzed. In this setup the solution vector search space is the largest. That makes those instances harder than the problems that have solution vector with Hamming weight different than $\frac{n}{2}$. That is why we chose to analyze the $\sum_{i=1}^{n} x_i = \frac{n}{2}$ instances as well.

The *problem density* is defined with

$$d := \frac{n}{\log_2 \max a_i}.$$

The instances of random subset sum problem have density close to 1, and such problems are hard to solve [19]. The hardest subset sum problems are the ones with density exactly 1, and that was shown in [20]. Instances with lower density can be solved efficiently, and we mention exact density bounds and algorithms used for successful attacks against instances with density less than those bounds in Section 3.2.

3.1 Combinatorial Approach

Currently the best (to our knowledge) algorithm for solving subset sum problem is the one by Becker, Coron and Joux [5] from 2012. This work is the extension of the work of Howgrave-Graham and Joux [19] and both of those are based on the meet-in-the-middle approach. We will first explain how does the Howgrave-Graham-Joux algorithm works.

3.1.1 Howgrave-Graham-Joux Algorithm

First let us assume that $\sum_{i=1}^{n} x_i = \frac{n}{2}$. Then the algorithm idea is the following:

- Split a subset sum problem instance into two sub-instances of dimension n and Hamming weight $\frac{n}{4}$ each.

$$s = \sum_{i=1}^{n} a_i y_i + \sum_{i=1}^{n} a_i z_i, \text{ with } y_i, z_i \in \{0,1\}, \sum_{i=1}^{n} y_i = \sum_{i=1}^{n} z_i = \frac{n}{4}.$$

  Combination of two solutions, y_i's and z_i's, gives a solution to original subset sum instance, if they do not overlap (if there does not exist i such that $y_i = z_i = 1$).

- Represent each x_i as a pair (y_i, z_i), with 0 being replaced by $(0,0)$ and 1 with either $(1,0)$ or $(0,1)$.

- Choose a modulus M and a random $r \in \mathbb{Z}_M$ and consider only sub-instances such that

$$\sigma_1 = \sum_{i=1}^{n} a_i y_i \equiv r \pmod{M} \text{ and } \sigma_2 = \sum_{i=1}^{n} a_i z_i \equiv s - r \pmod{M}.$$

  The expected number of solutions to each of these two (modular) sub-instance problems is

$$L = \frac{\binom{n}{n/4}}{M}.$$

- Search for a collision between values σ_1 and $s - \sigma_2$ in the lists of the solutions.

The authors first claimed that algorithm running time was $\tilde{\mathcal{O}}(2^{0.3113n})$, but later it was showed that the running time actually is $\tilde{\mathcal{O}}(2^{0.337n})$ [5].

3.1.2 Becker-Coron-Joux Algorithm

Becker, Coron and Joux extended the Howgrave-Graham-Joux algorithm in a way that they allowed -1s to be part of the sub-instances solution as well. That way, the 1s in the original subset sum problem instance solution could be split into pairs $(0,1)$ and $(1,0)$ as before, but the 0s could be split into pairs $(0,0)$, $(1,-1)$ and $(-1,1)$. They choose parameter α and search for sub-instances whose solution contains $(1/4+\alpha)n$ ones

and αn minus ones. The number of such decompositions of original instance into two sub-instances is

$$N_D = \binom{n/2}{n/4}\binom{n/2}{\alpha n, \alpha n, (1/2 - 2\alpha)n}.$$

The running time and memory requirements from the main algorithm presented in the paper were $\tilde{\mathcal{O}}(2^{0.291n})$. This extension of the Howgrave-Graham-Joux algorithm shows that adding more representations of the original solution can reduce the running time, but Becker, Coron and Joux argue that there are many hard obstacles in achieving that.

In the paper an algorithm with constant memory is also presented, with the running time $\tilde{\mathcal{O}}(2^{0.72n})$. Later in Chapter 4 we discuss how both versions of the Becker-Coron-Joux algorithm behave.

3.2 Lattice Reduction Approach

Solving of subset sum problem using lattice reduction was first presented in the paper by Ernest Brickell [8] in 1983. In the following years there were numerous papers improving Brickell's work and further exploring the combination of lattice reduction and subset sum algorithm. Lagarias and Odlyzko presented an algorithm that can solve almost all problems of density less than 0.6463 [23]. Two years later Coster et al. presented a modified version of Lagarias-Odlyzko algorithm that can solve almost all subset sum problems of density less than 0.9408 [11]. The work from Schnorr and Euchner was the first attempt to solve subset sum problems of density close to 1 [29], and since then that proved to be a hard task.

The paper which served as a reference point for this thesis is the one by Schnorr and Shevchenko [30] and in this work the authors report some results on solving the random subset sum problem of dimension 80. The theory in the paper is largely based on the aforementioned paper by Schnorr and Euchner [29] from 1994. In that paper also the BKZ algorithm was presented, but we will only deal with the part of the paper that concerns the usage of lattice reduction to solve subset sum problem.

3.2.1 Schnorr-Euchner Results

If we are given the subset sum problem instance $(a_1, a_2, ..., a_n, s)$ the authors proposed forming the following $(n+2)$-by-$(n+1)$ lattice basis:

$$B = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_{n+1}] = \begin{bmatrix} 2 & 0 & \cdots & 0 & 1 \\ 0 & 2 & \cdots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & 1 \\ na_1 & na_2 & \cdots & na_n & ns \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}, \tag{3.1}$$

where the vectors $\mathbf{b}_i$ are represented as column vectors.

Theorem 3.2.1. *If a lattice vector* $\mathbf{z} = (z_1, ..., z_n + 2) \in \mathcal{L}(B)$ *satisfies*

$$|z_1| = \cdots = |z_n| = |z_{n+2}| = 1 \text{ and } z_{n+1} = 0, \tag{3.2}$$

then the solution for the given subset sum problem instance is

$$x_i = \frac{|z_i - z_{n+2}|}{2}, \quad i \in \{1, ..., n\}.$$

Proof. If vector $\mathbf{z} \in \mathcal{L}(B)$ it means that there exist coefficients $c_1, ..., c_{n+1} \in \mathbb{Z}$ such that $c_1\mathbf{b}_1 + \cdots + c_{n+1}\mathbf{b}_{n+1} = \mathbf{z}$. Writing that equation down by coordinates we get:

$$\begin{aligned}
z_1 &= 2c_1 + c_{n+1} && (\,|z_1| = 1\,)\\
z_2 &= 2c_2 + c_{n+1} && (\,|z_2| = 1\,)\\
&\vdots && \vdots\\
z_n &= 2c_n + c_{n+1} && (\,|z_n| = 1\,)\\
z_{n+1} &= n\sum_{i=1}^{n} c_i a_i + c_{n+1} ns && (\,z_{n+1} = 0\,)\\
z_{n+2} &= c_{n+1} && (\,|z_{n+2}| = 1\,)
\end{aligned}$$

Suppose without loss of generality that $z_{n+2} = c_{n+1} = 1$.
Then c_i can be -1 or 0, for $1 \le i \le n$. It also holds $z_{n+1} = n\sum_{i=1}^{n} c_i a_i + ns = 0$. From the last two observations we can conclude that coefficients c_i uniquely determine which a_i contribute to the sum and as we know that $c_i = \frac{z_i - z_{n+2}}{2}$ it finally follows that

$$x_i = \frac{|z_i - z_{n+2}|}{2}, \text{ for } 1 \le i \le n$$

is the solution of the subset sum problem. □

3.2.2 Schnorr-Shevchenko Algorithm

The lattice used in Schnorr-Shevchenko paper is a little bit different than the one presented in Equation (3.1) and it is given by the following $(n + 3)$-by-$(n + 1)$ matrix:

$$B = \begin{bmatrix}
2 & 0 & \cdots & 0 & 1\\
0 & 2 & \cdots & 0 & 1\\
\vdots & \vdots & \ddots & \vdots & \vdots\\
0 & 0 & \cdots & 2 & 1\\
Na_1 & Na_2 & \cdots & Na_n & Ns\\
0 & 0 & \cdots & 0 & 1\\
N & N & \cdots & N & \frac{n}{2}N
\end{bmatrix}. \tag{3.3}$$

Note that the difference is having N instead of n, where $N > \sqrt{n}$, and there is an additional row $(N, ..., N, \frac{n}{2}N)$ in the matrix.
Also, the vector $\mathbf{z}$ from this "extended" matrix has length $n + 3$, and in addition to the requirements given in (3.2) there is a new requirement:

$$z_{n+3} = 0.$$

The additional row and new requirement for z_{n+3} assure that Hamming weight of the solution will be exactly $\frac{n}{2}$. Let us now explain why is having $N > \sqrt{n}$ important. Suppose we have a lattice vector $\mathbf{v} = B\mathbf{c}$. Then $\mathbf{v}$ would look like

$$\mathbf{v} = (2c_1 + c_{n+1}, ..., 2c_n + c_{n+1}, N\sum_{i=1}^{n} c_i a_i + c_{n+1}Ns, c_{n+1}, N\sum_{i=1}^{n} c_i + c_{n+1}\frac{n}{2}N.$$

Furthermore, if we take a look at the norm of $\mathbf{v}$ we would get

$$\|\mathbf{v}\|^2 = \sum_{i=1}^{n}(2c_i + c_{n+1})^2 + N^2(\sum_{i=1}^{n} c_i a_i + c_{n+1}s)^2 + c_{n+1}^2 + N^2(\sum_{i=1}^{n} c_i + c_{n+1}\frac{n}{2})^2.$$

Suppose vector $\mathbf{v}$ is short and has length $\|\mathbf{v}\| \leq \sqrt{n+1}$. Finally, if $N > \sqrt{n}$ it can be seen that the following three points would hold:

- $\sum_{i=1}^{n} c_i a_i + c_{n+1}s = 0$,
- $\sum_{i=1}^{n} c_i + c_{n+1}\frac{n}{2} = 0$ and
- $\sum_{i=1}^{n}(2c_i + c_{n+1})^2 + c_{n+1}^2 \leq n + 1$.

Were $N \leq \sqrt{n}$ (i.e. $N = 1$), we would have more possibilities for vector $\mathbf{c}$ that would lead to "incorrect" vector $\mathbf{v} \leq \sqrt{n+1}$.

The pseudocode of the algorithm authors used is given in Algorithm 7.

Algorithm 7 Schnorr-Shevchenko algorithm

Input: Subset sum problem instance $(a_1, ..., a_n, s)$.
Output: Solution $\{x_i \mid 1 \leq i \leq n\}$ for the instance $(a_1, ..., a_n, s)$.

```
Transform a instance into a basis B like in (3.3).
for β ∈ {2, 4, 8, 16, 32} do
    Permute the columns of basis B appropriately.
    Iteratively β-BKZ-reduce without pruning current state of basis B.
    if any vector z in reduced basis satisfies the requirements then
        return calculated set of solution coefficients x_i.
    end if
end for
for β ∈ {30, 31, ..., 61} do
    Independently β-BKZ-reduce with pruning the original basis B.
    if any vector z in reduced basis satisfies the requirements then
        return calculated set of solution coefficients x_i.
    end if
end for
```

For the sake of the algorithm readability we used some vague terms that we now expand on:

- *Permute* the columns of basis B *appropriately* - here appropriately means that the permuted basis should have nonzero entry in row $n + 2$ in first column, and then sort the vectors of the basis according to their length, except for the particular initial columns[1].
- *Iteratively* β-BKZ-reduce *current state* of basis B - by iteratively the authors had in mind that the output basis of the previous BKZ-reduction should be an input for the next BKZ-reduction with greater β.
- *Independently* β-BKZ-reduce *the original* basis B - here it is meant that for each β the original input basis B should be reduce ("from scratch").

The authors state that the β blocksizes for independent reduction ($\beta \in \{30, 31, ..., 61\}$) and the pruning parameters are adapted to dimension $n = 80$ and they do not further explain their choice.

Now it would be a good time to note that we have reason to suspect Schnorr-Shevchenko algorithm uses 32 threads to solve one instance of subset sum problem. Two things make us suspect that:

- In third step of their algorithm they restart the reduction of the original basis B independently for $\beta \in \{30, 31, ..., 61\}$ and do not use output bases of previous

[1] Which vectors are the "particular" ones is not explained by authors.

reductions. We do not see any particular reason to that, except that it is done because the reduction for each β was being done on a separate thread.

- If we suppose the algorithm solves instances using single thread, then the average running time per successfully solved problem for growing β does not seem to grow fast enough (as one would expect when BKZ algorithm is used).

We report on the results from this paper later on when we discuss about the performance of our algorithm in comparison to Schnorr-Shevchenko one.

3.2.3 Lattice for Modular Subset Sum Problem

If we want to solve instances of modular subset sum problem we do not directly use the Schnorr-Shevchenko lattice presented in Equation 3.3, but we use a modified version of it, which is presented in Equation 3.4.

$$B = \begin{bmatrix} 2 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 2 & \cdots & 0 & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & 0 & 1 \\ Na_1 & Na_2 & \cdots & Na_n & N2^n & Ns \\ 0 & 0 & \cdots & 0 & 0 & 1 \\ N & N & \cdots & N & 0 & \frac{n}{2}N \end{bmatrix}. \tag{3.4}$$

Here, B is a $(n+3)$-by-$(n+2)$ matrix. The difference between the two basis matrices is that we have added in the modular matrix version the $(0, 0, ..., 0, N2^n, 0, 0)^T$ column in front of the last column. This extra column is needed so that we could have coefficients $c_1, ..., c_n, c_{n+1}, c_{n+2}$, with $|c_1|, ..., |c_n|, |c_{n+2}| \in \{0, 1\}$, $c_{n+1} \in \mathbb{Z}$, and have the equation

$$\sum_{i=1}^{n} Na_ic_i + c_{n+1}N2^n + c_{n+2}Ns = 0$$

that holds. Under the assumption that vector $\mathbf{c}$ generates the vector $\mathbf{z}$ satisfying Schnorr-Shevchenko criteria, the equation above would be equivalent to

$$N\sum_{i=1}^{n} a_ic_i = -Nc_{n+2}s \pmod{2^n},$$

which is further equivalent to

$$\sum_{i=1}^{n} a_ic_i = -c_{n+2}s \pmod{2^n}.$$

4 Experiments

In this chapter we first introduce the library we used for our experiments, then present the general idea of our algorithm. After that, we report our preliminary results. This results are used to determine concrete parameters for the aforementioned general idea. Next, we present the detailed version of our algorithm and finally we discuss its runtime.

4.1 Test Environment

As already mentioned in Chapter 1, we try to solve instances of random subset sum problem and modular subset sum problem. We generated 300 problem instances, for each problem dimension $n \in \{50, 52, ..., 100\}$.

We used G6K lattice reduction library [12], which also incorporates fp(y)lll reduction library [15]. To the best of our knowledge, this is currently the most advanced lattice reduction library available. Experiments were ran on a server of the Chair for Embedded Security, EI Faculty, Ruhr-University Bochum and HGI Institute, and the CPU used was 2.3GHz AMD Opteron^{TM} 6276 CPU. Test run for each problem instance was performed on a single thread, though we used multiple threads to solve different problem instances parallelly.

Problem instances can be found on the *ssred* Bitbucket repository [22] in the folder named `instances`. In there, for each dimension x there is a folder called `n_x` where the 300 generated instances for that dimension are stored. Instances are represented in files with names `1`, ..., `300`. The first line in each of those files contains an array of a_i's in the given instance, second line contains the solution vector, third line contains the target sum and fourth line represents the problem dimension. Inside `n_x` folders, subfolders `N_x` and `modular_N_x` are present, where the lattices generated from the problem instances are stored. As the name suggests, in `modular*` subfolders, the lattices for solving modular versions of subset sum problem are stored.

4.1.1 G6K

The paper about the G6K (*The General Sieve Kernel*) library [2] was published in Eurocrypt 2019 conference. This paper reports on new records and speedups in lattice challenges, mainly thankfully to the idea of using sieving instead of enumeration routines when working with lattices of large dimension. As the authors report, the

G6K library is "abstract stateful machine supporting a wide variety of lattice reduction strategies based on sieving algorithms". For (sub)lattices of lower dimension the authors utilize the fp(y)lll library, that is, the fp(y)lll's implementation of LLL and BKZ2.0 algorithms. BKZ2.0 implemented in fp(y)lll uses enumeration as the SVP oracle subroutine.

G6K Internals

We will now try to explain briefly how does G6K library function. Internal state of G6K keeps track of three numbers $\kappa \leq l \leq r$, where $[l : r]$ is called the *sieving context* and $[\kappa : r]$ is called the *lifting context*. The sieving context determines the projected lattice $\mathcal{L}_{[l,r]}$ where all the vectors currently considered by G6K "live". The κ and r in the lifting context $[\kappa : r]$ determine a left and right limit of the sieving context that could potentially be extended in future steps. The following G6K instructions are important in our case:

- $\texttt{Reset}_{\kappa,l,r}$ - empty the sieving database, set sieving context to $[l, r]$ and lifting context to $[\kappa, r]$.
- `S` - run the chosen sieving algorithm. The choice of sieving algorithm can be given to G6K as an input.
- `EL` - extend the sieving context one place to the left ($l \leftarrow l - 1$), meaning expand the projected lattice we are taking into account with the projected basis vector $\mathbf{b}_{l-1}$.
- `I` - insert the best vector candidate for the reduced basis that is being constructed somewhere into the basis $B_{[\kappa,r]}$. Sieving context is updated to $[l + 1 : r]$. The insertion can happen anywhere between vectors $\mathbf{b}_\kappa$ and $\mathbf{b}_l$, but we always insert the vector in place of $\mathbf{b}_l$.

Fundamental operation in lattice reduction strategies G6K authors presented is `Pump`, which more precisely is defined as:

$$\texttt{Pump}_{\kappa,f,\beta} \ : \ \texttt{Reset}_{\kappa,\kappa+\beta,\kappa+\beta}, \ (\texttt{EL}, \ \texttt{S})^{\beta-f}, \ (\texttt{I}, \ \texttt{S})^{\beta-f}.$$

With `Reset` command, the $\texttt{Pump}_{\kappa,f,\beta}$ first initializes empty sieving database and sets up the β-wide sieving context. Then it extends left ("incorporates" the next basis vector into the sieve) the sieving context and sieves each time the `EL` operation is executed. That operation is repeated $\beta - f$ times, where the parameter f is the "dimensions for free" parameter. The "dimension for free" concept *"may be viewed as a hybrid of pruned enumeration with sieving ... In other words, we may consider these improvements as applying lessons learnt from enumeration to sieving algorithms"* [2]. In our case, "dimensions for free" parameter is 0, meaning we do not utilize the performance gains the concept offers. Next, we insert the best candidates from the sieve into the basis we are reducing, and sieve again after each insertion. Also, with each insertion the sieving context is "shrinking" back. That process is also repeated $\beta - f = \beta$ times. The G6K

sieving strategy we used is PumpNJump strategy, and it corresponds to the following G6K command:

$$\texttt{PumpNJumpTour}_{\beta',f,j} \ : \ \texttt{Pump}_{0,f,\beta'},\ \texttt{Pump}_{j,f,\beta'},\ \texttt{Pump}_{2j,f,\beta'},\ \ldots$$

In $\texttt{PumpNJumpTour}_{\beta',f,j}$ command the parameter f is the one we mentioned above and is 0 in our case, and parameter j is the "jump" parameter, which determines how much we want to shift the sieving and lifting context to the right after we execute one `Pump` command. In our case $j = 1$, meaning the β-wide "block" is moving properly and covering all β-wide subblocks of lattice basis, same as it is done in BKZ algorithm.

If we wanted to compare PumpNJump strategy to BKZ algorithm, we could say `Pump` somewhat corresponds to OneRoundBKZ subroutine and PumpNJump strategy as a sequence of `Pump` instructions corresponds to the whole BKZ algorithm. We will be using TripleSieve for our experiments, unless otherwise stated, because authors reported the TripleSieve performs the best in the experiments they executed.

4.2 Solving Random Problems

The algorithm we used to solve subset sum problem is based on the Schnorr-Shevchenko one. Our algorithm works in sequential setting - it applies iteratively BKZ2.0 or G6K PumpNJump BKZ-like algorithm with parameter β growing at each step. Whether we use BKZ2.0 or PumpNJump strategy depends on size of the β:

- For "small" β - use BKZ2.0 without pruning.
- For "medium" β - use BKZ2.0 with pruning, as non-pruning enumeration in BKZ2.0 becomes too slow at some point.
- For "large" β - use G6K PumpNJump strategy, as sieving becomes faster than non-pruned enumeration in BKZ2.0 at some point.

We determined exact meaning of "small", "medium" and "large" β using preliminary experiments and we report the details in the following subsection.

4.2.1 Choosing the Strategies

First we wanted to determine what is the "large" β for which G6K PumpNJump strategy becomes faster than BKZ2.0 using non-pruned enumeration. In the `G6K` paper [2], the authors showed that the simplest PumpNJump strategy (using TripleSieve) outperforms fp(y)lll's BKZ2.0 implementation already for $\beta \geq 62$, when ran over lattices with special structure of dimension 180. Their sample size was 8 lattices, that is, lattice bases.

As we are not working with the lattices they used, but with other kind of lattices that have special structure, we tested experimentally what is the minimal β for which

PumpNJump strategy (using TripleSieve) becomes faster than fp(y)lll BKZ2.0. We tested Schnorr-Shevchenko lattices of 32 randomly chosen instances of dimension 100 subset sum problem, and for $N = 16$. The parameter β ranged from 56 to 73. The average running time over all problems shows that in our case the PumpNJump becomes faster than BKZ2.0 for a slightly larger β than in `G6K` paper, namely $\beta = 69$, thus we choose $\beta_{\text{lim}} = 69$. The average running times are represented in Figure 4.1.

We see in previous paragraph that the BKZ2.0 execution for $\beta < \beta_{\text{lim}} = 69$ is faster than using G6K PumpNJump strategy. It can be concluded that for $\beta < 69$ we should use BKZ2.0. Here one can choose many different sets of β values that should be used for a β-BKZ2.0 reduction of the basis, and whether the pruning should be enabled or not. Schnorr and Shevchenko [30] noted that executing iterative[1] BKZ reduction with β doubling in each step is particularly fast for small β. We wanted to experimentally test that claim, and test some other strategies for BKZ2.0, in order to find out which of them performs the best.

We tested four different BKZ2.0 strategies:

- **Strategy 1.** pruned enumeration for $\beta \in \{2, 3, 4, ...\}$.
- **Strategy 2.** non-pruned enumeration for $\beta \in \{2, 3, ..., 32\}$, pruned enumeration for $\beta \in \{33, 34, ...\}$.
- **Strategy 3.** non-pruned enumeration for $\beta \in \{2, 4, 8, 16, 32\}$, pruned enumeration for $\beta \in \{33, 34, ...\}$.
- **Strategy 4.** non-pruned enumeration for $\beta \in \{2, 4, 6, ..., 32\}$, pruned enumeration for $\beta \in \{33, 34, ...\}$.

First we should note that in each test run on 300 instances, independent of the strategy we choose, there are some instances that are not solved. The average number of unsolved instances for each n and each strategy ranged from 5 (1.67 percent) to 20 (6.67 percent) problem instances.

In Figure 4.2 you can see the average running time taken over 300 problem instances, taking into account only running times of the problems that were successfully solved, and excluding 5 percent of fastest and slowest average running times.

It can be seen that Strategy 1 performs worse than all the remaining strategies. Strategy 2 is sligthly slower than Strategy 3 for all values of n, and Strategy 4 sometimes performs better than Strategy 2 and Strategy 3, and sometimes worse. The final choice of strategy then falls between Strategy 3 and Strategy 4. As the number of unsolved problems is fairly similar for the two strategies for a fixed n, that did not influence our choice. Strategy 3 is faster than Strategy 4 for more values of n, thus we finally choose Strategy 3 as best strategy for our algorithm. Let us remind the reader that non-pruning enumeration part

[1] the output basis of β_1-BKZ is the input basis for next β_2-BKZ.

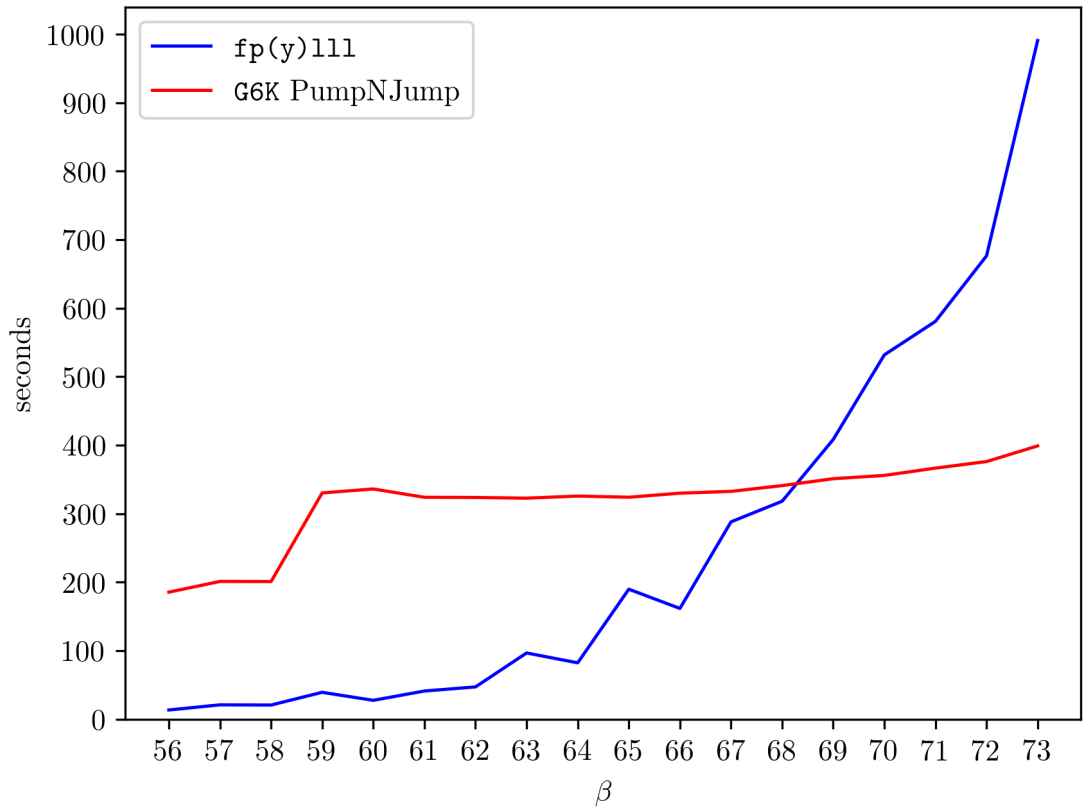


Figure 4.1: fp(y)lll BKZ2.0 versus G6K PumpNJump performance over Schnorr-Shevchenko lattices for dimension $n = 100$.

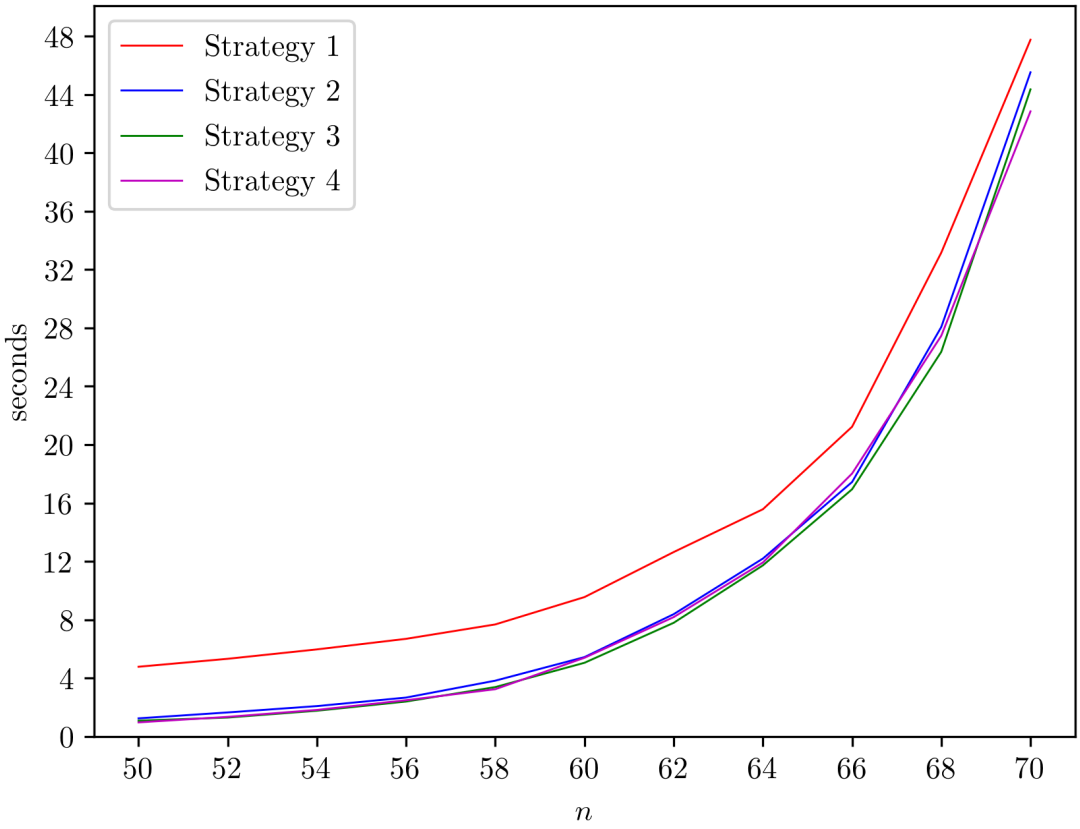


Figure 4.2: fp(y)lll BKZ2.0 average running time over 300 problems, excluding 5 percent of fastest and slowest solved problems.

of Strategy 3 is exactly the first part of algorithm described in Schnorr-Shevchenko paper [30].

As fp(y)lll's BKZ2.0 is faster than PumpNJump for $\beta < 69$, we will use BKZ2.0 exclusively for solving problems of the dimension $n \leq 66$, because the greatest β we can encounter in that case is $\beta = n + 1 = 67$. For $n \geq 68$ it can happen that we encounter $\beta \geq 69$ and for $\beta = 69$ we will switch to using BKZ-like PumpNJump strategy that, as already mentioned, uses sieving in a basis reduction routine. Taking all that into account, we present the detailed algorithm we later use for solving subset sum problems in the following subsection.

4.2.2 Our Algorithm

We named our algorithm *SSRed* (Subset Sum Reduction) and present it in Algorithm 8. The `solution_found` procedure checks if there exists vector $\mathbf{z}$ in the reduced basis that satisfies Schnorr-Shevchenko criteria. If there exists such vector, then the procedure `construct_solution` returns a solution of the subset sum problem instance, constructed as in Theorem 3.2.1.

Algorithm 8 SSRed algorithm

Input: Lattice B corresponding to a subset sum problem instance of dimension n.
Output: Solution $\{x_1, ..., x_n\}$ or FAIL.

for $\beta \in \{2, 4, 8, 16, 32\}$ **do**
  run BKZ2.0(β) with non-pruned enumeration.
  if `solution_found`() **then**
    solution $\leftarrow$ `construct_solution`().
    return solution.
  end if
end for
for $\beta \in \{33, 34, ..., 68\}$ **do**
  run BKZ2.0(β) with pruned enumeration.
  if `solution_found`() **then**
    solution $\leftarrow$ `construct_solution`().
    return solution.
  end if
end for
for $\beta \in \{69, 70, ..., n + 1\}$ **do**
  run PumpNJumpTour(β).
  if `solution_found`() **then**
    solution $\leftarrow$ `construct_solution`().
    return solution.
  end if
end for
return FAIL.

4.2.3 Our Results

We then finally ran our algorithm on all instances, for dimensions $n \in \{50, 52, ..., 100\}$ with $N = 16$ in all cases, and using TripleSieve in the PumpNJump subroutine. We used G6K `bkz.py` script and modified it a bit for managing the library. The modified script we used can be found in the repository under the name `g6k-tmp/bkz_ssred.py`. The total running time to execute experiments for random subset sum problems took over 11300 hours of CPU time which is equal to over 470 days. The graphical representation of the results can be seen in Figure 4.3. The reader should keep in mind that we discarded 10 percent of the slowest and fastest solved problem instances.

The crucial part of these results is that we clearly have the experimental confirmation that the running time needed for solving the instances of subset sum problem grows exponentially with the dimension of the problem. We see that in Subfigure 4.3a, as the logarithmic value to the base 2 of average running times seems to grow linearly. We did expect that, as the underlying algorithm uses the BKZ2.0 and G6K PumpNJump subroutines which do have theoretical bounds on the running time that are super-exponential and exponential.

Another interesting thing we looked at is median value of β's for which the instances were solved. We plotted that in the aforementioned subfigure as well. It can be seen that the median of β grows almost perfectly linearly together with n. We were interested whether the slowdown of logarithmic value of average runtime in Subfigure 4.3a has something to do with the switch from BKZ2.0 to PumpNJump strategy for $\beta_{\text{lim}} = 69$ in our algorithm, so we plotted the same logarithmic value in function of median β in Subfigure 4.3b. There it *seems* like the slowdown does happen somewhere around $\beta = 69$.

Next we tried to naively extrapolate the value of β in function of n, taking the points $(n, \beta) \in \{(50, 34), (100, 93)\}$ as an input. We got a function

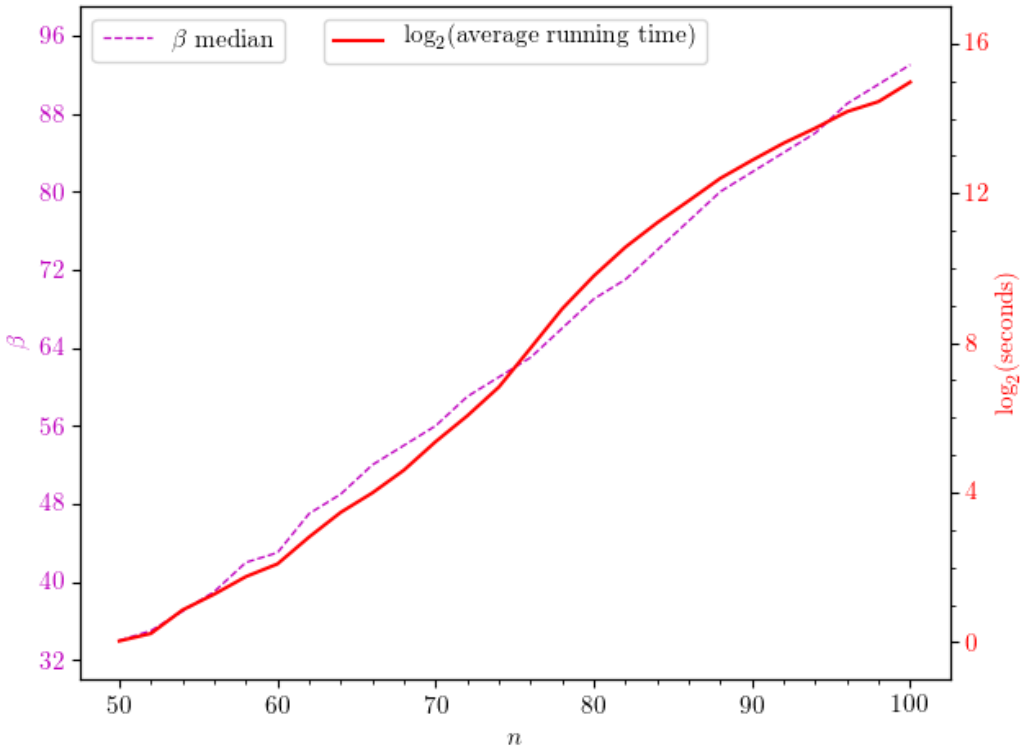
$$\beta(n) = 1.18n - 25.$$

Unfortunately, this is not a correct result in general, as for larger n the value of β will become larger than $n+1$ and that is not possible. More concretely

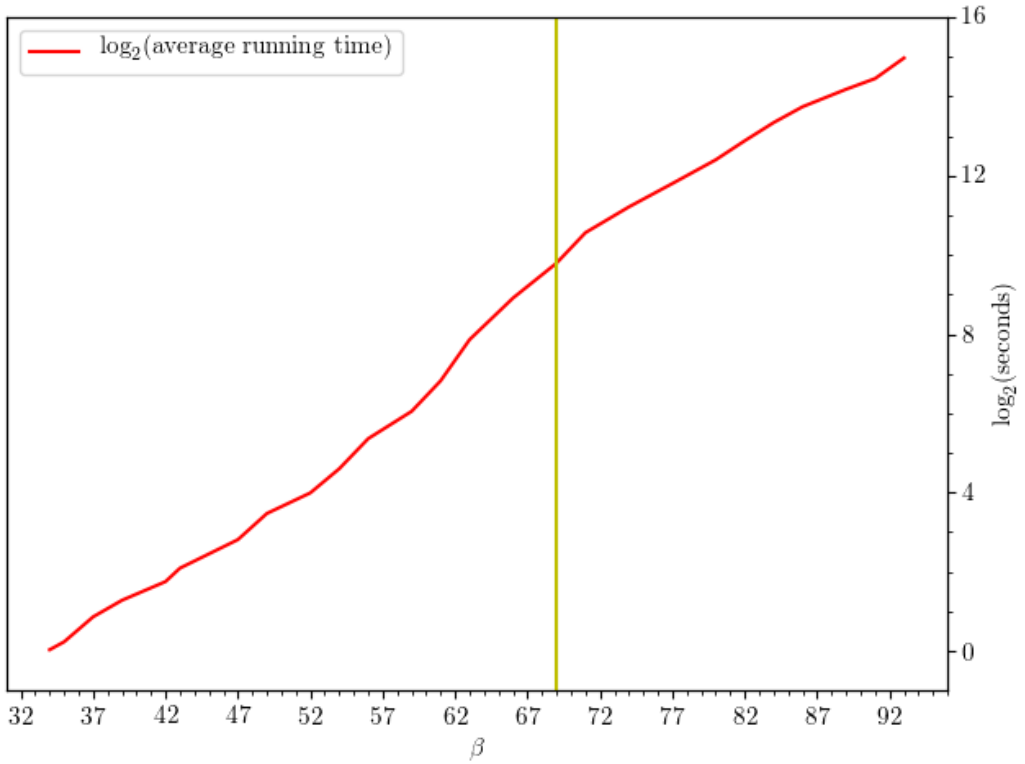
$$\forall n \geq 150 : \ \beta \geq n + 2.$$

In general, the $\beta(n)$ does not have to be linear function. Later on, we noticed that the value of $n - \beta(n)$ decreases with n growing larger. We graphically present the function $f(n) = n - \beta(n)$ for $n \in \{50, 52, ..., 100\}$ in Figure 4.4. This result indicates how could $\beta(n)$ relate to n, but we did not manage to extrapolate clear and accurate function $\beta(n)$.

Some instances do remain unsolved by our algorithm. The correct number of them is given in Table 4.5. We observed that if N is chosen differently, it may happen that some problems that were not solved for $N = 16$ become solved, and the other way around,



(a) Median of β for which the problems were solved; Logarithmic value of the average running time to the base 2.



(b) Logarithmic value of the average running time to the base 2 in the function of β median.

Figure 4.3: Graphical representation of our main results.

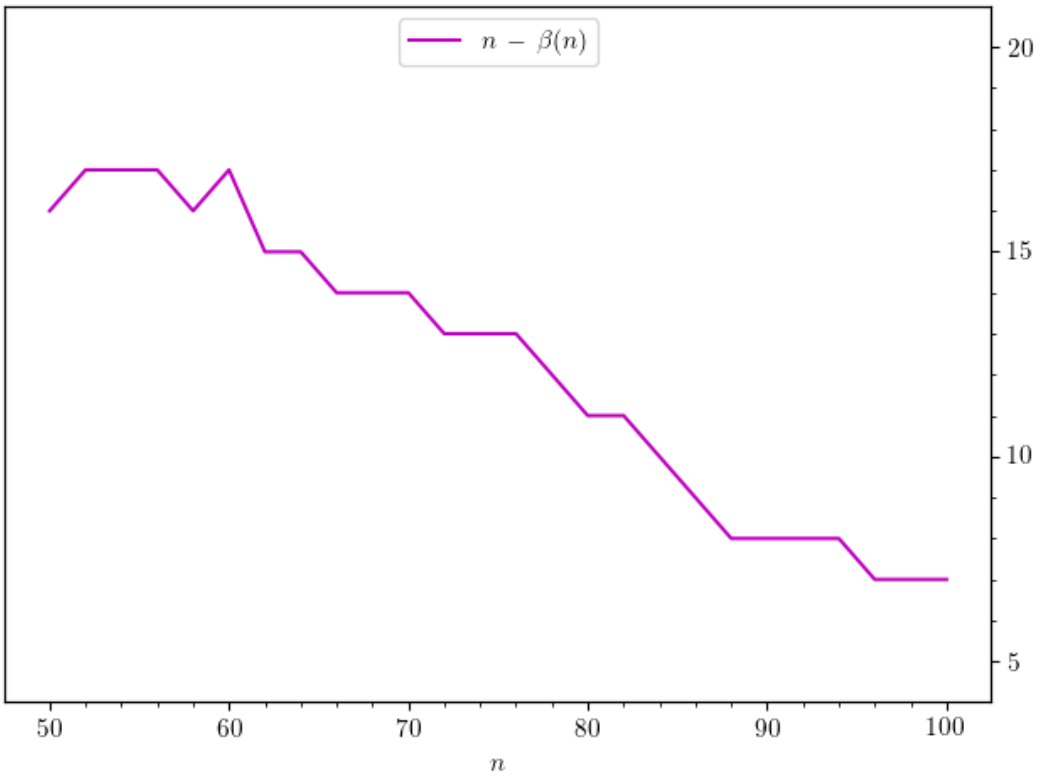


Figure 4.4: The difference between problem dimension n and median value of of β's for which the problems were solved.

n	# unsolved problems
50	0
52	7
54	2
56	2
58	3
60	2
62	1
64	0
66	0
68	0
70	0
72	2
74	0

n	# unsolved problems
76	4
78	5
80	5
82	7
84	6
86	10
88	9
90	16
92	9
94	17
96	21
98	17
100	13

Table 4.5: Number of unsolved problems per dimension n.

some problems that were solved for $N = 16$ are not solved for different N. Notice that with the growth of problem dimension n the number of unsolved problems slowly grows as well.

4.2.4 Comparing the Results

In this subsection we discuss the performance of our algorithm in comparison to the ones from Becker-Coron-Joux (combinatorial approach) and Schnorr-Shevchenko (lattice reduction approach). We must note here that Becker, Coron and Joux ran their experiments on unoptimized implementation of their algorithm and that should be taken into account while examining the results of all three algorithms.

First in Table 4.6 we report on the test machines and test results for the subset sum dimension 80 problems. For our results in "Average running time" row we give in parenthesis the average running time when we discard 10 percent of slowest and fastest solved instances. It can be seen that we used slower CPU than BCJ and Schnorr-Shevchenko used for algorithm execution. Clearly our algorithm seems to be faster for dimension 80 than BCJ ($\tilde{\mathcal{O}}(2^{0.291n})$ version) one. At the same time we report slower time than Schnorr-Shevchenko had reported, but let us recall the suspicions we have, that we presented in Section 3.2.2. We do believe the Schnorr-Shevchenko algorithm uses 32 threads to solve one instance of subset sum problem. If our suspicions are true, it does not make sense, and is not fair, to compare our results with Schnorr-Shevchenko ones.

We present results comparison for problems of dimension 96 in Table 4.7, comparing performance of just BCJ ($\tilde{\mathcal{O}}(2^{0.291n})$ version) and our algorithm, as Schnorr-Shevchenko tested only dimension 80 subset sum problem in their paper. For our results in "Average

	BCJ11	SS12	This thesis
CPU	2.67GHz	2.67GHz	2.3GHz
Sample size	50	50	300
Average running time	1068s	113.78s	974.34s (884.1s)
% unsolved problems	0	0	1.67

Table 4.6: Results comparison for problems of dimension 80.

	BCJ11	SS12	This thesis (TripleSieve)
CPU	2.8GHz	x	2.3GHz
Sample size	5	x	300
Average running time	2820s	x	19602s (18556s)
% unsolved problems	0	x	7

Table 4.7: Results comparison for problems of dimension 96.

running time" row we give in parenthesis the average running time when we discard 10 percent of slowest and fastest solved instances. Results show us that BCJ algorithm is almost 7 times faster than our algorithm. Though, we have to note that BCJ tests used 13GB of memory, while our algorithm used at most 0.3GB of memory. For this particular dimension we ran our algorithm with implemented version of Becker-Gama-Joux as the underlying sieve as well, and the average running time was approximately 18645 seconds, or 18249 seconds if we discard 10 percent of slowest and fastest solved instances. So, using the Becker-Gama-Joux sieve instead of TripleSieve in our algorithm seemingly gives us slightly better results, which are still far away from BCJ performance in this dimension.

Low memory consumption case. Now we would like to quickly report the results of the memory-constant algorithm presented in BCJ paper. For dimension 40 problems, the algorithm needs 933 seconds to solve a problem instance. Our algorithm for problems of dimension up to 66 uses polynomial memory because it uses the enumeration procedure as main component, and it could be helpful to say that our algorithm already for $n = 50$ has insignificant running time, 1.15 seconds (and 17.96 seconds for $n = 60$).

4.2.5 Other Results

In this subsection we briefly report our secondary findings.

Vector shortness in Schnorr-Shevchenko lattices. For each test case we also collected information whether shorter vector than the one satisfying Schnorr-Shevchenko criteria

appeared in the reduced basis before we found a solution. That information could be useful if we want to investigate how far away is the Schnorr-Shevchenko vector from the shortest vector in the lattice. The percentage of instances in which we found such shorter vector ranged from 0 to 7 percent across all problem dimensions that we tested. If we took the average of those percentages over all dimensions, we get that in approximately 2.5 percent of test cases vector shorter than the "solution" vector was found.

Influence of N. We tried to investigate whether the choice of N influences the running time and the percentage of solved problems. For problems of dimension $n = 70$, we generated lattices for all $N \in \{10, 15, ..., 300\}$ and ran the algorithm presented in Subsection 4.2.2 for all problem instances and all values of N. Results seem to indicate that both the average running time and percentage of solved problems are not influenced by specific choice of N.

4.3 Solving Modular Problems

We tried solving modular subset sum problems with our algorithm. As we already mentioned, the lattice used to solve modular problems is a bit different than original Schnorr-Shevchenko one, and can be seen in Equation 3.4. We tested all problems for $n \in \{50, 52, ..., 80\}$, having $N = 16$ in all generated lattices for modular subset sum problem instances.

The logarithmic value of average running times to the base 2 for both random and modular subset sum problems can be seen in Figure 4.8. It can be observed that for the same problem dimension, modular subset sum problem in average takes more time to be solved than the random subset sum problem. We suspect the one extra vector a lattice basis for modular instances has (in comparison to the original Schnorr-Shevchenko one) causes a larger average running time. At the same time, we do not claim the cause for larger running time is exclusively this extra column in the basis matrix. It is possible that other factors contribute to the larger running time as well.

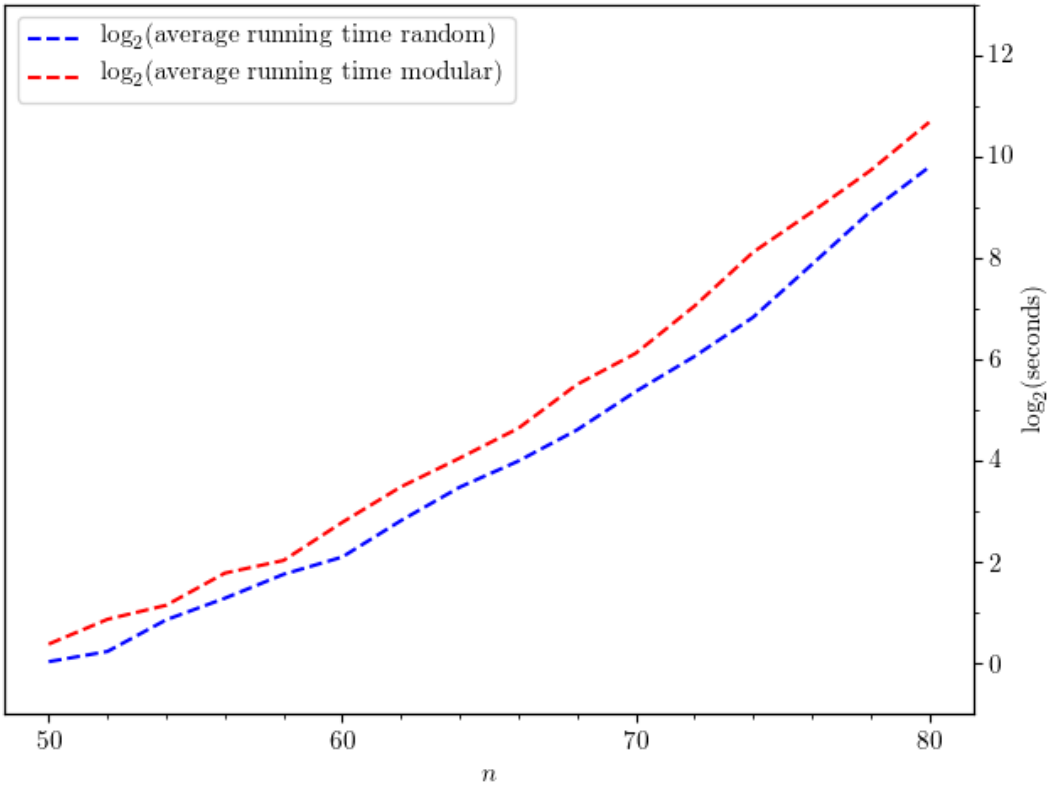


Figure 4.8: Logarithmic value to the base 2 of average running time for successfully solved, both random and modular, problems.

5 Mixing the Approaches (Work in Progress)

Two main techniques for solving subset sum problem are combinatorial, meet-in-the-middle-like, and lattice reduction algorithms. Can we mix those two approaches in order to get algorithm superior to both methods? The idea of using both combinatorial and lattice reduction algorithms for obtaining superior results, is not new. In 2007. Howgrave-Graham presented a meet-in-the-middle and lattice reduction hybrid attack on NTRU cryptosystem [18]. That was our inspiration for exploring the possibilities in our case. This exploration is still work in progress and we kindly ask the reader to keep that in mind.

When solving subset sum problem with lattice reduction we are looking for lattice vector $\mathbf{z}$ that satisfies Schnorr-Shevchenko criteria from Equation 3.2. We know that $\mathbf{z}$ can be represented as $\mathbf{z} = B\mathbf{c}$, $\mathbf{c} = (c_1, ..., c_{n+1})$. From the proof of Theorem 3.2.1 we know that $|c_{n+1}| = 1$ and $|c_j| \in \{0, 1\}$ for $j = 1, ..., n$. Roughly speaking, our idea was the following:

- Guess $c_i, ..., c_n$, for some $1 \leq i \leq n$.
- Conveniently divide B into 4 submatrices of smaller dimensions.
- Reduce upper left submatrix.
- Setup `CVP` problem instance and try to solve it.

Now we will try to explain our idea on an example.

Let $(a_1, ..., a_n, s)$ be dimension n subset sum problem instance. Let variable i be *separation parameter*. In Equation 5.1 the $B\mathbf{c} = \mathbf{v}$ is presented, with B having permuted rows in comparison to the original Schnorr-Shevchenko matrix and $\mathbf{v}$ being written as concatenation of left vector part $\mathbf{v}_1$ and right part $\mathbf{v}_2$. We divide matrix B such that in lower left part we have a $\mathbf{0}$ submatrix and then write B as

$$B = \left[\begin{array}{c|c} B_1 & B_2 \\ \hline B_3 & B_4 \end{array}\right] = \left[\begin{array}{c|c} B_1 & B_2 \\ \hline \mathbf{0} & B_4 \end{array}\right].$$

$$\begin{bmatrix} Na_1 & \cdots & Na_n & Ns \\ N & \cdots & N & N\frac{n}{2} \\ 2 & \cdots & 0 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 2 & 1 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \cdot \left[\begin{array}{c} c_1 \\ \vdots \\ c_{i-1} \\ \hline c_i \\ \vdots \\ c_n \\ c_{n+1} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{v_1} & \mathbf{v_2} \end{array}\right] \tag{5.1}$$

Next we write $\mathbf{v}_1$ and $\mathbf{v}_2$ as

$$\mathbf{v}_1 = B_1 \cdot (c_1, ..., c_{i-1}) + B_2 \cdot (c_i, ..., c_{n+1}) \tag{5.2}$$

and

$$\mathbf{v}_2 = \mathbf{0} \cdot (c_1, ..., c_{i-1}) + B_4 \cdot (c_i, ..., c_{n+1}) = B_4 \cdot (c_i, ..., c_{n+1}) \tag{5.3}$$

Our next step is to reduce the lattice basis B_1 (lattice reduction step). After that, we guess values of $(c_i, ..., c_n) \in \{0, -1\}^{n-i+1}$ and set c_{n+1} to be 1. This is the meet-in-the-middle step of our attack idea. Then in the Equations 5.2 and 5.3 above the value of $B_4 \cdot (c_i, ..., c_{n+1}) = \mathbf{v}$ is known, as well the value of $B_2 \cdot (c_n, ..., c_{n+1})$. The unknowns are $(c_1, ..., c_{i-1})$ and vector $\mathbf{v}_1$. But, from the Schnorr-Shevchenko criteria we would know that $\|\mathbf{v}_1\| = \sqrt{i+1}$. With reduced B_1 and $\mathbf{t} = B_2 \cdot (c_i, ..., c_{n+1})$ as a target vector we construct a `CVP` problem instance. We continue with Babai Nearest Plane algorithm in hope to get vector $(c'_i, ..., c'_{i-1})$ such that it will help us find a solution to subset sum problem instance together with previously guessed $(c_i, ..., c_{n+1})$.

Now let us get into more details. We are dealing with subset sum problem instances that have solution vector with Hamming weight $\frac{n}{2}$. If we suppose the 1s are evenly distributed over the solution vector, it would mean in any k (k is even) consecutive places in solution vector we should have $\frac{k}{2}$ ones. Even if the 1s are not evenly distributed, we can randomly permute the a_j's and automatically the solution vector will be permuted as well. After certain number of such permutations we can expect that 1s and 0s in solution vector are evenly distributed. That is why we will have a permutation step in the algorithm idea we present in the following items.

Algorithm idea:

- For certain number of times repeat:
  - Randomly permute vector $(a_1, ..., a_n)$.
  - Construct the Schnorr-Shevchenko lattice B with permuted rows[1].
  - Decompose B into 4 submatrices:
    - upper left B_1 of dimension $(i+1) \times (i-1)$.

[1] As showed in Equation 5.1.

  * upper right B_2 of dimension $(i+1) \times (n+2-i)$.
  * lower left B_3 of dimension $(n+2-i) \times (i-1)$, which is a $\mathbf{0}$ matrix.
  * lower right B_4 of dimension $(n+2-i) \times (n+2-i)$.

- Reduce B_1 with BKZ2.0 algorithm and denote the reduced basis by B_1'.
- For every combination of $\frac{(n+1-i)}{2}$ ones in binary vector of length $(n+1-i)$:
  - Set $\mathbf{c}_2 = (c_i', ..., c_n')$ to reflect current combination.
  - Calculate $\mathbf{t} = B_2 \cdot (c_i', ..., c_n')$.
  - Run Babai Nearest Plane algorithm on $(B_1', \mathbf{t})$ and denote the output with $\mathbf{y}$.
  - If $\|\mathbf{t} - \mathbf{y}\| = \sqrt{i+1}$:
    → Solve equation $B_1\mathbf{c}_1 = \mathbf{y}$, where B_1 is the original lattice basis and $\mathbf{c}_1$ an unknown.
    - Calculate $\mathbf{z}' = B\,(\mathbf{c}_1, \mathbf{c}_2)$ and check if $\mathbf{z}'$ satisfies Schnorr-Shevchenko criteria. If yes, **solution found**.

We expected to gain the most advantage by reducing the lattice basis B_1 of smaller dimension instead of reducing the original lattice basis B.

We could estimate, although not precise, the running time for separation parameter $1 \leq i \leq n$ with the following formula:

$$T_{\text{hybrid}}(n) = N_{\text{perm}}\ (T_{\text{BKZ}}(i)\ +\ 2^{\frac{n+1-i}{2}}\ poly(i)).$$

With N_{perm} we estimate the number of $(a_1, ..., a_n)$ permutations, with $T_{\text{BKZ}}(i)$ the running time of BKZ2.0 algorithm when reducing B_1 (of dimension $(i+1)\times(i-1)$), with $2^{(n+1-i)/2}$ the number of combinations of binary vectors we are testing and with $poly(i)$ we estimate the running time of Babai Nearest Plane algorithm.

The main question which arised is, how should one choose the separation parameter $i = c\ n$, that is, how should one choose the factor c, such that T_{hybrid} has the lowest theoretical bound or fastest practical results.

Last but not least, it was time to test our idea. We made a small script called `ssred_hybrid.py`, which you can find in the Bitbucket repository [22], and it contains the implementation of our algorithm idea. As this is still work in progress, the script did not produce favorable results and we are working on improvements.

6 Conclusion

It is always interesting to use different techniques and utilize technological advances to solve classical **NP**-complete problems. And that is what we did in this thesis. The problem of solving subset sum problem with lattice reduction algorithms is not closed and much can still be done in this area.

6.1 Takeaway

For already 35 years we know that subset sub problem can be solved using lattice reduction algorithms. Only thing that changed in the last 35 years is that we were able to solve denser or harder problems.

We executed more broader experiments than the works before this thesis, which allows us to add one more perspective in comparing the performance of lattice reduction algorithms to the meet-in-the-middle-like algorithms. Our results, together with Schnorr-Shevchenko results [30], give us reason to think that lattice reduction algorithms outperform these combinatorial algorithms in some dimensions. Concretely speaking about our results, our algorithm is slower than main algorithm presented in BCJ paper ([5]), but it does use a lot less memory than BCJ algorithm (e.g. for dimension 96 problems it uses more than 43 times less memory). However since the experiments were not conducted in the same settings (e.g. same computer), it makes it hard to conclude anything.

We must be honest and admit that from the theoretical point of view the combinatorial algorithms are superior, as there are no bounds on the running time of lattice reduction-based algorithms used for solving subset sum problem *yet*. We did not look into theoretical aspects of lattice reduction attacks a lot, but nevertheless we did notice a potential obstacles in proving the lattice reduction attacks running time. Some of those potential obstacles, that may be bypassed in the future, are:

- It is not known yet how much exactly far away is the shortest vector in the lattice presented in 3.3 from the vector satisfying the Schnorr-Shevchenko criteria that we are looking for.
- The good lower bound on the quality of the β-BKZ reduced basis is not known.

Two points mentioned above combined with the rough upper bound on the BKZ(2.0) running time make it hard to give a theoretical and stable estimate on the running time of lattice reduction-based algorithms, for now.

As currently the standardization of post-quantum cryptographic algorithms is taking place, where the lattice-based cryptosystems are taking the spotlight (if we look at the number of lattice-based submissions that made it to the second round [26] compared to the other cryptosystems), it may be possible that significant improvements in lattice reduction algorithms appear. If that happens, then automatically we will have improvements in algorithms for the subset sum problem.

6.2 Future Work

There is a lot of work that can be done in this topic. Surely in the future there will be much work done because both of the ingredients (subset sum problem and lattice reduction) are in focus. In the next few paragraphs we will present what we think it would be interesting and *needed* to be done in this topic, in order to get a clearer picture about solving subset sum problem with lattice reduction methods.

In Chapter 4 we reported on our results and the results from other papers. To be more precise, in Section 4.2.4 we presented and compared our result with results from BCJ and Schnorr-Shevchenko. There we noted that in both of other two papers the authors used faster CPUs than we did. The BCJ algorithm implementation was not optimized as well. Authors of BCJ paper also reported on memory usage of their algorithm. In that setting it is not totally fair to compare the experimental results of the algorithms, as they were not ran in the same environment. One interesting task would be to test all algorithms in the same setting (on the same CPU) and only then compare their performance.

Next interesting and important work would be to further investigate the mix of the combinatorial and lattice reduction-based approaches. Although at this time we did not get satisfying results, the idea seems perspective. We sincerely hope that this idea of hybrid algorithm will be looked at in the future.

Finally, it would be pleasing to establish theoretical bound on running time of lattice reduction-based algorithms. This could be shown, if for instance, the relation between problem dimension n and the β value for which the problem will be solved with certain probability is established. Other improvements in theoretical understanding of lattice reduction could help establishing this bound for lattice reduction-based algorithms as well.

List of Figures

Bibliography

[1] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, 2001.

[2] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 717–746, 2019. ISBN 978-3-030-17656-3.

[3] Shi Bai, Thijs Laarhoven, and Damien Stehle. Tuple lattice sieving. Cryptology ePrint Archive, Report 2016/713, 2016. `https://eprint.iacr.org/2016/713`.

[4] Shi Bai, Damien Stehle, and Weiqiang Wen. Improved reduction from the bounded distance decoding problem to the unique shortest vector problem in lattices. Cryptology ePrint Archive, Report 2016/753, 2016. `https://eprint.iacr.org/2016/753`.

[5] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 364–385, 2011. ISBN 978-3-642-20465-4.

[6] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. Cryptology ePrint Archive, Report 2015/1128, 2015. `https://eprint.iacr.org/2015/1128`.

[7] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522, 2015. `https://eprint.iacr.org/2015/522`.

[8] Ernest F. Brickell. *Solving Low Density Knapsacks*, pages 25–37. 1984. ISBN 978-1-4684-4730-9.

[9] Yuanmi Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Paris Diderot University, 11 2013.

[10] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 1–20, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25385-0.

[11] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *computational complexity*, 2(2):111–128, Jun 1992. doi: 10.1007/BF01201999.

[12] The G6K development team. G6K, a lattice reduction library. https://github.com/fplll/g6k, accessed October 17, 2019.

[13] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44:463–471, 1985.

[14] Ruhr University Bochum Chair for Network and Data Security. TeX Thesis Layout. https://github.com/RUB-NDS/thesis_layout, accessed October 17, 2019.

[15] The fplll development team. fplll, a lattice reduction library. https://github.com/fplll/fplll, accessed October 17, 2019.

[16] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 257–278. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13190-5.

[17] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 447–464, 2011. ISBN 978-3-642-22792-9.

[18] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against ntru. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, pages 150–169, 2007.

[19] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. Cryptology ePrint Archive, Report 2010/189, 2010. https://eprint.iacr.org/2010/189.

[20] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, Sep 1996.

[21] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[22] Vukasin Karadzic. ssred, Bitbucket repository. https://bitbucket.org/vukasin_karadzic/ssred, accessed October 17, 2019.

[23] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, January 1985. doi: 10.1145/2455.2461.

[24] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *MATH. ANN*, 261:515–534, 1982.

[25] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1468–1480, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-98-6. URL http://dl.acm.org/citation.cfm?id=1873601.1873720.

[26] NIST. Post-quantum cryptography standardization project - Round 2 submissions, 2019. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions.

[27] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2:181–207, 07 2008. doi: 10.1515/JMC.2008.009.

[28] Oded Regev. Lecture notes "Lattices in Computer Science". Tel Aviv University, 2004. https://cims.nyu.edu/~regev/teaching/lattices_fall_2009/, accessed October 17, 2019.

[29] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, Aug 1994. doi: 10.1007/BF01581144.

[30] Claus P. Schnorr and Taras Shevchenko. Solving Subset Sum Problems of Density close to 1 by "randomized" BKZ-reduction. Cryptology ePrint Archive, Report 2012/620, 2012. https://eprint.iacr.org/2012/620.