MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Short vector problem in small-dimensional lattices

Master's Thesis

BC. TOMÁŠ BOŽEK

Brno, Spring 2026

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Short vector problem in small-dimensional lattices

Master's Thesis

BC. TOMÁŠ BOŽEK

Advisor: Mgr. Marek Sýs, Ph.D.

Centre for Research on Cryptography and Security

Brno, Spring 2026

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

When creating this thesis, I used the tool *Gemini*[1] to help me with researching relevant academic sources and to assist me in understanding new concepts. I also used the tool *ChatGPT*[2] to improve my writing style.

Bc. Tomáš Božek

Advisor: Mgr. Marek Sýs, Ph.D.

1. https://gemini.google.com/
2. https://chatgpt.com/

Acknowledgements

I want to express my gratitude to my supervisor, Mgr. Marek Sýs, Ph.D., for his guidance and support throughout working on this thesis.

Abstract

The primary aim of this thesis was to develop an efficient method for finding sufficiently short lattice vectors in low-dimensional lattices (dimensions below 40). Such a method could then improve the practical performance of lattice basis reduction algorithms such as LLL and BKZ, which play an important role in Lattice-Based Cryptography and the underlying Shortest Vector Problem solving. Such an application of the found method could be of a significant value since Lattice-Based Cryptography is believed to be one of the leading candidates for Post-Quantum Cryptography.

Keywords

SVP, LLL, BKZ, Short Vector Enumeration, Lattice, Lattice basis reduction, Gram matrix, Cholesky decomposition

Contents

1 Introduction **1**

2 Used concepts **2**
2.1 Concepts context introduction 2
2.2 Concepts description 3
2.2.1 Lattice 3
2.2.2 Lattice basis reduction 5
2.2.3 Shortest vector problem 5
2.2.4 Short vector enumeration 5
2.2.5 Lenstra-Lenstra-Lovász algorithm 6
2.2.6 Korkine-Zolotarev algorithm 7
2.2.7 Gram matrix 8
2.2.8 Cholesky decomposition 9
2.2.9 Root Hermite factor 11

3 Preparation for method analysis **14**
3.1 Lattice basis generation process 14
3.2 Used enumeration variants 15
3.2.1 Implemented enumeration variants concept . . 15
3.2.2 Example of use of the DFS enumeration variant 17
3.2.3 Comparison of implemented and built-in variants 22

4 Optimal real basis vector coefficients **24**
4.1 Gradient 24
4.2 Concept description 25
4.3 Concept usage experiments 27
4.3.1 Lattice basis ordering key for BKZ reduction . . 27
4.3.2 Differences of coefficient sums from the shortest vector 29
4.3.3 Single coefficient differences from the shortest vector 33
4.3.4 Short vector enumeration heuristic 37

5 Cholesky matrix end diagonal values **39**
5.1 Concept usage experiments 42
5.1.1 Lattice basis ordering key for BKZ reduction . . 42

5.1.2 Lattice basis ordering key for enumeration tree size 44

6 Short vectors by Gram matrix quadrants **47**
6.1 Concept usage experiments 49
6.1.1 Short vectors coefficients in LLL reduced bases . 49
6.1.2 Sequential coefficient combination search 52
6.1.3 Random operations coefficient combination search 57

7 Precomputed transformation matrices **60**
7.1 Pearson correlation matrix 60
7.1.1 Pearson correlation matrix computation example 61
7.2 Concept description 62
7.3 Concept usage experiment 66

8 Conclusion **70**

Bibliography **72**

Appendix **76**

List of Tables

4.1 Comparison of results of BKZ reduction of lattice bases sorted by optimal real coefficients and of unsorted bases. 28

4.2 Computed optimal real values during enumeration which are in mod 1 threshold and which also predict next optimal integral coefficient value. 37

5.1 Comparison of results of BKZ reduction of lattice bases sorted by recursive biggest Cholesky end diagonal values and of unsorted bases. 43

5.2 Average ratios of enumeration tree sizes of unsorted lattice bases and lattice bases sorted by Cholesky decomposed Gram matrix recursive biggest end diagonal values belonging to lattice basis vectors (LLL reduced lattice bases). 45

5.3 Average ratios of enumeration tree sizes of unsorted lattice bases and lattice bases sorted by Cholesky decomposed Gram matrix recursive biggest end diagonal values belonging to lattice basis vectors (unreduced lattice bases). 45

6.1 Found rate of coefficient combination fulfilling required Gram matrix quadrant relation and rate of those that also represent vector with length in required vector length bound (sequential full variant). 54

6.2 Found rate of coefficient combination fulfilling required Gram matrix quadrant relation and rate of those that also represent vector with length in required vector length bound (sequential heuristic variant). 56

6.3 Rate of short vectors found by persisting random vector coefficient changes based on Gram matrix upper right quadrant negative sum. 59

List of Figures

2.1 Visualization of a 2 dimensional lattice. 4

3.1 Enumeration tree construction example with first coefficient value possibilities. 18

3.2 Enumeration tree construction example with second coefficient value for the $c_2 = -1$ case. 19

3.3 Enumeration tree construction example with third coefficient value for the $c_2 = -1$, $c_1 = 0$ case. 20

3.4 Enumeration tree construction example with final enumeration tree. 21

4.1 Distribution of sums of differences of optimal real coefficients from coefficients of the shortest vector (dimension 5, 2000 unreduced lattice bases). 30

4.2 Distribution of sums of differences of optimal real coefficients from coefficients of the shortest vector (dimension 5, 2000 reduced lattice bases). 31

4.3 Distribution of sums of differences of optimal real coefficients from coefficients of the shortest vector (dimension 10, 2000 unreduced lattice bases). 32

4.4 Distribution of sums of differences of optimal real coefficients from coefficients of the shortest vector (dimension 10, 2000 reduced lattice bases). 33

4.5 Distribution of differences of individual optimal real coefficient values of the next enumerated coefficient from the optimal integral coefficients (dimension 5, 1000 LLL reduced lattice bases). 35

4.6 Distribution of differences of individual optimal real coefficient values of the next enumerated coefficient from the optimal integral coefficients (dimension 10, 200 LLL reduced lattice bases). 36

6.1 Ratios across dimensions of enumerated short vectors from LLL reduced basis which have only {-1, 0, 1} coefficient values. 51

6.2 Average running times of short vector enumeration across dimensions. 51

7.1 Histogram of reduction rates (root Hermite factor) for precomputed transformation matrix mapping (dimension 5, interval size 0.4, 5000 lattice bases). 67
7.2 Histogram of reduction rates (root Hermite factor) of 5000 unreduced and 312 LLL reduced lattice bases (5 dimensional). 68

1 Introduction

Nowadays, it is widely believed that the age of quantum computers is approaching. However, quantum computers and the quantum attacks they could possibly perform pose a significant threat in many fields of information security. That is the reason why many researchers focus on potential future Post-Quantum Cryptography, which should be resistant to quantum attacks.

One of the leading candidates for this category of cryptography is believed to be Lattice-Based Cryptography because of its underlying problems being believed to be resistant to attacks from quantum computers. It is also a fact that one of key underlying concepts of the Lattice-Based Cryptography is the Shortest Vector Problem. It is however not necessarily needed to find the exact shortest vector of the lattice. [1, 2]

Therefore, the primary focus of this thesis was to find a method to swiftly find sufficiently short lattice vectors in small dimensions (< 40). The potentially found method could then be used as a practical utility for popular lattice basis reduction algorithms, such as LLL or BKZ. To be more precise, the main goal was to swiftly find a lattice vector short enough to be within the bounds defined by the Euclidean norms of LLL reduced lattice basis vectors.

2 Used concepts

This thesis assumes knowledge of linear algebra, calculus, statistics and computer science equal to that of a bachelor's degree graduate from Faculty of Informatics, Masaryk University. This includes the knowledge of concepts such as Euclidean space, vector space or transformation matrix from linear algebra, partial derivation from calculus, correlation or standard deviation from statistics, or BFS, DFS or error detection codes from computer science.

Concepts expected to be unknown are either explained in the following sections of this chapter or, if only used locally, explained at the start of the chapter with relevant context.

2.1 Concepts context introduction

Lattices are used in many fields, including Coding Theory for error-correction codes and Lattice-Based Cryptography, which is one of the leading candidates for Post-Quantum Cryptography because the underlying problems are believed to be resistant to quantum attacks. Key underlying concepts mainly include the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP), which are closely related to lattice basis reduction. [1, 2]

Q-ary lattices is a type of lattices which is used for lattice generating processes for lattice basis reduction algorithms testing for their adequate reduction difficulty. This type of lattices was also generated during all analyses in the scope of this thesis. Q-ary lattices are also known for their usage in the Learning With Errors (LWE) problem. [3, 4]

In terms of lattice basis reduction, there are many extents into which a lattice basis can be reduced, which include LLL reduction or KZ reduction. In the context of this thesis, the LLL reduction algorithm was used on places where the relevant testing required lattice bases which were partially reduced for various purposes. These purposes include computational complexity of a relevant test since for example short vector enumeration is significantly less computationally expensive on LLL reduced lattice bases. Other purpose might be the

fact that the relevant analysis required some LLL reduced lattice basis attributes like short vector coefficient values distribution.

The BKZ lattice basis reduction was, during the work on this thesis, used with block size parameter set to the dimension of the reduced lattice basis, which made it effectively equal to the KZ reduction. The BKZ reduction algorithm was used for its higher computational complexity and reduction rate in comparison to the LLL reduction algorithm in mentioned configuration for multiple analyses of lattice basis sorting methods which could potentially result in even better reduction rate or lower computational complexity (computational time) of the used BKZ reduction algorithm.

There are many methods that can be used to measure the degree to which a lattice basis is reduced, such as the Root Hermite factor or the Hermite factor. The root Hermite factor was used to determine the rate at which lattice bases were reduced when testing or comparing various reduction approaches or reduction algorithm inputs during the work on this thesis.

Also, the Gram matrix of a lattice basis was analyzed for possible relation between its quadrants. The Gram matrix was also used during Cholesky decomposition and for normalization for mapped reduction transformation matrix precomputation.

Furthermore, in this thesis, Cholesky decomposition was used on Gram matrix of lattice basis for lattice basis sorting by recursively found biggest end diagonal values in the computed lower triangular matrix. It was also used in my short vector enumeration implementations, since it was useful for computing intervals of lattice basis vector coefficient values to travel in the enumeration tree.

2.2 Concepts description

2.2.1 Lattice

Let $B = [b_0, ..., b_{n-1}]$ be a basis defined by a set of linearly independent vectors $\{b_0, ..., b_{n-1}\}$ in $\mathbb{R}^m$ (where $n \leq m$). A lattice $\mathcal{L}$ generated by this basis is the set of all possible integral linear combinations of these vectors. [1]

Mathematically, the lattice can be described as

$$\mathcal{L}(b_0, ..., b_{n-1}) = \left\{ \sum_{i=0}^{n-1} x_i b_i : x_i \in \mathbb{Z} \right\}.$$

On the following visualized example of a 2 dimensional lattice, there are 3 vectors. The 2 red vectors (b_1 and b_2) are lattice basis vectors. The third one, b_3, is a vector which is in the lattice but is not part of the lattice basis, and is defined as linear lattice basis vector combination $b_1 - 2b_2$. Every black point represents one possible linear lattice basis vector combination, and therefore one vector which is in the lattice, with black points continuing infinitely in the same fashion in all directions.

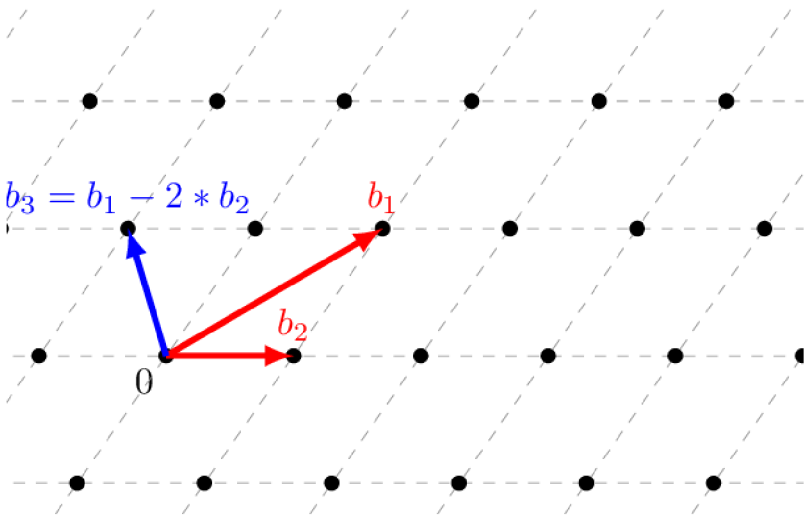

Figure 2.1: Visualization of a 2 dimensional lattice. [5]

Q-ary lattice

Q-ary lattice is a lattice with a generation process based on modular arithmetic. A lattice $\mathcal{L}$ is called q-ary, if it contains all integer vectors which are multiples of q, where $q \geq 2$, $q \in \mathbb{Z}$ is configurable quotient (usually prime number). [4, 6]

This can also be mathematically described, given $\mathcal{L}$ is a lattice and d is the dimension of the lattice, as

$$(q\mathbb{Z})^d \subseteq \mathcal{L} \subseteq \mathbb{Z}^d.$$

2.2.2 Lattice basis reduction

Lattice basis reduction is a process which aims to transform a lattice basis to a lattice basis with vectors of smaller Euclidean norm (vector length), or find a transformation matrix which transforms the lattice basis to one with shorter basis vectors. It can also be explained as searching for a sequence of linear combinations of lattice basis vectors with the purpose to transform the lattice basis to one with shorter basis vectors.

2.2.3 Shortest vector problem

Shortest vector problem (SVP) is a fundamental computational problem in the study of lattices. Given a lattice $\mathcal{L}$, the shortest vector problem aims to find a non-zero vector $v \in \mathcal{L}$ such that its length (Euclidean norm) is minimal and such that belongs to the given lattice. [7, 8]

Currently it is believed that this problem has exponential deterministic complexity and polynomial nondeterministic complexity (the problem is NP-hard). This problem is commonly solved by either

- algorithms searching for exact solution, such as enumeration
- or by algorithms searching for approximate solution, such as LLL lattice basis reduction algorithm. [9]

Approximate shortest vector problem

Approximate SVP, or short vector problem, aims to find a non-zero vector such that belongs to a lattice $\mathcal{L}$ and such that its Euclidean norm is lower or equal to given upper bound γ ($v \in \mathcal{L} \ \wedge ||v|| \leq \gamma$). [1]

2.2.4 Short vector enumeration

Short vector enumeration (SVE), is a fundamental algorithmic technique used to solve SVP. In its core, given a lattice $\mathcal{L}$ with basis $B = [b_0, ..., b_{n-1}]$, the goal is to find all vectors $v \in \mathcal{L}$ such that $||v|| \leq R$, where R is configurable upper bound, often called a radius. Hypersphere in an Euclidean space with R being its radius can be imagined. This approach is useful if the aim is to find all short vectors in a given

norm bounds, or in other words, to perform full short vector enumeration. This method is also useful for solving the approximate shortest vector problem. [10, 11]

However, if the goal is only to find the shortest vector, **pruned** SVE can be used. Pruning is a technique where a mechanism is applied which can make a decision leading to a branch of a tree being skipped, and therefore not traveled at all. There are various types of pruning types, including pruning methods which cut enumeration tree branches which have low chance of containing shortest vector, but not zero (called extreme pruning). Basic pruning methods which keep their exactness in finding shortest non-zero vector, but do not guarantee to find all short vectors, include continuous lowering of the hypersphere radius by the shortest currently found vector norm. [11]

Usually, the SVE views the collection of all possible lattice vectors in given bounds as a search tree using a DFS strategy to traverse it, where each level of the tree corresponds to a coefficient in the linear combination of the lattice basis. [10, 11]

For DFS variants, the typical space complexity is in $O(d^2)$, where d is input lattice basis dimension. Typical computation complexity is in $2^{\Theta(d \log d)}$. Therefore, since enumeration process is so computationally costly, it is usually used on lattice bases which are already reduced by less precise but faster reduction algorithms such as LLL. [12]

Example of the short vector enumeration process using DFS approach can be found in this thesis in the subsection (3.1.1).

2.2.5 Lenstra-Lenstra-Lovász algorithm

The LLL algorithm is a polynomial-time lattice basis reduction algorithm serving primarily as an efficient way to provide an approximate solution to the Shortest vector problem. Even though the SVP problem is NP-hard, the LLL algorithm is able to find, in polynomial time, a vector short enough to be within an exponential factor of the true minimum. [13, 14]

The LLL lattice basis reduction algorithm is an algorithm which transforms a lattice basis into a basis with short and nearly orthogonal vectors. After the LLL reduction algorithm finishes, the resulting lattice basis is considered to be **LLL reduced**. [14]

For a d-dimensional lattice with basis vectors of norm less than γ and of dimension n, the original LLL algorithm finishes execution in $O(d^5 n \, log^3 \gamma)$ bit operations. [15]

2.2.6 Korkine-Zolotarev algorithm

The KZ algorithm, or HKZ (Hermite-Korkine-Zolotarev) algorithm, is an exponential-time lattice basis reduction algorithm. It guaranties stronger lattice basis reduction and stronger lattice basis vectors orthogonality in comparison to the LLL reduction algorithm. Also, it guarantees to find the exact shortest vector λ_1 as b_0 of the resulting reduced basis, whereas LLL reduction algorithm does not. After the KZ reduction algorithm finishes, the resulting lattice basis is considered to be **KZ reduced**. [16]

For a d-dimensional lattice, the KZ lattice reduction algorithm computational complexity is in $2^{O(d)}$. [16]

Block Korkine-Zolotarev algorithm

The BKZ reduction algorithm was introduced because KZ reduction algorithm was too slow for high dimensions and LLL algorithm was fast, but not precise enough in its reduction result. The BKZ algorithm is an algorithm performing KZ reduction process flexibly on dimensional adjacency defined by the configuration parameter β, called **block size**. If $\beta = 2$, result of BKZ reduction is similar to the result of LLL reduction. If the $\beta = d$, where d is the dimension of the lattice basis which is to be reduced, the result is the same as the result of classical KZ reduction algorithm. Therefore, BKZ offers a configurable middle ground between classical KZ and LLL reduction algorithm with the option of balancing between computation complexity and reduction precision (lattice basis reduction rate). The essence of BKZ then logically leads to its resulting computational complexity being in $2^{O(\beta)}$. [17, 18]

2.2.7 Gram matrix

Given a lattice basis $B = [b_0, ..., b_{n-1}]$, Gram matrix G is a square matrix where each entry $G[i][j]$ is the inner product of the i-th and j-th vectors of the basis, also noted as

$$G[i][j] = \langle b_i, b_j \rangle.$$

Or, given these vectors are rows of the matrix B, then the Gram matrix can be computed simply by

$$G = BB^T.$$

Therefore, the Gram matrix is always symmetric ($G = G^T$), since the inner product is commutative ($\langle b_i, b_j \rangle = \langle b_j, b_i \rangle$). [19]

Gram matrix computation example

For example, lets assume that it is required to compute the Gram matrix from a lattice $\mathcal{L}$ with its basis with basis row vectors $B = [b_0, b_1, b_2]$ being

$$B = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 4 \end{bmatrix}.$$

Then it is true that $b_0 = (2, 1, 2)$, $b_1 = (1, 3, 1)$, $b_2 = (1, 1, 4)$. To compute the resulting Gram matrix, both of the described approaches can be used in this case, since the lattice basis vectors are row vectors of the lattice basis. Therefore, the following process can be either to solve

$$G = \begin{bmatrix} \langle b_0, b_0 \rangle & \langle b_0, b_1 \rangle & \langle b_0, b_2 \rangle \\ \langle b_1, b_0 \rangle & \langle b_1, b_1 \rangle & \langle b_1, b_2 \rangle \\ \langle b_2, b_0 \rangle & \langle b_2, b_1 \rangle & \langle b_2, b_2 \rangle \end{bmatrix}$$

or

$$G = BB^T = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 4 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 3 & 1 \\ 2 & 1 & 4 \end{bmatrix}.$$

For example, to solve the scalar vector product $\langle b_0, b_1 \rangle$, where $b_0 = (2, 1, 2)$, $b_1 = (1, 3, 1)$, it is simply required to compute $b_{00}b_{10} + b_{01}b_{11} + b_{02}b_{12} = 2 \cdot 1 + 1 \cdot 3 + 2 \cdot 1 = 7$.

In the end, the resulting Gram matrix in this case is

$$G = \begin{bmatrix} 9 & 7 & 11 \\ 7 & 11 & 8 \\ 11 & 8 & 18 \end{bmatrix}.$$

2.2.8 Cholesky decomposition

The goal of Cholesky decomposition process is to find an L lower triangular matrix, assuming G is a Gram matrix, such that

$$G = LL^T.$$

To compute elements, or row vectors L_i, assuming $L[i][j]$ is an entry of the matrix L, $G[i][j]$ is an entry of the Gram matrix G, $0 \leq i \leq d-1 \wedge i \in \mathbb{Z}$, $0 \leq j \leq d-1 \wedge j \in \mathbb{Z}$, where d is the dimension of the Gram matrix, the formulas used are [20]

$$L[i][j] = \begin{cases} \sqrt{G[i][i] - \sum_{k=0}^{i-1} L[i][k]^2} & \text{if } i = j \\ \frac{1}{L[j][j]} \left(G[i][j] - \sum_{k=0}^{j-1} L[i][k]L[j][k] \right) & \text{if } i > j \\ 0 & \text{if } i < j \end{cases}.$$

It is important to note that the resulting lower triangular lattice L computed from Gram matrix

1. is triangular and therefore hierarchical,
2. represents the lattice basis B and
3. has its diagonal value equal to length of according Gram-Schmidt Orthogonalization (GSO) basis vector b_i^* such that $L[i][i] = ||b_i^*||$.

Therefore, the enumerated attributes allow L to be effectively used to compute lattice basis vector coefficient intervals in usage with relevant configured short vector Euclidean norm upper bound R during full short vector enumeration.

Cholesky decomposition computation example

Lets assume that it is required to compute Cholesky decomposed lower triangular matrix L from a Gram matrix G where

$$G = \begin{bmatrix} 9 & 7 & 11 \\ 7 & 11 & 8 \\ 11 & 8 & 18 \end{bmatrix}.$$

A correct approach to compute the resulting lower triangular matrix L can be by defining $i = j = 0$ followed by:

1. First position of the resulting matrix $L[0][0]$ is computed by the formula for situation where $i = j$, which is $\sqrt{G[i][i] - \sum_{k=0}^{i-1} L[i][k]^2}$, and so the computation in this situation is $\sqrt{G[0][0] - \sum_{k=0}^{-1} L[0][k]^2}$ which is $\sqrt{9-0} = 3$.

2. Then, j is set to $j + 1 = 1$. And the process is repeated with the exception that now the relevant formula is different since $i \neq j$, because $i < j$. Therefore the resulting value of $L[0][1] = 0$, as defined by the computation formulas.

3. As next, j is set to $j + 1 = 2$. Again, the situation is $i < j$, and so $L[0][2] = 0$.

4. Now it is not possible to increment j since $j \leq d - 1$, where $d = 3$ is the dimension of the Gram matrix. j is therefore set to 0 and i is incremented, and so $i = 1$. Since the situation is now $i > j$, the used formula for the computation of $L[1][0]$ will be $\frac{1}{L[j][j]} \left(G[i][j] - \sum_{k=0}^{j-1} L[i][k] L[j][k] \right)$, which in this situation equals $\frac{1}{L[0][0]} \left(G[1][0] - \sum_{k=0}^{-1} L[1][k] L[0][k] \right)$ which is $\frac{G[1][0]}{L[0][0]} = \frac{7}{3}$, and so $L[1][0] = \frac{7}{3}$.

5. Then, by following this process all the way to the situation where $i = j = 2$, the situation where L is fully computed is reached. The resulting L is then

$$L = \begin{bmatrix} 3 & 0 & 0 \\ \frac{7}{3} & \frac{5\sqrt{2}}{3} & 0 \\ \frac{11}{3} & -\frac{1}{3\sqrt{2}} & \frac{9}{2} \end{bmatrix}.$$

2.2.9 Root Hermite factor

The Root Hermite Factor is a metric used to measure the rate at which a lattice has its basis reduced, or the quality of its reduction. To be more precise, for a lattice $\mathcal{L}$ of dimension d and a nonzero vector $v \in \mathcal{L}$, if it is assumed that $det(\mathcal{L})$ is the determinant of the lattice basis and v is the shortest vector from the lattice basis vectors, the RHF (δ_0) is defined as

$$\delta_0 = \left(\frac{||v||}{det(\mathcal{L})^{\frac{1}{d}}} \right)^{\frac{1}{d}}.$$

It is important to note that the closer to 1 the δ_0 is, the better the lattice reduction is. [21]

To explain the root Hermite factor differently, the shown formula represents how much the length of the shortest found vector ($||v||$) deviates from the "average" length of a vector in that lattice, since $det(\mathcal{L})^{\frac{1}{d}}$ represents geometric mean of the lattice. The outer exponent $\frac{1}{d}$ then represents a normalization of the factor result for better result comparison.

For comparison, usual values of δ_0 for LLL reduced lattice bases are approximately in (1.0219, 1.075) and for BKZ with $\beta = d$, where d is the dimension of the lattice basis, the value of δ_0 should theoretically be 1. [22, 23]

Root Hermite factor computation example

For example, lets assume it is desired to compute reduction rate, as shown by root Hermite factor, of a basis $B = [b_0, b_1, b_2]$ of a lattice $\mathcal{L}$, where basis vectors are row vectors and

$$B = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 4 \end{bmatrix}.$$

A correct approach is to do as follows:

1. Find the shortest vector from lattice basis vectors, which can be done by computing individual lattice basis vector lengths and choosing the vector with the shortest one. The resulting computations are

$$||b_0|| = \sqrt{b_{00}^2 + b_{01}^2 + b_{02}^2} = \sqrt{4+1+4} = 3,$$

$$||b_1|| = \sqrt{b_{10}^2 + b_{11}^2 + b_{12}^2} = \sqrt{1+9+1} = \sqrt{11}$$

and

$$||b_2|| = \sqrt{b_{20}^2 + b_{21}^2 + b_{22}^2} = \sqrt{1+1+16} = 3\sqrt{2}.$$

2. Since $3 < \sqrt{11} < 3\sqrt{2}$, it is found out that the shortest vector from lattice basis vectors is b_0.

3. Now, it is required to compute the determinant of the lattice $\det(\mathcal{L})$, and so the determinant of the lattice basis $\det(B)$, and since $\det(\mathcal{L}) = |\det(B)|$, because B is square matrix, the resulting computation can for example be, if rule of Sarrus is used,

$$\det(B) = (B_{00}B_{11}B_{22} + B_{01}B_{12}B_{20} + B_{02}B_{10}B_{21}) -$$

$$(B_{20}B_{11}B_{02} + B_{21}B_{12}B_{00} + B_{22}B_{10}B_{01}) =$$

$$(2 \cdot 3 \cdot 4 + 1 \cdot 1 \cdot 1 + 2 \cdot 1 \cdot 1) - (1 \cdot 3 \cdot 2 + 1 \cdot 1 \cdot 2 + 4 \cdot 1 \cdot 1) =$$

$$27 - 12 = 15$$

and

$$\det(\mathcal{L}) = |\det(B)| = |15| = 15.$$

4. Now, to compute the resulting root Hermite factor, since it is known that in this case, $d = 3$, $v = b_0$, it is required to solve

$$\delta_0 = \left(\frac{||v||}{det(\mathcal{L})^{\frac{1}{d}}}\right)^{\frac{1}{d}} = \left(\frac{3}{15^{\frac{1}{3}}}\right)^{\frac{1}{3}} = \frac{3^{\frac{2}{9}}}{5^{\frac{1}{9}}} = \sqrt[9]{\frac{9}{5}} \approx 1.067.$$

3 Preparation for method analysis

In the first place, it was crucial to prepare tools for analysis of methods proposed. These tools include lattice basis generator and short vector enumeration variants used for comparison with analyzed methods.

3.1 Lattice basis generation process

To generate lattice bases for analyses of proposed concepts (methods), a built-in SageMath method[1] was used as a black box in a specific configuration.

The configuration was designed so that it is sufficient for generating lattices that have mediocre-to-hard basis reduction difficulty. This was accomplished, assuming that *dimension* is dimension of the generated lattice, by configuring

- the **quotient** parameter $q = 11^{1.6dimension}$ so that the resulting lattice basis vector values are close in size to those of a result of widely used SVP challenge generator[2],
- the **lattice basis type** parameter $type = random$, which is recommended for testing basis reduction algorithms [3] and causes resulting lattice basis to be a q-ary lattice,
- the **determinant size** parameter $n = 1$, which needs to be set to this value in order to use the $type = random$ parameter configuration
- and the **lattice dimension** parameter $m = dimension$.

It is important to note that even though q parameter was set with the mentioned intention, the resulting generated lattice bases, when compared to those generated by the mentioned SVP challenge generator, had better (lower value) reduction rate computed by root Hermite factor ($\delta_0 \approx 45$ instead of $\delta_0 \approx 950$, compared in dimensions 40 and 100).

1. https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/lattice.html#sage.crypto.lattice.gen_lattice
2. https://www.latticechallenge.org/svp-challenge/generator.php

3.2 Used enumeration variants

Next, it was crucial to find a correct and reliable method to find short lattice vectors with lengths in configured bounds for comparison with later analyzed methods. For this purpose, 3 variants of full short vector enumeration were used, which include

1. an **built-in SageMath method**[3],
2. an **implemented BFS variant**
3. and an **implemented DFS variant**.

The first of the mentioned methods, the built-in SageMath method, proved to be the most precise and the fastest of all the variants used. This was due to its internal implementation in C and various possible optimizations.

However, although this variant was correct for the required purpose, it was not sufficient for testing heuristics or enumeration-pruning methods because it functioned as a black box. Therefore, it was not possible to modify its implementation to integrate any tested pruning method or heuristic.

This fact led to the creation of the other two mentioned variants, which were dedicated implementations designed with greater flexibility in mind.

3.2.1 Implemented enumeration variants concept

Both of implemented variants were based on keeping computed lower triangular matrix L from Gram matrix Cholesky decomposition process. Lattice basis vector indices were then iterated in reverse order because of the structure of the triangular matrix.

3. `https://doc.sagemath.org/html/en/reference/modules/sage/modules/free_quadratic_module_integer_symmetric.html#sage.modules.free_quadratic_module_integer_symmetric.FreeQuadraticModule_integer_symmetric.short_vectors`

In each such iteration, enumeration tree depth or basis vector coefficient index, possible chosen coefficient values interval was computed from

1. configured vector length bound squared R^2,
2. already chosen coefficients in given enumeration tree path P
3. and relevant triangular matrix L values.

It is important to note that the interval was computed by computing current length bound reduction squared from P and values of L. This reduction squared R^2_{red} was computed, assuming that d is Gram matrix dimension, by

```
procedure LengthBoundRed(L, d, P)
    R²_red ← 0
    for j in 0..|P| do
        LColSum ← 0
        for i in (d − 1)..(d − (j + 1) − 1) do
            LColSum ← LColSum + L[i][d − 1 − j] · P[d − 1 − i]
        end for
        R²_red ← R²_red + LColSum²
    end for
    return R²_red
end procedure.
```

Then, the resulting coefficient interval was $[-coeffB, coeffB]$, where $coeffB$ was computed by

```
procedure CoefficientBound(R², R²_red, L, d, P)
    LColSum ← 0
    for i in (d − 1)..(d − 1 − |P|) do
        LColSum ← LColSum + L[i][d − 1 − |P|] · P[d − 1 − i]
    end for
    dividend ← √(R² − R²_red) - LColSum
    coeffB ← dividend / L[d−1−|P|][d−1−|P|]
    return coeffB
end procedure.
```

Then, for each coefficient value of the computed interval, the next coefficient index was taken into account instead and the process was repeated, until the last coefficient index was reached. In that case, the current lattice basis vector coefficient combination, or enumeration tree path P, was added into the resulting collection of found short vector coefficient combinations.

Implemented enumeration variants difference

First implemented variant was full short vector enumeration in configured Euclidean norm bounds based on **DFS approach**. Global Euclidean norm upper bound was mutable and shared across the whole process. This approach allowed to reach first complete coefficient combinations very soon. These early complete coefficient combinations could then be useful for global norm bound reduction very soon in the enumeration process. This early reduction could then lead to a significant impact on resulting tree size and, consequently, the enumeration computation time.

In comparison, the second created variant was based on **BFS approach** and therefore reached all complete coefficient combinations at a similar time, since the enumeration process was traveling through the enumeration tree in depth layers. This variant was primarily designed to be easily extensible by various heuristics.

3.2.2 Example of use of the DFS enumeration variant

Lets assume there is a 3 dimensional lattice $\mathcal{L}$ with the lattice basis $B = [b_0, b_1, b_2]$ where the basis vectors are row vectors and B is an identity matrix. This then means that

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Lets also assume that $R = 1$ is used when enumerating all short vectors. The implemented DFS enumeration process is then following:

1. First, Gram matrix G is calculated from B and then a lower triangular matrix L is calculated from the Gram matrix G using the Cholesky decomposition process. Since B is an identity matrix, then G and L are also identity matrices.

2. Next, its calculated, which coefficient values c_2 can be picked for the basis vector b_2. This is done by using

   - current bound reduction R_{red}, which is 0 since no coefficient values are chosen for this enumeration tree path yet,
   - values of L, which is an identity matrix,
   - the configured bound R, which is 1,
   - current enumeration tree path P, which is an empty array
   - and current basis dimension d, which is 3

   in the procedure call

   $$coeffB_{c_2} = CoefficientBound(R^2,\ R^2_{red},\ L,\ d,\ P) = 1.$$

   And since $c_2 \in [-coeffB_{c_2},\ coeffB_{c_2}]$, $c_2 \in \mathbb{Z}$, then $c_2 \in \{-1,\ 0,\ 1\}$.

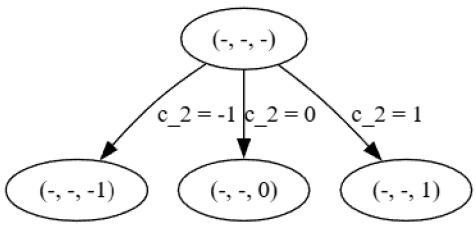

Figure 3.1: Enumeration tree construction example with first coefficient value possibilities.

3. Next, it is moved to the first of the possible cases, where $c_2 = -1$, and the process repeats, but with the change that $P = [-1]$ and that now the upper bound reduction R_{red} must be calculated since P is not empty. This will be done by the procedure call

   $$R^2_{red} = LengthBoundRed(L, d, P) = 1.$$

4. Now, since R^2_{red} is known for current case, the same process to find out current coefficient bound $coeffB(-, -, -1)$ as before will be used, and so

   $$coeffB(-, -, -1) = CoefficientBound(R^2, R^2_{red}, L, d, P) = 0.$$

   Therefore, only possible coefficient value to pick in the current case for c_1 is 0.

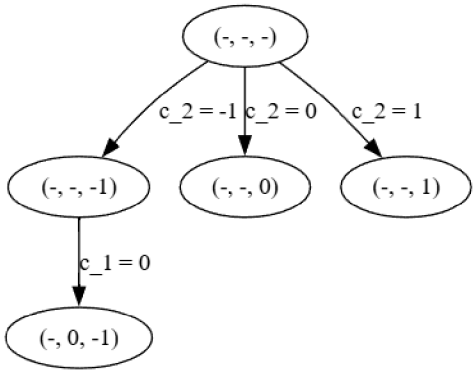

Figure 3.2: Enumeration tree construction example with second coefficient value for the $c_2 = -1$ case.

5. Then, it is moved to the only possible case, where $c_1 = 0$, and the same process is repeated to figure out the value for c_0 in this case, which leads to computing that $c_0 \in \{0\}$.

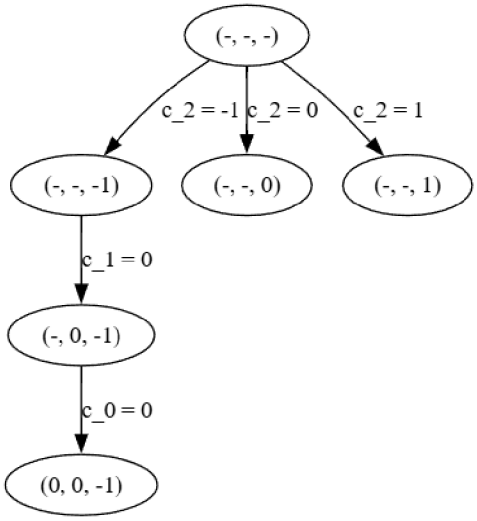

Figure 3.3: Enumeration tree construction example with third coefficient value for the $c_2 = -1$, $c_1 = 0$ case.

6. Now, the first leaf node of the enumeration tree was reached, and so it is moved back to the case, where $c_2 = 0$, and the process is repeated in the same way, until the whole tree is known.

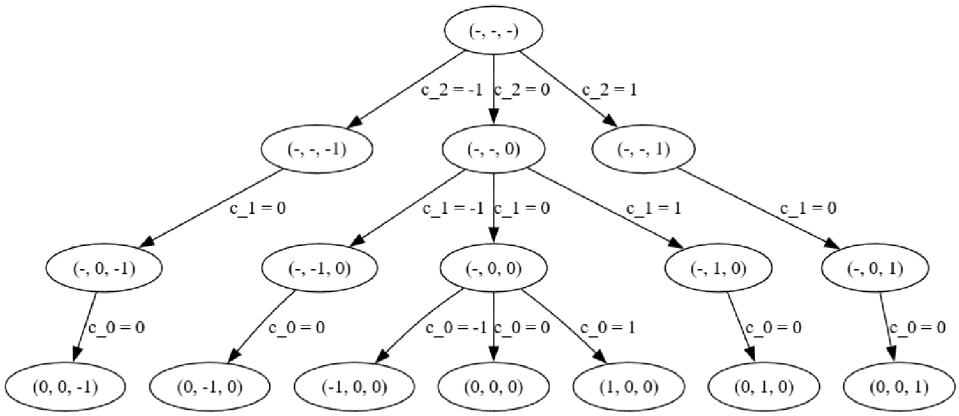

Figure 3.4: Enumeration tree construction example with final enumeration tree.

7. Finally, all leaf coefficient combination nodes of the enumeration tree, where each of them represents one found short vector $v \in \mathcal{L}$, $||v|| \leq R$, or its lattice basis vector coefficient value combination row vector to be more precise, are retrieved. Usually, the zero vector, or the coefficient combination $(0, 0, 0)$, is discarded from the result, since it has no use.

8. To find out, what the resulting short vector is, given that x is one of the resulting coefficient combination row vectors, all that is needed is to simply compute $v = xB$. For example, if $x = (-1, 0, 0)$, then $v = (-1, 0, 0)$ because B is an identity matrix.

Therefore, in the end, the resulting found meaningful short vectors in this example are $\{(-1, 0, 0), (1, 0, 0), (0, -1, 0), (0, 1, 0), (0, 0, -1), (0, 0, 1)\}$. It is important to note that all resulting found short vectors are equal to their coefficient combination row vectors because B is an identity matrix.

3.2.3 Comparison of implemented and built-in variants

During testing of enumeration variants, the built-in SageMath enumeration variant and the implemented BFS enumeration variant were compared.

Lets assume that

- $S_{inbuilt}$ is a set of all complete short vector coefficient combinations from enumeration process using the built-in Sagemath enumeration variant
- and S_{impl} is a set of all complete short vector coefficient combinations from enumeration process using the implemented BFS variant.

The test process

1. generated a set of lattice bases of a given sample size and dimension,
2. reduced generated lattice bases by using built-in LLL reduction algorithm method[4] with default configuration for sufficiently small enumeration tree sizes,
3. enumerated each of the LLL reduced lattice bases using both mentioned variants, with short vector length bound set to the norm of the longest basis vector, and therefore created $S_{inbuilt}$ and S_{impl} for each of the reduced bases
4. and compared, whether $S_{inbuilt} = S_{impl}$ for each of the reduced bases.

4. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL`

The results were following for given sample sizes and dimension configurations:

- **dimension 3, sample size 200**: 183 / 200
- **dimension 5, sample size 200**: 172 / 200
- **dimension 10, sample size 100**: 75 / 100

The test results propose that the compared enumeration variants results can differ. This led to further analysis of $S_{inbuilt}$ and S_{impl} and of the enumeration process of the implemented BFS variant.

It was then found out that both compared variants find short vectors in requested vector length bound. However, the implemented BFS variant has a low chance of not finding short vectors which have their vector length very close to the configured bound. This is thanks to frequent numeric type conversions which occur during the enumeration process of the BFS variant.

For example, lets assume that $B = [b_0, b_1, b_2, b_3, b_4]$ is an LLL reduced lattice basis, where basis vectors are row vectors, and

$$B = \begin{bmatrix} 0 & -14 & -10 & -9 & 22 \\ 1 & 3 & -23 & 23 & -15 \\ 15 & 40 & 4 & -5 & 40 \\ 56 & 1 & 4 & 20 & 11 \\ 40 & 27 & -45 & -39 & -10 \end{bmatrix}.$$

Lets also assume that both compared enumeration variants are executed on this reduced lattice basis with upper bound $R = ||b_4|| = \sqrt{5975} = 5\sqrt{239}$, since b_4 is the longest basis vector.

In this case, both compared variants find the same 61 short vectors, with the difference that the built-in SageMath variant also finds one additional short vector with complete coefficient combination $c_{61} = (-2, -2, 0, 0, 0)$ and therefore $sv_{61} = c_{61}B = (-2, 22, 66, -28, -14)$ where $||sv_{61}|| = \sqrt{5824} = 8\sqrt{91}$.

This lead to the conclusion that the SageMath enumeration variant will be used for cases where no enumeration pruning method or enumeration heuristic is tested.

4 Optimal real basis vector coefficients

Since reliable method for comparison of analyzed methods results was prepared, it was time to start searching for possibly suitable methods for the purpose of this thesis. The first analyzed method was the usage of optimal real lattice basis vector coefficients computation. To understand, how this method works, it is first needed to understand what a gradient is.

4.1 Gradient

For a multivariable function $f(x_0, ..., x_{n-1})$, the gradient (∇) is a vector-valued function whose components are the partial derivatives of f. [24]

This can also be mathematically noted as

$$\nabla f(x_0, ..., x_{n-1}) = \begin{pmatrix} \frac{\partial f}{\partial x_0} & \frac{\partial f}{\partial x_1} & ... & \frac{\partial f}{\partial x_{n-1}} \end{pmatrix}^T.$$

For example, if there is a 3 variable function

$$f(x, y, z) = 3x^2 + sin(y) - e^z,$$

then computing gradient of the mentioned function would mean computing

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \end{pmatrix}^T.$$

This computation then can be divided into three parts, where each part represents one row, or one partial derivation, of the resulting matrix, which leads to computation resulting in

$$\nabla f = \begin{pmatrix} 6x \\ cos(y) \\ -e^z \end{pmatrix}.$$

4.2 Concept description

Lets assume that there is

- a lattice $\mathcal{L}$ with basis $B = [b_0, ..., b_{d-1}]$,
- a Gram matrix G constructed from the lattice basis
- and a column vector of lattice basis vector coefficients $c = (c_0, ..., c_{d-1})$, where each value represents either lattice basis coefficient value or a symbol c_i, $0 \leq i \leq d-1$.

Then, by solving

$$\nabla_c \left(c^T G c \right) = 2Gc = 0\,,$$

optimal values for coefficient symbols c_i can be computed such that

$$\exists c_i (c_i \in \mathbb{R} \ \wedge \ c_i \notin \mathbb{Z})\,.$$

It is important to note that at least one nonzero nonsymbolic coefficient value c_i is recommended in the input column vector c, since otherwise the result would be a zero vector.

To formulate it differently, this concept focuses on computing real minimum of a quadratic form by solving a system of equations of partial derivations resulting from $\nabla_c \left(c^T G c \right)$.

However, it is also important to note that this concept cannot be used directly to find the shortest vector with given fixated coefficient values, since the computed coefficient values c_i are not guaranteed to be integral, and therefore do not guarantee to represent a vector which is in the given lattice $\mathcal{L}$.

For example, let there be a lattice $\mathcal{L}$ with its basis of row vectors $B = [b_0, b_1, b_2]$, where

$$B = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 4 \end{bmatrix}.$$

The resulting Gram matrix G computed from the basis B is therefore

$$G = \begin{bmatrix} 9 & 7 & 11 \\ 7 & 11 & 8 \\ 11 & 8 & 18 \end{bmatrix}.$$

Lets also assume that there is a single fixated coefficient value $c_0 = 1$, and therefore $c = (1, c_1, c_2)$. The process to compute optimal real values for unknown coefficient values is then solving

$$\nabla_c \left(c^T G c \right) = 2Gc = 0 \rightarrow Gc = 0 \rightarrow$$

$$\rightarrow \begin{bmatrix} 9 & 7 & 11 \\ 7 & 11 & 8 \\ 11 & 8 & 18 \end{bmatrix} \begin{bmatrix} 1 \\ c_1 \\ c_2 \end{bmatrix} = 0 \rightarrow$$

$$\begin{aligned} & 9 + 7c_1 + 11c_2 = 0 \\ \rightarrow \quad & 7 + 11c_1 + 8c_2 = 0 \\ & 11 + 8c_1 + 18c_2 = 0 \end{aligned} \qquad \rightarrow \qquad \begin{aligned} c_1 &= -\frac{1}{13} \\ c_2 &= -\frac{10}{13}. \end{aligned}$$

The resulting optimal real values computed by this process for unknown coefficients in this context are therefore $c_1 = -\frac{1}{13}$ and $c_2 = -\frac{10}{13}$, and so the result is $c = (1, -\frac{1}{13}, -\frac{10}{13})$.

4.3 Concept usage experiments

All experiments with this method tried to examine hypothesis:

- *If a computed real optimal coefficient value is close to an integer value, then this integer value is likely to be the searched optimal integral coefficient value.*

Four experiments with this method were conducted in this thesis, which contain following ideas related to the examination of the mentioned hypothesis:

- Use the computed coefficient closeness to an integer value as a sorting key for a lattice basis.
- Use the computed coefficient closeness to an integer value as an indicator that this integral value is optimal coefficient value choice in short vector enumeration.

4.3.1 Lattice basis ordering key for BKZ reduction

In the first experiment related to this method, it was attempted to

1. generate a lattice basis with basis row vectors $B = [b_0, ..., b_{d-1}]$,
2. sort the vectors of this basis by their lengths in ascending order to create B',
3. fixate coefficient of the sorted basis vector with the lowest length to 1 ($c_{B'}[0] = 1$),
4. compute Gram matrix G' from B',
5. compute optimal real coefficient values for the rest of the basis vectors with
$$c_{B'} = (1, ..., c_{B'}[d-1]),$$
where $c_{B'}$ is a column vector, and therefore to solve
$$\nabla_{c_{B'}} \left(c_{B'}^T G' c_{B'} \right) = 2G' c_{B'} = 0,$$

6. sort the B' basis vectors by remainder of computed optimal real values ($c_{B'}$) modulo 1 ($c_{B'}[i]$ mod 1, $0 \leq i \leq d-1$) in descending order to create B'',

7. and to compute BKZ lattice basis reduction by built-in SageMath method[1] with

   - $blocksize = d$ ($\beta = d$)
   - and $proof = true$ to force the usage of full BKZ reduction,

   on both B'' and B individually and compare the results and BKZ reduction process factors.

It is important to note that B'' has its basis vectors sorted in descending order because the used built-in BKZ algorithm implementation internally uses short vector enumeration which iterates through basis vectors in reversed order.

It is also important to note that s means sample size of lattice bases, d stands for dimension, RHF stands for lattice reduction rate indicated by root Hermite factor, $||\lambda_1||$ is a norm of shortest vector in given reduced lattice basis and t stands for BKZ reduction time.

Also, RHF, $RHF\,sor.$, $||\lambda_1||$ and $||\lambda_1||\,sor.$ factors were measured only in 2 cases, because they were measured in 2 extended test runs separately later on for better test result clarity.

Table 4.1: Comparison of results of BKZ reduction of lattice bases sorted by optimal real coefficients and of unsorted bases.

d	s	RHF	RHF sor.	$\|\|\lambda_1\|\|$	$\|\|\lambda_1\|\|$ sor.	t [s]	t sor. [s]
35	500	1.0124	1.0124	71.3459	71.3470	0.0626	0.0638
40	100	-	-	-	-	0.8306	0.6898
44	50	1.0121	1.0121	77.1715	77.1715	12.7529	14.7730
45	20	-	-	-	-	24.6474	18.4356
46	20	-	-	-	-	51.5460	43.6920

1. https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.BKZ

In the shown results, declining sample sizes are due to increasing BKZ reduction computation times.

From shown testing results, it seems that sorting unreduced lattice basis by computed optimal real coefficients' modulo 1 remainder does not need to improve BKZ reduction computation time or reduction rate.

4.3.2 Differences of coefficient sums from the shortest vector

In the process of the next analysis, 2000 unreduced lattice bases and 2000 LLL reduced lattice bases were generated for each of the two tested lattice basis dimension configurations ($d = 5$ and $d = 10$). For every such lattice $\mathcal{L}$ with basis of row vectors $B = [b_0, ..., b_{d-1}]$

1. the shortest vector λ_1 was computed using an *fplll* library method[2],
2. the lattice basis vector coefficient column vector c_{λ_1} was computed for λ_1 by using λ_1 as a column vector in $\lambda_1 = Bc_{\lambda_1} \rightarrow c_{\lambda_1} = B^{-1}\lambda_1$,
3. the Gram matrix G was computed from the basis B,
4. for each coefficient $c_{\lambda_1}[i]$, $0 \leq i \leq d-1$, the coefficient was individually fixated and the values of the rest of the coefficients were computed to be real optimal using the analyzed method, which led to computing vector $c_{\lambda_1 i}$, where $c_{\lambda_1 i}[i] = c_{\lambda_1}[i]$, by
$$\nabla_{c_{\lambda_1 i}} \left(c_{\lambda_1 i}^T G c_{\lambda_1 i} \right) = 2Gc_{\lambda_1 i} = 0\,,$$
5. and for each such vector $c_{\lambda_1 i}$, the sum of differences of its coefficients with coefficients of the same index of the shortest found vector λ_1 was computed by
$$\sum_{j=0}^{d-1} \left| c_{\lambda_1 i}[j] - c_{\lambda_1}[j] \right|\,.$$

2. `https://github.com/fplll/fpylll/blob/master/src/fpylll/fplll/svpcvp.pyx`

It is imporant to note that LLL reduced lattice bases were reduced by using built-in LLL reduction algorithm method[3] with default configuration.

Finally, a histogram describing distribution of sums was constructed from all such computed sums for each tested lattice basis configuration of a lattice dimension and lattice basis reduction state (LLL reduced/unreduced) combination.

It is important to note that the retrieved histograms for LLL reduced lattice bases configurations are without vectors with length equal to 0, since zero vectors are useless for the intended purpose and are the result of computing optimal real coefficients with no initial coefficient value fixation or with fixation of only coefficient values equal to 0.

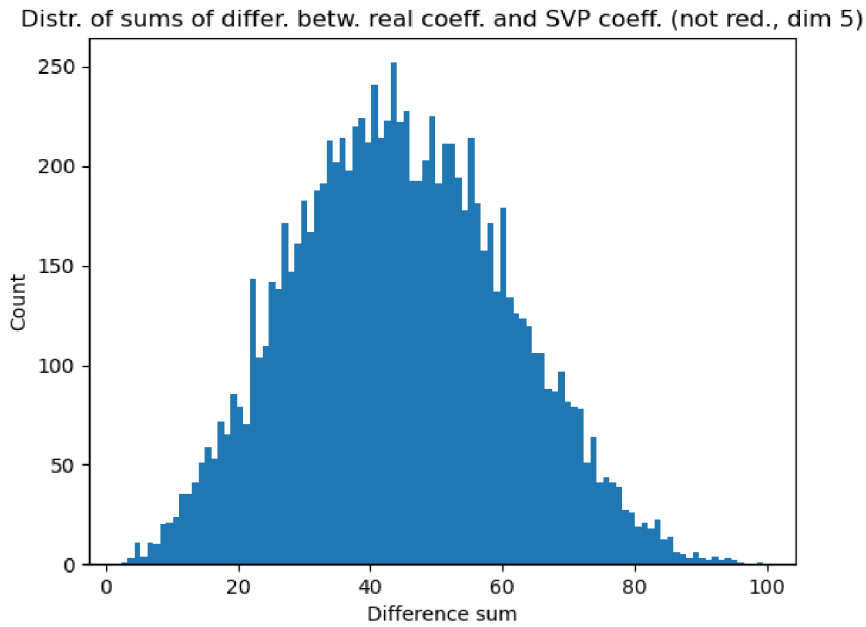

Figure 4.1: Distribution of sums of differences of optimal real coefficients from coefficients of the shortest vector (dimension 5, 2000 unreduced lattice bases).

3. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL`

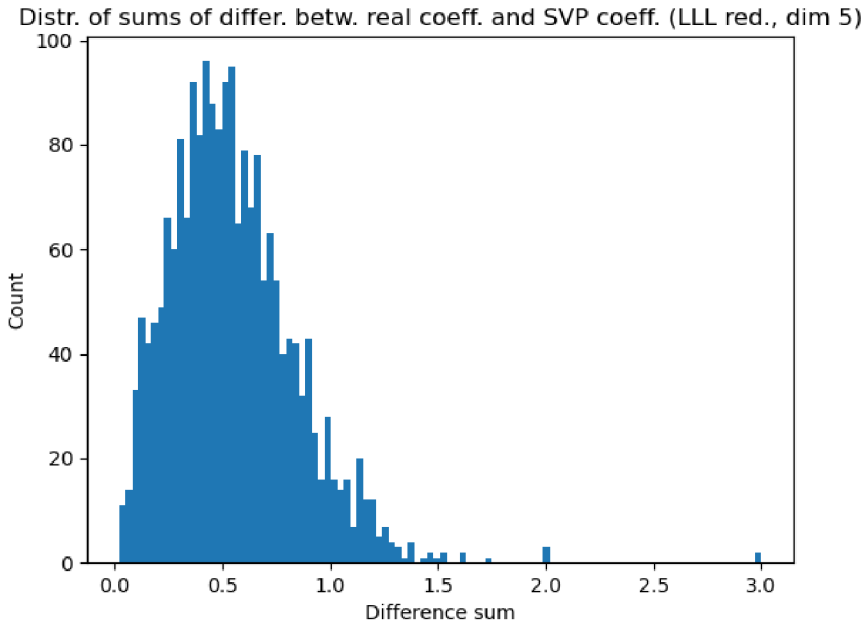

Figure 4.2: Distribution of sums of differences of optimal real coefficients from coefficients of the shortest vector (dimension 5, 2000 LLL reduced lattice bases).

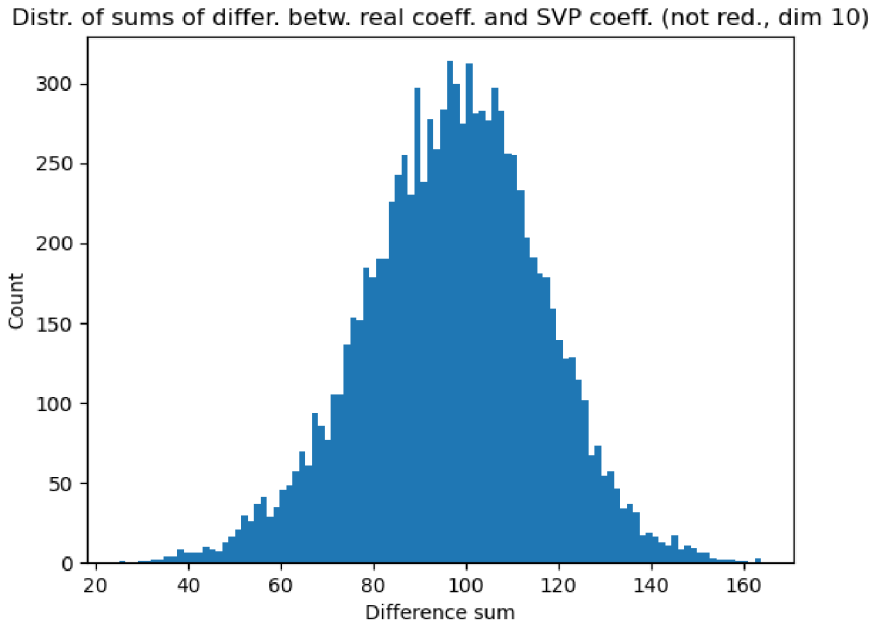

Figure 4.3: Distribution of sums of differences of optimal real coefficients from coefficients of the shortest vector (dimension 10, 2000 unreduced lattice bases).

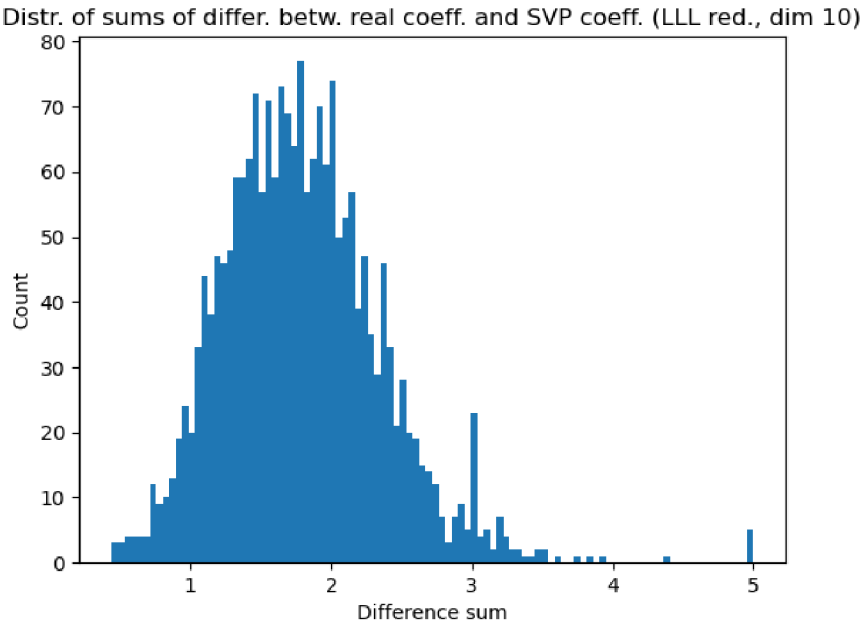

Figure 4.4: Distribution of sums of differences of optimal real coefficients from coefficients of the shortest vector (dimension 10, 2000 LLL reduced lattice bases).

From displayed results, it seems that differences from shortest vector coefficients are much smaller in case of the initial lattice basis being LLL reduced, which indicates that the analyzed method might be much more potent if used on LLL reduced lattice bases rather than on unreduced lattice bases.

4.3.3 Single coefficient differences from the shortest vector

Given the result of the previous analysis, the process of this experiment generated 1000 LLL reduced 5 dimensional lattice bases and 200 LLL reduced 10 dimensional lattice bases. Then, for every such lattice basis $B_{red} = [b_0, ..., b_{d-1}]$

1. the Gram matrix G was computed from the basis B_{red},

2. the implemented BFS short vector enumeration process was executed on B_{red}, with vector length bound R set to length of the longest basis vector of B_{red}, and with the integration of optimal real coefficient value computation such that

   - for each enumeration tree node represented by a tuple $n = (c, -)$, where c is a column vector, where its value c_i, $0 \leq i \leq d-1$ can be either fixated integral value or empty (symbol), optimal real coefficient values for all unassigned coefficient indexes were computed by $\nabla_c \left(c^T G c\right)$,
   - then, assuming that c_{last}, $1 \leq last \leq d-1$ was the last chosen fixated coefficient in the current enumeration context, the computed optimal real value c_{last-1} is stored in the enumeration node n so that $n = (c, c_{last-1})$,
   - however, if all values in c were either empty or equal to 0, an empty value remained in $n[1]$, since otherwise the resulting stored value would always be 0 in this case,
   - also, if the enumeration node n was a leaf node, empty value also remained in $n[1]$, since there is no meaning in computing real optimal coefficients, when all coefficients are already chosen and fixated

3. and the enumeration tree returned by the mentioned enumeration process was then processed in a way such that for every enumeration node represented by n which

   - had nonempty stored computed optimal real value c_{last-1}
   - and had only one child node n_{child} in the enumeration tree (for easy decision of what is the optimal integral coefficient value for the next coefficient in the given context),

   the absolute value of difference between the optimal integral coefficient value of the child node and the computed stored optimal real coefficient value was computed by

$$diff = \left| n[1] - n_{child}[0][last-1] \right| .$$

It is imporant to note that LLL reduced lattice bases were reduced by using built-in LLL reduction algorithm method[4] with default configuration.

Finally, a histogram describing distribution of the computed differences was constructed from all such computed differences for each tested lattice basis configuration.

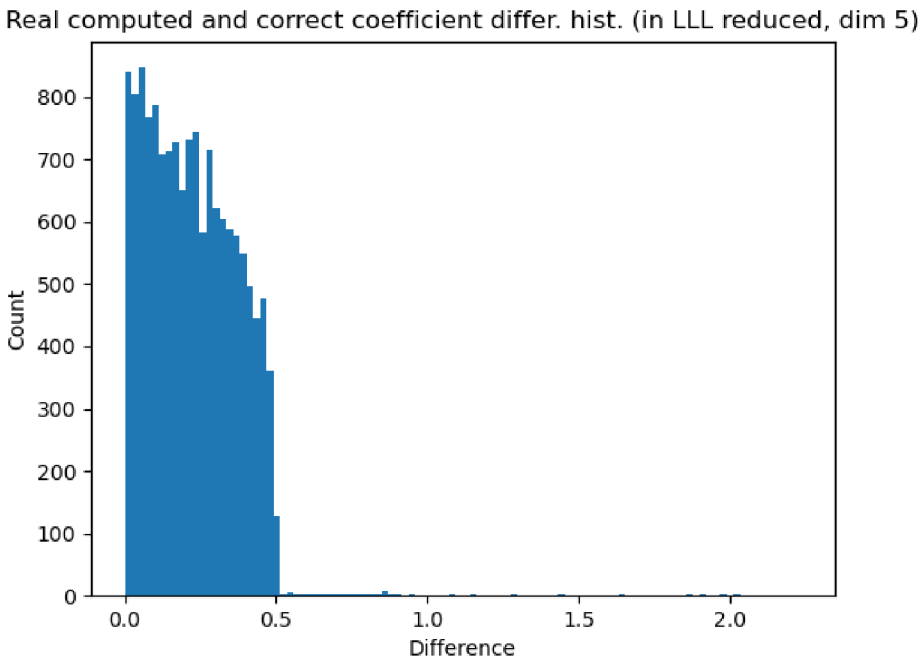

Figure 4.5: Distribution of differences of individual optimal real coefficient values of the next enumerated coefficient from the optimal integral coefficients (dimension 5, 1000 LLL reduced lattice bases).

4. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL`

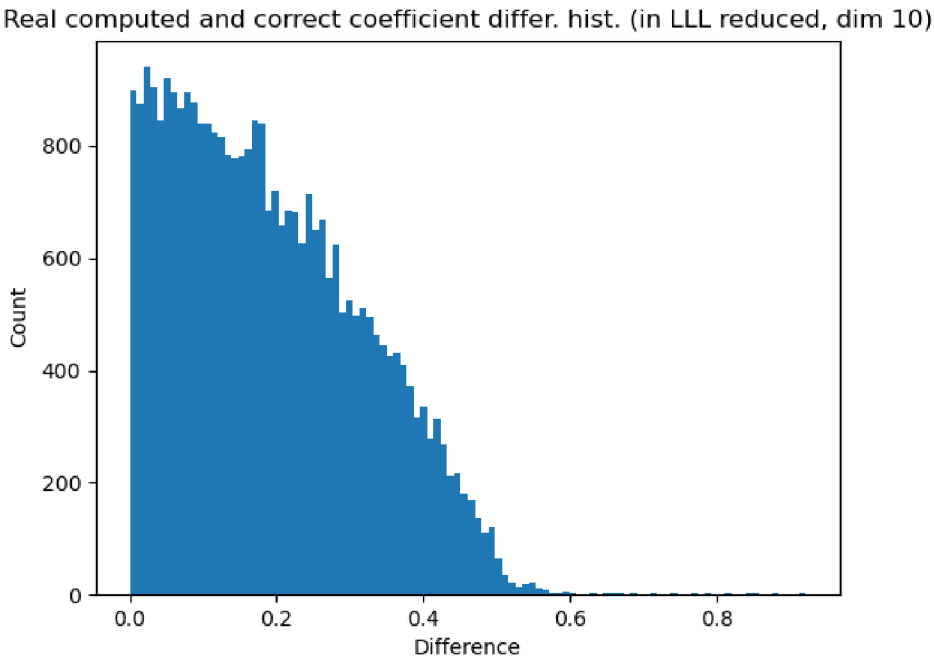

Figure 4.6: Distribution of differences of individual optimal real coefficient values of the next enumerated coefficient from the optimal integral coefficients (dimension 10, 200 LLL reduced lattice bases).

From shown results, it seems that for LLL reduced lattice bases, the optimal real coefficients computed during the short vector enumeration might be, given their closeness to optimal integral coefficients, a strong indicator of which coefficient to pick to find the shortest possible vector in the given enumeration context.

4.3.4 Short vector enumeration heuristic

In the final realized analysis of this method, based on results of previous 2 experiments, the same edited implemented BFS short vector enumeration variant was used as in the last experiment on generated LLL reduced lattice bases.

On the start of each test, both global counts *in threshold* and *in threshold predicting* were set to 0. Then, for each enumeration tree resulting from the mentioned enumeration process

1. for every enumeration node of the tree represented by $n = (c, c_{last-1})$, which
   - had nonempty stored computed optimal real value c_{last-1},
   - had only one child node n_{child} in the enumeration tree (for easy decision of what is the optimal integral coefficient value for the next coefficient in the given context)
   - and had $n[1] \ mod \ 1 < threshold$, where *threshold* is a configured parameter of the test,

   (a) the global count *in threshold*, representing the count of all enumeration tree nodes that conformed to the mentioned conditions, was incremented

   (b) and, if also $\lfloor n[1] \rfloor = \lfloor n_{child}[0][last-1] \rfloor$, the global count *in threshold predicting* was also incremented.

It is imporant to note that LLL reduced lattice bases were reduced by using built-in LLL reduction algorithm method[5] with default configuration.

Results of test runs of all LLL reduced lattice basis generation configurations described by s (sample size, number of generated LLL reduced lattice bases), d (dimension of the generated lattice bases) and *threshold* parameters, were following.

5. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL`

Table 4.2: Computed optimal real values during enumeration which are in mod 1 threshold and which also predict next optimal integral coefficient value.

d	s	threshold	*in threshold*	*in threshold predicting*
5	200	0.2	721	714
5	200	0.15	556	556
10	200	0.15	6796	6795
15	50	0.15	9692	9691
17	25	0.15	10928	10928

From the shown results, it seems that it can be assumed that if the $threshold = 0.15$ is used, it is highly likely that if the *threshold* is met by the computed optimal real coefficient during the complete short vector enumeration on LLL reduced lattice basis, then the mentioned real coefficient indicates the optimal integral coefficient.

5 Cholesky matrix end diagonal values

Next analyzed method's concept took generated lattice basis $B = [b_0, ..., b_{d-1}]$ as input and created a lattice basis B' which was sorted in a unique way defined below.

The sorting process

1. created empty basis B',
2. iterated through every lattice basis vector b_i, $0 \leq i < |B|$, and for each of them tried to
   (a) create B_i from B by removing b_i from B and append it to the end of B,
   (b) then, Gram matrix G_i and Cholesky decomposed lower triangular matrix L_i were created for this edited basis B_i,
   (c) and last, the end diagonal value of the resulting lower triangular matrix $L_i[d-1][d-1]$ was stored,
3. next, the biggest of the stored end diagonal values $max_j(L_j[d-1][d-1])$ was found and b_j basis vector was removed from B and put into the constructed basis B' on the first index,
4. then, this process was repeated from the step 2., until the basis B was empty and the new sorted basis B' was finished.

This sorting process can also be described by

```
procedure SortByCholesky(d, B)
    B' ← []
    for B_d in d..0 do
        biggEndVal ← -1
        biggEndValVecI ← -1
        for i in 0..|B| do
            B_i ← B[..i] + B[(i + 1)..] + [B[i]]
            G_i ← B_i B_i^T
            L_i ← G_i.cholesky()
            if L_i[B_d − 1][B_d − 1] > biggEndVal then
                biggEndVal ← L_i[B_d − 1][B_d − 1]
                biggEndValVecI ← i
            end if
        end for
        B' ← [B.pop(biggEndValVecI)] + B'
    end for
    return B'
end procedure.
```

For example, lets assume there is a lattice with basis with row basis vectors $B = [b_0, b_1, b_2]$, where

$$B = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 4 \end{bmatrix}.$$

To create the sorted basis B', the process is to

1. create the empty basis $B' = []$,
2. take b_0 and create basis $B_0 = [b_1, b_2, b_0]$,
3. create G_0 and L_0 from B_0, where

$$L_0 = \begin{bmatrix} \sqrt{11} & 0 & 0 \\ \frac{8}{\sqrt{11}} & \sqrt{\frac{134}{11}} & 0 \\ \frac{7}{\sqrt{11}} & \frac{65}{\sqrt{1474}} & \sqrt{\frac{225}{134}} \end{bmatrix},$$

4. create bases $B_1 = [b_0, b_2, b_1]$ and $B_2 = [b_0, b_1, b_2]$,

5. create Gram matrices G_1 from B_1 and G_2 from B_2 in the same way as G_0 from B_0,

6. create Cholesky decomposed lower triangular matrices L_1 from G_1 and L_2 from G_2 in the same way as L_0 from G_0, and so

$$L_1 = \begin{bmatrix} 3 & 0 & 0 \\ \frac{11}{3} & \frac{\sqrt{41}}{3} & 0 \\ \frac{7}{3} & -\frac{5}{3\sqrt{41}} & \frac{15}{\sqrt{41}} \end{bmatrix} \; and \; L_2 = \begin{bmatrix} 3 & 0 & 0 \\ \frac{7}{3} & \frac{5\sqrt{2}}{3} & 0 \\ \frac{11}{3} & -\frac{1}{3\sqrt{2}} & \frac{3}{\sqrt{2}} \end{bmatrix},$$

7. find the biggest out of $L_0[2][2]$, $L_1[2][2]$ and $L_2[2][2]$, which is $L_1[2][2] = \frac{15}{\sqrt{41}}$,

8. remove according vector b_1 from B and put it on the first index of the constructed B', and so $B' = [b_1]$, $B = [b_0, b_2]$

9. and repeat the process until B is empty and B' is fully constructed.

Result of this example is therefore $B' = [b_0, b_2, b_1]$ and so

$$B' = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 1 & 4 \\ 1 & 3 & 1 \end{bmatrix}.$$

5.1 Concept usage experiments

All experiments with this method attempted to examine the hypothesis:

- *If the lattice basis is sorted by biggest end diagonal values of Cholesky decomposed lower triangular matrices from back, then the enumeration tree size of a short vector enumeration should decrease.*

Two experiments with this method were conducted in this thesis, which contain following ideas of the mentioned hypothesis usage:

- Find out the impact of the described lattice basis sorting key on BKZ reduction algorithm process result.
- Find out the impact of the described lattice basis sorting key on short vector enumeration tree sizes in configurations with
  - pruned enumeration and LLL sorted lattice bases,
  - unpruned enumeration and LLL sorted lattice bases,
  - pruned enumeration and unsorted lattice bases,
  - and unpruned enumeration and unsorted lattice bases.

5.1.1 Lattice basis ordering key for BKZ reduction

In the first experiment related to this method, it was attempted to

1. generate a lattice basis with basis row vectors $B = [b_0, ..., b_{d-1}]$,
2. duplicate B to create B_{dupl},
3. calculate B_{sort} by $B_{sort} = SortByCholesky(d, B_{dupl})$

4. and to compute BKZ lattice basis reduction by built-in SageMath method[1] with
   - $blocksize = d$ ($\beta = d$)
   - and $proof = true$ to force the usage of full BKZ reduction,

   on both B_{sort} and B individually and compare the results and BKZ reduction process factors.

It is important to note that s means sample size of lattice bases, d stands for dimension, RHF stands for lattice reduction rate indicated by root Hermite factor, $||\lambda_1||$ is a norm of shortest vector in given reduced lattice basis and t stands for BKZ reduction time.

Also, RHF, $RHF\,sor.$, $||\lambda_1||$ and $||\lambda_1||\,sor.$ factors were measured only in 2 cases, because they were measured in 2 extended test runs separately later on for better test result clarity.

Table 5.1: Comparison of results of BKZ reduction of lattice bases sorted by recursive biggest Cholesky end diagonal values and of unsorted bases.

d	s	RHF	RHF sor.	λ_1	λ_1 sor.	t [s]	t sor. [s]
35	500	1.0124	1.0124	71.5037	71.5058	0.0611	0.0627
40	100	-	-	-	-	0.6949	0.7047
44	20	1.0121	1.0121	76.9519	76.9519	11.0600	9.5256
45	20	-	-	-	-	20.3666	24.2689
46	20	-	-	-	-	44.2453	50.4099

In the shown results, declining sample sizes are due to increasing BKZ reduction computation times. It is also important to note that this sorting approach was computationally expensive (in tested dimensions 4.5 - 13 seconds per lattice basis).

From shown testing results, it seems that sorting unreduced lattice basis by Cholesky decomposed Gram matrix recursive biggest end diagonal values belonging to lattice basis vectors does not need to improve BKZ reduction computation time or reduction rate.

1. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.BKZ`

5.1.2 Lattice basis ordering key for enumeration tree size

In the second experiment related to this method, it was attempted to

1. generate a lattice basis with basis row vectors $B = [b_0, ..., b_{d-1}]$,
2. reduce, if configured in the test parameters, the basis by using built-in LLL reduction algorithm method[2] with default configuration to get B_{red},
3. run on the basis B or B_{red}, with vector length bound R set to length of the longest basis vector, either
   - implemented DFS variant of the short vector enumeration process with pruning such that the short vector length upper bound is updated every time a shorter short vector is found,
   - or implemented BFS variant of the short vector enumeration process without pruning, if the pruning was not configured in the test parameters,

   with enumeration tree node counting enabled,
4. duplicate B or B_{red} to create B_{dupl},
5. calculate B_{sort} by $B_{sort} = SortByCholesky(d, B_{dupl})$,
6. run the same enumeration process, which was ran on B or B_{red}, and also on B_{sort}, and also get count of enumeration tree nodes,
7. and compute ratio of enumeration tree node counts before and after the sorting process such that $ratio = \frac{countBeforeSort}{countAfterSort}$.

To be more exact, mentioned ratios were always computed from average enumeration tree node count computed from results of 5 enumeration processes. It is also important to note that the execution time of this experiment was approximately 6 hours. Results of this experiment were following.

2. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL`

Table 5.2: Average ratios of enumeration tree sizes of unsorted lattice bases and lattice bases sorted by Cholesky decomposed Gram matrix recursive biggest end diagonal values belonging to lattice basis vectors (LLL reduced lattice bases).

dimension	pruned	not pruned	sample size
3	100.00 %	102.56 %	5
4	101.09 %	100.00 %	5
5	100.93 %	103.78 %	5
6	100.00 %	100.19 %	5
7	101.85 %	98.70 %	5
8	101.99 %	100.00 %	5
9	92.80 %	100.66 %	5
10	92.59 %	101.12 %	5
11	83.81 %	101.86 %	5
12	117.72 %	102.47 %	5
13	109.73 %	98.64 %	5
14	86.19 %	106.27 %	5
15	72.01 %	104.91 %	5

Table 5.3: Average ratios of enumeration tree sizes of unsorted lattice bases and lattice bases sorted by Cholesky decomposed Gram matrix recursive biggest end diagonal values belonging to lattice basis vectors (unreduced lattice bases).

dimension	pruned	not pruned	sample size
3	781.04 %	293.15 %	5
4	996.85 %	561.77 %	5
5	4717.36 %	495.93 %	5

It is important to note that tests regarding unreduced lattice bases were run only on very small dimensions since enumeration tree sizes of unreduced lattice bases are drastically wider, than of LLL reduced lattice bases.

This was evident even though the quotient indicating how much the generated lattice base is unreduced was set to be very small (parameter of the lattice basis generating process was set to be $q = 23$ in this case, to be exact).

This fact was later confirmed by further analysis, which included

- comparing averages of enumeration coefficient interval bounds $coeffB$ computed during enumeration in individual enumeration depths between
  - LLL reduced lattice bases being the input,
  - and unreduced lattice bases with $q = 23$ being set in the generating process input.

However, shown test results seem to indicate that using mentioned sorting strategy on lattice bases can have positive impact on resulting full short vector enumeration tree sizes in cases, when relevant lattice bases are unreduced. They also seem to indicate that this sorting strategy can have even more significant positive impact in cases, when relevant lattice bases are unreduced and the mentioned pruning method is used during the enumeration process.

6 Short vectors by Gram matrix quadrants

Another proposed concept was to

1. take as input lattice basis with row basis vectors $B = [b_0, ..., b_{d-1}]$, where $d \ mod \ 2 = 0$,
2. compute Gram matrix G from the basis B
3. and analyze potential relation between
   - **upper right quadrant** of the Gram matrix G, defined as $G_{ur} = G[..\frac{d}{2}][\frac{d}{2}..]$,
   - **upper left quadrant** of the Gram matrix G defined as $G_{ul} = G[..\frac{d}{2}][..\frac{d}{2}]$
   - and **lower right quadrant** of the Gram matrix G defined as $G_{lr} = G[\frac{d}{2}..][\frac{d}{2}..]$,
4. or other possible properties of the **upper right quadrant** G_{ur} of the Gram matrix G.

This was done, assuming that $c = (c_0, ..., c_{d-1})$ is a lattice basis B vector coefficient combination, by working with computed

- intersection values, where ***intersection value*** $iv_{i,j}$ is defined, given i is the row index and j is the column index, as multiplication of a row index coefficient, column index coefficient and value of given Gram matrix on given position, or can be described by
$$iv_{i,j} = G[i][j] \cdot c_i \ \cdot c_j \, ,$$

- and sums of intersection values for mentioned Gram matrix quadrants, which can be described by

$$sum_{G_{ul}} = \sum_{i=0}^{\frac{d}{2}-1} \sum_{j=0}^{\frac{d}{2}-1} iv_{i,j},\ sum_{G_{ur}} = \sum_{i=0}^{\frac{d}{2}-1} \sum_{j=\frac{d}{2}}^{d-1} iv_{i,j}$$

and

$$sum_{G_{lr}} = \sum_{i=\frac{d}{2}}^{d-1} \sum_{j=\frac{d}{2}}^{d-1} iv_{i,j}\,.$$

For example, lets assume that there is a lattice basis with row basis vectors $B = [b_0,\ b_1,\ b_2,\ b_3]$, lattice basis vector coefficient combination $c = (0,\ 1,\ 0,\ 1)$ and a Gram matrix G computed from B, where

$$B = \begin{bmatrix} 2 & -1 & 0 & 1 \\ 1 & 3 & -1 & 0 \\ 0 & 1 & 2 & -2 \\ -1 & 0 & 2 & 3 \end{bmatrix} \textit{ and so } G = \begin{bmatrix} 6 & -1 & -3 & 1 \\ -1 & 11 & 1 & -3 \\ -3 & 1 & 9 & -4 \\ 1 & -3 & -4 & 14 \end{bmatrix}.$$

Then it is true that

- the upper right quadrant G_{ur} of G is $G[..2][2..]$, the lower right quadrant G_{lr} of G is $G[2..][2..]$ and the upper left quadrant G_{ul} of G is $G[..2][..2]$, therefore

$$G_{ur} = \begin{bmatrix} -3 & 1 \\ 1 & -3 \end{bmatrix},\ G_{lr} = \begin{bmatrix} 9 & -4 \\ -4 & 14 \end{bmatrix} \textit{ and } G_{ul} = \begin{bmatrix} 6 & -1 \\ -1 & 11 \end{bmatrix},$$

- the Gram matrix intersection sum for upper left quadrant is

$$sum_{G_{ul}} = \sum_{i=0}^{2-1} \sum_{j=0}^{2-1} iv_{i,j} = iv_{0,0} + iv_{0,1} + iv_{1,0} + iv_{1,1} =$$

$$6 \cdot 0 \cdot 0 + (-1) \cdot 0 \cdot 1 + (-1) \cdot 1 \cdot 0 + 11 \cdot 1 \cdot 1 = 11$$

- and analogically, using relevant column index and row index intervals, $sum_{G_{ur}} = -3$ and $sum_{G_{lr}} = 14$.

6.1 Concept usage experiments

The analyses relevant to this concept were done by 3 experiments. Assuming that c, c_{first} and c_{second} are lattice basis B vector coefficient combinations, the examined hypotheses relevant to the described concept were:

1. *If c is found such that the absolute value of relevant negative sum of G_{ur} intersection values is bigger than the relevant positive sum of G_{ul} intersection values or than the positive sum of G_{lr} intersection values, then c represents a vector shorter or equally short to the longest basis vector of B.*

2. *If c_{first} is found such that it leads to bigger negative sum of G_{ur} intersection values than c_{second}, then c_{first} is more likely to represent a vector shorter than the longest basis vector of B.*

6.1.1 Short vectors coefficients in LLL reduced bases

Since the analyzed concept is tightly related to using lattice basis vector coefficient combinations, the first experiment related to this concept was focused on searching for optimal interval of coefficient values to use in following experiments related to this concept.

To narrow the coefficient interval to use,

- only lattice bases reduced by using built-in LLL reduction algorithm method[1] with default configuration would be used
- and ratio of enumerated short vectors with their lattice basis vector coefficients being only in $\{-1, 0, 1\}$ would be analyzed.

1. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL`

To analyze the mentioned ratio, the process of this experiment was to iterate through dimensions d, where $3 \leq d \leq 30$, and for each such dimension

1. generate 20 lattice bases,
2. reduce the bases by the mentioned LLL reduction algorithm method,
3. and for each such reduced basis
   (a) run the built-in SageMath short vector enumeration variant with vector length bound R set to length of the longest basis vector of the reduced basis,
   (b) store total count of enumerated short vector coefficient combinations,
   (c) store count of those enumerated short vector coefficient combinations, where all coefficient values are in $\{-1, 0, 1\}$
   (d) and store enumeration run time.

Finally, a boxplot was constructed from stored data, showing ratios of enumerated short vector coefficient combinations, where all coefficient values are in $\{-1, 0, 1\}$, for each dimension. A plot of average running times for each dimension was also constructed.

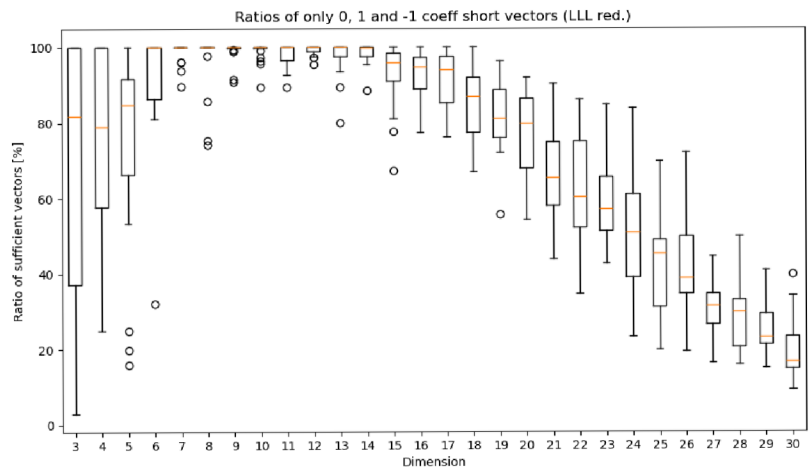

Figure 6.1: Ratios across dimensions of enumerated short vectors from LLL reduced basis which have only {-1, 0, 1} coefficient values.

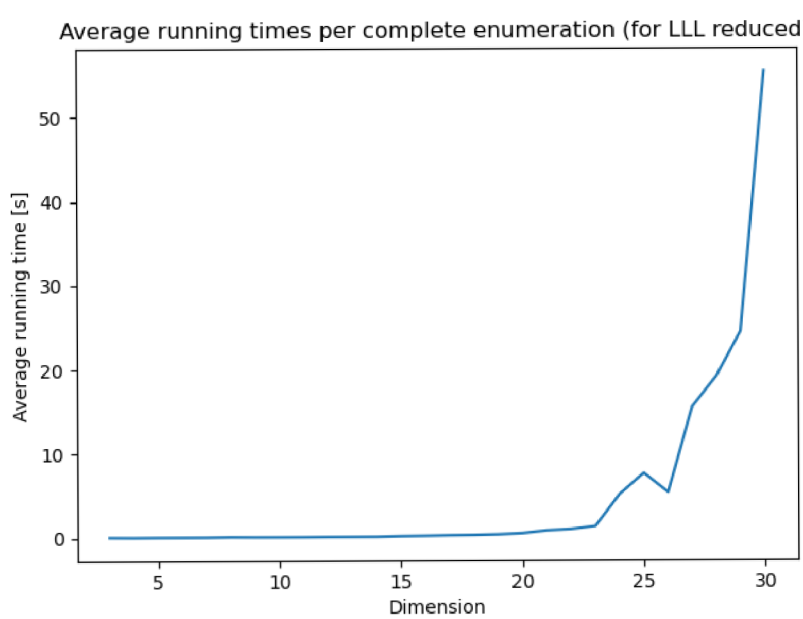

Figure 6.2: Average running times of short vector enumeration across dimensions.

It is important to note that it was not possible to explain the cause of the seen dip after 25th dimension in the graph of average enumeration running times because the relevant method used for short vector enumeration was used as a black box.

From shown results, it seems that across all tested dimensions, large portion of enumerated short vector coefficient combinations contained only mentioned coefficient values, even though the ratio seems to have become smaller with growing dimension. However, the resulting ratio was still high enough to imply that the analyzed lattice basis vector coefficient value interval could be used in the remaining experiments of this chapter.

6.1.2 Sequential coefficient combination search

In the next experiment, the first of the two mentioned hypotheses was analyzed. The process of this experiment was to

1. generate a lattice basis $B = [b_0, ..., b_{d-1}]$,
2. reduce this lattice basis by using built-in LLL reduction algorithm method[2] with default configuration to get B_{red},
3. compute Gram matrix G from B_{red},
4. iterate through coefficient combination space, assuming that
   - i was the integral coefficient combination index iterated from $i = 0$ to $i = 3^d - 1$
   - and $cvo = [0, 1, -1, -1, 1, 0]$ was coefficient value oscillation array,

   in a way that for each i, coefficient combination c_i was retrieved by

2. https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL

```
procedure CoefficientCombination(cvo, d, i)
    c_i ← []
    for j in 0..d do
        oscillDuplicRate ← 3^j
        oscillIntervSize ← 2(3^(j+1))
        coeff ← cvo [ (i mod oscillIntervSize) / oscillDuplicRate ]
        c_i.append(coeff)
    end for
    return c_i
end procedure,
```

5. and for each such coefficient combination c_i it was

   (a) checked whether resulting G upper right quadrant intersection value sum $sum_{G_{ur}}$ had negative sum low enough to either

   $$sum_{G_{ur}} < 0 \ \wedge \ sum_{G_{ul}} > 0 \ \wedge \ sum_{G_{ur}} + sum_{G_{ul}} < 0$$

   or

   $$sum_{G_{ur}} < 0 \ \wedge \ sum_{G_{lr}} > 0 \ \wedge \ sum_{G_{ur}} + sum_{G_{lr}} < 0$$

   (b) and if yes, the coefficient combination c_i was returned and the process stopped.

It is important to note that the described way of iterating through coefficient combination indexes led to

- always only one coefficient value of a coefficient combination changing in each iteration
- and thus, by also keeping track of column sums for G_{ur}, G_{ul} and G_{lr} throughout the process, led to very little computation needed to get $sum_{G_{ul}}$, $sum_{G_{ur}}$ and $sum_{G_{lr}}$ for given coefficient combination index.

To explain further, in each iteration, by knowing mentioned quadrant column sums of previous iteration and with the fact that only one coefficient value changes from the previous iteration, it was achieved that only

1. one row and one column for G_{ul} or G_{lr}
2. and one row or one column for G_{ur}

had to be recomputed in the column sums. This led to minimizing the required operation count during the iteration process.

Relevant results follow. It is important to note that s stands for sample size, or count of generated lattice bases in given test configuration. Next, *found rate* stands for ratio of lattice bases where coefficient combination that fulfilled the described Gram matrix quadrant relation was found. Also, *in bounds* stands for ratio where vector with such found coefficient combination was also shorter or equally short to the longest vector of the relevant LLL reduced lattice basis.

Next, t stands for average computation time spent on single lattice basis by the described process. Finally, *t enum* stands for average computation time spent on single LLL reduced lattice basis by built-in SageMath short vector enumeration variant with vector bound parameter set to the length of the longest basis vector of the LLL reduced lattice basis.

Table 6.1: Found rate of coefficient combination fulfilling required Gram matrix quadrant relation and rate of those that also represent vector with length in required vector length bound (sequential full variant).

d	s	found rate	in bounds	t [s]	t enum [s]
10	200	100 %	0 %	0.1770	0.0142
14	200	100 %	0 %	0.3202	0.0214
18	200	100 %	0 %	2.5698	0.0528
20	200	100 %	0 %	8.1303	0.0973

Before any conclusion from shown results was reached, a more heuristic variant of this process was also analyzed in the same way.

Heuristic variant

A heuristic alternative of the process of the described experiment was also implemented and analyzed. The only difference from the previous variant was that not the whole described coefficient combination space was searched through.

The heuristic variant searched only through all coefficient combinations of the middle half of the coefficient indexes. To be more exact, assuming that d is the dimension of the lattice basis, in the heuristic version, integral coefficient combination index i was iterated from $i = 0$ to

- $i = 3^{\lfloor \frac{d}{2} \rfloor} - 1$ if $d \ mod \ 4 = 0$,
- else to $i = 3^{\lfloor \frac{d}{2} \rfloor + 1} - 1$.

Also, the resulting coefficient combination c_i in a given iteration was, if $d \ mod \ 4 = 0$, retrieved by

$$c_i = \overbrace{[0, ..., 0]}^{\lfloor \frac{d}{4} \rfloor \text{ times}} + CoefficientCombination(cvo, \lfloor \frac{d}{2} \rfloor, i) + \overbrace{[0, ..., 0]}^{\lfloor \frac{d}{4} \rfloor \text{ times}},$$

else it was retrieved by

$$c_i = \overbrace{[0, ..., 0]}^{\lfloor \frac{d}{4} \rfloor \text{ times}} + CoefficientCombination(cvo, \lfloor \frac{d}{2} \rfloor + 1, i) + \overbrace{[0, ..., 0]}^{\lfloor \frac{d}{4} \rfloor \text{ times}}.$$

It is important to note that the dimension division result increments affected by the $d \ mod \ 4 \neq 0$ condition were performed to ensure that the resulting coefficient combination c_i had correct value count ($|c_i| = d$).

As was already mentioned, on the described heuristic alternative, the same analysis as on the previous variant was conducted. Results were following.

Table 6.2: Found rate of coefficient combination fulfilling required Gram matrix quadrant relation and rate of those that also represent vector with length in required vector length bound (sequential heuristic variant).

d	s	found rate	in bounds	t [s]	t enum [s]
10	200	34 %	0 %	0.0325	0.0247
14	200	88 %	0 %	0.1626	0.0202
18	200	99 %	0 %	0.5025	0.0554
20	200	99.5 %	0 %	0.5083	0.1013
24	200	100 %	0 %	0.6965	0.4645
28	200	100 %	0 %	1.1330	4.5960

From results of the full variant analysis and the heuristic variant analysis, it can be assumed that the full variant is sufficient for finding vector whose coefficient combination results in Gram matrix upper right quadrant negative sum being low enough to surpass upper left quadrant positive sum or lower right quadrant positive sum.

It also seems that used heuristic variant can be faster than mentioned short vector enumeration and is still very likely to find vector whose coefficient combination results in Gram matrix upper right quadrant negative sum being low enough to surpass upper left quadrant positive sum or lower right quadrant positive sum.

However, it also seems that the mentioned Gram matrix quadrant relation has a very low chance of indicating that the found vector is shorter or equally short to the longest vector of relevant LLL reduced lattice basis.

Further analysis of the quadrant relation

To further confirm that the first proposed hypothesis related to this concept was false, further analysis was done. The analysis focused on the ratio of how many enumerated short vectors in given vector length bounds also fulfill the analyzed Gram matrix quadrant relation.

The mentioned short vector enumeration was performed by using the SageMath short vector enumeration variant with vector bound parameter set to the length of the longest basis vector of the LLL reduced lattice basis.

The LLL reduced lattice basis was reduced by the built-in LLL reduction algorithm method[3] with default configuration.

The analysis results were following:

- **100 LLL reduced 10 dimensional bases**: ratio of 0.00340 (44 / 12950 of found short vectors also fulfilled the Gram matrix quadrant relation)
- **100 LLL reduced 10 dimensional bases**: ratio of 0.01416 (374 / 26414 of found short vectors also fulfilled the Gram matrix quadrant relation)

The shown results seem to further confirm very low, if any, relation between the mentioned Gram matrix quadrant relation and resulting short vector length being in the mentioned bound.

6.1.3 Random operations coefficient combination search

In the last analysis related to this concept, the second of the two mentioned hypotheses was analyzed. The process of this experiment was to

1. generate a lattice basis $B = [b_0, ..., b_{d-1}]$,
2. reduce this lattice by using the built-in LLL reduction algorithm method[4] with default configuration to get B_{red},
3. compute a Gram matrix G from B_{red},
4. initialize a coefficient combination c with random coefficient values where $c_i \in \{-1, 0, 1\}$, $0 \leq i \leq d-1$,

3. https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL
4. https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL

5. repeat $N = 10000$ times (or other configured count) a **change**, where
   (a) random number cci, $cci \in [0, d-1]$, $cci \in \mathbb{Z}$ was generated,
   (b) cci distinct coefficient indexes of c were chosen and according coefficient values of c were changed each to a random value from $\{-1, 0, -1\}$ to create $c_{changed}$
   (c) and $sum_{G_{ur}}$ was computed for $c_{changed}$ and if it was smaller than $sum_{G_{ur}}$ for c, then c was set to $c_{changed}$
6. and final c along with count of successful changes, where c was set to $c_{changed}$, were returned.

Results were following. It is important to note that s stands for sample size, or count of generated lattice bases in given test configuration. Next, N stands for configured number of mentioned **changes** to be performed.

Also, $SVfound\ ratio$ stands for ratio of runs in the sample, where in the end a coefficient combination c resulting in vector shorter than the longest vector of the LLL reduced lattice basis was returned. Finally, $change\ success\ ratio$ stands for ratio of random **changes** that resulted in upper right Gram matrix quadrant negative sum $sum_{G_{ur}}$ being lower than before the **change**.

Table 6.3: Rate of short vectors found by persisting random vector coefficient changes based on Gram matrix upper right quadrant negative sum.

d	s	N	SV found ratio	change success ratio
5	100	10 000	0.0	0.0004
10	100	10 000	0.0	0.0009
15	100	10 000	0.0	0.0014
20	100	10 000	0.0	0.0019
25	100	10 000	0.0	0.0022

From the shown results, it seems that this approach has a negligible chance of finding a coefficient combination representing a vector short enough to be shorter than the longest vector of the relevant LLL reduced lattice basis.

It also seems that the random **changes** proposed by this approach have a very low chance of resulting in lower $sum_{G_{ur}}$, and therefore of changing the relevant coefficient combination.

7 Precomputed transformation matrices

The last analyzed concept in the scope of this thesis was focused on using precomputed transformation matrices, representing reduction of lattice basis of configurable rate of similarity, for reduction of random generated lattice bases. To understand, how this concept works, it is first needed to understand what a Pearson correlation matrix is.

7.1 Pearson correlation matrix

Pearson correlation matrix is a square, symmetrical matrix used to summarize the linear relationships between multiple variables in a dataset. It displays the Pearson product-moment correlation coefficient (r) for every possible pair of variables. [25, 26]

Diagonal elements of this matrix always contain 1, since every variable has a perfect linear correlation with itself. The correlation coefficient r measures the strength and direction of a linear relationship between two continuous variables. The values of this coefficient range from -1 to 1 (where 1 means perfect positive linear relationship, -1 perfect negative linear relationship and 0 no linear relationship). [26, 27, 28]

Each element of a Gram matrix represents the dot product between two vectors. To normalize it into a Pearson correlation matrix, each entry is scaled by the product of the standard deviations of the corresponding variables. [29]

Assuming R is the resulting correlation matrix, or Pearson correlation matrix, $R[i][j]$ is an element of the resulting Pearson correlation matrix on row i and column j, and $G[i][j]$ is an element of the input Gram matrix, the normalization operation is

$$R[i][j] = \frac{G[i][j]}{\sqrt{G[i][i]G[j][j]}}\,.$$

7.1.1 Pearson correlation matrix computation example

For example, lets assume that there is a Gram matrix G such that

$$G = \begin{bmatrix} 9 & 7 & 11 \\ 7 & 11 & 8 \\ 11 & 8 & 18 \end{bmatrix}.$$

To compute the Pearson correlation matrix from this Gram matrix, assuming that R is the resulting Pearson correlation matrix, d is the dimension of the Gram matrix G and $R[i][j]$ is the value of the resulting correlation matrix on the ith row and jth column, where $0 \leq i \leq d-1$, $0 \leq j \leq d-1$, the process can be following:

1. First, the $R[0][0]$ can be computed by computing $R[i][j] = \frac{G[i][j]}{\sqrt{G[i][i]G[j][j]}}$, which in this case is equivalent to $R[0][0] = \frac{G[0][0]}{\sqrt{G[0][0]G[0][0]}} = \frac{9}{\sqrt{9 \cdot 9}} = 1$.

2. Then, the next computed value can be $R[0][1]$, which can be computed by the same formula, which in this case is equivalent to $R[0][1] = \frac{G[0][1]}{\sqrt{G[0][0]G[1][1]}} = \frac{7}{\sqrt{9 \cdot 11}} = \frac{7}{3\sqrt{11}}$.

3. Next, the same computation steps can be repeated for the values $R[0][2]$, $R[1][0]$, $R[1][1]$, $R[1][2]$, $R[2][0]$, $R[2][1]$ and $R[2][2]$.

Now, the resulting Pearson correlation matrix is fully computed, and

$$R = \begin{bmatrix} 1 & \frac{7}{3\sqrt{11}} & \frac{11}{9\sqrt{2}} \\ \frac{7}{3\sqrt{11}} & 1 & \frac{8}{3\sqrt{2}\sqrt{11}} \\ \frac{11}{9\sqrt{2}} & \frac{8}{3\sqrt{2}\sqrt{11}} & 1 \end{bmatrix}.$$

7.2 Concept description

This concept first created a database of transformation matrices mapped to normalized Gram matrices of value intervals. Given input parameters

- *avbs* representing absolute start bound of normalized Gram matrix value intervals,
- *avbe* representing absolute end bound of normalized Gram matrix value intervals,
- *is* representing normalized Gram matrix value interval size
- and d representing relevant lattice basis dimension,

the process was to

1. create an empty database $DB_{G_{iv},U}$ prepared for mapping Gram matrices with interval values to transformation matrices,
2. create a collection of Gram matrices of given dimension d with all possible combinations of value intervals in configured bounds *avbs* and *avbe* with granularity given by the configured interval size *is*,
3. for each Gram matrix of interval values G_{iv} of created Gram matrices,
   (a) create a new Gram matrix G_{miv} by taking each interval value of the G_{iv} and putting middle value of this interval into G_{miv} on the according position,

(b) use built-in LLL reduction method[1] with default configuration to compute lattice basis reduction transformation matrix U from the G_{miv} Gram matrix and

- if U can not be computed because G_{miv} is not positive definite, then skip the relevant G_{iv},
- otherwise, store the transformation matrix U as a value under G_{iv} key into the constructed database $DB_{G_{iv},U}$.

To use the mentioned database, assuming that there is a generated lattice basis $B = [b_0, ..., b_{d-1}]$, it was then needed to

1. compute Gram matrix G from the lattice basis B,
2. compute Pearson correlation matrix R, representing the normalized Gram matrix, from the Gram matrix G,
3. create a Gram matrix of interval values G_{iv} from the Pearson correlation matrix R by replacing each value of R by according value interval derived from parameters *avbs*, *avbe* and *is*,
4. use the created G_{iv} as a key to retrieve according transformation matrix U from the $DB_{G_{iv},U}$ database
5. use the retrieved transformation matrix U in attempt to reduce the initial lattice basis B to create B_{pr} by $B_{pr} = UB$,
6. and measure reduction rate of the resulting lattice basis B_{pr} by computing its root Hermite factor.

It is important to note that in the case of this thesis, since Pearson correlation matrix computation was used as a Gram matrix normalization method, the parameter *avbs* was always set to be $avbs = -1$ and the parameter *avbe* was always set to be $avbe = 1$. This is because Pearson correlation matrix cannot have values outside of these bounds, and so it is true that $-1 \leq R[i][j] \leq 1$, $0 \leq i \leq d-1$, $0 \leq j \leq d-1$.

1. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix2.html#sage.matrix.matrix2.Matrix.LLL_gram`

It is also important to note that Pearson correlation matrix computation was used as a Gram matrix normalization method in this concept of this thesis for

- its resulting entry values being in $[-1, 1]$
- and all its resulting diagonal values being equal to 1,

since these attributes lead to drastic reduction of the storage space required for the created database $DB_{G_{iv},U}$.

For example, lets assume that the database $DB_{G_{iv},U}$ is to be stored with G_{iv} according to

- $avbs = -1$, $avbe = 1$, $is = 0.2$ and $d = 3$.

Then, to fill the database, the process is to

1. create first G_{iv} according to set parameters, which is

$$G_{iv} = \begin{bmatrix} 1 & [-1,\ -0.8) & [-1,\ -0.8) \\ [-1,\ -0.8) & 1 & [-1,\ -0.8) \\ [-1,\ -0.8) & [-1,\ -0.8) & 1 \end{bmatrix},$$

2. create G_{miv} according to G_{iv} and so

$$G_{miv} = \begin{bmatrix} 1 & -0.9 & -0.9 \\ -0.9 & 1 & -0.9 \\ -0.9 & -0.9 & 1 \end{bmatrix},$$

3. try to retrieve transformation matrix U from the mentioned LLL reduction process ran on G_{miv}, however, G_{iv} in this case is not positive definite, and therefore is skipped at this moment,

4. create second G_{iv} according to set parameters, where

$$G_{iv} = \begin{bmatrix} 1 & [-1,\ -0.8) & [-1,\ -0.8) \\ [-1,\ -0.8) & 1 & [-0.8,\ -0.6) \\ [-1,\ -0.8) & [-0.8,\ -0.6) & 1 \end{bmatrix},$$

   and repeat the process, potentially storing retrieved transformation matrix U in $DB_{G_{iv},U}$ under the G_{iv} key, if possible, until all unique G_{iv} are processed.

Then, lets also assume there is a generated lattice basis with row basis vectors $B = [b_0, ..., b_{d-1}]$ such that

$$B = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 4 \end{bmatrix}.$$

The process to try to reduce it by a transformation matrix retrieved from $DB_{G_{iv},U}$ is then to

1. compute Gram matrix G from B, and so

$$G = \begin{bmatrix} 9 & 7 & 11 \\ 7 & 11 & 8 \\ 11 & 8 & 18 \end{bmatrix},$$

2. compute Pearson correlation matrix R from G, and so

$$R = \begin{bmatrix} 1 & \frac{7}{3\sqrt{11}} & \frac{11}{9\sqrt{2}} \\ \frac{7}{3\sqrt{11}} & 1 & \frac{8}{3\sqrt{22}} \\ \frac{11}{9\sqrt{2}} & \frac{8}{3\sqrt{22}} & 1 \end{bmatrix},$$

3. compute G_{iv} from R, and so

$$G_{iv} = \begin{bmatrix} 1 & [0.6, 0.8) & [0.8, 1] \\ [0.6, 0.8) & 1 & [0.4, 0.6) \\ [0.8, 1] & [0.4, 0.6) & 1 \end{bmatrix},$$

4. retrieve according transformation matrix U from $DB_{G_{iv},U}$ by using G_{iv} as key

5. and use U to try to reduce basis B by computing $B_{pr} = UB$.

7.3 Concept usage experiment

The experiment with the described concept attempted to examine, assuming that B_{first} and B_{second} are both lattice bases of the same dimension, following hypothesis:

- *If B_{first} has a similar Pearson correlation matrix computed from its Gram matrix to that of B_{second}, then their reduction transformation matrices are also similar.*

The process of this experiment was to

1. create database $DB_{G_{iv},U}$ by the earlier described process, such that
   - $avbs = -1$, $avbe = 1$, $is = 0.4$ and $d = 5$,
2. generate 5000 random 5 dimensional lattice bases,
3. for each of the generated lattice bases, retrieve according transformation matrices from the $DB_{G_{iv},U}$ by the earlier described process,
4. and for each of the generated lattice bases, apply the retrieved transformation matrix and measure resulting reduction rate by computing root Hermite factor.

Finally, a histogram was constructed from all retrieved reduction rates. Also, for comparison purposes, a histogram containing reduction rates, retrieved by computing root Hermite factor, of

- 312 random unreduced 5 dimensional lattice bases
- and 5000 random 5 dimensional lattice bases reduced by the built-in LLL reduction algorithm method[2] with default configuration

was also constructed. Results were following.

It is important to note that *LBR* stands for *lattice basis reduction*, value range in $<>$ represents global interval space $(< avbs, avbe >)$ and value in [] stands for interval size of interval values of G_{iv} $([is])$.

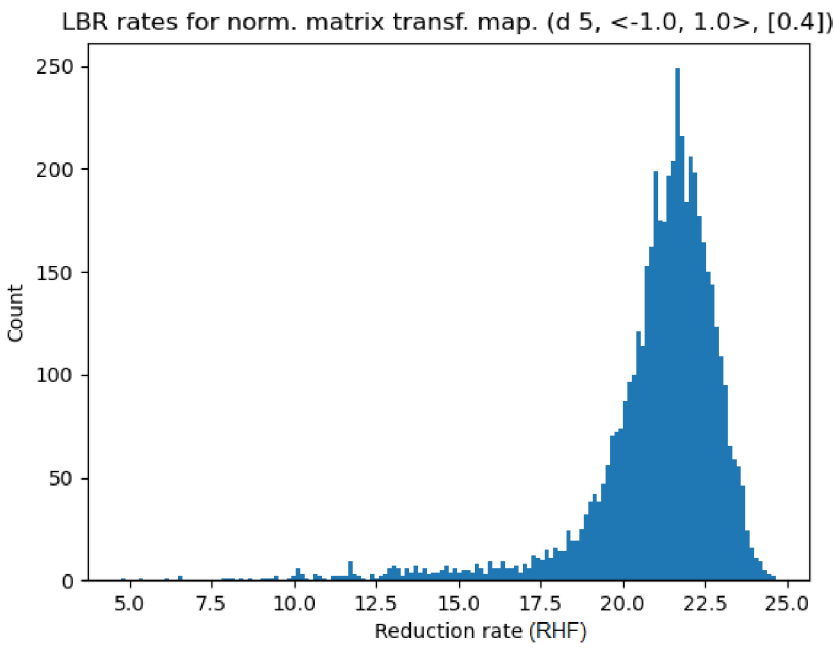

Figure 7.1: Histogram of reduction rates (root Hermite factor) for precomputed transformation matrix mapping (dimension 5, interval size 0.4, 5000 lattice bases).

2. `https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html#sage.matrix.matrix_integer_dense.Matrix_integer_dense.LLL`

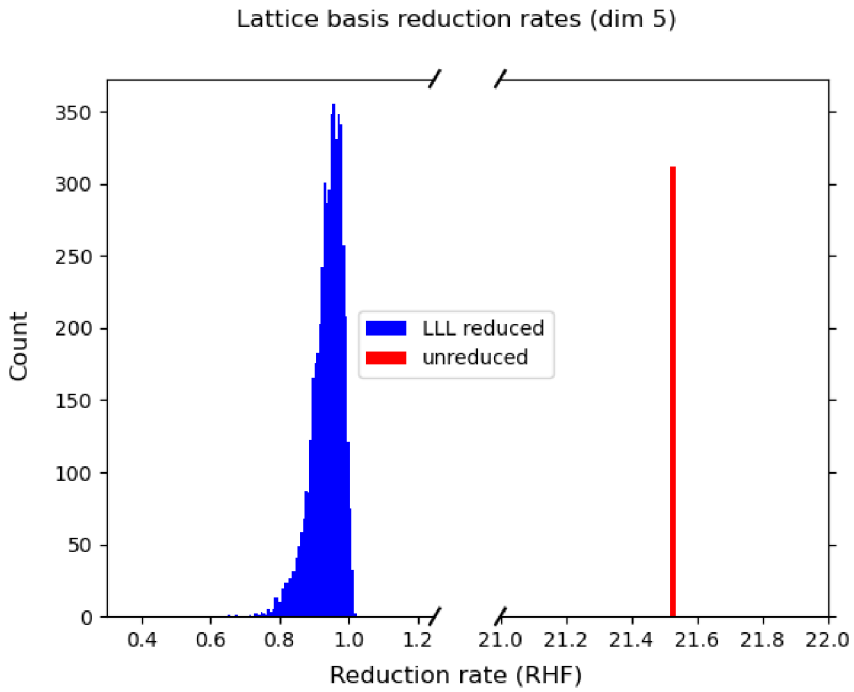

Figure 7.2: Histogram of reduction rates (root Hermite factor) of 5000 unreduced and 312 LLL reduced lattice bases (5 dimensional).

It is important to note, that the only tested configuration was such where $d = 5$ and $is = 0.4$, because

- $d = 5$ was the biggest possible dimension of lattice bases
- and $is = 0.4$ was smallest possible interval size for given dimension

where it was possible to create transformation matrix database in the scope of limitations of this concept's implementation. The limitation was primarily due to the database being stored in operation memory of a computer.

From shown results, it seems that the transformation matrices retrieved from the precomputed database have a negligible chance of significantly reducing lattice bases with similar normalized Gram matrices.

However, it is recommended that this concept is analyzed on bigger lattice basis dimensions and lower interval sizes to either find out that it potentially can be useful for the mentioned purpose, or to further confirm the result of this experiment.

8 Conclusion

Throughout working on this thesis, properties of computation of optimal real lattice basis vector coefficients were explored. As first, results show that sorting an unreduced lattice basis by computed optimal real coefficients' modulo 1 remainder does not need to improve BKZ reduction computation time or reduction rate. Next, results also show that for LLL reduced lattice bases, the optimal real coefficients computed during the short vector enumeration might be, given their closeness to optimal integral coefficients, a strong indicator of which coefficient value to pick to find the shortest possible vector in the given enumeration context, with the optimal closeness threshold value seemingly being $threshold = 0.15$.

Next, sorting an unreduced lattice basis by biggest recursively computed Cholesky decomposed Gram matrix end diagonal values belonging to lattice basis vectors was tested. Results show that this sorting method does not need to improve BKZ reduction computation time or reduction rate. However, results also indicate that using mentioned sorting strategy on lattice bases can have positive impact on resulting short vector enumeration tree sizes in cases, where relevant lattice bases are unreduced. Results also indicate that using mentioned sorting strategy can have even more significant positive impact on resulting short vector enumeration tree sizes in cases, when relevant lattice bases are unreduced and short vector enumeration pruning method, with updating global vector length bound every time a shorter vector is found, is used.

Then, properties of quadrants of a Gram matrix computed from LLL reduced lattice basis were analyzed. As first, results show that the analyzed full variant of sequential search through limited lattice basis vector coefficient combination space is sufficient for finding vector whose coefficient combination results in Gram matrix upper right quadrant negative sum being low enough to surpass upper left quadrant positive sum or lower right quadrant positive sum.

Results also indicate that heuristic variant of the mentioned search seems to be faster than full short vector enumeration with short vector length bound set to the length of the longest lattice basis vector while

still being able to find vector whose coefficient combination fulfills the mentioned quadrant relation.

However, results show that the mentioned quadrant relation has a very low chance, if any, of indicating that the found vector is shorter or equally short to the longest vector of the LLL reduced lattice basis.

Next, an approach which executed configurable amount of random changes to vector coefficient combination, where the change persisted only when Gram matrix upper right quadrant sum was lower than before, was analyzed. However, results indicate that this approach seems to have a negligible change of finding a coefficient combination representing a vector short enough to be shorter than the longest vector of the relevant LLL reduced lattice basis. Also, the chance, that the performed random change was persisted, was very low.

Finally, a method, where a finite database of normalized Gram matrices, with interval values with configurable interval width, was constructed, was analyzed. The mentioned database would then map the mentioned Gram matrices to transformation matrices meant to significantly reduced unreduced lattice basis with similar normalized Gram matrix. Results indicate that the transformation matrices retrieved from the precomputed database have a negligible chance of significantly reducing the mentioned unreduced lattice bases.

However, it is recommended that the last proposed method is analyzed on bigger lattice basis dimensions and lower interval sizes (widths), since the implementation created during the work on this thesis has severe memory limitations.

Bibliography

1. MICCIANCIO, Daniele; GOLDWASSER, Shafi. *Complexity of Lattice Problems: A Cryptographic Perspective*. 1st ed. San Diego, Cambridge: Springer, 2002. ISBN 978-0792376880.
2. KHAN, Jaleel. The World of Lattices: Structure, Applications, and Future Directions. *Advanced Materials Science Research*. 2024. Available also from: `https://www.openaccessjournals.com/articles/the-world-of-lattices-structure-applications-and-future-directions-18218.html`.
3. GOLDSTEIN, Daniel; MAYER, Andrew. On the equidistribution of Hecke points. *Forum Mathematicum*. 2003. Available also from: `https://www.degruyterbrill.com/document/doi/10.1515/form.2003.009/html`.
4. ALBRECHT, M. R.; SHEN, Y. Quantum Augmented Dual Attack. *arXiv preprint arXiv:2205.13983*. 2022. Available also from: `https://arxiv.org/abs/2205.13983`.
5. MARIANO, Artur; LAARHOVEN, Thijs; CORREIA, Fabio; RODRIGUES, Manuel. A Practical View of the State-of-the-Art of Lattice-Based Cryptanalysis. *IEEE Access*. 2017. Available also from: `https://ieeexplore.ieee.org/document/8023973/`.
6. WENGER, E.; SAXENA, E.; MALHOU, M.; THIEU, E.; LAUTER, K. Benchmarking Attacks on Learning with Errors. *arXiv preprint arXiv:2408.00882*. 2024. Available also from: `https://arxiv.org/abs/2408.00882`.
7. CHEN, Y.; CHUNG, K.-M.; LAI, C.-Y. Space-efficient classical and quantum algorithms for the shortest vector problem. *Quantum Information and Computation*. 2018. Available also from: `https://arxiv.org/abs/1709.00378`.
8. MICCIANCIO, Daniele. Shortest Vector Problem. *Encyclopedia of Cryptography, Security and Privacy*. 2025.
9. NGUYEN, P. Q.; VIDICK, T. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology, 2*. 2008. Available also from: `https://people.csail.mit.edu/vidick/JoMC08.pdf`.

10. DOULGERAKIS, E.; LAARHOVEN, T.; WEGER, B. de. Sieve, Enumerate, Slice, and Lift: Hybrid Lattice Algorithms for SVP via CVPP. *Lecture Notes in Computer Science*. 2020. Available also from: `https://scispace.com/pdf/sieve-enumerate-slice-and-lift-hybrid-lattice-algorithms-for-4ms2748gp8.pdf`.

11. LUAN, L.; GU, C.; ZHENG, Y.; SHI, Y. Lattice Enumeration with Discrete Pruning: Improvements, Cost Estimation and Optimal Parameters. *Mathematics*. 2023. Available also from: `https://www.mdpi.com/2227-7390/11/3/766`.

12. TATEIWA, N.; SHINANO, Y.; NAKAMURA, S.; YOSHIDA, A.; KAJI, S.; YASUDA, M.; FUJISAWA, K. Massive Parallelization for Finding Shortest Lattice Vectors Based on Ubiquity Generator Framework. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2020. Available also from: `https://catalog.lib.kyushu-u.ac.jp/opac_download_md/4771852/4771852.pdf`.

13. MOHSIN, J.; AL-SHABLY, Z. H. Implementations of the Lenstra Lenstra Lovász algorithm in MATLAB. *AIP Conference Proceedings*. 2024.

14. THIEMANN, R.; BOTTESCH, R.; DIVASÓN, J.; HASLBECK, M. W.; JOOSTEN, S. J. C.; YAMADA, A. Formalizing the LLL Basis Reduction Algorithm and the LLL Factorization Algorithm in Isabelle/HOL. *Journal of Automated Reasoning*. 2020.

15. STEHLÉ, D. Floating-point LLL: Theoretical and practical aspects. *Information Security and Cryptography*. 2009.

16. HANROT, G.; STEHLÉ, D. Worst-Case Hermite Korkine Zolotarev Reduced Lattice Bases. *arXiv preprint arXiv:0801.3331*. 2008. Available also from: `https://arxiv.org/pdf/0801.3331`.

17. SCHNORR, C. P.; EUCHNER, M. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*. 1994. Available also from: `https://d-nb.info/1169615635/34`.

18. TATEIWA, N.; SHINANO, Y.; YASUDA, M.; KAJI, S.; YAMAMURA, K.; FUJISAWA, K. Development and analysis of massive parallelization of a lattice basis reduction algorithm. *Japan Journal of Industrial and Applied Mathematics*. 2023.

19. HORN, Roger A.; JOHNSON, Charles R. *Matrix Analysis*. 2nd ed. Cambridge: Cambridge University Press, 2013. ISBN 978-0-521-54823-6.

20. HARBRECHT, H.; PETERS, M.; SCHNEIDER, R. On the low-rank approximation by the pivoted Cholesky decomposition. *Applied Numerical Mathematics*. 2012.

21. ZHAO, Z.; DING, J. Practical Improvements on BKZ Algorithm. *Lecture Notes in Computer Science*. 2023.

22. SHEN, Tongchen; LI, Xiangxue. A Survey on Classical Lattice Algorithms. *MDPI Cryptography, 10*(2). 2026. Available also from: `https://www.mdpi.com/2410-387X/10/2/17`.

23. SCHNORR, C.P. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science, Volume 53, Issues 2–3*. 1987. Available also from: `https://www.sciencedirect.com/science/article/pii/0304397587900648`.

24. JAMES, Stewart; DANIEL, Clegg; SALEEM, Watson. *Calculus: Early Transcendentals*. 9th ed. Cengage Learning, Inc, 2019. ISBN 978-0-357-11351-6.

25. KIJSIPONGSE, E.; U-RUEKOLAN, S.; NGAMPHIW, C.; TONGSIMA, S. Efficient large Pearson correlation matrix computing using hybrid MPI/CUDA. *2011 Eighth International Joint Conference on Computer Science and Software Engineering* (*JCSSE*). 2011. Available also from: `https://ieeexplore.ieee.org/document/5930127`.

26. SCHOBER, P.; BOER, C.; SCHWARTE, L. A. Correlation Coefficients: Appropriate Use and Interpretation. *Anesthesia and Analgesia*. 2018. Available also from: `https://journals.lww.com/anesthesia-analgesia/fulltext/2018/05000/correlation_coefficients__appropriate_use_and.50.aspx`.

27. RANGANATHAN, P.; AGGARWAL, R. Common pitfalls in statistical analysis: The use of correlation techniques. *Perspectives in Clinical Research*. 2016. Available also from: https://pmc.ncbi.nlm.nih.gov/articles/PMC5079093/.

28. ASUERO, A. G.; SAYAGO, A.; GONZÁLEZ, A. G. The Correlation Coefficient: An Overview. *Critical Reviews in Analytical Chemistry*. 2006. Available also from: https://www.researchgate.net/publication/244588315_The_Correlation_Coefficient_An_Overview.

29. CHEN, H.; WANG, C. Large dimensional Spearman's rank correlation matrices: The central limit theorem and its applications. *arXiv preprint arXiv:2411.15861*. 2024. Available also from: https://arxiv.org/abs/2411.15861.

Appendix

1. **Relevant GitHub repository** containing all analyzed concepts' and relevant experiments' implementations is public and available here.