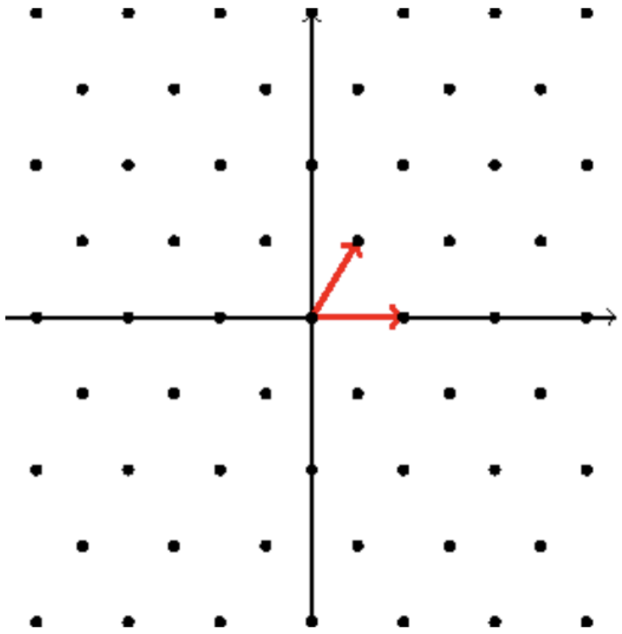
Cryptography: Shortest Vector Problem

Systems Programming

COMP2221

Pulindu Fonseka

cfcf66



1. Introduction

There is a rising risk that modern encryption will not be sufficient in the future due to the rapid development of quantum computers, and their ability to quickly solve current cryptography. To counter this, many post-quantum cryptographies look at using lattice-based cryptography to keep data secure. One prominent example is the Shortest Vector Problem (SVP). The following article is dedicated to solving the SVP. I begin by using a brute force algorithm, but alone it is ineffective for a large basis. So, I later implement LLL reduction.

2. My algorithm

2.1. Parsing

My algorithm starts by parsing the vector to be stored correctly for use.
This is done by checking for '[' in the input to find the start of each vector, and ']' to find the end. After parsing the input vectors, the algorithm allocates memory for the basis vector. For debugging purposes, the basis vector is also printed in the terminal.

2.2. LLL reduction

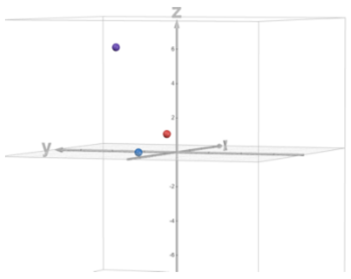
Size Condition: $| \mu_{i,j} | \geq 0.5$, *when* $\mu_{i,j} = \langle b_i, b_j^* \rangle / \langle b_j^*, b_j^* \rangle$

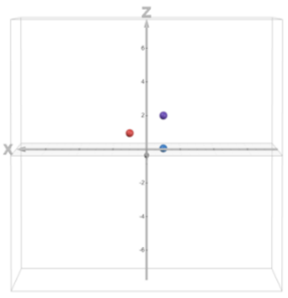
Lovász Condition: $|b_k^*|^2 \geq (delta - \mu_{k,k-1})^2 \times |b_{k-1}^*|^2$, *for* $k = 1, \ldots., n$

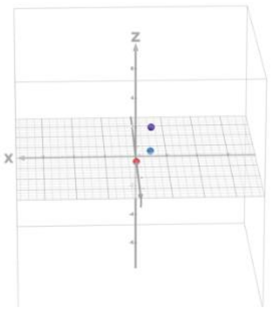
The LLL algorithm uses 2 main conditions: The Size Condition and Lovász Condition. After being correctly parsed, we must reduce the basis. LLL works by repeatedly going over the basis and making it more orthogonal. Each time the size condition is met during the loop, the kth vector in the basis is reduced in size. Whether the size condition is met or not, the algorithm checks for whether the Lovász Condition is met (and increments k if it is). If not met, the current vector will be swapped with the previous vector.

2.3. Post-LLL Processing

The function 'further_processing()' takes in the reduced basis and iteratively tries to optimise the basis vectors (it stops after a set maximum number of loops). This is done by subtracting rows from other rows. If the Euclidean length is smaller after this subtraction, then it is saved. To visually show that the algorithm does further orthogonalize a basis I have included the following graphs from desmos. From left to right: basis before LLL, basis after LLL, basis after processing.







2.3. Enumeration

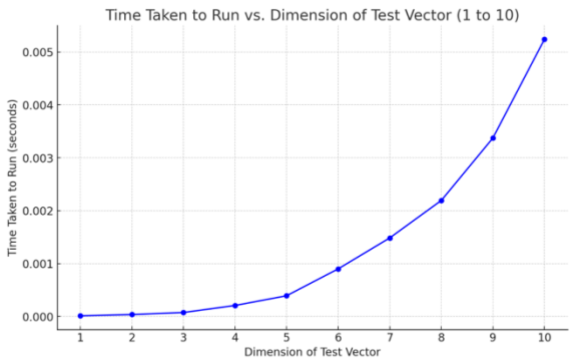
The ‘enumeration()’ function sets up the conditions for enumeration and allocates memory. It then calls the function ‘enum_recursive()’. This function recursively explores the lattice defined by the basis to find the shortest vector. Since the lattice is infinite, a finite bound has been set ‘boundary = 10’. This reduces the radius of the lattice searched.

3. Optimisation and time/space complexity

3.1 Optimisation

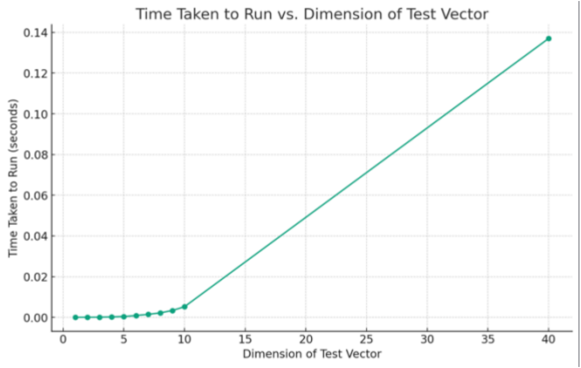
Initially, I used a brute force approach. For smaller dimensions of the basis vectors, it worked quickly. However, for larger vectors, the time taken for the program to run was very impractical. This is why I implemented LLL. LLL can reduce the basis to be much smaller in while keeping the vectors in the lattice, making the enumeration algorithm work much faster. The worst-case scenario remains the same; however, the average case is significantly faster. Although LLL reduction simplifies the basis, it won’t be optimally reduced. Therefore, I added a final reduction step. By calling the function ‘further_processing()’, I can further orthogonalize the vectors to be closer to an optimal vector for enumeration to run quickly. If no progress is made after one iteration of subtracting pairs of vectors, the loop is exited early.

3.2 Time complexity



The time complexity of the brute force requires super-exponential time O($2^{\omega(n)}$).
Time complexity of my algorithm:

- LLL: n^2 (due to the nested for-loops)
- Further processing: n^3
- Enumeration: exponential (as demonstrated in the graphs)



After LLL reduction and further_processing(), it works much faster in practice. In the worst case the algorithm is still super-exponential time, but most cases will run faster. So, my algorithm can solve even 40-dimensional vectors in a short amount of time. However, since the algorithm still runs in exponential time, for an extremely large basis, it can’t find the shortest vector in a reasonable amount of time since speed and accuracy must be balanced. If a small enough boundary is set, then it will be solved fast however it may not return the smallest possible Euclidean length.

3.3 Space complexity

The space complexity of:

- LLL: n^2
- Post-processing: constant
- Enumeration: n^2

So, according to my mathematical calculations, the algorithm has n^2 overall space complexity.

References

[1] Külshammer, J. and Polách, J. (2022). *Lattice Basis Reduction Using LLL Algorithm with Application to Algorithmic Lattice Problems*. [online] Available at: https://uu.diva-portal.org/smash/get/diva2:1641300/FULLTEXT01.pdf [Accessed 30 Feb. 2023].

[2] Schaefer, S. (2020). *LLL Algorithm*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=U8MI2a_BHHo&t=286s [Accessed 1 Jan. 2024].

[3] kel.bz. (2017). *Building Lattice Reduction (LLL) Intuition – kel.bz*. [online] Available at: https://kel.bz/post/lll/.

[4] Silverman, Joseph. "Introduction to Mathematical Cryptography Errata"(PDF). *Brown University Mathematics Dept*.
[5] Orisano (n.d.). *olll/olll.py at master · orisano/olll*. [online] GitHub. Available at: https://github.com/orisano/olll/blob/master/olll.py [Accessed 1 Jan. 2024].

[6] GitHub. (n.d.). *Lattice-Reduction/lll_reduce.c at master · gradyrw/Lattice-Reduction*. [online] Available at: https://github.com/gradyrw/Lattice-Reduction/blob/master/lll_reduce.c [Accessed 2 Jan. 2024].

[7] Wickr (2018). *What is Lattice-based cryptography & why should you care*. [online] Medium. Available at: https://medium.com/cryptoblog/what-is-lattice-based-cryptography-why-should-you-care-dbf9957ab717 [Accessed 18 Jan. 2024].

[8] Kannan, Ravi (1983). "Improved algorithms for integer programming and related lattice problems". *Proceedings of the fifteenth annual ACM symposium on Theory of computing - STOC '83*. New York, NY, USA: ACM. pp. 193–206. doi:10.1145/800061.808749. ISBN 978-0-89791-099-6. S2CID 18181112

[9] *Fincke, U.; Pohst, M. (1985). "Improved Methods for Calculating Vectors of Short Length in a Lattice, Including a Complexity Analysis". Math. Comp.* ***44*** *(170): 463–471. doi:10.1090/S0025-5718-1985-0777278-8*.

[10] Ajtai, Miklós; Kumar, Ravi; Sivakumar, D. (2001). "A sieve algorithm for the shortest lattice vector problem". *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. Hersonissos, Greece: ACM. pp. 601–610. doi:10.1145/380752.380857. ISBN 1-58113-349-9. S2CID 14982298

[11] Desmos. (n.d.). *Desmos | 3D Graphing Calculator*. [online] Available at: https://www.desmos.com/3d.