Solving the Shortest Vector Problem

INITIAL THOUGHTS

Initially speaking, the Shortest Vector Problem is an interesting one. Being NP-hard, getting the right answer for higher dimension lattices can take a long time with certain methodologies, such as a greedy brute force search. With brute force being the first implemented algorithm, concern was raised when testing for 5 dimensions took over 6 minutes for the program to complete, returning the correct answer.

The brute force search generated all possible combinations of all the basis vectors within a range, enumerated through every single one within a bound, slowly shrinking the bound down until the shortest one was reached. This was a novel solution, meaning there was little to no support or resources available online. Thus, the approach that was being taken had to be changed.

A GLIMPSE INTO LLL REDUCTION

LLL reduction is a method of lattice reduction that transforms a "bad" lattice into a "good" one, by transforming the basis vectors to be nearly orthogonal while representing the same lattice. It first performs Gram-Schmidt reduction on a basis, uses the μ values for size reduction and checks the basis against the Lovasz condition. This algorithm is especially helpful at reducing the time needed, as LLL reduction "runs in polynomial time" as it "requires $O(n^2 \log(X))$ iterations", according to proof seen in Galbraith's paper. (Galbraith, Steven, *Mathematics of Public Key Cryptography*, 2018, p.375-379)[1]

SCHNORR-EUCHNER ENUMERATION AT A GLANCE

Schnorr-Euchner enumeration becomes the next step in the algorithm following from LLL reduction. According to Yasuda, this enumeration algorithm exhaustively searches for an integer combinations of basis vectors such that the lattice vector is the shortest. It acts as a depth first search of the enumeration tree, with a running time depending on the number of tree nodes.[2] (Yasuda, M., *A Survey of Solving SVP Algorithms and Recent Strategies for Solving the SVP Challenge,* 2021, p196-197). The basic pseudocode on page 197 laid a foundation for the enumeration algorithm used in the code.

When combined with LLL reduction. Schnorr-Euchner enumeration gives an exact answer in a short amount of time. From Gama, Nguyen and Regev, the enumeration complexity of Schnorr-Euchner enumeration is $\sum_{l=1}^{n} q^{(n-l)/2} 2^{O(n)}$, where 'q' is a constant depending on the basis.[3] (Gama, N., *Lattice Enumeration using Extreme Pruning*, 2010, p257-278)

APPROACH TO THE PROBLEM

The first step to writing this program in C++ was to not write it in C++. The foundation for my program was entirely written in Python since it is a language that I'm more familiar with. Initially, the earlier specified brute force algorithm was being used. As specified earlier, this was not the best way to start. Initially, it took 6:40 for a 5-dimensional lattice basis. After some more testing and optimization, this was reduced to 1:20.

The next logical step was to write an accurate that was fast *and* accurate. Therefore, I implemented LLL reduction and Schnorr-Euchner enumeration together, in the hopes of making a quick and good algorithm. The same 5 dimensional test now took only a fraction of a second.

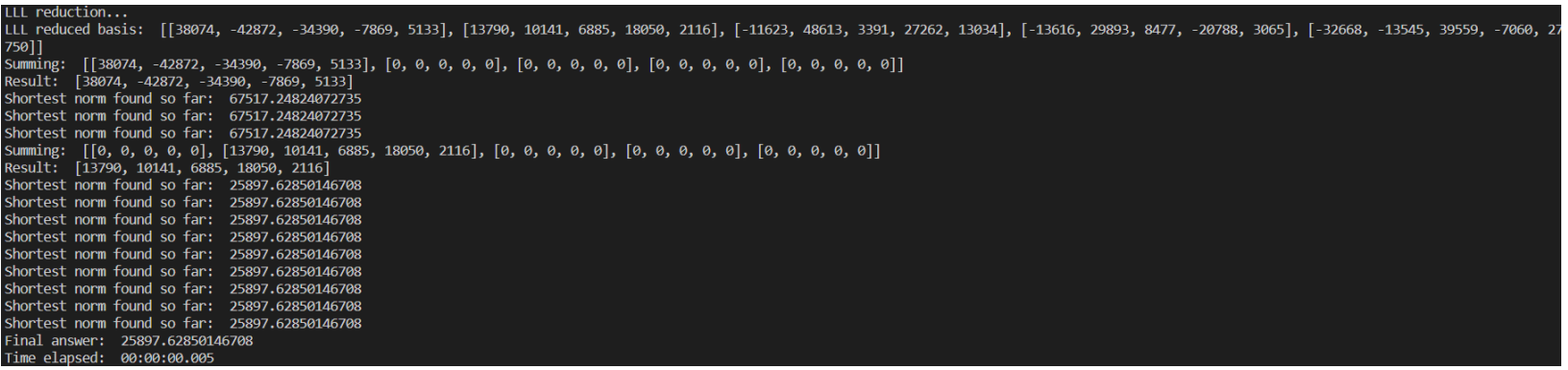


Figure 1: A screenshot of the terminal running the SVP solving problem with a 5D lattice

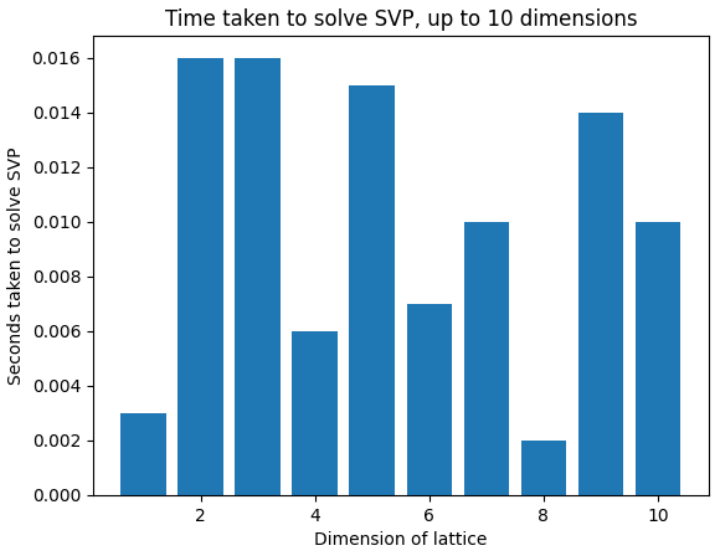
Once I knew this was correct, I translated it to C++, researching equivalents of Pythonic functions in C++ and writing it out by hand.

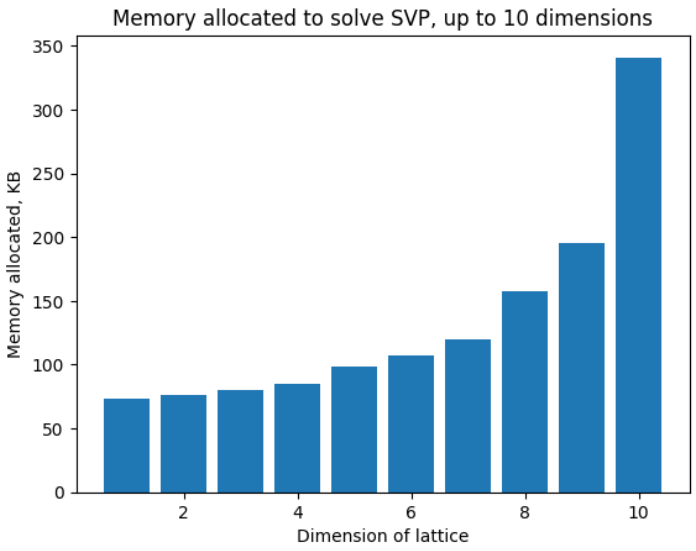
The inputted basis has commas taken out of it, if there are any, and each basis vector is represented by the square brackets around the numbers. A closing square bracket indicates the end of a vector, and an open square bracket is the start of a new one.

This basis is then passed to a function that performs LLL reduction, transforming the basis vectors to be nearly orthogonal, making it easier to calculate the SVP. It then feeds this LLL-reduced basis into Schnorr-Euchner enumeration, which, as specified earlier on, performs a depth first search on the enumeration tree to find the shortest vector.

PERFORMANCE

Below are some graphs to show the performance of my algorithm. The graphs show the time taken and memory allocated for randomly generated 16-bit integer lattices from 1 – 10 dimensions.





One optimization while writing this code was the use of the 'double' data type. This struck a good balance between ensuring that bases of many sizes could be input as possible lattice bases, but was not so large that unnecessary memory was being allocated, such as with the long double type. Double also supports decimals, which is good for calculations within the code, such as the 'mu' within the Gram-Schmidt process.

REFERENCES

[1]Steven D. Galbraith, *Mathematics of Public Key Cryptography,* 2018: https://www.math.auckland.ac.nz/~sgal018/crypto-book/main.pdf

[2]Masaya Yasuda, *A Survey of Solving SVP Algorithms and Recent Strategies for Solving the SVP Challenge*, 2021 In: Takagi, T., Wakayama, M., Tanaka, K., Kunihiro, N., Kimoto, K., Ikematsu, Y. (eds) *International Symposium on Mathematics, Quantum Theory, and Cryptography. Mathematics for Industry, vol 33*: https://doi.org/10.1007/978-981-15-5191-8_15

[3]Nicolas Gama, Phong Q. Nguyen, Oded Regev, *Lattice Enumeration using Extreme Pruning*. In: EUROCRYPT '10, 2010, p.257-278: https://www.iacr.org/archive/eurocrypt2010/66320257/66320257.pdf