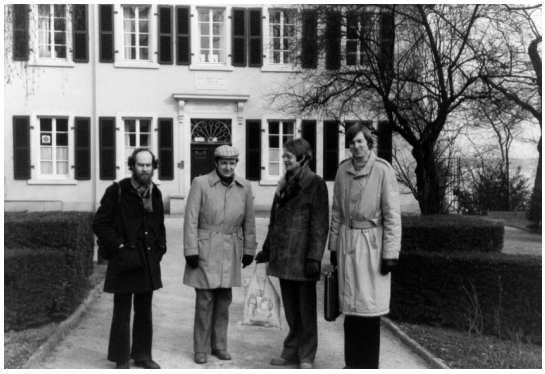
Nina Py Brozovich, MATH1580, S2024

The Lenstra, Lenstra, Lovász Algorithm



Introduction

The Lenstra, Lenstra, Lovász algorithm, better known as LLL, is a **polynomial time algorithm** used to find the shortest vector in a Lattice for large dimensions. The algorithm was published in 1982 by mathematicians László Lovász, Hendrik Lenstra and Arjen Lenstra.

Why is the algorithm useful?

Many current lattice-based cryptographic systems rely on hard mathematical problems in a lattice like the Short Vector Problem (SVP) and the Closest Vector Problem (CVP). Having algorithms that allow us to approximate the solution fast for these problems can help mathematicians improve the efficiency of their protocols while maintaining their security. In particular, fast mathematical algorithms will be key in developing protocols that resist the emergence of quantum computers.

Intuition: how to find a short vector?

To find the shortest vector in a lattice from the basis we can intuitively start by reducing the size of the vectors in the basis. LLL suggests we also need to orthogonalize the basis vectors, but why? In an orthogonal basis, the shortest basis vector is the shortest vector in the lattice. Therefore, orthogonalizing the basis to the maximum and reducing the vectors to their minimum size will produce the shortest vector in the basis to be *similar* to the shortest vector of the lattice.

A good starting point could be Gauss's Lattice Reduction Algorithm to solve the Shortest Vector Problem in 2 dimensions. The algorithm consists of swapping the vectors in the basis in order to have the shortest in the first position. We then continuously subtract a multiple of the smallest vector in the tuple $m = \lfloor \frac{v_1 \cdot v_2}{|v_1|^2} \rceil$ to the largest one until m = 0. We continuously check the size condition and swap again if possible. By analogy in higher dimensions, we would want to reduce our basis by subtracting linear combinations of vectors in the basis and swapping them to draw the smallest to the first position. In the end, all vectors should be minimal in size and organized such that the first vector is the smallest and the rest grow in size as slowly as possible. Additionally, as stated above, all vectors in the basis should be as close to orthogonality as possible to simplify the computations.

The algorithm

Input: A basis of a lattice.

Output: Returns a reduced basis of the lattice, satisfying

(1) The Lovász Condition (explained in detail below)

(2) $\mu_{i,j} = \frac{v_i \cdot v_j^*}{|v_j^*|^2} \leq \frac{1}{2}$, where v_j and v_i^* represent vectors in the input and provisional reduced basis respectively, for all $1 \leq j < i \leq n$ (the size of the basis).

Vectors in the basis have slowly increasing size. The first vector in the reduced basis is an approximation of the shortest vector in the lattice.

Algorithm: The algorithm consists of two operations performed successively until achieving the **Lovász Condition**:

(1) Orthogonalization of the vectors using the Gram-Schmidt algorithm.

(2) Reduction of the size of the vectors by subtracting linear combinations the rest of the vectors in the provisional basis.

In essence, this corresponds to the same idea as the Gaussian Basis Reduction or the Euclidean Algorithm to find the GCD of two numbers.

Explained Pseudo-code

Please refer to the rest of the submission for the python implementation and relevant test cases extracted from cryptography textbooks.

Let **n** be the number of vectors in the input basis.

Let **k** be the index of the "working vector," the vector in the provisional basis we will be working with throughout the iteration of the loop.

Loop through the code while $k \leq n$ to make sure all vectors in the initial basis are reduced.

Reduce the size of each vector in the basis using the vectors in the provisional reduced basis computed in the steps before: the loop goes through $k - 1 \leq j \leq 1$.

> More precisely, subtract the projection of the working vector onto the provisional reduced vectors.

If the Lovász Condition is satisfied, then we have found the smallest version of the vector we can have. Keep this version of the vector as the provisional vector for index k, and move on to index k + 1.

If Lovász is not satisfied, then the working vector is smaller than some other vector in the provisional basis. Since the basis has vectors of increasing size, we must swap the two vectors and go back to index max(k - 1, 2) (either the vector swapped with the current working vector, or the first vector in the basis) to make sure the conditions are still satisfied for all vectors computed before.

Once we have checked that all vectors satisfy Lovász's Condition, return the reduced basis.

The Lovász's Condition

$$|v_i^*|^2 \geq (\delta - \mu_{i,i-1}^2)|v_{i-1}^*|^2 \qquad \text{for all } 1 \leq j < i \leq n$$

Where v_j and v_i^* represent vectors in the input and provisional reduced basis respectively, $\mu_{i,j} = \frac{v_i \cdot v_j}{|v_j|^2}$, and $\delta = \frac{3}{4}$ (in our case).

This condition checks for both size and orthogonality. For every vector in the reduced basis, the condition ensures that the size of the currently reduced vector is not significantly larger than the previously reduced vector in the basis. This comparison helps ensure that the vectors in the basis are not increasing too rapidly in size, indicating that they are relatively short. Specifically, it compares the length of the v_i^{th} vector in the reduced basis to the length of the v_{i-1}^{th} vector which is supposed to be smaller. Additionally, when $\mu_{i,i-1}$ is close to 0, it indicates that the vectors are nearly orthogonal. In general $\delta \in [\frac{1}{4}, 1[$ to guarantee polynomial runtime. In our case $\delta = \frac{3}{4}$ since greater values of δ lead to stronger reductions of the basis.

Size of the shortest vector in the basis

Claim: the first vector in the basis is an approximation of the shortest vector in the lattice, and has size satisfying $|v_1| \leq (\frac{2}{\sqrt{4\delta-1}})^{n-1}\lambda$, where $\lambda \geq min_i |v_i^*|$.

Proof: By the LLL reduction, we know the vectors in our basis satisfy,

$$|v_i^*|^2 \geq (\delta - \mu_{i,i-1}^2)|v_{i-1}^*|^2$$

The LLL algorithm ensures that $\mu_{i,i-1} \leq \frac{1}{2}$. Using this fact,

$$\delta - (\frac{1}{2})^2 = \frac{4\delta-1}{4}$$

$$\Leftrightarrow |v_i^*|^2 \geq (\frac{4\delta-1}{4})|v_{i-1}^*|^2$$

Starting with $i = 1$ (the first index) and applying induction gives,

$$|v_1^*|^2 \leq (\frac{4}{4\delta-1})|v_2^*|^2 \qquad \text{and} \qquad |v_2^*|^2 \leq (\frac{4}{4\delta-1})|v_3^*|^2 \qquad \text{etc.}$$

$$\Leftrightarrow |v_1^*|^2 \leq (\frac{4}{4\delta-1})((\frac{4}{4\delta-1})|v_3^*|^2) = (\frac{4}{4\delta-1})^2|v_3^*|^2$$

$$\Leftrightarrow |v_1^*|^2 \leq (\frac{4}{4\delta-1})^{i-1}|v_i^*|^2$$

$$\Leftrightarrow \sqrt{|v_1^*|^2} \leq \sqrt{(\frac{4}{4\delta-1})^{i-1}|v_i^*|^2}$$

$\Leftrightarrow |v_1{}^*| \leq (\frac{2}{\sqrt{4\delta-1}})^{i-1}|v_i{}^*|$

$\Leftrightarrow |v_1{}^*| \leq (\frac{2}{\sqrt{4\delta-1}})^{n-1}\lambda$ □

More precisely, this means the shortest vector in the basis is significantly smaller than any other vector in the basis, ensuring we have a good approximation of the shortest vector in the lattice by the end of the algorithm.

Runtime

One of the reasons why the LLL Algorithm is so relevant is because it can be solved in polynomial time. To roughly understand this consider the operations computed in the algorithm:

Consider we have m vectors in the basis for a lattice of dimension n. Gram-Schmidt orthogonalization takes approximately $n \times m^2$ steps: each step takes roughly $O(n)$ steps, since we are only computing simple operations like sums, products and dot products at each step. Additionally, the algorithm is run m times to account for all the vectors in the process, therefore the total runtime for Gram-Schmidt turns out to be $O(n \times m^2)$. LLL runs Gram-Schmidt every time it updates a vector in the basis, at every reduction and swap step. Further analysis leads to concluding the algorithm executes the main k loop no more than $O(n^2 log\, n + n^2 log\, B)$ times, where B is the maximal size of the norms of initial vectors, which results in a polynomial runtime for LLL. More precisely, as explained in detail in HPS 7.13. the LLL algorithm terminates in no more than $O(n^6 B^3)$, but has proved to take significantly less time in practice.

Application example

Consider 2 vectors of the basis of a lattice, $v_1 = [3, 8]$, $v_2 = [5, 14]$

The basis resulting from the algorithm gives $v_1{}^* = [-1, 0]$, $v_2{}^* = [0, -2]$

Using the Gaussian expected shortest length,

$$exp\,|v_{shortest}| = \sqrt{\frac{2}{2\pi e}}(det\, L)^{\frac{1}{n}} \simeq 0.5$$

And comparing the size of the shortest vector and Hadamard's Ratio of each basis,

$|v_1| \simeq 8.5$ $\qquad$ $H(v_1, v_2) = (\frac{det\, L}{|v_1|\cdot|v_2|})^{\frac{1}{n}} = (\frac{-70}{\sqrt{73}\cdot\sqrt{221}})^{\frac{1}{2}} = 0.74$

$|v_1{}^*| = 1$ $\qquad$ $H(v_1{}^*, v_2{}^*) = (\frac{2}{2})^{\frac{1}{2}} = 1$

Note the Hadamard Ratio was significantly improved with the reduction. The equality indicates the vectors are orthogonal. We can check that by computing $v_1 \cdot v_2 = 0$, as expected.

The size of the shortest vector in the basis was greatly improved compared to the initial basis as well. It now approximates the size of the expected shortest vector by approximately 0.5.

Conclusions and opening

The LLL algorithm is currently one of the most used lattice reduction algorithms. Upon publication, the LLL algorithm was received with great success. To this date, it was cited in 6091 articles. LLL was used to break dozens of public-key cryptosystems including an "earlier version of RSA", the Merkle-Hellman cryptosystem published in 1978 and broken by mathematician Shamir the year of the publication of LLL. The simplicity of the mathematical concept and the fast runtime makes LLL a promising candidate for post-quantum cryptography.

Interesting subjects to study in the future include real life applications of LLL in cryptosystems related or unrelated to SVP and CVP, and the study of possible improvement in the efficiency of the algorithm.

References

Hoffstein, J., Pipher, J., & Silverman, J. H. (2014). An Introduction to Mathematical Cryptography. In Undergraduate texts in mathematics. https://doi.org/10.1007/978-1-4939-1711-2

Lenstra, A. K., Lenstra, H., & Lovász, L. (1982). Factoring polynomials with rational coefficients. Mathematische Annalen, 261(4), 515–534. https://doi.org/10.1007/bf01457454

Smeets, Ionica & Lenstra, Arjen & Lenstra, Hendrik & László, Lovász & van Emde Boas, Peter. (2009). The history of the LLL-algorithm. 10.1007/978-3-642-02295-1_1. (Picture)

Deng, X. (n.d.). An introduction to Lenstra-Lenstra-Lovasz Lattice Basis Reduction Algorithm. Massachusetts Institute of Technology. https://math.mit.edu/~apost/courses/18.204-2016/18.204_Xinyue_Deng_final_paper.pdf

Lienart, T. (2021, June 25). Gram-Schmidt Orthogonalisation. https://tlienart.github.io/posts/2021/06/25-gram-schmidt/#computational_complexity_of_gs

Galbraith, S. (2012). Lattice basis reduction. In Mathematics of Public Key Cryptography. Cambridge University Press. https://www.math.auckland.ac.nz/~sgal018/crypto-book/ch17.pdf

Slides: https://www.di.ens.fr/~pnguyen/SLIDES/SlidesLuminy2010.pdf

Micciancio, D. & Lenstra, Lenstra and Lovasz. (2012). CSE 206A: Lattice Algorithms and Applications Winter 2012. In UCSD CSE. https://cseweb.ucsd.edu/classes/wi12/cse206A-a/lec3.pdf

Phong Nguyen (2010). The LLL algorithm. https://www.di.ens.fr/~pnguyen/SLIDES/SlidesLuminy2010.pdf