THE LLL ALGORITHM

ADVAITH MOPURI

1. Introduction

A common problem in mathematics and computer science is to find the shortest nonzero vector in a given lattice. In fact, we will see many cases where this is required in a later section. The Lenstra-Lenstra-Lovàsz (LLL) algorithm gives an approximation (up to an exponential factor of the actual solution) of the shortest vector in a lattice by reducing a basis of the lattice into a shorter and almost orthogonal basis.

2. Short Vectors and Lattice Reduction

Before we begin with the actual algorithm, we first define what lattices are in the context of LLL.

Definition 2.1 (Lattice)**.** A lattice $\mathcal{L}$ is a subgroup of $\mathbb{R}^n$ generated by all integer linear combinations of the vectors in some basis B.

In other words, given a basis $B = (b_0, b_1, \ldots, b_n)$, the lattice $\mathcal{L}$ consists of all vectors which can be expressed as $\sum_{i=0}^{n} c_i b_i$ for constant integers c_i.

As we will be discussing LLL in the context of the shortest vector problem (SVP), it is important to note that there does exist a single shortest distance between any 2 points in the lattice.

One of the earliest results regarding SVP is Minkowski's theorem, which proves an upper bound on the length of the shortest nonzero vector in a given lattice. The theorem requires the determinant of a lattice, which is given by the determinant of the matrix formed by that lattice's basis vectors.

Theorem 2.2 (Minkowski's Theorem)**.** *We call a subset S of $\mathbb{R}^n$ convex if for any 2 vectors $x, y \in S$, $x - y \in S$.*

Consider a lattice $\mathcal{L}$ with determinant $\det(\mathcal{L})$ in $\mathbb{R}^n$ that is symmetric about the origin ($v \in \mathcal{L} \iff -v \in \mathcal{L}$). For a convex subset S of $\mathbb{R}^n$ with a volume greater than $2^n \det(\mathcal{L})$, there exists a nonzero lattice point in S.

Though Minkowski's theorem gives a nice upper bound on the size of the smallest vector in a lattice, it does not provide any information on how exactly to find such a vector. Currently, there has been no algorithm that *can* find such a vector which also runs in polynomial time – in fact, it has been conjectured that the problem is NP hard.

However, as mentioned previously, LLL gives a polynomial time algorithm to find close

Date: October 2023.

to the shortest vector in a lattice. The algorithm's key component is basis reduction, which is done via the Gram-Schmidt method.

3. Basis Reduction

Basis reduction is the procedure of reducing the size of a basis (the lengths of the vectors that make up the basis) while keeping the lattice generated by the basis the same. Two obvious methods for doing this are turning a vector b_i in the lattice into $-b_i$ and swapping 2 vectors in the basis. However, note that these adjustments do not shorten the vectors in the basis.

To do so, another common method is to subtract out linear combinations of all other vectors in the basis from the vector b_i. Clearly, this keeps the generated lattice the same while also (possibly) having the effect of shortening b_i. In fact, this observation is at the core of the Gram-Schmidt method which will be covered shortly.

Before we cover the more general basis reduction methods of Gram-Schmidt and LLL, we first solve the two dimensional case in order to gain a better understanding of what is happening in the more general n dimensional case.

It turns out that Gauss solved this case of the SVP. His algorithm is as follows:

(1) Start with a basis of vectors (b_0, b_1) in $\mathbb{R}^2$.
(2) If $|b_0| > |b_1|$, then swap the 2 vectors
(3) Compute the coefficient $u = \frac{b_0 \cdot b_1}{|b_0|^2}$
(4) If $u > \frac{1}{2}$, let $m = \lfloor u \rceil$, and let $b_1 = b_1 - mb_0$.
(5) If $|b_0| > |b_1|$, then swap the 2 vectors and go back to step 3. Otherwise, output b_0.

As an example, consider the basis consisting of the vectors $b_0 = (10, 0)$ and $b_1 = (3, 4)$. Since $|b_0| = 10 > 5 = |b_1|$, we swap the 2 vectors to turn the basis into $b_0 = (3, 4)$ and $b_1 = (10, 0)$. Then,

$$u = \frac{3 \cdot 10 + 4 \cdot 0}{5^2} = \frac{6}{5},$$

so b_1 gets shortened to

$$b_1 - \left\lfloor \frac{6}{5} \right\rceil \cdot b_0 = \left(\frac{32}{5}, \frac{-24}{5} \right).$$

The length of this shortened vector is 8, which is greater than the length of b_0, so we are done. We output $b_0 = (3, 4)$, which is the shortest vector in the lattice generated by the original basis, according to Gauss' algorithm.

Note that the coefficient u is called the orthogonal projection coefficient, and as the name suggests, is used to compute orthogonal projections of 2 vectors. It is crucial in the Gram-Schmidt process, as we will see shortly.

To understand why this algorithm works, we must define and prove a few more things. We begin by considering a $2D$ basis of vectors (b_0, b_1). It is clear that we only need 2 vectors for the basis as the space since it is only 2 dimensional. Such a basis is considered to be *reduced* if the following conditions hold:

Definition 3.1 (2D reduced basis)**.** A basis (b_0, b_1) in $\mathbb{R}^2$ is called *reduced* if

$$|b_0| \leq |b_1|$$

and

$$u = \frac{b_0 \cdot b_1}{|b_0|^2} \leq \frac{1}{2}.$$

It turns out that in the $2D$ case, if (b_0, b_1) is a reduced basis for a lattice $\mathcal{L}$, then b_0 itself is the shortest vector in the lattice. This observation is what Gauss used to create his algorithm – he simply starts from some random basis for lattice $\mathcal{L}$, and utilizes swaps (in order to ensure that the smallest vector is at the "front" of the basis) and subtractions to shorten the basis. The subtractions serve a similar purpose to the subtractions in the Euclidean algorithm. By subtracting out a well-chosen number of copies of b_0 from b_1, the algorithm effectively finds b_1 mod b_0, and the process continues as in the Euclidean algorithm.

Additionally, it is possible to bound the shortest vector in a $2D$ lattice. We prove the following theorem:

Theorem 3.2 (2D shortest vector)**.** *In a lattice $\mathcal{L}$ which is $2D$ (and cannot be expressed in $1D$), the length of the shortest vector $\lambda \in \mathcal{L}$ satisfies*

$$\lambda \leq \sqrt{\frac{2}{\sqrt{3}} \det(\mathcal{L})}.$$

Here, the condition about the dimension of $\mathcal{L}$ is equivalent to saying that the basis rank (number of independent vectors in a basis) of $\mathcal{L}$ is 2.

Proof. Consider a reduced basis (b_0, b_1). Let b_1^* be such that b_0 and b_1^* are orthogonal, and also such that $b_1 = b_1^* + ub_0$. Once again, the orthogonal projection coefficient u makes an appearance. Since the basis is reduced, we know that $u \leq \frac{1}{2}$.

Figure 1 provides a more pictorial representation of what is going on. We now have the following:

$$\begin{aligned}
& b_1 = b_1^* + ub_0 \\
\iff\ & |b_1|^2 = |b_1^* + ub_0| \\
\iff\ & |b_1|^2 = |b_1^*|^2 + u^2|b_0|^2 && \text{(since } b_0 \text{ and } b_1 \text{ are orthogonal)} \\
& \geq |b_0|^2 - \left(\frac{1}{2}\right)^2 |b_0|^2 = \frac{3}{4}|b_0|^2 \\
\iff\ & |b_1^*| \geq \frac{\sqrt{3}}{2}|b_1|.
\end{aligned}$$

Next, note that the determinant of the lattice $\mathcal{L}$ may be expressed in terms of the orthogonal vectors as

$$|b_1^*||b_0| \geq \frac{\sqrt{3}}{2}|b_0|^2.$$

Rearranging,

$$|b_0| \leq \sqrt{\frac{2}{\sqrt{3}} \det(\mathcal{L})}.$$

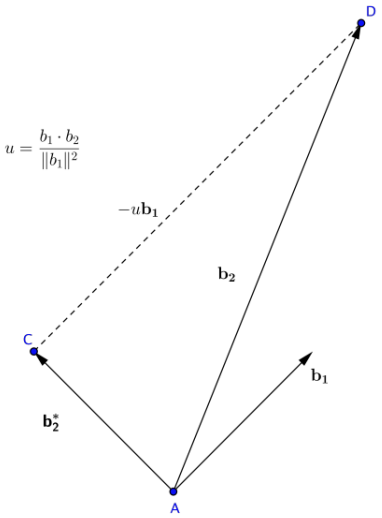

Figure 1. A 1-indexed version of the current scenario [1].

Since the smallest vector in the lattice is b_0 itself, we have $\lambda = b_0$, and we are done. ■

4. Gram-Schmidt

In the 2 dimensional case, Gauss's algorithm implements a basic version of the Gram-Schmidt method. In his algorithm, the generated basis was not always orthogonal, but via Gram-Schmidt, we can turn a basis B into a possibly different basis B^* that spans the same subspace as B while being comprised of pairwise orthogonal vectors. The algorithm requires the use of vector projection. We define $\text{proj}_v(w)$ as the orthogonal projection of w onto the line spanned by v.

Definition 4.1 (Gram-Schmidt Algorithm)**.** Given the basis $B = (b_0, \ldots, b_n)$, we define its Gram-Schmidt orthogonalized basis $B^* = (b_0^*, \ldots, b_n^*)$ as

$$b_k^* = b_k - \sum_{i=0}^{k-1} \text{proj}_{b_i^*}(b_k).$$

In other words,

$$\begin{aligned}
b_0^* &= b_0 \\
b_1^* &= b_1 - \text{proj}_{b_0^*}(b_1) \\
b_2^* &= b_2 - \text{proj}_{b_0^*}(b_2) - \text{proj}_{b_1^*}(b_2) \\
&\vdots \\
b_n^* &= b_n - \text{proj}_{b_0^*}(b_n) - \ldots \text{proj}_{b_{n-1}^*}(b_n).
\end{aligned}$$

5. The LLL Algorithm

The LLL algorithm makes use of the Gram-Schmidt algorithm and some of the ideas that have been mentioned throughout the paper in order to generate an almost-orthogonal basis of small vectors.

We begin by defining the basis $B = (b_0, b_1, \ldots, b_n)$, and its Gram-Schmidt basis as $B^* = b_0^*, b_1^*, \ldots, b_n^*$ obtained by applying the Gram-Schmidt process to B.

Our goal is to find a basis B that satisfies the following properties:

Definition 5.1 (LLL reduced basis)**.** We call a basis *LLL-reduced* if the following definition holds.

- (Size condition) For $1 \leq j < i \leq n$, the Gram-Schmidt coefficient $\mu_{i,j}$ satisfies $|\mu_{i,j}| \leq 0.5$.
- (Lovàsz condition) For $k = 1, 3, \ldots, n$, $\delta |b_{k-1}^*|^2 \leq |b_k^*|^2 + \mu_{k,k-1}^2 |b_{k-1}^*|^2$. The value of δ is taken to be $\frac{3}{4}$ by convention, though the algorithm will work for any $\delta \in \left(\frac{1}{4}, 1\right)$.

An algorithm to find such a reduced basis is the LLL algorithm, shown below [2]:

```
k = 1
while k ≤ n:
    for j from 0 to k − 1:
        if μ_{j,k} > 0.5:
            b_k = b_k − ⌊μ_{j,k}⌉ · b_j
            update Gram-Schmidt basis B*
    if Lovasz condition met by b_k:
        k = k + 1
    else:
        swap b_k, b_{k−1}
        update Gram-Schmidt basis B*
        k = max(k − 1, 1)
return B.
```

Just like in the $2D$ case, the shortest vector of an LLL reduced basis is the first one, b_0. We have that b_0 differs by an exponential factor from the actual shortest vector in a lattice, which is good enough for most applications. In fact, if λ is the true shortest vector in the lattice, then $|b_0| \leq 2^{n/2}\lambda$.

6. Use Cases

The algorithm has various applications in breaking cryptographic systems, factoring polynomials over the integers/rationals, and more. My favorite application of LLL is in integer relation algorithms.

For example, consider the number $r = 1.6180$. If r is a root of some unknown quadratic, then LLL can be used to determine a relatively "small" quadratic that contains a root approximating r.

More advanced uses of this technique, along with many of the previous applications of LLL, make this algorithm a powerful lattice reduction and shortest vector finding tool. Improvements continue to be made to this algorithm in order to tailor it for more specific situations, and as such, LLL continues to have a place in the scientific ecosystem of today.

References

[1] Xinyue Deng.
[2] Steven Schaefer. *LLL Algorithm*. YouTube, Dec 2020.