6.S897 Algebra and Computation March 7, 2012

Lecture 9

Lecturer: Eli Ben-Sasson *Scribe: Alan Guo*

Today, we will cover the **Lenstra-Lenstra-Lovasz (LLL) Algorithm**, which is one of the most important algorithms dealing with the geometry of numbers, and has applications to cryptanalysis, complexity, and number theory, in particular, to the factorization of polynomials in $\mathbb{Z}[x]$.

1 Lattices

Definition 1 *A **lattice** $L \subseteq \mathbb{R}^n$ is a discrete additive set. A set L is*

- ***discrete** if $\forall x \in L\ \exists \delta > 0$ such $B(x, \delta) \cap L = \{x\}$;*
- ***additive** if $x, y \in L \implies x + y, x - y \in L$.*

Definition 2 *A **basis** for L is a list $B = (b_1, \ldots, b_n) \in \mathbb{R}^{n\times n}$ of $\mathbb{R}$-linearly independent vectors such that $L = L(B) = \{B \cdot x \mid x \in \mathbb{Z}^n\}$. Typically $B \in \mathbb{Q}^{n\times n}$ or even $\mathbb{Z}^{n\times n}$. (We can define m-dimensional $L \subseteq \mathbb{R}^n$ for $m < n$, but we can reduce this to the full-dimensional case.)*

We will not prove the following theorem.

Theorem 1 *L is a lattice if and only if $L = L(B)$ for some basis B.*

An example of a non-lattice is the following. Consider the set generated by $\mathbb{Z}$-linear combinations of $b_1 = 1$ and $b_2 = \sqrt{2}$ in $\mathbb{R}$. One can show that this set is not discrete and hence not a lattice.

Problems related to lattices. An easy problem is the following: given x and L, is $x \in L$? This can be solved easily using linear algebra. Once we start asking about the geometry of our lattice, questions get harder. A hard question is: how close is x to L? Or, find the closest point in L to x. Another hard problem is the **smallest vector problem (SVP)**: given a basis B, find $v \in L(B) - \{0\}$ which minimizes $\|v\|_2$.

2 LLL Algorithm

The **LLL algorithm** is an approximation algorithm for the SVP. It finds $v' \in L(B) - \{0\}$ that satisfies

$$\|v'\|_2 \le 2^n \|v\|_2$$

in polynomial time, where v is the smallest vector. There is a randomized algorithm that approximates within a factor of $2^{n/\log n}$. Solving the SVP exactly is NP-hard. In fact, finding a $2^{\log^{1-\varepsilon} n}$-approximation is NP-hard. The problem of $\sqrt{n}$-gap SVP (a promise problem where there is a "gap" between two given lattices, and you must distinguish between the two) is in the intersection **NP** $\cap$ co**NP**, which suggests that it may be easier than the exact SVP problem. Many cryptography applications assume that n^c-approximation for the gap-SVP requires super-polynomial time, for $c > 1$ (say, $c = 10$).

2.1 Application: Factoring polynomials

Before we go into how the algorithm works, we show an application. Recall the problem from last time.

- **Input:** $g \in \mathbb{Z}[x]$, a degree parameter d, coefficient bound N, modulus M
- **Output:** $f \in \mathbb{Z}[x]$ satisfying $\deg f \leq d$, $|\text{coeffs}(f)| \leq N$ and there exists $h \in \mathbb{Z}[x]$ such that $f = g \cdot h \pmod{M}$.

Claim 2 *Using the LLL algorithm, we can either find f if there exists such an f' with $|coeffs(f')| \leq \frac{N}{(d+1)2^{d+1}}$.*

Proof We go back and forth between polynomials and a lattice. We encode polynomials as vectors. In particular, $g = \sum_{i=0}^{k} g_i x^i$ gets encoded as the vector $v_1 = (g_0, g_1, \ldots, g_k, 0, \ldots, 0) \in \mathbb{Z}^{d+1}$. We encode multiples of g by powers of x as

$$\begin{aligned} v_2 &= (0, g_0, g_1, \ldots, g_k, 0, \ldots, 0) \\ v_3 &= (0, 0, g_0, g_1, \ldots, g_k, 0, \ldots, 0) \\ &\vdots \\ v_{d-k+1} &= (0, \ldots, 0, g_0, g_1, \ldots, g_k). \end{aligned}$$

Let $e_1, \ldots, e_{d+1}$ be the standard basis vectors, i.e. e_i has 1 in the i-th coordinate and 0 elsewhere. Then the lattice we want is generated by the vectors $(v_1, \ldots, v_{d-k+1}, Me_1, \ldots, Me_{d+1})$. Suppose we have a basis B for this lattice. Then we can feed this to the LLL algorithm and get a 2^{d+1}-approximation of the SVP, which will give us the desired f.

First we show how to get the basis B. Starting with the $(2d-k+1) \times (d+1)$ matrix B_0 whose rows are the vectors $(v_1, \ldots, v_{d-k+1}, Me_1, \ldots, Me_{d+1})$, we can convert this into a matrix B_1 using row swaps and additions so that the first row of B_1 has first entry equal to the gcd of the first column of B_0, and zeros in all entries below. Continuing in this way, we get a matrix whose first $d+1$ rows form a upper triangular matrix, which is the basis B we want, and zeros everywhere else.

Next we show that the LLL algorithm gives us the f we want. We will abuse notation and write f to denote both the polynomial as well as the vector. Clearly the f returned by LLL has degree at most d. Moreover, there exists h such that $f = gh \pmod{M}$. This is simply because the resulting f lies in the $\mathbb{Z}$-span of $g, xg, x^2 g, \ldots, x^{d-k} g, M, Mx, Mx^2, \ldots, Mx^d$ and the corresponding coefficients give us h. It remains to show that the coefficients of f are bounded by N; equivalently, $\|f\|_\infty \leq N$. Suppose there exists $f' \in L$ with $\|f\|_\infty \leq \frac{N}{(d+1)2^{d+1}}$. Then the LLL algorithm guarantees that

$$\|f\|_2 \leq 2^{d+1}\|f'\|_2 \leq 2^{d+1}\|f'\|_1 \leq (d+1)2^{d+1}\|f'\|_\infty \leq N$$

and furthermore $\|f\|_\infty \leq \|f\|_2$, which completes the proof.

■

2.2 The algorithm

We define an important invariant of lattices.

Definition 3 *The* ***determinant*** *(or index) of a lattice L is $|\det(B)|$ for any basis B of L.*

This is well-defined since changing basis requires elementary row operations on the matrix B, the composition of which is a unimodular matrix with determinant ± 1.

Before we cover the actual LLL algorithm, we show Gauss' algorithm for solving the SVP in dimension 2.

Gauss' Algorithm. **Input:** $a, b \in \mathbb{Z}^2$

1. $i \leftarrow \arg\min_j \|b - ja\|$
2. $b \leftarrow b - ia$
3. If $\|b\| \leq \frac{1}{\sqrt{3}}\|a\|$, swap a and b and go to 1, else output $\arg\min\{\|a\|, \|b\|\}$.

The running time is polynomial in the bit length of (a, b) since each swap reduces the sum of the bit lengths of a and b by a constant. Now we show that the algorithm gives us the correct answer.
Proof Let $v = ia + jb$ be the smallest vector. Write $b = b^* + \alpha \cdot a$, for $\alpha \in \mathbb{R}$, where $b^* \perp a$ and $|\alpha| < \frac{1}{2}$. Then

$$\|v\|^2 = j^2\|b^*\|^2 + (i + \alpha j)^2\|a\|^2 \geq j^2\|b^*\|^2$$

and so $\|v\| \geq j\|b^*\|$. We claim that $\|b^*\| > \frac{1}{2}\|b\|$. This implies that $\|v\| > \frac{j}{2}\|b\|$ which implies that $j < 2$, so $j \in \{0, 1\}$ If $j = 0$, then $i = 1$ and $v = a$, otherwise $j = 1$ and $i = 0$ so $v = b$. Now we prove the claim. We have that $\|b\| > \frac{1}{\sqrt{3}}\|a\|$ and $\|b\|^2 = \|b^*\|^2 + \alpha^2\|a\|^2$ with $|\alpha| < \frac{1}{2}$. Therefore,

$$\begin{aligned} \|b^*\|^2 &= \|b\|^2 - \alpha^2\|a\|^2 \\ &> \|b\|^2 - 3\alpha^2\|b\|^2 \\ &> \frac{1}{4}\|b\|^2. \end{aligned}$$

■

The LLL algorithm is similar to Gauss' algorithm. The motivation is to define b_i^* to be b_i minus the projection of b_i to $\text{span}(b_1, \ldots, b_{i-1})$, so $\prod_i b_i^* = \det(B)$. Let $\mu_{ij} \in \mathbb{R}$ be the coefficients such that $b_i = \sum_{j \leq i} \mu_{ij} b_j^*$, $\mu_{ii} = 1$.

LLL Algorithm.

1. "Orthogonalize": make sure $|\mu_{ij}| \leq \frac{1}{2}$, for $j < i$ (takes time $\binom{n}{2}$)
2. "Swap": if there is i such that swapping b_i and b_{i+1} reduces b_i^* by a $\frac{3}{4}$ factor, then swap, else return b_1.

Claim 3 $\|b_1\| \leq 2^n \min_j \|b_j^*\|$, *and therefore* $\|b_1\| \leq 2^n\|SVP\|$ *since* $\|SVP\| \geq \min_j \|b_j^*\|$.

Proof Since $\|b_{i+1}^*\| \geq \frac{1}{2}\|b_i^*\|$, by induction $\|b_1\| = \|b_1^*\| \leq 2^n\|b_i\|$ for any i. ■

Claim 4 *The running time of LLL is polynomial in n and the bit lengths of the b_i.*

Proof (Sketch) Define

$$\phi_t = \sum_{i=1}^{n}(n + 1 - i)\log\|b_1^*\| \text{ after } t \text{ swaps}$$

and it happens that $\phi_{t+1} \leq \phi_t - C$ for some constant $C > 0$. Moreover, $\phi_0 \leq \text{poly}(n, \text{bit length of input})$
■