18.409 An Algorithmist's Toolkit Nov. 19, 2009

Lecture 20

Lecturer: Jonathan Kelner

1 Brief Review of Gram-Schmidt and Gauss's Algorithm

Our main task of this lecture is to show a polynomial time algorithm which approximately solves the Shortest Vector Problem (SVP) within a factor of $2^{O(n)}$ for lattices of dimension n. It may seem that such an algorithm with exponential error bound is either obvious or useless. However, the algorithm of Lenstra, Lenstra and Lovász (LLL) is widely regarded as one of the most beautiful algorithms and is strong enough to give some extremely striking results in both theory and practice.

Recall that given a basis $b_1, \ldots, b_n$ for a vector space (no lattices here yet), we can use the Gram-Schmidt process to construct an orthogonal basis $b_1^*, \ldots, b_n^*$ such that $b_1^* = b_1$ and
$b_k^* = b_k -$ [projection of b_k onto $\text{span}(b_1, \ldots, b_{k-1})$] for all $2 \le k \le n$ (note that we do not normalize b_k^*). In particular, we have that for all k:

- $\text{span}(b_1, \ldots, b_k) = \text{span}(b_1^*, \ldots, b_k^*)$,
- $b_k = \sum_{i=1}^{k} \mu_{ki} b_i^*$, and
- $\mu_{kk} = 1$.

The above conditions can be rewritten as $B = MB^*$, where basis vectors are rows of B and B^*, and

$$M = \begin{bmatrix} \mu_{11} & 0 & 0 & \ldots & 0 \\ \mu_{21} & \mu_{22} & 0 & \ldots & 0 \\ \vdots & & \ddots & & \\ \mu_{n1} & \mu_{n2} & \mu_{n3} & \ldots & \mu_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ \mu_{21} & 1 & 0 & \ldots & 0 \\ \vdots & & \ddots & & \\ \mu_{n1} & \mu_{n2} & \mu_{n3} & \ldots & 1 \end{bmatrix}.$$

Obviously $\det(M) = 1$, and thus $\text{vol}(B) = \text{vol}(B^*)$. However, the entries of M are not integers, and thus $L(B) \neq L(B^*)$. We have proved last time that

$$\textit{for any } b \in L,\ ||b|| \ge \min_i\{||b_i^*||\}.$$

We'll use this to prove useful bound for the shortest vector on lattice.

Recall also that last time we saw the Gauss's algorithm which solves SVP for $d = 2$. There are two key ingredients of the algorithm. The first is a definition of "reduced basis" which characterizes the discrete version of bases being orthogonal: namely,

a basis $\{u, v\}$ for a 2-d lattices is said to be *reduced*, if $|u| \le |v|$ and $|u \cdot v| \le \frac{|u|^2}{2}$.

The second is an efficient procedure that produces a reduced basis. The procedure consists of two stages: First is a Euclid-like process which subtracts a multiple of the shorter vector from the longer one to get a vector as short as possible. The second stage is, if the length ordering is broken, we swap the two vectors and repeat, otherwise (i.e., $|u| \le |v|$) the procedure ends. To make the above procedure obviously terminate in polynomial time, we change the termination criterion to be $(1 - \epsilon)|u| \le |v|$. This only gives us a $(1 - \epsilon)$-approximation, but is good enough. The basic idea of LLL algorithm is to generalize Gauss's algorithm to higher dimensions.

2 LLL Algorithm

2.1 Reduced Basis

In order to find a short vector in the lattice, we would like to perform a discrete version of GS procedure. To this end, we need to formalize the notion of being orthogonal in lattice problems. One way to do this is to say that the result of our procedure is "almost orthogonalized" so that doing Gram-Schmidt does not change much.

Definition 1 (Reduced Basis) *Let $\{b_1, \ldots, b_n\}$ be a basis for a lattice L and let M be its GS matrix defined in Section 1. $\{b_1, \ldots, b_n\}$ is a reduced basis if it meets the following two conditions:*

- *Condition 1: all the non-diagonal entries of M satisfy $|\mu_{ik}| \leq 1/2$.*
- *Condition 2: for each i, $||\pi_{S_i} b_i||^2 \leq \frac{4}{3}||\pi_{S_i} b_{i+1}||^2$, where S_i is the orthogonal complement of (i.e., the subspace orthogonal to) span$(b_1, \ldots, b_{i-1})$, and π_{S_i} is the projection operator to S_i.*

Remark The constant $4/3$ here is to guarantee polynomial-time termination of the algorithm, but the choice of the exact value is somewhat arbitrary. In fact, any number in $(1, 4)$ will do.

Remark Condition 2 is equivalent to $||b^*_{i+1} + \mu_{i+1,i} b^*_i||^2 \geq \frac{3}{4}||b^*_i||^2$ and one may think it as requiring that the projections of any two successive basis vectors b_i and b_{i+1} onto S_i satisfy a gapped norm ordering condition, analogous to what we did in Gauss's algorithm for 2D case.

2.2 The algorithm

Given $\{b_1, \ldots, b_n\}$, the LLL algorithm works as below.

```
LLL Algorithm for SVP

Repeat the following two steps until we have a reduced basis

Step 1: Gauss Reduction

    Compute the GS matrix $M$
    for $i = 1$ to $n$
        for $k = i - 1$ to 1
            $m \leftarrow$ nearest integer to $\mu_{ik}$
            $b_i \leftarrow b_i - m b_k$
        end
    end

Step 2: Swapping

    if exists $i$ s.t. $||\pi_{S_i} b_i||^2 > \frac{4}{3}||\pi_{S_i} b_{i+1}||^2$
        then swap $b_i$ and $b_{i+1}$
             go to Step 1
```

3 Analysis of LLL Algorithm

The LLL algorithm looks pretty intuitive, but it is not obvious at all that it converges in polynomial number of steps or gives a good answer to SVP. We'll see that it indeed works.

3.1 LLL produces a short vector

We first show that reduced basis gives a short vector.

Claim 2 *If $b_1, \ldots, b_n$ is a reduced basis, then $||b_1|| \le 2^{\frac{n-1}{2}} \lambda_1(L)$.*

Proof Note that

$$\begin{aligned}||b_i^*||^2 = ||\pi_{S_i} b_i||^2 &\le \frac{4}{3}||\pi_{S_i} b_{i+1}||^2 \\ &= \frac{4}{3}||b_{i+1}^* + \mu_{i+1,i} b_i^*||^2 = \frac{4}{3}||b_{i+1}^*||^2 + \frac{4}{3}\mu_{i+1,i}^2 ||b_i^*||^2 \\ &\le \frac{4}{3}||b_{i+1}^*||^2 + \frac{1}{3}||b_i^*||^2,\end{aligned}$$

which gives $||b_{i+1}^*||^2 \ge \frac{1}{2}||b_i^*||^2$. By induction on i, we have

$$||b_i^*||^2 \ge \frac{1}{2^{i-1}}||b_1^*||^2 = \frac{1}{2^{i-1}}||b_1||^2.$$

Recall that $\forall b \in L$, $||b|| \ge \min_i ||b_i^*||$. Therefore $\lambda_1(L) \ge \min_i ||b_i^*||$, which combined with the inequality above yields

$$||b_1||^2 \le \min_i \{2^{i-1}||b_i^*||^2\} \le 2^{n-1} \min_i \{||b_i^*||^2\} \le 2^{n-1}\lambda_1(L)^2$$

as desired. ■

3.2 Convergence of LLL

Now we show that the LLL algorithm terminates in polynomial time. Note that in each iteration of LLL, Step 1 takes polynomial time and Step 2 takes $O(1)$ times. What we need to show is that we only need to repeat Step 1 and Step 2 a polynomial number of times. To this end, we define a potential function as follows:

$$D(b_1, \ldots, b_n) = \prod_{i=1}^{n} ||b_i^*||^{n-i}.$$

It is clear that Step 1 does not change D since we do not change the Gram-Schmidt basis.

We are going to show that each iteration of Step 2 decreases D by a constant factor. In Step 2, we swap i and $i+1$ only when $||b_i^*||^2 > 4/3||\pi_{S_i} b_{i+1}||^2 \ge 4/3||b_{i+1}^*||^2$. Therefore each swapping decreases D by a factor of at least $2/\sqrt{3}$, as desired.

It is left to show that D can be upper- and lower-bounded. Since $||b_i^*|| \le ||b_i||$, the initial value of D can be upper bounded by $(\max_i ||b_i||)^{n(n-1)/2}$. On the other hand, we may rewrite D as $\prod_{i=1}^n |\det(\Lambda_i)|$, where Λ_i is the lattice spanned by $b_1, \ldots, b_i$. Since we assume that the lattice basis vectors are integer-valued, so D is at least 1.

In sum, the algorithm must terminate in $\log_{2/\sqrt{3}}(\max_i ||b_i||)^{n(n-1)/2} = \text{poly}(n)$ iterations.

4 Application of LLL–Lenstra's Algorithm for Integer Programming

4.1 Applications of LLL

LLL algorithm has many important applications in various fields of computer science. Here are a few (many taken from Regev's notes):

1. Solve integer programming in bounded dimension as we are going to see next.

2. Factor polynomials over the integers or rationals. Note that this problem is harder than the same task but over reals, e.g. it needs to distinguish $x^2 - 1$ from $x^2 - 2$.

3. Given an approximation of an algebraic number, find its minimal polynomial. For example, given 0.645751 outputs $x^2 + 4x - 3$.

4. Find integer relations among a set of numbers. A set of real numbers $\{x_1, \ldots, x_n\}$ is said to have an integer relation if there exists a set of integers $\{a_1, \ldots, a_n\}$ not identically zero such that $a_1x_1 + \cdots + a_nx_n = 0$. As an example, if we are given $\arctan(1), \arctan(1/5)$ and $\arctan(1/239)$, we should output $\arctan(1) - 4\arctan(1/5) + \arctan(1/239) = 0$. How would you find this just given these numbers as decimals?

5. Approximate to SVP, CVP and some other lattice problems.

6. Break a whole bunch of cryptosystems. For example, RSA with low public exponent and many knapsack based cryptographic systems.

7. Build real life algorithms for some NP-hard problems, e.g. subset sum problem.

4.2 Integer Programming in Bounded Dimension

4.2.1 Linear, Convex and Integer Programming

Consider the following feasibility version of the linear programming problem:

- Linear Programming (feasibility)

  Given: An $m \times n$ matrix A and a vector $b \in \mathbb{R}^n$

  Goal: Find a point $x \in \mathbb{R}^n$ s.t. $Ax \leq b$, or determine (with a certificate) that none exists

One can show that other versions, such as the optimization version, are equivalent to feasibility version.
If we relax the searching regions from polytopes to convex bodies, we get convex programming.

- Convex Programming (feasibility)

  Given: A separation oracle for a convex body K and a promise that

  - K is contained in a ball of singly exponential radius R
  - if K is non-empty, it contains a ball of radius r which is at least 1/(singly exponential)

  Goal: Find a point $x \in \mathbb{R}^n$ that belongs to K, or determine (with a certificate) that none exists

Integer programming is the same thing as above, except that we require the program to produce a point in $\mathbb{Z}^n$, not just $\mathbb{R}^n$. Although linear programming and convex programming are known to be in **P**, integer programming is a well-known NP-complete problem.

4.2.2 Lenstra's algorithm

Theorem 3 (Lenstra) *If our polytope/convex body is in $\mathbb{R}^n$ for any constant n, then there exists a polynomial time algorithm for integer programming.*

Remark.

- For linear programming (LP), the running time of the algorithm will grow exponentially in n, but polynomially in m (the number of constrains) and the number of bits in the inputs.
- For convex programming, the running time is polynomial in $\log(R/r)$.
- As before, we could also ask for maximum of $c \cdot x$ over all $x \in K \cap Z^n$, which is equivalent to the feasibility problem, as we can do a binary search on the whole range of $c \cdot x$.

The main idea of Lenstra's algorithm is the following. The main difficulty of integer programming comes from the fact that K may not be well-rounded, therefore it could be exponentially large but still contain no integral point, as illustrated in the following figure:

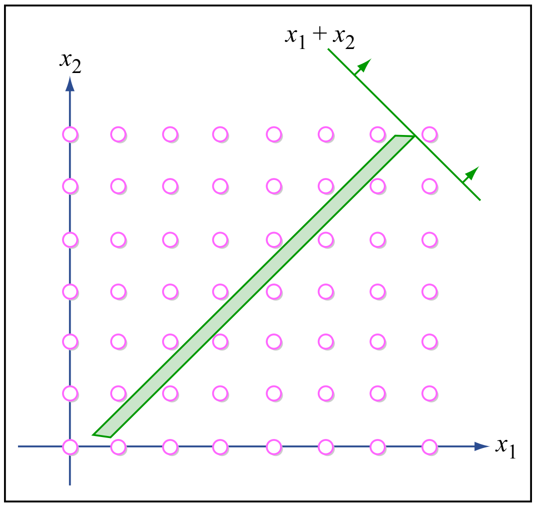

Figure by MIT OpenCourseWare.

Figure 1: A not-well-rounded convex body

Our first step is thus to change the basis so that K is well-rounded, i.e., K contains a ball of radius 1 and is contained in a ball of radius $c(n)$ for some function that depends only on n. Such a transformation will sends $\mathbb{Z}^n$ to some lattice L. Now our convex body is well-rounded but the basis of lattice L may be ill-conditioned, as shown in the following figure:

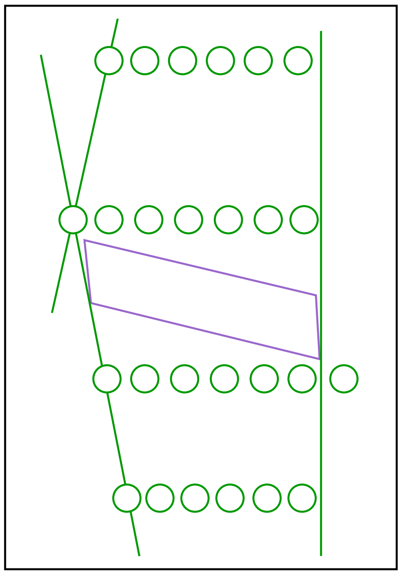

Figure by MIT OpenCourseWare.

Figure 2: A well-rounded convex body and an ill-conditioned lattice basis

It turns out that the lattice points are still well-separated and we can remedy the lattice basis by a basis reduction procedure of LLL (i.e., discrete Gram-Schmidt). Finally we chop the lattice space up in some intelligent way and search for lattice points in K.

Note that in the first step of Lenstra's algorithm, what we need is an algorithmic version of Fritz John's theorem. As we saw in the problem set, there is an efficient algorithm which, for any convex body K specified by a separation oracle, constructs an ellipsoid E such that

$$E(P') \subseteq K \subseteq O(n^{3/2})E(P').$$

Next let $T : \mathbb{R}^n \to \mathbb{R}^n$ be the linear transformation such that $E(P')$ is transformed to $\mathbf{B}(P, 1)$. Now K is sandwiched between two reasonably-sized balls:

$$\mathbf{B}(P, 1) \subseteq TK \subseteq \mathbf{B}(P, R),$$

where $R = O(n^{3/2})$ is the radius of the outer ball.

Let $L = T\mathbb{Z}^n$ with basis $Te_1, \ldots, Te_n$. Our goal is to find a point (if it exists) in $TK \cap T\mathbb{Z}^n = TK \cap L$. Our next step is to apply the basis reduction in LLL algorithm. We will need the following two lemmas in analyzing Lenstra's algorithm. The proofs of the lemmas are left as exercises.

Lemma 4 *Let $b_1, \ldots, b_n$ be any basis for L with $||b_1||^2 \leq \cdots \leq ||b_n||^2$. Then for every $x \in \mathbb{R}^n$, there exists a lattice point y such that*

$$\begin{aligned} ||x - y||^2 &\leq \frac{1}{4}(||b_1||^2 + \cdots + ||b_n||^2) \\ &\leq \frac{1}{4}n||b_n||^2. \end{aligned}$$

Lemma 5 *For a reduced basis $b_1, \ldots, b_n$ ordered as above,*

$$\prod_{i=1}^{n} ||b_i|| \leq 2^{n(n-1)/4} det(L).$$

Consequently, if we let $H = span(b_1, \ldots, b_{n-1})$, then

$$2^{-n(n-1)/4}||b_n|| \leq dist(H, b_n) \leq ||b_n||.$$

Let $b_1, \ldots, b_n$ be a reduced basis for L. Applying Lemma 4 gives us a point $y \in L$ such that $||y - P|| \leq \frac{1}{2}\sqrt{n}||b_n||$.

- case 1: $y \in TK$. We find a point in $TK \cap L$.
- case 2: $y \notin TK$, hence $y \notin \mathbf{B}(P, 1)$. Consequently, $||y - P|| \geq 1$ and $||b_n|| \geq \frac{2}{\sqrt{n}}$.

This means that the length of b_n is not much smaller than R. In the following we partition L along the sublattice "orthogonal" to b_n and then apply this process recursively.

Let L' be the lattice spanned by $b_1, \ldots, b_{n-1}$ and let $\mathcal{L}_i = L' + ib_n$ for each $i \in \mathbb{Z}$. Clearly $L = \bigcup_{i\in\mathbb{Z}} \mathcal{L}_i$. From Lemma 5 the distance between two adjacent hyperplanes is at least

$$\begin{aligned} \text{dist}(b_n, \text{span}(b_1, \ldots, b_{n-1})) &\geq 2^{-n(n-1)/4}||b_n|| \\ &\geq \frac{2}{\sqrt{n}}2^{-n(n-1)/4}||b_n|| = c_1(n), \end{aligned}$$

where $c_1(n)$ is some function that depends only on n. This implies that the convex body TK can not intersect with too many hyperplanes. That is

$$|\{i \in \mathbb{Z} : \mathcal{L}_i \cap \mathbf{B}(P, R) \neq \emptyset\}| \leq 2R/c_1(n) = c_2(n)$$

for some function $c_2(n)$ that depends only on n. Now we have reduced our original searching problem in n-dimensional space to $c_2(n)$ instances of searching problems in $(n - 1)$-dimensional space. Therefore we can apply this process recursively and the total running time will be a polynomial in the input size times a function that depends only on n.