6.876 Advanced Topics in Cryptography: Lattices September 16, 2015

Lecture 3

Lecturer: Vinod Vaikuntanathan *Scribe: Itay Berman*

In the first two lectures we introduces lattices and covered the following topics:

- Minkowski's first and second convex body theorems.
- Gram-Schmidt Orthogonalization and used it to lower-bound the length of the shortest non-zero vector in a lattice ($\lambda_1 \geq \min_i \|\mathbf{b}_i\|$).
- Proof for the four squares theorem using lattices.

In the next few lectures we will cover:

- Computational problem with lattices (SVP, CVP, SIVP).
- Overview of the complexity landscape.
- The LLL algorithm, which finds an approximation to the shortest vector. We will then use the LLL algorithm to find the shortest vector exactly (while increasing the running time), and to find an approximation to the closest vector problem.

1 Computational Problems

We will focus on three computational problems regarding lattices:

- The Shortest Vector Problem (SVP): find the shortest non-zero vector in the lattice.
- The closest Vector Problem (CVP): find the closest lattice vector to a given vector.
- The Shortest Independent Vectors Problem (SIVP): find n independent and "short" vectors.

We define search, approximation and decision variants of the above problems. Recall that $\lambda_1 = \lambda_1(\mathbf{B})$ is the first successive minima of $\mathcal{L}(\mathbf{B})$ (i.e., is the length of the shortest non-zero vector in $\mathcal{L}(\mathbf{B})$), that λ_n is the n'th successive minima of $\mathcal{L}(\mathbf{B})$ and that $\text{dist}(t, \mathcal{L}(\mathbf{B})) = \min_{u \in \mathcal{L}(\mathbf{B})} \|u - t\|$.

Search	Approximation for $\gamma > 1$	Decision (Promise[1]) for $\gamma > 1$
SVP: given a basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, find $u \in \mathcal{L}(\mathbf{B}) \setminus \{0\}$ s.t. $\|u\| = \lambda_1$.	SVP_γ**:** given a basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, find $u \in \mathcal{L}(\mathbf{B}) \setminus \{0\}$ s.t. $\|u\| \leq \gamma \cdot \lambda_1$.	gapSVP_γ**:** $YES = \{(\mathbf{B}, s) \colon \lambda_1(\mathbf{B}) \leq s\}$, $NO = \{(\mathbf{B}, s) \colon \lambda_1(\mathbf{B}) > \gamma \cdot s\}$
CVP: given a basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$ and $t \in \mathbb{Z}^n$, find $u \in \mathcal{L}(\mathbf{B})$ s.t. $\|u - t\| = \text{dist}(\mathcal{L}(\mathbf{B}), t)$.	CVP_γ**:** given a basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$ and $t \in \mathbb{Z}^n$, find $u \in \mathcal{L}(\mathbf{B})$ s.t. $\|u - t\| \leq \gamma \cdot \text{dist}(\mathcal{L}(\mathbf{B}), t)$.	gapCVP_γ**:** $YES = \{(\mathbf{B}, t, s) \colon \text{dist}(\mathcal{L}(\mathbf{B}), t) \leq s\}$, $NO = \{(\mathbf{B}, t, s) \colon \text{dist}(\mathcal{L}(\mathbf{B}), t) > \gamma \cdot s\}$
SIVP: given a basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, find independent vectors $u_1, \ldots, u_n$ s.t. $\|u_i\| \leq \lambda_n$ for $i \in [n]$.	N/A	N/A

[1] A promise problem is such that the input domain is $YES \cup NO$. An algorithm decides a promise problem if it accepts inputs from YES and rejects inputs from NO.

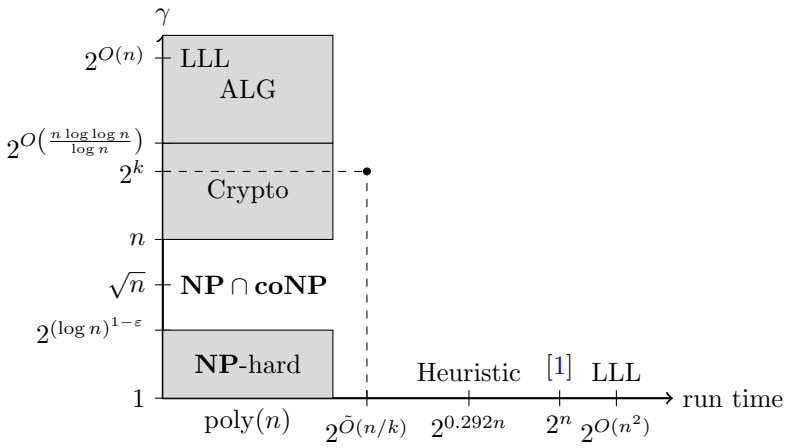

Figure 1: The complexity landscape of SVP_γ.

2 Complexity Landscape

We present the landscape for the Shortest Vector Problem. The landscape for the Closest Vector Problem is very similar. In the following we assume we are given a lattice basis in $\mathbb{Z}^{n \times n}$. The running times of the described algorithms will be a function of n, and we ignore polynomial factors in the length of the bit representation of the given basis.

Algorithms for SVP_γ: The first algorithm to solve the SVP_γ was the LLL algorithm [2]. It gave a $2^{O(n)}$-approximation and its running time is $\mathrm{poly}(n)$. The best know approximation factor achieved by a polynomial time algorithm is $2^{O(\frac{n \log\log n}{\log n})}$.

If we want to solve the exact SVP (i.e., $\gamma = 1$), then the LLL algorithm can do so with running time of $2^{O(n^2)}$. The fastest known algorithm to solve the exact SVP was recently given in [1], and its running time is 2^n. There are also Heuristics algorithms with running time $2^{0.292n}$.

Finally, One can have a trade off between the approximation factor and the running time — achieving 2^k-approximation can be done in $2^{\tilde{O}(n/k)}$ time.

Hardness of SVP_γ: It is no surprise that we don't know of a polynomial time algorithm to solve the exact SVP, since it was shown to be **NP**-hard. In fact, even achieving $2^{(\log n)^{1-\varepsilon}}$-approximation is **NP**-hard. On the other hand, it was shown that $\mathsf{SVP}_{\sqrt{n}}$ is in $\mathbf{NP} \cap \mathbf{coNP}$, and thus unlikely to be **NP**-hard.

Cryptography from SVP_γ: The smallest approximation factor from which we know how to build cryptographic primitives is $\gamma = n$. Since $\mathsf{SVP}_{\sqrt{n}} \in \mathbf{NP} \cap \mathbf{coNP}$, this will likely not suffice to base cryptography on **NP**-hardness.

A pictorial presentation of the above description if given in Figuire 1. Throughout this course we'll cover all the aforementioned results, starting with the LLL algorithm. However, before presenting the LLL algorithm, we warm up by reducing the Shortest Vector Problem to the Closest Vector Problem.

Theorem 1. *For any $\gamma > 1$, given access to CVP_γ oracle, there exists a polynomial time algorithm that solves SVP_γ.*

Proof. We prove for the case $\gamma = 1$, and the proof generalizes to $\gamma > 1$ naturally. For a basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ and $i \in [n]$, let $\mathbf{B}_i = [\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, 2\mathbf{b}_i, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_n]$. The polynomial time algorithm that solves SVP sets $v_i := \mathsf{CVP}(\mathbf{B}_i, \mathbf{b}_i)$ for every $i \in [n]$, and return $v_{i^*} - \mathbf{b}_{i^*}$ such that $i^* = \arg\min_i \|v_i - \mathbf{b}_i\|$.

Let $v = \sum_{i=1}^n c_i \mathbf{b}_i$ be the shortest vector in $\mathcal{L}(\mathbf{B})$. We show that the algorithm returns v. The correctness of the algorithm follows from the next claim.

Claim 2. *Assume that c_j is odd, then $v + \mathbf{b}_j$ is the closest vector to $\mathbf{b}_j$ in $\mathcal{L}(\mathbf{B}_j)$.*

Proof. First, it is easy to see that $v + \mathbf{b}_j \in \mathcal{L}(\mathbf{B}_j)$. Assume towards a contradiction that there exists $v' \in \mathcal{L}(\mathbf{B}_j)$ such that $\|v' - \mathbf{b}_j\| < \|(v + \mathbf{b}_j) - \mathbf{b}_j\| = \|v\|$. But, it is easy to see that $v' - \mathbf{b}_j$ belongs to $\mathcal{L}(\mathbf{B})$, a contradiction to the fact that v is the shortest vector in $\mathcal{L}(\mathbf{B})$. □

Since at least one of the c_i's is odd (as otherwise $v/2$ would be a shorter lattice vector), say c_j, the algorithm will set $v_j = v + \mathbf{b}_j$, and will return v. □

3 The LLL Algorithm

We begin describing the algorithm of Lenstra, Lenstra and Lovász for approximating the shortest vector of a lattice.

Theorem 3. *Given a basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$ there is a $\mathrm{poly}(n, W_{\mathbf{B}})$-time algorithm for $\mathsf{SVP}_{2^{O(n)}}$, where $W_{\mathbf{B}}$ is the length of the bit representation of $\mathbf{B}$; namely, the algorithm returns a vector $v \in \mathcal{L}(\mathbf{B})$ such that $\|v\| \leq 2^{O(n)} \cdot \lambda_1(\mathbf{B})$.*

The LLL algorithm actually transforms, in polynomial time, the given basis into a "LLL-reduced" basis for the same lattice. The above theorem holds since a LLL-reduced basis has an important property — its shortest vector is a $2^{O(n)}$-approximation for the shortest vector in entire the lattice. In this lecture we'll give define a LLL-reduced basis and give some intuition for this definition; in the next lecture we'll describe and analyze the LLL algorithm itself.

3.1 LLL-reduced Basis

Our goal is to transform the given basis to one with "short" vectors. Consider the case $n = 2$, i.e., $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2]$. Our starting point is the Gram-Schmidt orthogonalization process in which we set $\tilde{\mathbf{b}}_1 := \mathbf{b}_1$ and $\tilde{\mathbf{b}}_2 := \mathbf{b}_2 - \mu_{21}\tilde{\mathbf{b}}_1$, where $\mu_{21} = \langle \mathbf{b}_2, \tilde{\mathbf{b}}_1 \rangle / \langle \tilde{\mathbf{b}}_1, \tilde{\mathbf{b}}_1 \rangle$. Intuitively, $\tilde{\mathbf{b}}_2$ is the shortest vector we can hope for, since we removed all $\tilde{\mathbf{b}}_1$'s components from it (this fact is what make the Gram-Schmidt orthogonal). However, $\tilde{\mathbf{b}}_2 \notin \mathcal{L}(\mathbf{B})$, and thus cannot be in a basis of $\mathcal{L}(\mathbf{B})$. To fix this issue, we transform $\mathbf{B}$ into the following basis: $\mathbf{b}'_1 = \mathbf{b}$ and $\mathbf{b}'_2 = \mathbf{b}_2 - \lfloor \mu_{21} \rceil \mathbf{b}'_1$, where $\lfloor \cdot \rceil$ means rounding to the closest integer. $\mathbf{b}'_2$ is the shortest *lattice* vector we can hope for, as we removed all the *integer* components of $\tilde{\mathbf{b}}_1$. Note that when projecting $\mathbf{b}'_2$ to the line generated by $\mathbf{b}'_1$, then this projection is between $-\mathbf{b}'_1/2$ to $\mathbf{b}'_1/2$. The latter fact also guarantees that the resulting basis is "close" to orthogonal — the angle between $\mathbf{b}'_1$ to $\mathbf{b}'_2$ is at least 60 degrees. See Figure 2 for an example of this transformation.

So far we reduced $\mathbf{b}_2$, but left $\mathbf{b}_1$ as is. But what if $\mathbf{b}_1$ is very long to begin with? there is no guarantee that the reduced basis is short. At this point we adopt an idea from Euclid's greatest common divisor (gcd) algorithm: we reduce $\mathbf{b}_2$ with respect to $\mathbf{b}_1$ as much as we can, then swap the roles of $\mathbf{b}_1$ and $\mathbf{b}_2$ and repeat the process. The process will stop when the basis meets the following conditions, which are the definition of LLL-reduced basis in two dimensions.

Definition 4 (LLL-reduced basis in two dimensions)**.** *A basis $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2]$ is LLL-reduced if*

1. $|\mu_{21}| \leq 1/2$.
2. $\|\mathbf{b}_2\| \geq \|\mathbf{b}_1\|$.

Note that the second condition can be written as $\left\|\tilde{\mathbf{b}}_2\right\|^2 \geq (1 - \mu_{21}^2) \left\|\tilde{\mathbf{b}}_1\right\|^2$. Generalizing for n dimensions we get the following definition.

Definition 5 (LLL-reduced basis)**.** *Let $\delta \in [1/4, 1]$. A basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ is δ-LLL-reduced if*

1. $|\mu_{ij}| \leq 1/2$ *for every* $1 \leq i < j \leq n$.

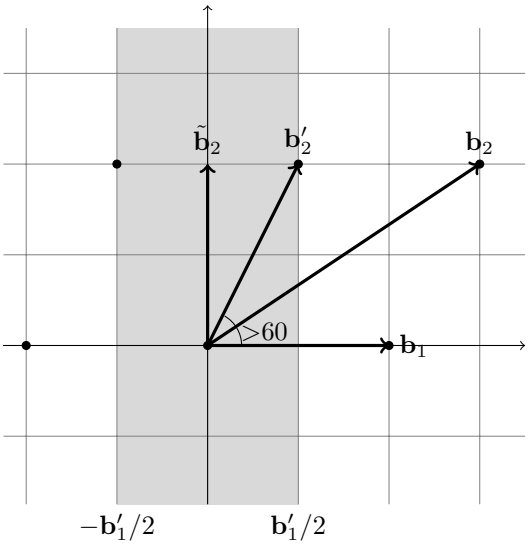

Figure 2: The LLL-reduced basis of $[\mathbf{b}_1 = (2,0), \mathbf{b}_2 = (3,2)]$ is $[\mathbf{b}'_1 = (2,0), \mathbf{b}'_2 = (1,2)]$.

2. $\left\|\tilde{\mathbf{b}}_{i+1}\right\|^2 \geq (\delta - \mu_{i+i,i}^2) \left\|\tilde{\mathbf{b}}_i\right\|^2$ *for every* $1 \leq i \leq n-1$.

Note that the projection of a (partial) LLL-reduced basis $[\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_i, \mathbf{b}_{i+1}]$ to $\mathsf{Span}(\tilde{\mathbf{b}}_i, \ldots, \tilde{\mathbf{b}}_n)$ is $[0, \ldots, 0, \tilde{\mathbf{b}}_i, \mathbf{b}_{i+1} + \mu_{i+1,i}\tilde{\mathbf{b}}_i]$. The last two vectors meet the definition of LLL-reduced basis in two dimensions.

References

[1] Divesh Aggarwal and Daniel Dadush and Oded Regev and Noah Stephens-Davidowitz. *Solving the Shortest Vector Problem in 2^n Time Using Discrete Gaussian Sampling: Extended Abstract.* STOC 2015.

[2] Lenstra, A.K. and Lenstra, H.W.jun. and László Lovász. *Factoring polynomials with rational coefficients.* Math. Ann. 1982