FHE-MPC Notes

Lecturer: Nigel Smart
Scribe: Joop Van de Pol

Lecture # 1
10th October 2011

Lattices

A lattice consists of all linear combinations with integer coefficients of some set of linearly independent vectors in a Euclidean space. If $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$ are linearly independent, then the lattice spanned by these vectors is given by $L = \mathbb{Z}\mathbf{b}_1 + \ldots + \mathbb{Z}\mathbf{b}_n = \{\sum_{i=1}^{n} z_i \mathbf{b}_i : z_i \in \mathbb{Z}\}$. The vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are called a *basis* of the lattice. Such a basis is generally not unique. For example, the vectors $-\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ and the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}, (\mathbf{b}_1 + \mathbf{b}_n)$ both span the same lattice as our original vectors.

You can think of a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ as the matrix containing these vectors as columns, i.e., $B = (\mathbf{b}_1|\mathbf{b}_2|\ldots|\mathbf{b}_n)$. Then, the lattice can be described as the set of all vectors $B\mathbf{z}$ for $\mathbf{z} \in \mathbb{Z}^n$. Now, two bases B and B' are *equivalent* (they span the same lattice) if and only if there exists some *unimodular* matrix U ($U \in GL_n(\mathbb{Z})$) such that $B' = BU$. Since unimodular matrices U have determinant $\det(U) = \pm 1$, it immediately follows that for any two equivalent bases B and B' we have that $\det(B') = \det(BU) = \det(B)$. Thus, the quantity $|\det(B)|$ is independent of the chosen basis. It can hence be written as $\det(L)$ and it is called the *fundamental volume* of the lattice.

Because a lattice is a discrete subgroup of $\mathbb{R}^n$, there must be a lattice vector that has a non-zero minimal length. For a given lattice L, let us define this length as the first minimum of the lattice: $\lambda_1(L) = \min\{\|\mathbf{x}\| : \mathbf{x} \in L, \mathbf{x} \neq \mathbf{0}\}$, where $\|.\|$ can be any norm (but is most commonly the Euclidean norm). Note that such a 'shortest' vector is not unique, because if $\mathbf{x}$ is a shortest lattice vector, then $-\mathbf{x}$ is a shortest lattice vector as well.

Besides the first minimum, there is also a second minimum, a third minimum and so on. However, some care must be taken in defining these successive minima. It may very well be that the 'second shortest' vector in a lattice is merely a multiple of the shortest vector in the lattice. Therefore, linearly independence is used to exclude these cases. Formally, the i-th minimum is defined as

$$\lambda_i(L) = \min\{\max\{\|\mathbf{x}_1\|, \ldots, \|\mathbf{x}_i\|\} : \mathbf{x}_1, \ldots, \mathbf{x}_i \in L \setminus \{\mathbf{0}\} \text{ linearly independent}\}.$$

Lattice problems

The concept of a shortest vector gives rise to the following classical lattice problem:

Shortest Vector Problem (SVP): Given a basis B for a lattice $L = L(B)$, find a lattice vector $\mathbf{v} \in L$ such that $\|\mathbf{v}\| = \lambda_1(L)$.

Another classical lattice problem is the following, which can be seen as the inhomogeneous version of SVP:

Closest Vector Problem (CVP): Given a basis B for a lattice $L = L(B)$ and some vector $\mathbf{x} \in \mathbb{R}^n$ (generally not in L), find a lattice vector $\mathbf{v} \in L$ such that $\|\mathbf{x} - \mathbf{v}\|$ is minimal.

Both of these problems are NP-hard, even when allowing approximations within constant factors (as well as some slowly increasing sub-polynomial functions). In fact, it is even considered to be a hard problem to find the shortest length $\lambda_1(L)$ of non-zero vectors of a lattice L.

In both of these problems, you are given a lattice basis B. Generally, a lattice is almost always represented by one of its bases. However, since each unimodular matrix U gives rise to a new basis of the same lattice, it follows that any lattice has infinitely many bases as soon as the dimension is at least 2. While these bases are equivalent, they are not equal. In fact, there exist 'good' and ‘bad’ bases for a lattice L. Informally, good means that the basis consists of short vectors that are somewhat orthogonal to each other, whereas bad means that it consists of long vectors that generally point in the same (or opposite) direction. The idea is that SVP and CVP become easier to solve when you have a good basis at your disposal. In the case of SVP, ideally the shortest vector is already in your basis, but there are other advantages to having a good basis as well. In the case of CVP, heuristic methods such as Babai’s nearest plane algorithm and the embedding technique give better results for better bases.

Basis reduction in theory

Ideally, we would like a basis where $\|\mathbf{b}_1\| = \lambda_1(L)$, $\|\mathbf{b}_2\| = \lambda_2(L)$ and so forth. However, from dimension $n = 5$ on such bases do not have to exist. Furthermore, this would amount to solving SVP outright, which is a hard problem. Therefore, we instead focus on the question: how can we turn a bad basis into a good basis? This is what so-called basis reduction methods try to achieve.

The aim is to retrieve a basis which consists of short and orthogonal vectors. However, since we know that the volume of a lattice is a fixed quantity that is independent of the choice of basis, ‘short’ and ‘orthogonal’ must be related. Indeed, when considering a fully orthogonal basis B, its determinant will be equal to the product of the length of the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$. This suggests that the more orthogonal basis vectors get, the shorter they must be as well. Therefore, let us focus on the orthogonality of our basis vectors for now.

Fortunately, we have a tool at our disposal that allows us to orthogonalize bases of vector spaces. This tool is called the Gram-Schmidt process, which, on input of a basis of a vector space, produces an orthonormal basis spanning the same space. Since we are only interested in orthogonal, we forget the normalization for now. The Gram-Schmidt process works iteratively, as follows:

$$\begin{aligned} &\mathbf{b}_1^* := \mathbf{b}_1 \\ &\mathbf{b}_i^* := \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^*, \quad \text{where } \mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2} \text{ for all } 1 \leq j < i \leq n. \end{aligned}$$

We were working with lattices, where only integer multiples of our basis vectors are allowed. However, the quantities μ_{ij} are generally not integers and hence the Gram-Schmidt vectors $\mathbf{b}_i^*$ are not lattice vectors. Therefore, we are going to do the next best thing: instead of using the Gram-Schmidt vectors $\mathbf{b}_i^*$, we will consider the vectors $\mathbf{b}_i' = \mathbf{b}_i - \sum_{j=1}^{i-1} \lfloor \mu_{ij} \rceil \mathbf{b}_j^*$, where $\lfloor . \rceil$ rounds to the nearest integer (rounding up if this is not unique). Note that applying the Gram-Schmidt process to our new vectors $\mathbf{b}_1', \ldots, \mathbf{b}_n'$ results in the same vectors $\mathbf{b}_i^*$ as our original basis, but the Gram-Schmidt coefficients μ_{ij} all satisfy $|\mu_{ij}| \leq 1/2$. In some sense, our new basis $\mathbf{b}_1', \ldots, \mathbf{b}_n'$ is as close to the Gram-Schmidt basis as possible, while still spanning the lattice. A basis B such that the Gram-Schmidt coefficients satisfy $|\mu_{ij}| \leq 1/2$ is called *size-reduced*.

However, the concept of a size-reduced basis is not enough. The quality of our size-reduced basis is highly dependent on the lengths of the Gram-Schmidt vectors, which are in turn highly dependent on the order of our basis vectors. As a different order of basis vectors gives different results, we would now like to be able to improve our basis by changing the order of basis vectors in a smart way. It was not known that efficient algorithms of this kind were possible until the invention of the LLL-algorithm in 1982, which is the first polynomial-time basis reduction algorithm. It turns out that being too greedy in the ordering of the vectors leads to inefficient algorithms. So how does the LLL-algorithm do it?

What happens when basis vectors $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ are swapped? Only $\mathbf{b}_i^*$ and $\mathbf{b}_{i+1}^*$ would change, and the new $\mathbf{b}_i^{*'} = \mathbf{b}_{i+1}^* + \mu_{i+1,i} \mathbf{b}_i^*$. Therefore, the creators of the LLL-algorithm added the condition

$$\delta \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^* + \mu_{i+1,i} \mathbf{b}_i^*\|^2, \tag{1}$$

where $\delta \in (1/4, 1)$ (default $\delta = 3/4$). This condition basically says that swapping vectors $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ will not gain you much. The LLL-algorithm (in its simplest form) repeatedly uses a procedure to sizereduce the basis using the Gram-Schmidt vectors while swapping vectors that violate condition (1) until no vector pairs violate the condition anymore.

The proof that the LLL-algorithm terminates (in polynomially many steps) is based on some integer quantity that strictly decreases with each swap and can hence only decrease a fixed number of times. When the LLL-algorithm terminates, it results in a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ such that, approximately, $\mathbf{b}_1 \leq \left(\frac{4}{3}\right)^{(n-1)/2} \lambda_1(L)$. For $n = 2$, LLL is equivalent to Gaussian reduction, continued fractions and the Euclidean GCD algorithm.

Now, LLL considers two vectors at a time and computes the gain of swapping these. Schnorr came up with a blockwise generalization of this procedure, which resulted in the algorithm BKZ. In BKZ, instead of considering only 2 vectors at a time, you consider m vectors at a time, where m is called the blocksize. It constructs a so-called projected lattice of dimension m using the vectors $\mathbf{x}_l, \ldots, \mathbf{x}_{l+m-1}$, for some $1 \leq l \leq n - m + 1$ where

$$\mathbf{x}_i := \mathbf{b}_i - \sum_{j=1}^{l-1} \mu_{ij} \mathbf{b}_j^*,$$

with the μ_{ij} and $\mathbf{b}_i^*$ as before and $l \leq i \leq l+m-1$. The $\mathbf{x}_i$ are similar to the Gram-Schmidt vectors, with only the contribution of the first $l - 1$ basis vectors removed. BKZ then uses a subprocedure to find the shortest vector (or close to it) in the lattice spanned by the vectors $\mathbf{x}_l, \ldots, \mathbf{x}_{l+m-1}$). It then decides in a way similar to LLL whether this vector should be inserted into the basis or not. BKZ repeats this procedure for varying l until no change occurs.

BKZ uses a procedure that actually solves SVP, but in a lattice of lower dimension. In practice enumeration methods are often used to solve SVP, but it is also possible to use sieving or other kinds of SVP-solvers. These routines are generally exponential in m. Note that, for $m = 2$, BKZ is equal to LLL, whereas for $m = n$, BKZ solves SVP exactly. BKZ will return a better approximation to the shortest vector as m increases, but its complexity increases (at least) exponentially in m as well.

Basis reduction in practice

People that used basis reduction algorithms for practical applications eventually discovered that the algorithms inexplicably found vectors of significant smaller norm than their theoretical upper bounds. Why are is the practical behaviour of basis reduction algorithms important? Basis reduction methods are the most prominent tools to attack lattice-based cryptosystems, and specifically, fully homomorphic encryption schemes that are often based on lattices. Understanding the practical performance of basis reduction algorithms allows us to understand the security of these lattice-based cryptosystems against such attacks.

In a Eurocrypt paper of 2008, Gama and Nguyen examined this unexpectedly good performance in more detail and found that, experimentally, these basis reduction algorithms do indeed perform much better than the theoretical upper bounds, but still within limits. In a 2011 Asiacrypt paper, Chen and Nguyen expand these results. To understand these results we must first have a way to describe the practical performance of basis reduction algorithms. Since finding the first minimum $\lambda_1(L)$ is in itself a hard problem, this quantity is generally not known. Hence, our basis might return some short vector, but we will have no idea how its length compares to the length of a shortest vector of the lattice. Therefore, the following lattice problem is often considered when working with basis reduction algorithms:

Hermite Shortest Vector Problem (HSVP$_\alpha$): Given a basis B for a lattice $L = L(B)$ and some approximation factor α, find a lattice vector $\mathbf{v} \in L$ such that $\|\mathbf{v}\| \leq \alpha \det(L)^{1/n}$.

This problem is related to a classical result by Hermite, who showed that $\lambda_1(L)^2 \leq \gamma_n \det(L)^{1/n}$, where γ_n is Hermite's constant in dimension n. In fact, the length of the shortest vector is proportional to $\det(L)^{1/n}$. Given any positive real number $t > 0$, let tL denote the lattice obtained by scaling all vectors of L by t. It is easily seen that the shortest length scales homogeneously $\lambda_1(tL) =$

$t\lambda_1(L)$ and that the n-th root of the volume scales by $\sqrt[n]{\det(tL)} = \sqrt[n]{t^n \det(L)} = t \det(L)^{1/n}$. Since we can easily compute $\det(L)$, we know the approximation factor α that our algorithms achieve.

In their 2008 paper, Gama and Nguyen showed that in practice, the Hermite factor α that basis reduction algorithms such as LLL and BKZ achieve is roughly δ^n for some δ, which is called the Hermite root factor. Thus, it is still exponential in the dimension of the lattice. However, they also showed that the Hermite root factor δ achieved by the algorithms was much lower in practice than the upper bounds from theory. LLL theoretically finds a vector of length, approximately, $\|\mathbf{b}_1\| \leq (4/3)^{(n-1)/4} \det(L)^{1/n} \approx 1.075^n \det(L)^{1/n}$. In practice, it finds a vector of length, approximately, $\|\mathbf{b}_1\| \leq 1.022^n \det(L)$. For BKZ with blocksize 20 this gap was about 1.033^n in theory versus 1.012^n in practice. Gama and Nguyen further claimed that a Hermite root factor of $\delta = 1.01$ seemed possible, whereas $\delta = 1.005$ did not seem possible yet. In the Asiacrypt 2011 paper, Chen and Nguyen claim that $\delta = 1.005$ is just about solvable in practice, whereas $\delta = 1.001$ will not be solvable by BKZ.

Further reading

For more information on basis reduction algorithms in practice, consider the EC2008 paper by Gama and Nguyen as well as the AC2011 paper by Chen and Nguyen. For a mix between theoretical and practical basis reduction, consider some of Schnorr's papers on the subject (from the 1980's onward). The chapter 'Lattice-based Cryptography' by Micciancio and Regev is a good introduction to this subject and can be found on Regev's website (`http://www.cs.tau.ac.il/~odedr/`).

For the applications of lattices to (the cryptanalysis of) other cryptosystems, search for the Coppersmith method or papers by Boneh and Durfee or Alex May. It is also interesting to see how lattices were used to break knapsack-based cryptosystems.