2.1 The Shortest Vector Problem

In our last lecture, we considered short solutions to modular equations. We showed how to find such solutions in two dimensions, and we proved that relatively short solutions always exist. We then generalized to arbitrary lattices, $\mathcal{L} := \{a_1\boldsymbol{b}_1 + \cdots + a_n\boldsymbol{b}_n \ : \ a_i \in \mathbb{Z}\}$ for arbitrary linearly independent basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$. We defined $\lambda_1(\mathcal{L})$ to be the length of the shortest non-zero vector (in the Euclidean norm) and showed Minkowski's bound,

$$\lambda_1(\mathcal{L}) \leq \sqrt{n}\det(\mathcal{L})^{1/n} .$$

But, we are computer scientists. So, we're not only interested in the existence of short lattice vectors; we want algorithms that find them. We therefore define the following computational problem, which is the central computational problem on lattices.

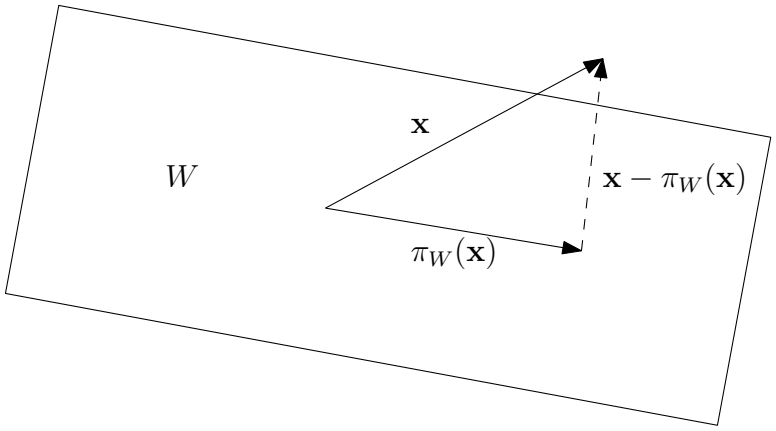
Definition 2.1. *For any approximation factor $\gamma = \gamma(n) \geq 1$, the γ-Shortest Vector Problem (γ-SVP) is defined as follows. The input is a basis for a lattice $\mathcal{L} \subset \mathbb{R}^n$. The goal is to output a non-zero lattice vector $\boldsymbol{x} \in \mathcal{L} \setminus \{\boldsymbol{0}\}$ with $\|\boldsymbol{x}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$.*

SVP is extremely well-studied for many different approximation factors. It turns out to be NP-hard (under randomized reductions) for any constant approximation factor, and it is hard for approximation factors up to $n^{c/\log\log n}$ for some constant $c > 0$ under reasonable complexity-theoretic assumptions [Mic01, Kho05, HR12]. Even for polynomial approximation factors $\gamma = n^C$ for any constant C, the fastest known algorithm for γ-SVP has running time that is exponential in the dimension! In this lecture, we will show the celebrated LLL algorithm, which efficiently solves γ-SVP for $\gamma = 2^{O(n)}$. This might not sound very impressive, but it is undoubtedly the most important result in the study of computational lattice problems.

2.2 The Gram-Schmidt Orthogonalization

In order to present the LLL algorithm, we need to introduce the Gram-Schmidt orthogonalization (which will turn out to be useful throughout this course).

We start with two natural definitions. Given any set $S \subseteq \mathbb{R}^n$, we write $S^\perp$ for the subspace of vectors perpendicular to all vectors in S. For any subspace $W \subseteq \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{R}^n$, we write $\pi_W(\boldsymbol{x})$ to be the *orthogonal projection of $\boldsymbol{x}$ onto W*. In other words, $\pi_W(\boldsymbol{x})$ is the unique vector in W such that $\boldsymbol{x} - \pi_W(\boldsymbol{x})$ is in $W^\perp$. Formally, $\pi_W(\boldsymbol{x}) := \boldsymbol{x} - \sum_i \langle \boldsymbol{v}_i, \boldsymbol{x}\rangle \boldsymbol{v}_i/\|\boldsymbol{v}_i\|^2$, where the $\boldsymbol{v}_i$ are any orthogonal basis of $W^\perp$, but the picture should be clearer than the formal definition:



.

Given a basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, we define its *Gram-Schmidt orthogonalization* (or just the Gram-Schmidt vectors) $(\widetilde{\boldsymbol{b}}_1, \ldots, \widetilde{\boldsymbol{b}}_n)$ such that

$$\widetilde{\boldsymbol{b}}_i := \pi_{\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}\}^\perp}(\boldsymbol{b}_i) \ .$$

In other words, $\widetilde{\boldsymbol{b}}_i$ is the component of $\boldsymbol{b}_i$ that is orthogonal to the preceding vectors in the basis. Note that the Gram-Schmidt vectors are mutually orthogonal with $\widetilde{\boldsymbol{b}}_1 = \boldsymbol{b}_1$.[1] And, notice that the Gram-Schmidt vectors depend crucially on the *order* of the basis vectors.

Claim 2.2. *For any lattice $\mathcal{L} \subset \mathbb{R}^n$ with basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$,*

$$\lambda_1(\mathcal{L}) \geq \min_i \|\widetilde{\boldsymbol{b}}_i\| \ .$$

Proof. Let $\boldsymbol{x} = \sum a_i \boldsymbol{b}_i \in \mathcal{L}$ be a shortest non-zero vector. Since $\boldsymbol{x}$ is non-zero, it has a non-zero coordinate. Let j be maximal such that $a_j \neq 0$. Then, since $a_i = 0$ for all $i > j$,

$$\|\boldsymbol{x}\| \geq \|\pi_{\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{j-1}\}^\perp}(\boldsymbol{x})\| = |a_j| \|\widetilde{\boldsymbol{b}}_j\| \geq \|\widetilde{\boldsymbol{b}}_j\| \geq \min_i \|\widetilde{\boldsymbol{b}}_i\| \ ,$$

where we have used the fact that a_j is a non-zero integer. □

It is often convenient to rotate our coordinates so that the Gram-Schmidt vectors are aligned with the standard basis. In these coordinates, our basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ becomes upper triangular,

$$\begin{pmatrix} \|\widetilde{\boldsymbol{b}}_1\| & \mu_{1,2}\|\widetilde{\boldsymbol{b}}_1\| & \mu_{1,3}\|\widetilde{\boldsymbol{b}}_1\| & \cdots & \mu_{1,n}\|\widetilde{\boldsymbol{b}}_1\| \\ 0 & \|\widetilde{\boldsymbol{b}}_2\| & \mu_{2,3}\|\widetilde{\boldsymbol{b}}_2\| & \cdots & \mu_{2,n}\|\widetilde{\boldsymbol{b}}_2\| \\ 0 & 0 & \|\widetilde{\boldsymbol{b}}_3\| & \cdots & \mu_{3,n}\|\widetilde{\boldsymbol{b}}_3\| \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \|\widetilde{\boldsymbol{b}}_n\| \end{pmatrix} \ . \tag{2.1}$$

[1] Some authors choose to work with the Gram-Schmidt *orthonormalization*, which in our notation is $(\widetilde{\boldsymbol{b}}_1/\|\widetilde{\boldsymbol{b}}_1\|, \ldots, \widetilde{\boldsymbol{b}}_n/\|\widetilde{\boldsymbol{b}}_n\|)$. I.e., they normalize so that the Gram-Schmidt vectors have unit length. We intentionally do not do this, as the lengths of the Gram-Schmidt vectors will play a key role in our study of bases.

By convention, we write the off-diagonal entries as $\mu_{i,j}\|\widetilde{\boldsymbol{b}}_i\|$. I.e., for $i < j$, we define

$$\mu_{i,j} := \frac{\langle \widetilde{\boldsymbol{b}}_i, \boldsymbol{b}_j \rangle}{\|\widetilde{\boldsymbol{b}}_i\|^2} \,. \tag{2.2}$$

In particular, note that we can trivially modify the basis to make $|\mu_{i,j}| \leq 1/2$ by simply subtracting an integer multiple of $\boldsymbol{b}_i$ from $\boldsymbol{b}_j$. We say that a basis is *size-reduced* if $|\mu_{i,j}| \leq 1/2$ for all $i < j$.

The above representation immediately shows that $\det(\mathcal{L}) = \prod_i \|\widetilde{\boldsymbol{b}}_i\|$. Combining this with Minkowski's theorem from the previous lecture and Claim 2.2, we have

$$\min_i \|\widetilde{\boldsymbol{b}}_i\| \leq \lambda_1(\mathcal{L}) \leq \sqrt{n} \cdot \Big(\prod_i \|\widetilde{\boldsymbol{b}}_i\| \Big)^{1/n} \,. \tag{2.3}$$

One fairly natural notion of a "good basis" for a lattice is one that yields relatively tight bounds in Eq. (2.3). In particular, such a basis immediately implies a good estimate for $\lambda_1(\mathcal{L})$. (And, with a bit more work, one can show that it yields a short vector.) Clearly, Eq. (2.3) becomes tighter if the Gram-Schmidt vectors have relatively similar lengths. So, this suggests that we might prefer bases whose Gram-Schmidt vectors all have roughly the same length.

We also have the bound

$$\min_i \|\widetilde{\boldsymbol{b}}_i\| \leq \lambda_1(\mathcal{L}) \leq \|\widetilde{\boldsymbol{b}}_1\| \,, \tag{2.4}$$

since $\widetilde{\boldsymbol{b}}_1 = \boldsymbol{b}_1$ is a non-zero lattice vector. In other words, the first vector in our basis will be a good solution to SVP if the lengths of the Gram-Schmidt vectors "decay" slowly. I.e., $\max_i \|\widetilde{\boldsymbol{b}}_1\|/\|\widetilde{\boldsymbol{b}}_i\|$ bounds the ratio of $\|\boldsymbol{b}_1\|$ to $\lambda_1(\mathcal{L})$.

This "decay rate of the Gram-Schmidt vectors," $\max_i \|\widetilde{\boldsymbol{b}}_1\|/\|\widetilde{\boldsymbol{b}}_i\|$ (and the similar expression $\max_{i \geq j} \|\widetilde{\boldsymbol{b}}_j\|/\|\widetilde{\boldsymbol{b}}_i\|$) turns out to be a very good way to judge a basis, and it leads naturally to the LLL algorithm. (We will later see that the performance of Babai's nearest-plane algorithm also depends on this rate.) In particular, as we will see, the LLL algorithm works by attempting to control this decay.

2.3 The LLL Algorithm

2.3.1 LLL-reduced bases

The idea behind the LLL algorithm, due to Lenstra, Lenstra, and Lovász [LLL82], is quite simple and elegant. Instead of looking at the "global decay" of the Gram-Schmidt vectors $\|\widetilde{\boldsymbol{b}}_1\|/\|\widetilde{\boldsymbol{b}}_i\|$, we consider the "local decay" $\|\widetilde{\boldsymbol{b}}_i\|/\|\widetilde{\boldsymbol{b}}_{i+1}\|$. We try to improve this local decay by swapping adjacent vectors in the basis. I.e., we switch the places of the vectors $\boldsymbol{b}_i$ and $\boldsymbol{b}_{i+1}$ if $\|\widetilde{\boldsymbol{b}}_i\|/\|\widetilde{\boldsymbol{b}}_{i+1}\|$ would become significantly lower as a result.

By only considering adjacent pairs of vectors, the LLL algorithm effectively reduces the problem of finding a good lattice basis to a simple two-dimensional lattice problem. So, let's look at the two-dimensional case, with a basis $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ and corresponding Gram-Schmidt vectors $(\widetilde{\boldsymbol{b}}_1, \widetilde{\boldsymbol{b}}_2)$. Let $\widetilde{\boldsymbol{b}}'_1$ and $\widetilde{\boldsymbol{b}}'_2$ be the Gram-Schmidt vectors of the "swapped

basis" $(\boldsymbol{b}_2, \boldsymbol{b}_1)$. (I.e., $\widetilde{\boldsymbol{b}}_1' = \boldsymbol{b}_2$ and $\widetilde{\boldsymbol{b}}_2' = \pi_{\boldsymbol{b}_2^\perp}(\boldsymbol{b}_1)$.) We would like to know when the "decay" of the first basis is no worse than the decay of the second basis. I.e., when do we have $\|\widetilde{\boldsymbol{b}}_1\|/\|\widetilde{\boldsymbol{b}}_2\| \le \|\widetilde{\boldsymbol{b}}_1'\|/\|\widetilde{\boldsymbol{b}}_2'\|$?

Note that

$$\det(\mathcal{L}) = \|\widetilde{\boldsymbol{b}}_1\|\|\widetilde{\boldsymbol{b}}_2\| = \|\widetilde{\boldsymbol{b}}_1'\|\|\widetilde{\boldsymbol{b}}_2'\| \,.$$

After rearranging a bit, we see that $\|\widetilde{\boldsymbol{b}}_1\|/\|\widetilde{\boldsymbol{b}}_2\| \le \|\widetilde{\boldsymbol{b}}_1'\|/\|\widetilde{\boldsymbol{b}}_2'\|$ if and only if $\|\widetilde{\boldsymbol{b}}_1\| \le \|\widetilde{\boldsymbol{b}}_1'\|$. Recalling that $\widetilde{\boldsymbol{b}}_1 = \boldsymbol{b}_1$ and $\widetilde{\boldsymbol{b}}_1' = \boldsymbol{b}_2$, we see that "swapping" will fail to improve the decay if and only if $\|\boldsymbol{b}_1\| \le \|\boldsymbol{b}_2\|$. That is a very simple condition!

To extend this to higher dimensions, it is convenient to write this simple condition using the $\mu_{i,j}$ notation from Eq. (2.2). In this notation, $\|\boldsymbol{b}_2\|^2 = \|\widetilde{\boldsymbol{b}}_2\|^2 + \mu_{1,2}^2\|\widetilde{\boldsymbol{b}}_1\|^2$. So, we see that a swap will fail to improve the decay of the Gram-Schmidt vectors if and only if

$$\|\widetilde{\boldsymbol{b}}_1\|^2 \le \|\widetilde{\boldsymbol{b}}_2\|^2 + \mu_{1,2}^2\|\widetilde{\boldsymbol{b}}_1\|^2 \,.$$

Lenstra, Lenstra, and Lovász extended this condition to an n-dimensional basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ and added a relaxation parameter $\delta \in (1/4, 1]$ to obtain the *Lovász condition*:

$$\delta\|\widetilde{\boldsymbol{b}}_i\|^2 \le \|\widetilde{\boldsymbol{b}}_{i+1}\|^2 + \mu_{i,i+1}^2\|\widetilde{\boldsymbol{b}}_i\|^2 \,. \tag{2.5}$$

(It is common to take $\delta = 3/4$.)

Definition 2.3. *For $\delta \in (1/4, 1]$, we say that a basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ is δ-LLL reduced (or just δ-LLL) if (1) it is size-reduced (i.e., $|\mu_{i,j}| \le 1/2$ for all $i < j$); and (2) it satisfies the Lovász condition in Eq.* (2.5).

The next theorem shows that a δ-LLL basis yields a relatively short vector. (It also shows why we take $\delta > 1/4$.)

Theorem 2.4. *If a basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ is δ-LLL reduced for some $\delta \in (1/4, 1]$, then $\|\boldsymbol{b}_1\| \le (\delta - 1/4)^{-n/2}\lambda_1(\mathcal{L})$.*

Proof. By Eq. (2.5), we have

$$\delta\|\widetilde{\boldsymbol{b}}_i\|^2 \le \|\widetilde{\boldsymbol{b}}_{i+1}\|^2 + \mu_{i,i+1}^2\|\widetilde{\boldsymbol{b}}_i\|^2 \le \|\widetilde{\boldsymbol{b}}_{i+1}\|^2 + \frac{1}{4}\|\widetilde{\boldsymbol{b}}_i\|^2 \,,$$

where we have used the fact that the basis is size-reduced. After rearranging, we have $\|\widetilde{\boldsymbol{b}}_i\|^2 \le (\delta - 1/4)^{-1} \cdot \|\widetilde{\boldsymbol{b}}_{i+1}\|^2$, and applying this repeatedly, we see that

$$\|\widetilde{\boldsymbol{b}}_1\|^2 \le (\delta - 1/4)^{-n}\|\widetilde{\boldsymbol{b}}_i\|^2$$

for all i. The result then follows from Claim 2.2. □

2.3.2 The actual algorithm

Of course, Theorem 2.4 is only interesting if we can find LLL-reduced bases. A very simple algorithm actually suffices! Here is the full LLL algorithm, which takes as input a basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$:

1. Size-reduce the basis.
2. If the basis is δ-LLL-reduced, halt and output the basis.
3. Otherwise, pick any index i such that the Lovasz condition in Eq. (2.5) is not satisfied, and switch the positions of $\boldsymbol{b}_i$ and $\boldsymbol{b}_{i+1}$.
4. Repeat.

Theorem 2.5. *The LLL algorithm terminates in time* $\mathrm{poly}(S)/\log(1/\delta)$*, where* S *is the bit length of the input basis.*

Proof. We assume for simplicity that the input basis is integral. (If the input is rational, then this is without loss of generality, as we can simply scale the basis by the least common multiple of the denominators in the input. If there are irrational entries in the input basis, then one needs to first choose a representation of irrational numbers.) We will not bother to prove that the numbers encountered by the algorithm remain bounded by $2^{\mathrm{poly}(S)}$. (I.e., we will only bound the arithmetic complexity of the algorithm.)

Consider the rather strange potential function

$$\Phi(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) := \prod_{i=1}^{n} \|\widetilde{\boldsymbol{b}}_1\| \|\widetilde{\boldsymbol{b}}_2\| \cdots \|\widetilde{\boldsymbol{b}}_i\| \,. \tag{2.6}$$

Notice that

$$\Phi(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) = \prod_{j=1}^{n} \|\widetilde{\boldsymbol{b}}_j\|^{n-j+1} = \det(\mathcal{L})^n \cdot \prod_{j=1}^{n} \|\widetilde{\boldsymbol{b}}_j\|^{1-j}$$

gets larger as the later Gram-Schmidt vectors get shorter. So, we can think of Φ as some sort of measure of the decay of the Gram-Schmidt vectors.

We observe some basic properties about Φ. If the $\boldsymbol{b}_i$ are integral, then $\Phi(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ is a product of determinants of integral lattices, so that $\Phi(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \geq 1$.[2] Furthermore, $\Phi(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \leq 2^{\mathrm{poly}(S)}$ for the input basis. Finally, note that Φ only depends on the Gram-Schmidt vectors, so that size-reduction does not affect Φ.

It therefore suffices to show that $\Phi(B')/\Phi(B) \leq \mathrm{poly}(\delta)$, where $B' := (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i+1}, \boldsymbol{b}_i, \ldots, \boldsymbol{b}_n)$ is the basis that results from applying a single "swapping step" of the LLL algorithm to $B := (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$. Let the Gram-Schmidt vectors of B' be $(\widetilde{\boldsymbol{b}}'_1, \ldots, \widetilde{\boldsymbol{b}}'_n)$. Notice that for $j \neq i$, the lattice generated by the first j vectors of B is the same as the lattice generated by the first j vectors of B'. So, the determinants of these prefixes must be the same. I.e., $\|\widetilde{\boldsymbol{b}}_1\| \cdots \|\widetilde{\boldsymbol{b}}_j\| = \|\widetilde{\boldsymbol{b}}'_1\| \cdots \|\widetilde{\boldsymbol{b}}'_j\|$. It follows from Eq. (2.6) that

$$\frac{\Phi(B')}{\Phi(B)} = \frac{\|\widetilde{\boldsymbol{b}}'_1\| \cdots \|\widetilde{\boldsymbol{b}}'_i\|}{\|\widetilde{\boldsymbol{b}}_1\| \cdots \|\widetilde{\boldsymbol{b}}_i\|} = \frac{\|\widetilde{\boldsymbol{b}}'_i\|}{\|\widetilde{\boldsymbol{b}}_i\|} \,.$$

[2] Here, we are glossing over the subtlety of what we mean by the determinant of a lattice spanned by a basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$ for $d < n$. Formally, we can take the square root of the determinant of the Gram matrix $B^T B$. Since the Gram matrix is integral, the determinant must be the square root of an integer.

Finally, we note that $\widetilde{\boldsymbol{b}}_i' = \widetilde{\boldsymbol{b}}_i + \mu_{i,i+1}\widetilde{\boldsymbol{b}}_{i+1}$. Therefore, $\|\widetilde{\boldsymbol{b}}_i'\|^2 = \|\widetilde{\boldsymbol{b}}_i\|^2 + \mu_{i,i+1}^2\|\widetilde{\boldsymbol{b}}_{i+1}\|^2$. Since the algorithm swapped $\boldsymbol{b}_i$ and $\boldsymbol{b}_{i+1}$, we must have

$$\|\widetilde{\boldsymbol{b}}_i'\|^2 = \|\widetilde{\boldsymbol{b}}_i\|^2 + \mu_{i,i+1}^2\|\widetilde{\boldsymbol{b}}_{i+1}\|^2 < \delta\|\widetilde{\boldsymbol{b}}_i\|^2 \ .$$

Therefore, $\Phi(B')/\Phi(B) < \sqrt{\delta}$, as needed. □

The following corollary is an immediate consequence of Theorems 2.4 and 2.5.

Corollary 2.6. *For any constant $\delta \in (1/4, 1)$, there is an efficient algorithm that solves γ-SVP with $\gamma := (\delta - 1/4)^{-n/2}$.*

For completeness, we note that a slightly better algorithm is known—the so-called Block Korkine-Zolotarev algorithm, or BKZ. The BKZ algorithm generalizes the LLL algorithm by considering k-tuples of adjacent vectors in the basis, as opposed to just pairs.

Theorem 2.7 ([Sch87, AKS01])**.** *For any constant $C > 0$, there is an efficient algorithm that solves γ-SVP with $\gamma := 2^{Cn \log\log n / \log n}$. More generally, for any $2 \le k \le n$, there is an algorithm that solves γ_k-SVP in time $2^{O(k)}$ times a polynomial in the input length with $\gamma_k := k^{n/k-1}$.*

2.3.3 A note on the importance of this algorithm

An approximation factor that is exponential in the dimension might not seem very impressive. But, the LLL algorithm has found innumerable applications. Lenstra, Lenstra, and Lovász originally used it to give an efficient algorithm for factoring rational polynomials—an application with obvious importance. The LLL algorithm is also used to solve computational lattice problems and integer programming exactly in low dimensions; to find integer relations and near-integer relations of irrational numbers; to identify unknown constants from their decimal approximations; etc. And, it is used as a subroutine in a huge number of lattice algorithms. In the next lecture, we will see how to use the LLL algorithm to break a large class of knapsack-based encryption schemes.

References

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

[HR12] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. *Theory of Computing*, 8(23):513–531, 2012. Preliminary version in STOC'07.

[Kho05] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *Journal of the ACM*, 52(5):789–808, September 2005. Preliminary version in FOCS'04.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[Mic01] Daniele Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, March 2001. Preliminary version in FOCS 1998.

[Sch87] C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(23):201 – 224, 1987.