Lattices

January 29, 2025

Acknowledgments

These lecture notes have been prepared using scribes contributed by Saswata Mukherjee, Aditya Morolia, Louise Xu, Ivan Lau, Duc Bui Hong, Dexter Kwan, Yu Peng Ng, Lin Haoxing, Sheyuan Yeo, Zheng Long Ong, Kel Zin Tan, Hua Jun Teow, and Chen Zhili, based on lectures delivered by Divesh Aggarwal. The notes have been edited and compiled by Aditya Morolia with valuable help by Ananta Mukherjee.

Contents

Chapter 1

Introduction to Lattices

Lattices are mathematical objects that have been studied since the time of Gauss, Lagrange, and Minkowski, starting in the 18^{th} century. They found numerous applications in cryptanalysis, post-quantum cryptography, integer programming, coding theory and computational number theory.

Definition 1.0.1 (Lattice). *A lattice is a discrete ablelian sub-group of $\mathbb{R}^d$, for some positive integer d. Equivalently, given a set $\{\vec{b}_1, \ldots, \vec{b}_n\} \subset \mathbb{R}^d$ of linearly independent vectors over $\mathbb{R}$, the set of all integer linear combinations of these vectors*

$$\mathcal{L}(\vec{b}_1, \vec{b}_2, \cdots, \vec{b}_n) := \left\{ \sum_{j \in [n]} a_j \vec{b}_j : \forall j \in [n],\ a_j \in \mathbb{Z} \right\} \tag{1.0.1}$$

is called a lattice.

The set $\{\vec{b}_1, \ldots, \vec{b}_n\}$ is called the basis of the lattice. We usually denote the matrix formed by stacking the basis vectors as

$$B := \begin{pmatrix} \vdots & \vdots & \cdots & \vdots \\ \vec{b}_1 & \vec{b}_2 & \cdots & \vec{b}_n \\ \vdots & \vdots & \cdots & \vdots \end{pmatrix}_{d \times n} \tag{1.0.2}$$

and the lattice generated by B as $\mathcal{L}(B) := \{B\vec{z} : \vec{z} \in \mathbb{Z}^n\}$.
The dimension of the ambient space d is called the dimension of lattice and the integer n is called the *rank* of the lattice. A lattice is called *full-rank* if its dimension is same as its rank ($n = d$). Clearly, a lattice is a discrete structure.

Definition 1.0.2 (Span of a lattice). *For a lattice $\mathcal{L}(B)$, its span is span of its basis vcetors, that is*

$$\mathit{span}(\mathcal{L}(B)) := \left\{ B\vec{z} : \vec{z} \in \mathbb{R}^n \right\}$$

It is easy to see that the set in eq. (1.0.1) is an *abelian* group under vector addition.

Lemma 1.0.3. *The tuple $(\mathcal{L}(B), +)$ (eq. (1.0.1)) forms an abelian group. Here $+$ is vector addition over $\mathbb{R}^d$.*

Proof. We check each property of the group axioms, which follow trivially from the properties of addition over $\mathbb{R}^n$.

This chapter is based on lectures 1 and 2.

- **Closed under addition:** Given two elements, $\vec{u} = \sum_j y_j \vec{b}_j$ and $\vec{v} = \sum_j z_j \vec{b}_j$ in $\mathcal{L}(B)$, their addition $\vec{u} + \vec{v} = \sum_j (y_j + z_j)\vec{b}_j$ is also in $\mathcal{L}(B)$.
- **Existence of additive unity:** The zero vector $\vec{0} = (0, \ldots, 0)^\top \in \mathcal{L}(B)$.
- **Existence of additive inverse:** For any $\vec{u} = \sum_j y_j \vec{b}_j$, there is $-\vec{u} = \sum_j (-y_j)\vec{b}_j \in \mathcal{L}(B)$, and $\vec{u} + (-\vec{u}) = \vec{0}$.
- $+$ **commutes:** $\forall \vec{u}, \vec{v} \in \mathcal{L}(B)$, $\vec{u} + \vec{v} = \vec{v} + \vec{u}$.

□

1.1 Fundamental Parallelepiped

Definition 1.1.1 (Fundamental Prallelepiped)**.** *For a lattice $\mathcal{L}(B)$ of rank n, we define its fundamental parallelepiped as*

$$\mathcal{P}(B) := \{B\vec{x} : \vec{x} \in [0,1)^n\} \tag{1.1.1}$$

Lemma 1.1.2. *Translates of $\mathcal{P}(B)$ by lattice vectors, i.e. the set $\mathcal{S} := \{\vec{v} + \mathcal{P}(B) : \vec{v} \in \mathcal{L}(B)\}$ is a tiling of $\mathbb{R}^d$.*

The proof is left as an exercise.

Lemma 1.1.3. *Let $\mathcal{L}$ be a lattice of rank n, and $B = \{\vec{b}_1, \ldots, \vec{b}_n\} \subset \mathcal{L}$ be a linearly independent set of n lattice vectors. Then B forms a basis of $\mathcal{L}$* if and only if

$$\mathcal{P}(\vec{b}_1, \ldots, \vec{b}_n) \cap \mathcal{L} = \{\vec{0}\}$$

Proof. ($\implies$) We prove this by contradiction. Assume B form a basis of $\mathcal{L}$. Let $\vec{u} \neq \vec{0} \in \mathcal{P}(\vec{b}_1, \ldots, \vec{b}_n) \cap \mathcal{L}$. Since B is a basis, we know that there exists some non-zero $\vec{x} \in \mathbb{Z}^n$ and $\vec{y} \in [0,1)^n$ such that

$$\begin{aligned} \vec{u} &= x_1\vec{b}_1 + \ldots + x_n\vec{b}_n = y_1\vec{b}_1 + \ldots + y_n\vec{b}_n \\ &\implies (x_1 - y_1)\vec{b}_1 + \ldots + (x_n - y_n)\vec{b}_n \neq \vec{0}. \end{aligned}$$

But B is linearly independent, so this is a contradiction. Thus $\mathcal{P}(\vec{b}_1, \ldots, \vec{b}_n) \cap \mathcal{L} = \{\vec{0}\}$.
($\impliedby$) We prove this by contradiction. Assume $\mathcal{P}(\vec{b}_1, \ldots, \vec{b}_n) \cap \mathcal{L} = \{\vec{0}\}$. Let $\vec{u} \neq \vec{0} \in \mathcal{L}$ but $\vec{u}$ is not generated by integer combinations of $\vec{b}_1, \ldots, \vec{b}_n$. Since $\mathcal{L}$ has rank n, and $\vec{b}_1, \ldots, \vec{b}_n$ are linearly independent, $\exists \vec{x} \in \mathbb{R}^n \setminus \mathbb{Z}^n : \vec{u} = x_1\vec{b}_1 + \ldots + x_n\vec{b}_n$. Consider the vector $\vec{u}' = \sum_{i=1}^n (x_i - \lfloor x_i \rfloor)\vec{b}_i$. We know that $\vec{u} \in \mathcal{L}$, and $\forall i \in [n] : (-\lfloor x_i \rfloor)b_i \in \mathcal{L}$. Since $\mathcal{L}$ is closed under addition, $\vec{u}' \in \mathcal{L}$. Also, $\forall i \in [n] : x_i - \lfloor x_i \rfloor \in [0,1)$. Thus $u' \in \mathcal{P}(\vec{b}_1, \ldots, \vec{b}_n) \cap \mathcal{L}$. By our assumption $\vec{u}' = \vec{0}$. So, $x_i - \lfloor x_i \rfloor = 0$ for all i, which implies $\vec{x} \in \mathbb{Z}^n$. This contradicts our assumption. □

1.2 Equality of Lattices

How do we decide if the lattices generated by two basis B_1 and B_2 are the same (i.e., $\mathcal{L}(B_1) = \mathcal{L}(B_2)$)? Observe that for a basis B, the following operations leave the lattice generated by B unchanged:

- **Adding of an integer multiple of basis vector to another:** $\vec{b}_i \leftarrow \vec{b}_i + k\vec{b}_j$ for some $k \in \mathbb{Z}$.
- **Swapping two basis vectors:** $\vec{b}_i \leftrightarrow \vec{b}_j$.

- **Negating a basis vector:** $\vec{b}_i \leftarrow -\vec{b}_i$.

This is true because the new set of basis generated after each of these operations can generate the original set of basis vectors by integer linear combinations. We can formalize this notion using linear algebraic properties of the basis matrix.

Definition 1.2.1 (Unimodular Matrix)**.** *A matrix $U \in \mathbb{Z}^{n\times n}$ is called unimodular if* $\det(U) = \pm 1$.

Lemma 1.2.2. *$U \in \mathbb{Z}^{n\times n} \implies U^{-1} \in \mathbb{Z}^{n\times n}$ and U^{-1} is unimodular.*

Proof. Note that, by generalized Cramer's rule,

$$(U^{-1})_{ij} = \frac{(\text{adj}(U))_{ij}}{\det(U)}.$$

Since the entries of adj(U) are integers ($(-1)^{i+j}$ times the (j,i)-minor of U), and $\det(U) = \pm 1$, $U^{-1} \in \mathbb{Z}^{n\times n}$. Moreover, $\det(I) = 1 = \det(UU^{-1}) = \det(U)\det(U^{-1}) \implies U^{-1}$ is unimodular. □

Now we are ready to decide if two basis are equivalent.

Lemma 1.2.3. *Two basis $B_1, B_2 \in \mathbb{R}^{d\times n}$ are equivalent (i.e. $\mathcal{L}(B_1) = \mathcal{L}(B_2)$) if and only if $B_2 = B_1U$, for some unimodular matrix U.*

Proof. ($\implies$) Let $\mathcal{L}(B_1) = \mathcal{L}(B_2)$. Then, each column of B_1 is in $\mathcal{L}(B_2)$, that is, $\forall j \in [n]\ \exists \vec{x} \in \mathbb{Z}^n : (B_1)_j = B_2\vec{x}$. So, $B_1 = B_2U$ for some $U \in \mathbb{Z}^{n\times n}$ (whose columns are the corresponding $\vec{x}$). By a similar argument, there is a $V \in \mathbb{Z}^{n\times n}$ so that $B_2 = B_1V = B_2UV$. Note that $B_2^\top B_2 \in \mathbb{R}^{n\times n}$ is a square matrix, and hence $\det(B_2^\top B_2)$ is well defined.

$$\begin{aligned}
\det(B_2^\top B_2) &= \det((B_2UV)^\top B_2UV) \\
&= \det(V^\top U^\top B_2^\top B_2UV) \\
&= \det(V^\top)\det(U^\top)\det(B_2^\top B_2)\det(U)\det(V) \quad \text{determinant of product is product of determinants.} \\
&\implies (\det(U)\det(V))^2 = 1 \\
&\implies (\det(U)\det(V)) = \pm 1.
\end{aligned}$$

Since U, V are integer matrices, the above implies that $\det(U), \det(V) \in \{\pm 1\}$, and thus they are unimodular.
($\impliedby$) Suppose $B_2 = B_1U$ for some unimodular matrix U. Then, $B_2U^{-1} = B_1$, and U^{-1} is unimodular by lemma 1.2.2. By definition of matrix multiplication, each column of B_2 is integer combination of columns of B_1. So, $\mathcal{L}(B_2) \subseteq \mathcal{L}(B_1)$. Similarly, $\mathcal{L}(B_2) \subseteq \mathcal{L}(B_1)$. Hence, $\mathcal{L}(B_1) = \mathcal{L}(B_2)$. □

1.3 Determinant of a Lattice

Definition 1.3.1 (Determinant of a Lattice)**.** *Let $\mathcal{L} = \mathcal{L}(B)$ be a lattice of rank n. We define its determinant* $\det(\mathcal{L})$ *as the n-dimensional volume of $\mathcal{P}(B)$. Equivalently*[1]*,*

$$\det(\mathcal{L}) = \sqrt{\det(B^\top B)}$$

[1] This is proved in section 1.6

Note that if the lattice is full-rank, then B is a square matrix and $\det(\mathcal{L}) = |\det(B)|$. Determinant of a lattice is well-defined. Say, we have another basis B' for $\mathcal{L}$. By lemma 1.2.3, there exists a unimodular matrix U such that $B' = BU$. Then,

$$\det(B'^{\top}B') = \det(U^{\top}B^{\top}BU) = \det(U^{\top})\det(U)\det(B^{\top}B) = \det(B^{\top}B) \quad \text{since, } \det(U) = \pm 1.$$

The determinant of a lattice is a measure of how *dense* a lattice is. The lesser the determinant, the denser the lattice is. To see why, observe that the density of $\mathcal{L}$ would be inversely proportional to the volume of the fundamental parallelepiped.

$$\text{density}(\mathcal{L}(B)) \propto \frac{1}{\text{Vol}_n(\mathcal{P}(B))} = \frac{1}{\det(\mathcal{L})}.$$

Consider an n-dimensional ball of radius $R > 0$ (in the ℓ_2 norm) centered at the origin, denoted $\mathcal{B}_2^n(R)$. As $R \to \infty$, the number of lattice points inside the ball,

$$|\mathcal{L}(B) \cap \mathcal{B}_2^n(R)| \to \frac{\text{Vol}_n(\mathcal{B}_2^n(R))}{\text{Vol}_n(\mathcal{P}(B))}.$$

In fact, we can prove the following.

Theorem 1.3.2. *Let $\mathcal{L}(B) : B \in \mathbb{R}^{n\times n}$ be a lattice of rank n. Then for any $\varepsilon > 0$, there is an $R := R(\varepsilon, n, B)$ such that*

$$(1-\varepsilon)\frac{Vol_n(\mathcal{B}_2^n(R))}{Vol_n(\mathcal{P}(B))} \le |\mathcal{L}(B) \cap \mathcal{B}_2^n(R)| \le (1+\varepsilon)\frac{Vol_n(\mathcal{B}_2^n(R))}{Vol_n(\mathcal{P}(B))} \tag{1.3.1}$$

1.4 Shortest Non-Zero Vector and Towards Minkowski's First Theorem

Given a set of lattice vectors, we would like to define a notion of length and distance. We define the length of a lattice vector to be the Euclidean norm of the vector. For $\vec{x} \in \mathbb{R}^d$, we denote its length by $\|\vec{x}\| := \sqrt{\sum_{i=1}^n x_i^2}$. Given two vectors $\vec{x}, \vec{y} \in \mathbb{R}^d$, define the Euclidean distance between them to be $\|\vec{x} - \vec{y}\|$.

Definition 1.4.1 (Successive minima)**.** *Let $\mathcal{L}$ be a lattice of rank n. For $i \in [n]$, define the i-th* successive minima *as*

$$\lambda_i := \inf\{R | \dim(\text{span}(\mathcal{L} \cap \mathcal{B}_n(R))) \ge i\} \tag{1.4.1}$$

Note that $\lambda_1(\mathcal{L})$ is the length of the shortest non-zero vector of the lattice. Observe that equivalently, $\lambda_1(\mathcal{L}) := \min\{\|\vec{x} - \vec{y}\| : \vec{x}, \vec{y} \in \mathcal{L} \text{ and } \vec{x} \ne \vec{y}\}$. The most important computational lattice problem is the problem of finding the shortest non-zero lattice vector.

Theorem 1.4.2 (Blichfeldt's Theorem)**.** *For any full rank lattice $\mathcal{L} \subseteq \mathbb{R}^n$ and any measurable set $S \subseteq \mathbb{R}^n$ such that $Vol_n(S) > \det(\mathcal{L})$, there exist distinct points $\vec{z}_1 \ne \vec{z}_2 \in S$ such that $\vec{z}_1 - \vec{z}_2 \in \mathcal{L}$.*

Proof. Let B be a basis of $\mathcal{L}$. Note that $\{\vec{x} + \mathcal{P}(B) : \vec{x} \in \mathcal{L}\}$ is a partition of $\mathbb{R}^n$ (lemma 1.1.2). For $x \in \mathcal{L}$, define $S_{\vec{x}} = S \cap (\vec{x} + \mathcal{P}(B))$. So we have that

$$\text{Vol}_n(S) = \sum_{\vec{x}\in\mathcal{L}} \text{Vol}_n(S_{\vec{x}}).$$

Also define $\hat{S}_{\vec{x}} := S_{\vec{x}} - \vec{x}$. Clearly, $\hat{S}_x \subseteq \mathcal{P}(B)$, and $\text{Vol}_n(\hat{S}_{\vec{x}}) = \text{Vol}_n(S_{\vec{x}})$. So,

$$\sum_{\vec{x}\in\mathcal{L}} \text{Vol}_n(\hat{S}_{\vec{x}}) = \sum_{\vec{x}\in\mathcal{L}} \text{Vol}_n(S_{\vec{x}}) = \text{Vol}_n(S) > \text{Vol}_n(\mathcal{P}(B)).$$

So, there must be $\vec{x} \neq \vec{y} \in \mathcal{L}$ for which $\hat{S}_{\vec{x}} \cap \hat{S}_{\vec{y}} \neq \phi$. Let $\vec{z} \in \hat{S}_{\vec{x}} \cap \hat{S}_{\vec{y}}$. Then, by definition we have that $\vec{z}+\vec{x} \in S_{\vec{x}}$ and $\vec{z}+\vec{y} \in S_{\vec{y}}$, and for these points $(\vec{z}+\vec{x})-(\vec{z}+\vec{y}) = \vec{x}-\vec{y} \in \mathcal{L}$.

□

Notice that the bound in theorem 1.4.2 is tight.
We call a set $S \subseteq \mathbb{R}^n$ *centrally symmetric* if $\vec{x} \in S \implies -\vec{x} \in S$. We call a set $S \subseteq \mathbb{R}^n$ *convex* if for any $\vec{x}, \vec{y} \in S$, $\lambda\vec{x} + (1-\lambda)\vec{y} \in S$ for all $\lambda \in [0,1]$. Using these notions, we can state Minkowski's *convex body theorem*, from which Minkowski's first theorem follows.

Theorem 1.4.3 (Minkowski's Convex Body Thorem)**.** *Let $\mathcal{L}$ be a full rank lattice of rank n. For any centrally symmetric convex set $S \subset \mathbb{R}^n$, if Vol$(S) > 2^n \det(\mathcal{L})$, then S contains a non-zero lattice point.*

Proof. Define $\hat{S} = \frac{1}{2}S := \{\vec{x} : 2\vec{x} \in S\}$. Then, $\text{Vol}(\hat{S}) = \frac{1}{2^n}\text{Vol}_n(S) > \det(\mathcal{L})$.
By theorem 1.4.2, there is $\vec{z}_1 \neq \vec{z}_2 \in \hat{S}$ so that $\vec{z}_1 - \vec{z}_2 \in \mathcal{L}$. Also, $2\vec{z}_1, 2\vec{z}_2 \in S$. Because S is centrally symmetric, $-2\vec{z}_2 \in S$. Because it is convex, $\frac{2\vec{z}_1 - 2\vec{z}_2}{2} = \vec{z}_1 - \vec{z}_2 \in S$. □

We would also need the following bound.

Lemma 1.4.4. *The volume of an n-dimensional ball of radius r, Vol$_n(\mathcal{B}(\vec{0}, r)) \geq \left(\frac{2r}{\sqrt{n}}\right)^n$*

Proof. Clearly, $\mathcal{B}(\vec{0}, r)$ contains the cube $\left\{\vec{x} \in \mathbb{R}^n : |x_i| < \frac{r}{\sqrt{n}}\right\}$. Therefore, the volume of the ball is at least the volume of this cube. □

Theorem 1.4.5 (Minkowski's First Theorem)**.** *For any full rank lattice $\mathcal{L}$ of rank n, $\lambda_1(\mathcal{L}) \leq \sqrt{n}(\det(\mathcal{L}))^{1/n}$.*

Proof. By definition, $\mathcal{B}(\vec{0}, \lambda_1(\mathcal{L}))$ does not contain any non-zero lattice point. From (the converse of) theorem 1.4.3 and lemma 1.4.4

$$\left(\frac{2\lambda_1(\mathcal{L})}{\sqrt{n}}\right)^n \leq \text{Vol}_n(\mathcal{B}(\vec{0}, \lambda_1(\mathcal{L}))) \leq 2^n \det(\mathcal{L})$$

Theorem follows from this. □

1.5 Gram-Schmidt orthogonalization

The Gram-Schmidt orthogonalization (GSO) takes linearly independent vectors $\vec{b}_1, ..., \vec{b}_n \in \mathbb{R}^d$ as input and computes an orthogonal basis $\vec{b}_1^*, ..., \vec{b}_n^*$ such that $\text{span}(\vec{b}_1, ..., \vec{b}_j) = \text{span}(\vec{b}_1^*..., \vec{b}_j^*)$ for all $j = 1, ..., n$. The idea of GSO is that we compute the vectors $\vec{b}_1^*, ..., \vec{b}_n^*$ in order such that $\vec{b}_i^*$ is the component of $\vec{b}_i$ that is orthogonal to $\text{span}(\vec{b}_1, ..., \vec{b}_{i-1}) = \text{span}(\vec{b}_1^*..., \vec{b}_{i-1}^*)$. This can be obtained by subtracting all components of $\vec{b}_1, ..., \vec{b}_{i-1}$ from $\vec{b}_i$ and call the remainder $\vec{b}_i^*$. Formally the method is as follows:

Remark 1.5.1. *Note that in general $\mathcal{L}(B) \neq \mathcal{L}(B^*)$, i.e., B^* may not be a basis for the lattice $\mathcal{L}(B)$. This is because the coefficient $\mu_{i,j}$ on Line 3 may not be an integer in general. For example, for basis $B = [(2,0)^\mathsf{T}, (1,2)^\mathsf{T}]$, the orthogonal basis $B^* = \text{GSO}(B) = [(2,0)^\mathsf{T}, (0,2)^\mathsf{T}]$, is not a lattice basis because the vector $(0,2)^\mathsf{T}$ does not belong to the lattice $\mathcal{L}(B)$.*

Algorithm 1 Gram-Schmidt orthogonalization (GSO)

Input: Basis $B = [\vec{b}_1, ..., \vec{b}_n] \in \mathbb{R}^{d\times n}$
Output: Orthogonal basis $B^* = [\vec{b}_1^*, ..., \vec{b}_n^*] \in \mathbb{R}^{d\times n}$

1: Set $\vec{b}_1^* = \vec{b}_1$
2: **for** $i = 2$ to n **do**
3: $\quad \vec{b}_i^* = \vec{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\, \vec{b}_j^*$ where $\mu_{i,j} = \frac{\langle \vec{b}_i, \vec{b}_j^* \rangle}{\langle \vec{b}_j^*, \vec{b}_j^* \rangle} = \frac{\langle \vec{b}_i, \vec{b}_j^* \rangle}{\|\vec{b}_j^*\|^2}$
4: **end for**

Remark 1.5.2. *Recall that the fundamental parallelepiped associated to a lattice basis $B \in \mathbb{R}^{d\times n}$ is the set of points*

$$\mathcal{P}(B) = \Big\{ B\vec{x} : \vec{x} \in [0,1)^n \Big\}, \tag{1.5.1}$$

and the determinant of lattice $\mathcal{L}(B)$, denoted by $\det(\mathcal{L})$, is given by the n-dimensional volume $\mathrm{Vol}(\mathcal{P}(B))$. By some linear algebraic properties, $\mathrm{Vol}(\mathcal{P}(B))$ is given by

$$\mathrm{Vol}(\mathcal{P}(B)) = \prod_{i=1}^{n} \|\vec{b}_i^*\|, \tag{1.5.2}$$

and so we have

$$\det(\mathcal{L}) = \mathrm{Vol}(\mathcal{P}(B)) = \prod_{i=1}^{n} \|\vec{b}_i^*\|, \tag{1.5.3}$$

1.6 A proof of the formula of determinant of a lattice.

It is stated earlier without proof that $\det(\mathcal{L}) = \sqrt{\det(B^\mathsf{T} B)}$. We now provide a proof.

Theorem 1.6.1. *For any lattice basis $B = [\vec{b}_1, ..., \vec{b}_n] \in \mathbb{R}^{d\times n}$, we have*

$$\det(\mathcal{L}) = \sqrt{\det(B^\mathsf{T} B)} \tag{1.6.1}$$

Proof. Let $B^* = [\vec{b}_1^*, ..., \vec{b}_n^*] = \mathrm{GSO}(B)$. By Line 3 of Algorithm 1, we have $\vec{b}_i = \vec{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j}\, \vec{b}_j^*$ for each $i = 1, ..., n$. In matrix notation, we can write this as $B = B^* U$ for some upper triangular matrix U with 1's on the diagonal and $U_{j,i} = \mu_{i,j}$ for all $j < i$. Note that $\det(U) = 1 = \det\big(U^\mathsf{T}\big)$ since the determinant of a triangular matrix is the product of its diagonal elements, which is 1 for U.
It follows that

$$\sqrt{\det(B^\mathsf{T} B)} = \sqrt{\det\Big(U^\mathsf{T} (B^*)^\mathsf{T} B^* U\Big)} = \sqrt{\det\big(U^\mathsf{T}\big)\det\Big((B^*)^\mathsf{T} B^*\Big)\det(U)} = \sqrt{\det\Big((B^*)^\mathsf{T} B^*\Big)}. \tag{1.6.2}$$

Since the columns of B^* are orthogonal, we have

$$(B^*)^\mathsf{T} B^* = \mathrm{diag}(\|\vec{b}_1^*\|^2, ..., \|\vec{b}_n^*\|^2), \tag{1.6.3}$$

and so

$$\det\Big((B^*)^\mathsf{T} B^*\Big) = \prod_{i=1}^{n} \|\vec{b}_i^*\|^2. \tag{1.6.4}$$

Combining (1.6.2), (1.6.4), and (1.5.3) gives

$$\sqrt{\det(B^\mathsf{T}B)} = \sqrt{\prod_{i=1}^{n} \|\vec{b}_i^*\|^2} = \prod_{i=1}^{n} \|\vec{b}_i^*\| = \det(\mathcal{L}) \tag{1.6.5}$$

as desired. □

1.7 Lower bound on the shortest vector

Theorem 1.7.1. *Let $\mathcal{L}$ be a lattice with basis $B = [\vec{b}_1, ..., \vec{b}_n]$, and let $B^* = \mathrm{GSO}(B) = [\vec{b}_1^*, ..., \vec{b}_n^*]$. Then the length $\lambda_1(\mathcal{L})$ of the shortest vector satisfies $\lambda_1(\mathcal{L}) \geq \min_{1\leq i\leq n} \|\vec{b}_i^*\|$.*

Proof. It sufficies to show that $\|\vec{v}\| \geq \min_{1\leq i\leq n} \|\vec{b}_i^*\|$ for each non-zero lattice vector $\vec{v} \in \mathcal{L}$. Let $\vec{v} = \sum_{i=1}^{n} x_i\vec{b}_i \in \mathcal{L}$ be a non-zero lattice vector, where $x_i \in \mathbb{Z}$. Since $\vec{v}$ is non-zero, it has a non-zero coefficient x_i. Let k be the largest index such that $x_k \neq 0$, then

$$\vec{v} = \sum_{i=1}^{k} x_i\vec{b}_i = \sum_{i=1}^{k} x_i \left(\vec{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j}\vec{b}_j^*\right) = x_k\vec{b}_k^* + \sum_{i=1}^{k-1} x_i\vec{b}_i^* + \sum_{i=1}^{k}\sum_{j=1}^{i-1} \mu_{i,j}\vec{b}_j^*. \tag{1.7.1}$$

It follows that

$$\pi_k(\vec{v}) := \pi_{\mathrm{span}(\tilde{\mathrm{b}}_1,\ldots,\tilde{\mathrm{b}}_{k-1})^\perp}(\vec{v}) = \pi_{\mathrm{span}(\tilde{\mathrm{b}}_1^*,\ldots,\tilde{\mathrm{b}}_{k-1}^*)^\perp}(\vec{v}) = \pi_{\vec{b}_k^*}(\vec{v}) = x_k\vec{b}_k^* \tag{1.7.2}$$

where $\pi_k(\vec{v})$ is the component of $\vec{v}$ in the direction of $\vec{b}_k^*$. Therefore, we have

$$\|\vec{v}\| = \left\|\pi_k(\vec{v}) + \pi_{\mathrm{span}(\tilde{\mathrm{b}}_1^*,\ldots,\tilde{\mathrm{b}}_{k-1}^*)}(\vec{v})\right\| \geq \|\pi_k(\vec{v})\| = \|x_k\vec{b}_k^*\| = |x_k|\|\vec{b}_k^*\| \geq \|\vec{b}_k^*\| \geq \min_{1\leq i\leq n} \|\vec{b}_i^*\|, \tag{1.7.3}$$

where the second last inequality follows from the fact that x_k is a nonzero integer, i.e. $|x_k| \geq 1$. □

Chapter 2

Lenstra Lenstra Lovász Algorithm

The LLL algorithm [LLL82] is the most important lattice algorithm known. It can be used to find the basis of a lattice, given some generating set, as a *basis reduction* algorithm, and to approximate the shortest non-zero lattice vector in polynomial time. In this chapter, we will see all of these applications.
Similar to vector space, we can define a lattice via a generating set of vectors. Given $\vec{a}_1, ..., \vec{a}_N \in \mathbb{Z}^d$, we define

$$\mathcal{L}(\vec{a}_1, ..., \vec{a}_N) := \left\{ \sum_{i=1}^{N} x_i \vec{a}_i : x_i \in \mathbb{Z} \right\}. \tag{2.0.1}$$

Remark 2.0.1. *Note that unlike basis, the vectors $\vec{a}_1, ..., \vec{a}_N$ are not necessarily independent. Specifically, N may be larger than d.*

Remark 2.0.2. *Without loss of generality, we can take vectors to be arbitrary rational vectors since we can scale the lattice by the lowest common multiple of all the denominators involved and obtain an isomorphic lattice. However, we cannot allow vectors to be arbitrarily real vectors.*

Goal: Given a set of vectors $\vec{a}_1, \ldots, \vec{a}_N \in \mathbb{Z}^d$ generating $\mathcal{L}$, we would like to find a basis of $\mathcal{L}(\vec{a}_1, \ldots, \vec{a}_N)$.

2.1 LLL as a basis finding algorithm

Note that for $d = 1$, we have $\mathcal{L}(a_1, \ldots, a_N) = \gcd(a_1, \ldots, a_N) \cdot \mathbb{Z}$, and so the basis is simply $\gcd(a_1, \ldots, a_N)$.

Example 2.1.1. *When $d = 1$ with $N = 2$, this can be found using the Euclidean algorithm. We use the following example to illustrate this*

$$\gcd(264, 936) = \gcd(144, 264) = \gcd(120, 144) = \gcd(24, 120) = \gcd(0, 24) = 24. \tag{2.1.1}$$

For $d = 1$ with general $N \geq 3$, we progressively find pairs of numbers (x_i, y_i) and replace them with (x_{i+1}, y_{i+1}) such that

- $\gcd(x_{i+1}, y_{i+1}) = \gcd(x_i, y_i)$
- (x_{i+1}, y_{i+1}) is "sufficiently" less than (x_i, y_i)

This chapter is based on lectures 2, 3 and 4.

This can be done by picking an x_i and updating each (x_i, y_i) to (x_i, r_i) where

$$r_i \in \left\{ -\left\lfloor \frac{x_i - 1}{2} \right\rfloor, \ldots, 0, 1, \ldots, \left\lfloor \frac{x_i}{2} \right\rfloor \right\} \tag{2.1.2}$$

Example 2.1.2. *Consider computing* $g = \gcd(468, 366, 552, 738, 822, 966, 264)$. *Using 468 as the divisor, we update the pairs* $(468, 366) \to (468, -102)$, $(468, 552) \to (468, 84)$, *...,* $(468, 264) \to (468, -204)$ *and obtain*

$$g = \gcd(468, -102, 84, -198, -114, 30, -204). \tag{2.1.3}$$

Using -102 as the divisor and performing similar updates, we get

$$g = \gcd(-102, 468, 84, -198, -114, 30, -204) = \gcd(-102, -42, -18, 6, -12, 30, 0). \tag{2.1.4}$$

Removing 0 from the list and using 6 as the divisor yields

$$g = \gcd(6, -102, -42, -18, 6, -12, 30) = \gcd(6, 0, 0, 0, 0, 0, 0) = 6, \tag{2.1.5}$$

and we conclude that $\gcd(468, 366, 552, 738, 822, 966, 264) = 6$.

Formally Euclid's GCD algorithm is as follows.

Algorithm 2 Euclid's GCD Algorithm

INPUT: Given n integers $a_1, \ldots, a_n$.
OUTPUT: $\gcd(a_1, \ldots, a_n)$

Ensure $a_1 \neq 0$ by swapping.

Reduction Step:
for $i = 2 \ldots, n$ **do**
    $r_i \leftarrow a_i \mod |a_1|$
    if $r_i > \frac{|a_1|}{2}$ **then**
        $r_i \leftarrow r_i - |a_1|$
    end if
    $a_i \leftarrow r_i$
end for
Drop all the zero elements.
if a_1 is the only non-zero element **then**
    return a_1
end if

Swap Step:
if There is some $a_i \neq 0$ left other than a_1 **then**
    Swap $a_1 \leftrightarrow a_i$
    Go to **Reduction Step**
end if

In the above algorithm, initially at 0th iteration the numbers were $(a_1, \ldots, a_n)$. After one iteration of the reduction step the numbers will be replaced by $(a_1, a_2', \ldots, a_n')$ where $|a_i| < \frac{|a_1|}{2}$ for all $i \geq 2$ and $\gcd(a_1, a_2', \ldots, a_n') = \gcd(a_1, \ldots, a_n)$. In the swap step a_1 is replaced by some a_i' whose absolute value is at most half of the absolute value of a_1 and perform reduction step with respect to that a_i' in next iteration. In general we maintain two conditions in the algorithm:

1. If after ith iteraion we have numbers $(a_1^{(i)}, \ldots, a_m^{(i)})$ and after $(i+1)$th iteraion we have numbers $(a_1^{(i+1)}, \ldots, a_s^{(i+1)})$ for $s \leq m$, then,
$\gcd(a_1^{(i)}, \ldots, a_m^{(i)}) = \gcd(a_1^{(i+1)}, \ldots, a_s^{(i+1)})$

2. And, $|a_1^{(i+1)}| < \frac{|a_1^{(i)}|}{2}$.

So, the algorithm correctly answers the gcd in $O(\mathsf{poly}(\max_i \log |a_i|), n))$ time.

Now we shall describe the algorithm of finding basis of a lattice from a given generating set, as a generalization of Euclid's GCD algorithm. The current presentation is taken from [BGPS23]. The same principle applies, where we keep updating each $\vec{a}_i$ by adding some integer multiplies of $\vec{a}_j$ where $j \neq i$, i.e., $\vec{a}_i \to \vec{a}_i + \sum_{j \neq i} x_{i,j}\vec{a}_j$ to make the length of $\vec{a}_i$ "shorter".

Example 2.1.3. *Consider finding a basis for the lattice generated by*

$$\vec{a}_1 = \begin{bmatrix} 28 \\ 0 \\ 0 \end{bmatrix}, \vec{a}_2 = \begin{bmatrix} 34 \\ -42 \\ 0 \end{bmatrix}, \vec{a}_3 = \begin{bmatrix} -50 \\ 90 \\ 0 \end{bmatrix}, \vec{a}_4 = \begin{bmatrix} 24 \\ 108 \\ 12 \end{bmatrix}, \vec{a}_5 = \begin{bmatrix} -92 \\ 0 \\ 0 \end{bmatrix}, \tag{2.1.6}$$

$$\vec{a}_6 = \begin{bmatrix} 62 \\ -2 \\ 8 \end{bmatrix}, \vec{a}_7 = \begin{bmatrix} -4 \\ 48 \\ 0 \end{bmatrix}, \vec{a}_8 = \begin{bmatrix} 81 \\ -25 \\ 1 \end{bmatrix}, \vec{a}_9 = \begin{bmatrix} -13 \\ -87 \\ -3 \end{bmatrix}. \tag{2.1.7}$$

Using the update rule

$$\vec{a}_2 \to \vec{a}_2 - \vec{a}_1 \tag{2.1.8}$$
$$\vec{a}_3 \to \vec{a}_3 + 2\vec{a}_2 + \vec{a}_1 \tag{2.1.9}$$
$$\vec{a}_4 \to \vec{a}_4 + 3\vec{a}_2 - \vec{a}_1 \tag{2.1.10}$$
$$\vec{a}_5 \to \vec{a}_5 + 3\vec{a}_1 \tag{2.1.11}$$
$$\vec{a}_6 \to \vec{a}_6 - 8\vec{a}_8 - 33\vec{a}_3 - 32\vec{a}_5 \tag{2.1.12}$$
$$\vec{a}_7 \to \vec{a}_7 - 8\vec{a}_3 - \vec{a}_1 + 6\vec{a}_5 \tag{2.1.13}$$
$$\vec{a}_8 \to \vec{a}_8 + 4\vec{a}_3 + 5\vec{a}_5 \tag{2.1.14}$$
$$\vec{a}_9 \to \vec{a}_9 + 2\vec{a}_2, \tag{2.1.15}$$

we now only have to find a basis for the lattice generated by

$$\begin{bmatrix} 28 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 6 \\ -42 \\ 0 \end{bmatrix}, \begin{bmatrix} -10 \\ 6 \\ 0 \end{bmatrix}, \begin{bmatrix} 14 \\ -18 \\ 12 \end{bmatrix}, \begin{bmatrix} -8 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -3 \\ -3 \end{bmatrix}. \tag{2.1.16}$$

In the example above, the coefficients $x_{i,j}$ of $\vec{a}_i$ are determined by "trial and error" during the lecture. We now provide a more systematic way to do this, which is similar to how the coefficient $\mu_{i,j}$ is obtained in Line 3 of GSO. The main difference being that we pick the coefficient to be the nearest integer to $\mu_{i,j}$, since $\mu_{i,j}$ may not be an integer in general. This gives us the LLL algorithm:

Algorithm 3 LLL algorithm [LLL82]

Input: $\vec{a}_1, ..., \vec{a}_N \in \mathbb{Z}^d$ (Generating set of a lattice.)
Output: Basis of $\mathcal{L}(\vec{a}_1, ..., \vec{a}_N)$

Reduction Step:
Compute $\vec{a}_1^*, ..., \vec{a}_N^* = \text{GSO}(\vec{a}_1, ..., \vec{a}_N)$
for $i = 2$ to N **do**
  for $j = i - 1$ to 1 **do**
    if $\vec{a}_j^* \neq 0$ **then**
      $\mu_{i,j} := \dfrac{\langle \vec{a}_i, \vec{a}_j^* \rangle}{\langle \vec{a}_j^*, \vec{a}_j^* \rangle}$
      $m \leftarrow$ nearest integer to $\mu_{i,j}$
      $\vec{a}_i \leftarrow \vec{a}_i - m\vec{a}_j$
    end if
  end for
end for
Remove all zero vectors and update N

Swap Step:
if there exists i such that $\vec{a}_i^* = 0$ **then**
  Swap the order of $\vec{a}_i$ and $\vec{a}_k$ where $k < i$ is the smallest index satisfying $\vec{a}_i \in \text{span}(\vec{a}_1, ..., \vec{a}_k)$.
  Go to **Reduction Step**
end if
Return $\vec{a}_1, ..., \vec{a}_N$

Remark 2.1.4. *Note that $\mathcal{L}(\vec{a}_1, ..., \vec{a}_N)$ remains unchanged throughout the algorithm since the operations on the vectors involved only swapping, adding an integer multiple of another basis vector $\vec{a}_i \leftarrow \vec{a}_i - m\vec{a}_j$, and removal of zero vectors.*

Remark 2.1.5. *We expect the length of each $\vec{a}_i$ to go down after each reduction step since we are subtracting from $\vec{a}_i$ the components that are "orthogonal".*

Remark 2.1.6. *The algorithm terminates when there is no i so that $\vec{a}_i^* = 0$. $\vec{a}_i^* = 0$ if and only if $\vec{a}_i$ is in span of $\vec{a}_1, \ldots, \vec{a}_{i-1}$. So, we stop when we get a linearly independent set of vectors for the lattice.*

We now show that the LLL algorithm terminates in time poly(n), where n is the rank of $\mathcal{L}(\vec{a}_1, ..., \vec{a}_N)$. Note that for each iteration of LLL, the reduction step takes polynomial time and the swap step takes $O(1)$ times, and so it is sufficient to show that we have to repeat the two steps only a polynomial number of times.

Theorem 2.1.7. *Let $\vec{a}_1, ..., \vec{a}_N$ be an input of Algorithm 3, and let $\mathcal{L}(\vec{a}_1, ..., \vec{a}_N)$ be the lattice generated as defined in (2.0.1) with $n = \text{rank}(\mathcal{L})$. Then the reduction step and swap step are run only poly(n) times.*

Proof. We first define a potential function at iteration $t \geq 1$ as follows:

$$P_t = P_t(\vec{a}_1, ..., \vec{a}_N) = \prod_{i:\vec{a}_i^* \neq 0} \|\vec{a}_i^*\|, \tag{2.1.17}$$

where $\vec{a}_1, ..., \vec{a}_N$ are the vectors at the start of iteration t and $\vec{a}_1^*, ..., \vec{a}_N^* = \text{GSO}(\vec{a}_1, ..., \vec{a}_N)$.

Note that the initial value of P is upper bounded by

$$P_1(\vec{a}_1, ..., \vec{a}_N) = \prod_{i:\vec{a}_i^* \neq 0} \|\vec{a}_i^*\| \leq \prod_{i:\vec{a}_i^* \neq 0} \|\vec{a}_i\| \leq (\max_i \|\vec{a}_i\|)^n, \tag{2.1.18}$$

where the inequalities follow from $\|\vec{a}_i^*\| \leq \|\vec{a}_i\|$ and $n = \text{rank}(\mathcal{L})$. Note that the vectors $\vec{a}_1, ..., \vec{a}_N$ here are the input of Algorithm 3, i.e., the vectors at the start of the first iteration. We also have

$$P_{\text{final}}(\vec{a}_1, ..., \vec{a}_N) = P_{\text{final}}(\text{basis of } \mathcal{L}) = \det(\mathcal{L}) \geq 1, \tag{2.1.19}$$

where the equality follows from (1.5.3) and the inequality follows from the assumption that the lattice vectors are integer-valued.
We now show that the value of P goes down by at least a factor of 2 every time swap step is run, which implies that the algorithm will terminate after at most $n \log(\max_i \|\vec{a}_i\|)$ iterations. Suppose we swap the order of $\vec{a}_i$ and $\vec{a}_k$ for some $k < i$. By Lines 11–13, we have $\vec{a}_i \neq 0$, $\vec{a}_i^* = 0$, $\vec{a}_k^* \neq 0$ and $\vec{a}_i \in \text{span}(\vec{a}_1, ..., \vec{a}_k) = \text{span}(\vec{a}_1^*, ..., \vec{a}_k^*)$. Furthermore, we can write

$$\vec{a}_i = \sum_{j=1}^{k} x_j \vec{a}_j \tag{2.1.20}$$

for some $x_k \neq 0$ since k is the smallest such index. By argument similar to (1.7.2), we have

$$\pi_k(\vec{a}_i) \coloneqq \pi_{\vec{a}_k^*} \vec{a}_i = \frac{\langle \vec{a}_i, \vec{a}_k^* \rangle}{\langle \vec{a}_k^*, \vec{a}_k^* \rangle} \vec{a}_k^* = x_k \vec{a}_k^*, \tag{2.1.21}$$

where

$$x_k = \frac{\langle \vec{a}_i, \vec{a}_k^* \rangle}{\langle \vec{a}_k^*, \vec{a}_k^* \rangle} = \frac{\langle \vec{a}_i^{\text{old}}, \vec{a}_k^* \rangle}{\langle \vec{a}_k^*, \vec{a}_k^* \rangle} - m \frac{\langle \vec{a}_k, \vec{a}_k^* \rangle}{\langle \vec{a}_k^*, \vec{a}_k^* \rangle} = \frac{\langle \vec{a}_i^{\text{old}}, \vec{a}_k^* \rangle}{\langle \vec{a}_k^*, \vec{a}_k^* \rangle} - m \in \left[-\frac{1}{2}, \frac{1}{2} \right]. \tag{2.1.22}$$

Here, $\vec{a}_i^{\text{old}}$ is the $\vec{a}_i$ before the update on Line 7 with respect to $\vec{a}_k^*$, and m is the nearest integer to $\frac{\langle \vec{a}_i^{\text{old}}, \vec{a}_k^* \rangle}{\langle \vec{a}_k^*, \vec{a}_k^* \rangle}$. It follows that

$$\|\pi_k(\vec{a}_i)\| \leq \frac{1}{2} \|\vec{a}_k^*\|. \tag{2.1.23}$$

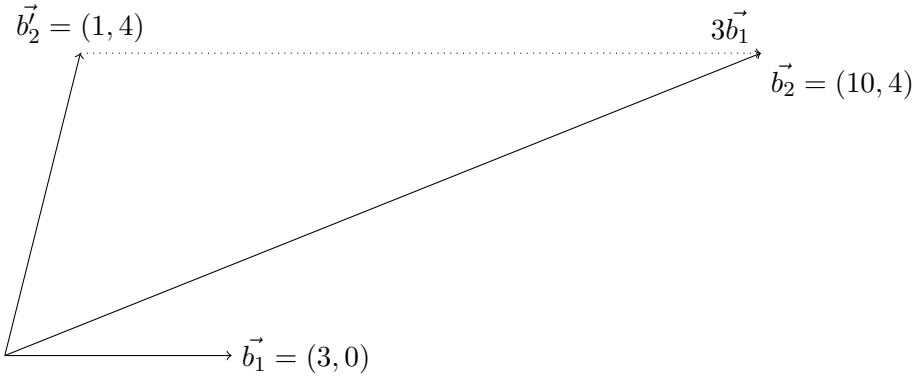
After the swap of $\vec{a}_i$ and $\vec{a}_k$, in the next iteration, we have $\vec{a}_j^*$ remains the same for $j \neq k$ and $\|\vec{a}_k^*\| \leq \frac{1}{2} \|\vec{a}_k^{\text{old},*}\|$, and so P goes down by at least a factor of 2 as claimed. □

2.2 LLL as Basis Reduction

Often, we would have a set of vectors, and we would be interested in understanding the lattice generated by that set. In the last lecture we saw the LLL algorithm, which can be used to find a basis for the lattice generated by an arbitrary set of vectors over rationals. An important problem in the field of lattices is the *shortest vector problem* (SVP). Given a lattice, it asks to find the shortest non-zero vector in that lattice. We have already seen some nice properties of such vectors in the second lecture (Minkowski's theorems.) Today we will see that the LLL algorithm can in fact be used to solve an *approximate* version of this problem. In fact, the LLL algorithm can be seen as an example of *basis reduction*, where we are given a lattice basis, and the goal is to find *better basis* vectors for that lattice. Qualitativey, a basis is better if its vectors are short and orthogonal. We begin by looking at the SVP in just two dimensions.

2.2.1 SVP in 2 Dimensions

Suppose we are given two vectors $\vec{b}_1, \vec{b}_2$, such that $\left\|\vec{b}_2\right\| > \left\|\vec{b}_1\right\|$, as in the figure below. What operations can we perform on them to make them *as orthogonal as possible*, while preserving the lattice generated?



We can subtract $\lfloor \mu_{2,1} \rceil \vec{b}_1$ from $\vec{b}_2$ to obtain $\vec{b}_2'$. Clearly, this does not change the lattice generated by the two vectors. What this achieves is that the coefficients of a particular basis vector, written in the GSO basis, become shorter. Thus, the inner products reduce, and vectors become more orthogonal.
We can now swap our vectors (if $\vec{b}_2'$ is now shorter than $\vec{b}_1$), and continue doing this. This idea can clearly be used to construct an algorithm for SVP in 2 dimensions. For simplicity, we present this for vectors over the integer field. Without loss of generality, we assume that $\left\|\vec{b}_2\right\| \geq \left\|\vec{b}_1\right\|$.

Algorithm 4 SVP in 2D

Input: $\left\{\vec{b}_1, \vec{b}_2\right\} \subset \mathbb{Z}^2$

Output: Shortest non-zero vector in $\mathcal{L}(\vec{b}_1, \vec{b}_2)$

1: $\vec{b}_2 \leftarrow \vec{b}_2 - \lfloor \mu_{2,1} \rceil \vec{b}_1$ ▷ $\mu_{2,1} = \dfrac{\langle \vec{b}_2, \vec{b}_1 \rangle}{\langle \vec{b}_1, \vec{b}_1 \rangle}$
2: **if** $0.9 \cdot \|\vec{b}_1\| > \|\vec{b}_2\|$ **then**
3: Swap($\vec{b}_1$, $\vec{b}_2$)
4: Go To Step 1
5: **end if**
6: Return the shorter of $\vec{b}_1, \vec{b}_2$

Lemma 2.2.1. *Algorithm 4 terminates after $O(\log \left\|\vec{b}_2\right\|)$ steps.*

Proof. Each Swap step will decrease the length of one of the vectors by at least a factor of 0.9. Therefore, the total number of Swap steps will be at most $2 \cdot \log_{1/0.9}(\max(\|\vec{b}_1\|, \|\vec{b}_2\|))$. □

Lemma 2.2.2. *Algorithm 4 outputs the shortest vector.*

Proof. Let $\vec{v}$ be a shortest non-zero vector in $\mathcal{L}$, and $\vec{b}_1, \vec{b}_2$ be the final basis vectors returned by the algorithm. Suppose $\|\vec{v}\| < \min(\|\vec{b}_1\|, \|\vec{b}_2\|)$. Since $\vec{b}_1, \vec{b}_2$ form a basis, we can write $\vec{v} = \alpha\vec{b}_1 + \beta\vec{b}_2$ where $\alpha, \beta \in \mathbb{Z}$. It is easy to see that $\alpha, \beta \neq 0$, because if any of the coefficients is zero, $\vec{v}$ is an integer multiple of a basis vector which contradicts the assumption that $\vec{v}$ is the shortest vector.

We claim that $|\beta| = 1$. To prove the claim, suppose that $|\beta| \geq 2$. Observe that the basis returned from the algorithm has the property

$$\|\vec{b}_2\| \geq 0.9 \cdot \|\vec{b}_1\|$$

Squaring on both sides,

$$\|\vec{b}_2\|^2 = \|\vec{b}_2^* + \mu_{2,1}\vec{b}_1\|^2 = \|\vec{b}_2^*\|^2 + \mu_{2,1}^2\|\vec{b}_1\|^2 \geq 0.81 \cdot \|\vec{b}_1\|^2$$

Due to rounding in line 1 of algorithm 4 and a similar analysis like (2.1.22), $|\mu_{2,1}| \leq 0.5$, therefore

$$\begin{aligned}\|\vec{b}_2^*\|^2 &\geq (0.81 - 0.25) \cdot \|\vec{b}_1\|^2 \\ &= 0.56\|\vec{b}_1\|^2 \\ &> 0.56\|\vec{v}\|^2\end{aligned}$$

The last inequality follows from our assumption that $\vec{v}$ is shorter than the returned vector. Since $|\beta| \geq 2$ and $\vec{b}_2^*, \vec{b}_1$ are orthogonal,

$$\begin{aligned}&\vec{v} = \alpha\vec{b}_1 + \beta\vec{b}_2 = \alpha\vec{b}_1 + \beta(\vec{b}_2^* + \mu_{2,1}\vec{b}_1) \\ &\implies \|v\| \geq \beta\|\vec{b}_2^*\| \geq 2\|\vec{b}_2^*\| > 2\sqrt{0.56}\|\vec{v}\| \geq \|\vec{v}\|,\end{aligned}$$

which is a contradiction. Therefore $|\beta| = 1$. Without loss of generality, we assume $\beta = 1$. Therefore,

$$\vec{v} = (\alpha + \mu_{2,1})\vec{b}_1 + \vec{b}_2^*$$

Since $\alpha \neq 0$, so $|\alpha| \geq 1$. WLOG we assume $\alpha \geq 1$. Let $\vec{u} = \vec{v} - \vec{b}_1$. Clearly $\vec{u} \in \mathcal{L}(\vec{b}_1, \vec{b}_2)$. Then,

$$\|u\|^2 = (\alpha + \mu_{2,1} - 1)^2||\vec{b}_1||^2 + ||\vec{b}_2^*||^2 < (\alpha + \mu_{2,1})^2||\vec{b}_1||^2 + ||\vec{b}_2^*||^2 = \|\vec{v}\|^2$$

Therefore $\vec{u}$ is a shorter vector than $\vec{v}$, which contradicts the assumption that $\vec{v}$ is the shortest vector. So $\|\vec{v}\| \geq \min(\|\vec{b}_1\|, \|\vec{b}_2\|)$ and one of the returned basis must be the shortest vector in lattice.

□

2.2.2 LLL Reduced Basis

LLL (Lenstra–Lenstra–Lovász) is one of the most important algorithms for basis reduction. It aims to reduce the size of the basis and can be used to find decently short vectors in the lattice in polynomial time.

Definition 2.2.3 (δ-LLL Reduced Basis)**.** *A basis $B = (\vec{b}_1, \ldots, \vec{b}_n)$ where $\vec{b}_i \in \mathbb{Z}^d$ is a δ-LLL reduced basis if the following holds*

1. Coefficient reduced. $\forall i \in [n], \forall j : 1 \leq j < i, |\mu_{i,j}| \leq \dfrac{1}{2}$
2. Lovász condition. $\forall i \in [n], \delta \left\|\vec{b}_i^*\right\| \leq \|\mu_{i+1,i}\vec{b}_i^* + \vec{b}_{i+1}^*\|$

where $\mu_{i,j} = \dfrac{\langle \vec{b}_i, \vec{b}_j^* \rangle}{\langle \vec{b}_j^*, \vec{b}_j^* \rangle}$ *and* $0.25 < \delta^2 \leq 1$

Remark 2.2.4. *The Lovász condition is equivalently stated as*

$$\left(\delta^2 - \frac{1}{4}\right)\left\|\vec{b}_i^*\right\|^2 \leq \left\|\vec{b}_{i+1}^*\right\|^2 \tag{2.2.1}$$

Proof. Condition 2 is equivalent to

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

Since $|\mu_{i+1,i}| \leq \frac{1}{2}$, we get

$$\begin{aligned}\delta^2\|\vec{b}_i^*\|^2 &\leq \frac{1}{4}\|\vec{b}_i^*\|^2 + \|\vec{b}_{i+1}^*\|^2\\ \implies \left(\delta^2 - \frac{1}{4}\right)\left\|\vec{b}_i^*\right\|^2 &\leq \left\|\vec{b}_{i+1}^*\right\|^2\end{aligned}$$

□

Remark 2.2.5. *Suppose $\delta^2 = \frac{3}{4}$. Then, $\left\|\vec{b}_i^*\right\|^2 \leq 2\left\|\vec{b}_{i+1}^*\right\|^2$.*

Lemma 2.2.6. *For a δ-LLL reduced basis $B = (\vec{b}_1, \ldots, \vec{b}_n)$ for lattice $\mathcal{L}$,*

$$\|\vec{b}_1\| \leq \left(\sqrt{\frac{1}{\delta^2 - 0.25}}\right)^{n-1}\lambda_1(\mathcal{L}) \tag{2.2.2}$$

Proof. From eq. (2.2.1), we get that

$$\begin{aligned}(\delta^2 - 0.25)\|\vec{b}_i^*\|^2 &\leq \|\vec{b}_{i+1}^*\|^2\\ \|\vec{b}_i^*\| &\leq \left(\sqrt{\frac{1}{\delta^2 - 0.25}}\right)\|\vec{b}_{i+1}^*\|\end{aligned}$$

Chaining these inequalities over $i \in [n]$ we get that

$$\begin{aligned}\|\vec{b}_1\| = \|\vec{b}_1^*\| &\leq \left(\sqrt{\frac{1}{\delta^2 - 0.25}}\right)\|\vec{b}_2^*\|\\ &\leq \left(\sqrt{\frac{1}{\delta^2 - 0.25}}\right)^2\|\vec{b}_3^*\|\\ &\vdots\\ &\leq \left(\sqrt{\frac{1}{\delta^2 - 0.25}}\right)^{n-1}\|\vec{b}_n^*\|\end{aligned}$$

We know that $\lambda_1(\mathcal{L}) \geq \min_i \|\vec{b}_i^*\|$. Minimizing over all i, we achieve the desired result. □

Remark 2.2.7. *For a $\sqrt{\frac{3}{4}}$-reduced LLL basis, $\|\vec{b}_1\| \leq (\sqrt{2})^{n-1}\lambda_1(\mathcal{L})$. That is, $\vec{b}_1$ is a $(\sqrt{2})^{n-1}$ approximation to the shortest non-zero vector.*

Remark 2.2.8. *The condition that $0.25 < \delta^2$ ensures that the approximation factor is always well-defined. A higher factor will lead to a better approximation, however, polynomial runtime is only guaranteed when $\delta^2 < 1$.*

2.2.3 Algorithm

Algorithm 5 LLL

Input: Lattice basis $B = (\vec{b}_1, \ldots, \vec{b}_n)$
Output: $\delta-$LLL reduced basis for $\mathcal{L}(B)$

1: Perform Gram-Schmidt orthogonalization on B to get $(\vec{b}_1^*, \ldots, \vec{b}_n^*)$
`/* Reduction Step */`
2: **for** $i = 1$ to n **do**
3: **for** $j = i - 1$ to 1 **do**
4: $\vec{b}_i \leftarrow \vec{b}_i - \lfloor \mu_{i,j} \rceil \vec{b}_j$ ▷ $\mu_{i,j} = \dfrac{\langle \vec{b}_i, \vec{b}_j^* \rangle}{\langle \vec{b}_j^*, \vec{b}_j^* \rangle}$
5: **end for**
6: **end for**
`/* Swap Step */`
7: **if** $\exists i$ such that $\delta \|\vec{b}_i^*\| > \|\mu_{i+1,i}\vec{b}_i^* + \vec{b}_{i+1}^*\|$ **then**
8: Swap$(\vec{b}_i, \vec{b}_{i+1})$
9: Go To Step 1
10: **end if**
11: Return B

2.2.4 Correctness

Lemma 2.2.9. *If the algorithm terminates, LLL produce a δ-LLL reduced basis for $\mathcal{L}(B)$*

Proof. To obtain a δ-LLL reduced basis, check if the returned basis satisfy the conditions in definition 2.2.3.

- Condition 1 is satisfied because of the reduction step.

  First notice that reducing $\vec{b}_i \leftarrow \vec{b}_i - \lfloor \mu_{i,j} \rceil \vec{b}_j$ for $j < i$ does not change $\vec{b}_i^*$. This is because $\vec{b}_i^* = \vec{b}_i - \sum_{j<i} \mu_{i,j}\vec{b}_j^*$ and we can represent $\vec{b}_j$ as $\vec{b}_j = \sum_{k \le j} c_k \vec{b}_k^*$ for some c_k. The change to $\vec{b}_i$ will be reflected in $\mu_{i,j}$. More precisely, $\mu_{i,j}$ will decrease by $\lfloor \mu_{i,j} \rceil c_j$. Therefore, summing them up will nullify the changes and give back the original $\vec{b}_i^*$. Note that the condition that $j < i$ is important to have this property

  Next, since $\vec{b}_j = \vec{b}_j^* + \sum_{k<j} \mu_{j,k}\vec{b}_k^*$, after the reduction the coefficient of $\vec{b}_j^*$ in $\vec{b}_i$ will be less than $\frac{1}{2}$ due to rounding, and doing the reduction from $j = n - 1$ to 1 ensures that further reduction will not affect the property achieved by previous reduction. Therefore condition 1 is satisfied

- Condition 2 is satisfied due to the swap steps. If the algorithm terminates, for all i, we will have $\delta \|\vec{b}_i^*\| \le \|\mu_{i+1,i}\vec{b}_i^* + \vec{b}_{i+1}^*\|$

□

Now we will show that the LLL algorithm runs in polynomial time if $0.25 < \delta^2 < 1$.

Definition 2.2.10. *For a basis $B := \left\{\vec{b}_1, \ldots \vec{b}_n\right\}$, define a potential Δ as,*

$$\Delta := \|\vec{b}_1^*\|^n \cdot \|\vec{b}_2^*\|^{n-1} \cdot \ldots \cdot \|\vec{b}_n^*\| = \prod_{i \le n} \|\vec{b}_i^*\|^{n-i} \tag{2.2.3}$$

Since $(\vec{b}_1^*, \ldots, \vec{b}_n^*)$ is orthogonal basis and Gram-Schmidt orthogonalization does not change the determinant, $\|\vec{b}_1^*\| \cdot \|\vec{b}_2^*\| \cdot \ldots \cdot \|\vec{b}_n^*\| = \det(\mathcal{L}(\vec{b}_1^*, \ldots, \vec{b}_n^*)) = \det(\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_n))$.Thus Δ can be rewriten as

$$\Delta = \det(\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_n)) \cdot \det(\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{n-1})) \cdot \ldots \cdot \det(\mathcal{L}(\vec{b}_1)) \tag{2.2.4}$$

Lemma 2.2.11. *LLL can have at most $O(n^2 \log(\max \|\vec{b}_i\|))$ swap steps*

Proof. Notice that for every swap step on i, all factors of Δ in eq. (2.2.4) stay the same except for one, which changes as:

$$\det(\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{i-1}, \vec{b}_i)) \to \det(\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{i-1}, \vec{b}_{i+1}))$$

Also a swap occurs when $\delta\|\vec{b}_i^*\| > \|\mu_{i+1,i}\vec{b}_i^* + \vec{b}_{i+1}^*\|$ which implies that

$$\delta^2\|\vec{b}_i^*\|^2 > \mu_{i+1,i}^2\|\vec{b}_i^*\|^2 + \|\vec{b}_{i+1}^*\|^2 \geq \|\vec{b}_{i+1}^*\|^2$$

$$\implies \delta\|\vec{b}_i^*\| > \|\vec{b}_{i+1}^*\|$$

So $\vec{b}_{i+1}^*$ is a factor δ smaller than $\vec{b}_i^*$, which means

$$\det(\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{i-1}, \vec{b}_{i+1})) < \delta \det(\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{i-1}, \vec{b}_i))$$

Hence after a swap step, the potential Δ will decrease at least by a factor of δ. This implies there will only be $O(\log_{1/\delta} \Delta)$ number of swaps. Since $\|\vec{b}_i\| \geq \|\vec{b}_i^*\|$, we can write the expression to,

$$O(\log_{1/\delta} \Delta) = O(\log_{1/\delta}(\max \|\vec{b}_i\|^{1+\cdots+n})) = O(n^2 \log(\max \|\vec{b}_i\|))$$

□

We still have one more thing to take care of. To prove that LLL runs in polynomial time, we must also make sure that the bit representation of vectors $\vec{b}_i^*$ and $\vec{b}_i$ at any stage do not grow exponentially during the reduction process. We first analyse $\vec{b}_i^*$:

Lemma 2.2.12. *$\forall i \in [n]$, we have $\Delta^2 \cdot \vec{b}_i^* \in \mathbb{Z}^d$ and $\dfrac{1}{\Delta^2} \leq \|\vec{b}_i^*\| \leq \Delta^2$*

Remark 2.2.13. *If this lemma is true, then $\vec{b}_i^*$ can be represented with $O(\log \Delta)$ bits*

Proof. First, let's prove $\Delta^2 \cdot \vec{b}_i^* \in \mathbb{Z}^d$. We start by writing each basis vector in the GSO basis:

$$\vec{b}_i = \vec{b}_i^* + \sum_{j<i} \mu_{i,j}\vec{b}_j^*$$

Let's say that

$$\vec{b}_i^* = \vec{b}_i + a_1\vec{b}_1 + \cdots + a_{i-1}\vec{b}_{i-1} \tag{2.2.5}$$

where $a_1, \ldots, a_{i-1} \in \mathbb{R}$. Since b_i^* is orthogonal to b_j for all $j < i$, taking inner products of b_i^* with b_j gives $i-1$ linear equations where the unknowns are a_j:

$$\begin{aligned}
0 = \langle b_i^*, b_1\rangle &= \langle \vec{b}_i, \vec{b}_1\rangle + a_1\langle \vec{b}_1, \vec{b}_1\rangle + \cdots + a_{i-1}\langle \vec{b}_{i-1}, \vec{b}_1\rangle \\
0 = \langle b_i^*, b_2\rangle &= \langle \vec{b}_i, \vec{b}_2\rangle + a_1\langle \vec{b}_1, \vec{b}_2\rangle + \cdots + a_{i-1}\langle \vec{b}_{i-1}, \vec{b}_2\rangle \\
&\vdots \\
0 = \langle b_i^*, b_j\rangle &= \langle \vec{b}_i, \vec{b}_j\rangle + a_1\langle \vec{b}_1, \vec{b}_j\rangle + \cdots + a_{i-1}\langle \vec{b}_{i-1}, \vec{b}_j\rangle
\end{aligned}$$

Notice that the coefficients of the equations are all integers. We can solve the system of linear equations using Cramer's rule. The value of each variable a_j will be

$$a_j = \frac{x_j}{\begin{vmatrix} \langle \vec{b}_1, \vec{b}_1 \rangle & \dots & \langle \vec{b}_{i-1} \vec{b}_1 \rangle \\ \langle \vec{b}_1, \vec{b}_2 \rangle & \dots & \langle \vec{b}_{i-1} \vec{b}_2 \rangle \\ \vdots & \ddots & \vdots \\ \langle \vec{b}_1, \vec{b}_{i-1} \rangle & \dots & \langle \vec{b}_{i-1} \vec{b}_{i-1} \rangle \end{vmatrix}} = \frac{x_j}{\det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_{i-1}))^2} \tag{2.2.6}$$

for some $x_j \in \mathbb{Z}$. Notice that if we multiply eq. (2.2.5) by the denominator of eq. (2.2.6), we get an integer vector. Then, by definition of Δ, we get:

$$\det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_{i-1}))^2 \cdot \vec{b}_i^* \in \mathbb{Z}^d \implies \Delta^2 \cdot \vec{b}_i^* \in \mathbb{Z}^d$$

This completes the proof of the first part. For the second part, since $\det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_{i-1}))^2 \cdot \vec{b}_i^* \in \mathbb{Z}^d$, and it is a non-zero vector, the ℓ_2 norm must be at least 1. This gives us one side of the required inequality:

$$\det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_{i-1}))^2 \cdot \|\vec{b}_i^*\| \geq 1 \implies \|\vec{b}_i^*\| \geq \frac{1}{\det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_{i-1}))^2} \geq \frac{1}{\Delta^2}$$

For the other side,

$$\begin{aligned} \|\vec{b}_i^*\| \cdot \prod_{j<i} \|\vec{b}_j^*\| &= \det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_i)) \\ \implies \|\vec{b}_i^*\| &= \frac{\det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_i))}{\prod_{j<i} \|\vec{b}_j^*\|} \\ &\leq \det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_i)) \prod_{j<i} \det(\mathcal{L}(\vec{b}_1, \dots, \vec{b}_j))^2 \\ &\leq \Delta^2 \end{aligned}$$

□

Remark 2.2.14. *Notice that this is a rather loose bound. However, we just want to prove that we can represent the GSO vectors with polynomial (in n) numbers of bits.*

Now let's look at $\vec{b}_i$.

Lemma 2.2.15. *After the reduction step, $\|\vec{b}_i\| \leq n\Delta^2$*

Proof. With

$$\vec{b}_i = \vec{b}_i^* + \sum_{j<i} \mu_{i,j} \vec{b}_j^*$$

After the reduction step,

$$\|\vec{b}_i\| \leq \|\vec{b}_i^*\| + \sum_{j<i} \frac{1}{2} \|\vec{b}_j^*\| \leq n\Delta^2$$

where the last inequality is due to lemma 2.2.12 □

Lemma 2.2.16. *During the reduction step, $\|\vec{b}_i\| \leq n\Delta^2 (2n\Delta^4)^{n-1}$*

Proof. While reducing the i vector, $\vec{b}_i \leftarrow \vec{b}_i - \lfloor \mu_{i,j} \rceil \vec{b}_j$

$$\begin{aligned}
\|\vec{b}_i - \lfloor \mu_{i,j} \rceil \vec{b}_j\| &\leq \|\vec{b}_i\| + |\lfloor \mu_{i,j} \rceil| \vec{b}_j \\
&\leq \|\vec{b}_i\| + \left(1 + \left|\frac{\langle \vec{b}_i, \vec{b}_j^* \rangle}{\langle \vec{b}_j^*, \vec{b}_j^* \rangle}\right|\right) \|\vec{b}_j\| \\
&\leq \|\vec{b}_i\| + \left(1 + \frac{\|\vec{b}_i\| \|\vec{b}_j^*\|}{\|\vec{b}_j^*\|^2}\right) \|\vec{b}_j\| \\
&= \|\vec{b}_i\| + \|\vec{b}_j\| + \frac{\|\vec{b}_i\| \|\vec{b}_j\|}{\|\vec{b}_j^*\|}
\end{aligned}$$

Now use lemma 2.2.12 and lemma 2.2.15,

$$\|\vec{b}_i - \lfloor \mu_{i,j} \rceil \vec{b}_j\| \leq \|\vec{b}_i\|(1 + n\Delta^4) + n\Delta^2 \leq 2n\Delta^4 \|\vec{b}_i\|$$

Since at most $n - 1$ steps is performed, the final $\vec{b}_i$ is at most

$$(2n\Delta^4)^{n-1} \|\vec{b}_i\| \leq n\Delta^2 (2n\Delta^4)^{n-1}$$

□

Corollary 2.2.17. *LLL terminates in polynomial time.*

Proof. From lemma 2.2.12, lemma 2.2.15, lemma 2.2.16, all the vectors in the algorithm can be represented in $O(n \log \Delta)$ bits. From lemma 2.2.11, it is clear that the algorithm runs in polynomial time □

Remark 2.2.18. *In practice, the swapping of vectors can be done for all vectors i that don't satisfy condition 2 before jumping back to the first step. There are also many possible optimizations to reduce the runtime dependence on the bit size of the vector. This will increase the overall runtime of the algorithm. The current known fastest implementation of LLL is from [RH23], which has a heuristic running time of $O(n^\omega (\log \|B\|_{max} + n)^{1+\epsilon})$, where $\omega \in (2, 3]$ and B is the basis. To play around with LLL, you may use [dt23].*

2.3 Application of LLL: Cryptanalysis of Subset Sum based Cryptosystems

The LLL algorithm is probably the most important lattice algorithm. Today we will see how it can be used for cryptanalysis of a certain subset sum based cryptosystem.

2.3.1 Public-Key Encryption

Suppose two parties (say our old friends Alice and Bob) want to communicate securely. A *public key encryption scheme* is a protocol that allows for this. Alice publishes a *public key*, which Bob can use to encrypt the message she wishes to send to Alice. Alice retains a *private key*, that allows her to decrypt the message that Bob sent. Any eavesdropper who does not know the secret key can't decrypt the messages. The protocol itself consists of a key generation mechanism, and reliable encryption and decryption mechanisms. It is defined as follows.

Definition 2.3.1 (Public-Key Encryption Scheme)**.** *A public-key encryption scheme consists of three polynomial time algorithms (Gen, Enc, Dec):*

- *$Gen(1^\lambda) \to (pk, sk)$: A probabilistic key generation algorithm that takes a security parameter λ and outputs a public key pk and a secret key sk.*
- *$Enc(pk, m) \to c$: A probabilistic encryption algorithm that takes the public key pk and a message m, and outputs a ciphertext c.*
- *$Dec(sk, c) \to m$ or $\perp$: A deterministic decryption algorithm that takes the secret key sk and a ciphertext c, and outputs either a message m or an error symbol $\perp$.*

The scheme must satisfy the correctness property: for all messages m in the message space,

$$\Pr[Dec(sk, Enc(pk, m)) = m] = 1 - negl(\lambda)$$

where $(pk, sk) \leftarrow Gen(1^\lambda)$ and $negl(\lambda)$ is a negligible function in λ.

We will need to define a *trapdoor one way function*, which is a function that is hard to invert in general, except if you have a secret *trapdoor*, which can be used to invert the function in polynomial time. Given such a function, it should be clear that the trapdoor can act as a secret key that can allow one to decrypt messages that are encrypted using such functions.

Definition 2.3.2 (Trapdoor One-Way Function)**.** *A trapdoor one-way function is a tuple of three polynomial-time algorithms (G, F, F^{-1}):*

- *$G(1^\lambda) \to (ek, td)$: A probabilistic algorithm that generates an evaluation key ek and a trapdoor td.*
- *$F(ek, x) \to y$: A deterministic algorithm that evaluates the function on input x using the evaluation key ek to produce output y.*
- *$F^{-1}(td, y) \to x$: A deterministic algorithm that inverts y to recover x using the trapdoor td.*

The function must satisfy:

1. *One-wayness: For any probabilistic polynomial-time adversary $\mathcal{A}$,*

   $$\Pr[F(ek, \mathcal{A}(ek, F(ek, x))) = F(ek, x)] \leq negl(\lambda)$$

   where $(ek, td) \leftarrow G(1^\lambda)$, x is chosen uniformly from the domain, and $negl(\lambda)$ is a negligible function in λ.

2. *Trapdoor property: For all x in the domain,*

   $$\Pr[F^{-1}(td, F(ek, x)) = x] = 1 - negl(\lambda)$$

   where $(ek, td) \leftarrow G(1^\lambda)$.

2.3.2 Subset-Sum

Definition 2.3.3 (Subset-Sum Problem)**.** *Given a set of non-negative real numbers $A = \{a_1, \ldots, a_n\}$ and a target sum S, the Subset-Sum Problem asks whether there exists a vector $x \in \{0, 1\}^n$ such that:*

$$\sum_{i=1}^{n} x_i \cdot a_i = S$$

More formally:

- ***Input:*** *A vector* $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_{\geq 0}$ *and a target* $S \in \mathbb{Z}$.
- ***Question:*** *Does there exist a vector* $\vec{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ *such that* $\vec{a} \cdot \vec{x} = t$*?*

The search version of the problem asks to find such a subset (or equivalently, the vector $\vec{x}$*) if it exists.*

Remark 2.3.4. *The Subset-Sum Problem is known to be NP-complete [Goo22], which means it is believed to be computationally hard in the worst case.*

Example 2.3.5 (Easy Instance of Subset-Sum Problem)**.** *An easy instance of the Subset-Sum Problem occurs when the sequence* $\{a_1, \ldots, a_n\}$ *is superincreasing, i.e., when:*

$$a_i > \sum_{j=1}^{i-1} a_j \quad \textit{for all } i = 2, \ldots, n$$

For example, the sequence $(1, 3, 7, 15, 31)$ *is superincreasing because:*

- $3 > 1$
- $7 > 1 + 3 = 4$
- $15 > 1 + 3 + 7 = 11$
- $31 > 1 + 3 + 7 + 15 = 26$

For such instances, the subset-sum problem can be solved efficiently using a greedy algorithm:

1. *Start with the largest element* a_n *(binary search).*
2. *If* $a_n \leq t$*, include it in the subset and subtract it from* t*.*
3. *Continue with* $a_{n-1}, a_{n-2}, \ldots$ *until either* $t = 0$ *(solution found) or all elements have been considered.*

This property is crucial in certain subset-sum based cryptosystems, where the ease of solving superincreasing instances is used as a trapdoor. Note that the superincreasing property is easy to hide from a sequence using modular arithmetic. One simply samples a large prime number P and a random integer $M \in \mathbb{Z}_P^*$, and computes a new sequence $\vec{a}' = M \cdot \vec{a} \bmod P$. This sequence can then be published as the public key. If a sender wants to send a message $\vec{x} \in \{0,1\}^n$, she computes the sum $S := \sum_j a'_j x_j$, and send it to the receiver. For an evesdropper who does not know P, M, this is an instance of the subset sum problem, where the sequence is not superincreasing. But the receiver who knows P, M can easily invert the initial map, and recover an instance of subset sum over the initial superincreasing sequence. Let's see what such a function formally looks like.

2.3.3 A Cryptosystem Based on Subset Sum

We construct a trapdoor One-Way Function Based on Subset-Sum. Let $\vec{a} = (a_1, \ldots, a_n)$ be a superincreasing sequence as defined in the previous example and $a_i \in [2^n]$. We construct a trapdoor one-way function as follows:

1. **Key Generation** $G(1^\lambda)$:
   - Sample a large prime P with $|P| = O(2^{n^2})$.

- Sample $M \xleftarrow{random} \mathbb{Z}_P^*$.
- Compute $\vec{a}' = M \cdot \vec{a} \bmod P = (Ma_1 \bmod P, \ldots, Ma_n \bmod P)$.
- Output evaluation key $ek = \vec{a}'$ and trapdoor $td = (M, P)$.

2. **Function Evaluation** $F(ek, \vec{x})$:
   - Input: $\vec{x} \in \{0,1\}^n$.
   - Output: $c = \langle \vec{x}, \vec{a}' \rangle = \sum_{i=1}^{n} x_i a_i'$.
3. **Inversion** $F^{-1}(td, c)$:
   - Compute $S = c \cdot M^{-1} \bmod P$.
   - Note that $S = \langle \vec{x}, \vec{a} \rangle \bmod P = \langle \vec{x}, \vec{a} \rangle$, since P is much larger than any subset-sum of $\vec{a}$.
   - Recover $\vec{x}$ by solving the easy instance of the subset-sum problem with target sum S and set $\vec{a}$.

The security of this construction relies on the hardness assumption of the worst case subset-sum problem. Without knowledge of M and the easy instance $\vec{a}$, finding $\vec{x}$ (decryption) given $\vec{a}'$ and c would be computationally difficult. The security of the trapdoor one way function described above relies of the fact that a random instance of the subset sum problem with large coefficients (as produced by the key generation algorithm above) would be a hard instance. Interestingly, this is not the case, (no) thanks to the LLL algorithm. It turns out that the $2^{n/2}$-approximation (to the shortest vector) guarantee we saw in the last lecture is enough to solve such *sparse* instances.

2.3.4 Breaking the Subset-Sum Based Trapdoor One-Way Function Using LLL

We now demonstrate how the Lenstra-Lenstra-Lovász (LLL) algorithm can be used to break the subset-sum based trapdoor one-way function described earlier. This attack showcases the power of lattice reduction techniques in cryptanalysis. The idea is to come up with a lattice (using the given set and the target), such that a $2^{n/2}$-approximate shortest vector is the solution to the sparse subset sum instance with high probability.

Problem Statement. Let $\mathcal{M} := 2^{4n^2}$. Given $\vec{a} \in \mathbb{Z}^n$, such that each entry a_i is sampled independently, uniformly at random from $U := \left\{\frac{1}{2}\mathcal{M}, \frac{1}{2}(\mathcal{M}+1), \ldots, \mathcal{M}\right\}$, and a target S computed as $S := \langle \vec{a}, \vec{x} \rangle$, recover $\vec{x}$.
Note that finding $\vec{x}$ amounts to recovering the message.

Remark 2.3.6. *Even though the analysis below relies on the instance (the set $\vec{a}$) being chosen randomly, in practice the LLL has seen a lot more success breaking subset sum based cryptosystems.*

Consider the lattice generated by the following basis.

$$B = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 2^n \cdot S & -2^n \cdot a_1 & -2^n \cdot a_2 & \cdots & -2^n \cdot a_n \end{pmatrix} \tag{2.3.1}$$

The lattice $\mathcal{L}(B)$ contains the vector $\vec{b_0} + \sum_{i=1}^{n} x_i \vec{b_i} = (\vec{x}, 0)^T$ (by construction). Because $\vec{x}$ is a binary vector, $\left\|(\vec{x}, 0)^T\right\|_2 = \|\vec{x}\|_2 \leq \sqrt{n}$. We can use the LLL algorithm to find a lattice vector of length at most $\|\vec{x}\|_2 \cdot 2^{n/2} \ll 2^n$. We now show that any such short vector recovered by the LLL algorithm is a multiple of $(\vec{x}, 0)^T$.

Lemma 2.3.7. *With high probability, the only vectors $\vec{z} \in \mathcal{L}(B)$ with $\|\vec{z}\| < 2^n$ are multiples of $(\vec{x}, 0)^T$.*

Proof. Let $\vec{z} \in \mathcal{L}(B)$ such that $\|\vec{z}\|_2 < 2^n$. Then there exist $\alpha, y_1, y_2, \ldots y_n \in \mathbb{Z}$ such that $\vec{z} = \alpha\vec{b_0} + \sum_{i=1}^{n} y_i \vec{b_i}$. Since $|z_n| = 2^n \cdot |(\alpha S - \sum_{i=1}^{n} y_i a_i)| < 2^n$, we get that

$$\alpha S = \sum_{i=1}^{n} y_i a_i \tag{2.3.2}$$

Now, since $\vec{x}$ is a binary vector, $\alpha S \geq \frac{1}{2}\mathcal{M}\alpha$. Also, $\|\vec{z}\|_2 < 2^n \implies \|\vec{y}\|_\infty \leq 2^n \implies \sum_{i=1}^{n} y_i a_i \leq n\mathcal{M}2^n$. Putting these two inequalities together with eq. (2.3.2) gives us $\alpha \leq 2n2^n$.
Next, observe that the number of triplets $(\vec{x}, \vec{y}, \alpha)$, such that $\vec{x} \in \{0,1\}^n, \|\vec{y}\|_\infty \leq 2^n, |\alpha| \leq 2n2^n$ is bounded by $2^n \cdot 2^{n(n+1)} \cdot 2n2^n \leq 2^{3n^2}$. Let's see what is the probability that eq. (2.3.2) is satisfied for a fixed $(\vec{x}, \vec{y}, \alpha)$ such that $\vec{y} \notin \mathbb{Z}\vec{x}$. That is, we want to upper bound

$$\Pr_{\vec{a} \sim U^n}\left[\alpha S = \sum_{i=1}^{n} y_i a_i\right] \tag{2.3.3}$$

By our assumption, $\exists j \in [n] : y_j \neq \alpha x_j$. Suppose we fix the values of $a_1, \ldots a_{j-1}, a_{j+1}, \ldots a_n$, and pick only a_j randomly. Then the picked a_j has to satisfy $a_j(y_j - \alpha x_j) = \alpha \sum_{i \neq j} x_i a_i - \sum_{i \neq j} y_i a_i$. Clearly, there is at most one choice of a_j that satisfies this. Therefore, the probability in the above equation is bounded by

$$\Pr_{\vec{a} \sim U^n}\left[\alpha S = \sum_{i=1}^{n} y_i a_i\right] \leq (\mathcal{M}/2 + 1)^{-1} \leq \left(2^{4n^2 - 1}\right)^{-1}. \tag{2.3.4}$$

Applying union bound over all possible triplets $(\vec{x}, \vec{y}, \alpha)$ (a maximum of 2^{3n^2} such triplets), for our choice of $\mathcal{M}$, we get our desired result. □

Chapter 3

Closest Vector Problem

Lattice-based cryptography has emerged as a promising approach for constructing secure cryptosystems. It allows constructions of cryptosystems based on *average case* hardness assumptions of key lattice problems, such as the shortest vector problem. Notably, Regev [Reg05] established a reduction from worst case lattice problems (GapSVP and SIVP) to the *Learning With Errors (LWE)* problems, and showed that it is possible to build public key cryptosystems using this problem. This allowed cryptosystems based on the hardness of polynomial-time approximations of the Shortest Vector Problem (SVP). We first define some lattice problems, and give an overview of the hardness of lattice problems.

Definition 3.0.1 (Decision SVP)**.** *Given a lattice basis B and a distance d, output* YES *if the shortest non-zero vector has length $\lambda(\mathcal{L}) \leq d$, and* NO *otherwise.*

Definition 3.0.2 (GapSVP$_\gamma$)**.** *Given a lattice basis B, a parameter γ and a distance d, output* YES *if the shortest non-zero vector has length $\lambda(\mathcal{L}) \leq d$, and* NO *if the length is $\lambda(\mathcal{L}) > \gamma d$.*

Definition 3.0.3 (γ-approximate SVP or SVP$_\gamma$)**.** *Given a lattice basis B and an approximation factor γ, output a vector $v \in \mathcal{L}$ such that $\|v\| \leq \gamma\lambda(\mathcal{L})$.*

Clearly, if SVP$_\gamma$ can be solved, GapSVP$_\gamma$ can be solved. SVP$_\gamma$ is interesting because we get surprising results for almost all approximation factors:

1. $1 \leq \gamma \leq \mathcal{O}(1) \implies$ SVP$_\gamma$ is NP-hard. Best known algorithm run in $\mathcal{O}(2^n)$ time.
2. $\gamma = \mathcal{O}(\sqrt{n}) \implies$ SVP$_\gamma \in$ coNP.
3. If $\gamma = \mathcal{O}(n)$ is hard, we have constructions for one-way functions and collision-resistant hash functions (minicrypt).
4. If $\gamma = \mathcal{O}(n^{3/2})$ is hard, we have constructions for public key cryptography (cryptomania).
5. $\gamma > 2^{n \log\log n / \log n}$: Solvable in $\mathrm{poly}(n)$ time (e.g., using LLL algorithm).

There exist algorithms for SVP$_{\mathcal{O}(n^k)}$ (equivalently, GapSVP$_{\mathcal{O}(n^k)}$) that run in time $\mathcal{O}(2^{n/k})$.

This chapter is based on lectures 5 and 6.

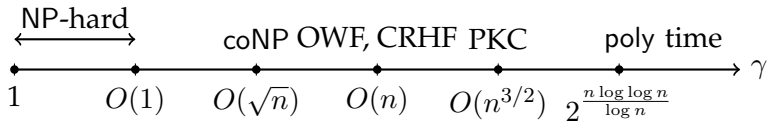


Figure 3.1: SVP Approximation Landscape

The hardness of SVP for various approximation factors says something about all of Impagliazzo's five worlds [Imp95][1]:

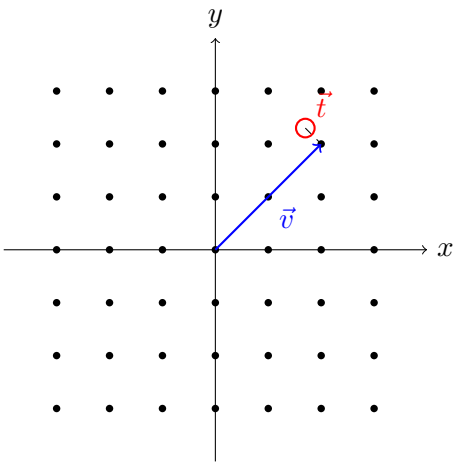
1. ALGORITHMICA: If $\mathsf{SVP}_{\mathcal{O}(1)}$ is easy in the worst case.
2. HEURISTICA: If $\mathsf{SVP}_{\mathcal{O}(1)}$ is easy on average (and $\mathsf{P} \neq \mathsf{NP}$).
3. MINICRYPT: If $\mathsf{SVP}_{\mathcal{O}(n)}$ is hard.
4. CRYPTOMANIA: If $\mathsf{SVP}_{\mathcal{O}(n^{1.5})}$ is hard.
5. PESSILAND: If $\mathsf{SVP}_{\mathcal{O}(\sqrt{n})}$ is hard, but $\mathsf{SVP}_{\mathcal{O}(n)}$ is easy (and $\mathsf{P} \neq \mathsf{NP}$).

3.1 Closest Vector Problem (CVP)

Given a lattice and a target vector in $\mathbb{R}^n$, this problem asks to find the lattice vector closest to the target.

Definition 3.1.1 (Search CVP). *Given a lattice $\mathcal{L}(B)$ with basis B and a target vector $\vec{t} \in \mathbb{R}^n$, find $\vec{v} \in \mathcal{L}(B)$ that minimizes $\|\vec{v} - \vec{t}\|_2$.*

Definition 3.1.2 (Decision CVP). *Given a lattice $\mathcal{L}(B)$ with basis B, a target vector $\vec{t} \in \mathbb{R}^n$, and a distance parameter $d > 0$, output* YES *iff there exists a lattice vector $\vec{v} \in \mathcal{L}(B)$ such that $\|\vec{v} - \vec{t}\|_2 \leq d$.*



CVP: Find closest lattice point $\vec{v}$ to target $\vec{t}$

Figure 3.2: Illustration of the Closest Vector Problem (CVP)

CVP is at least as hard as SVP.

[1] This is probably the only such problem?

Theorem 3.1.3. *There exists a polynomial-time reduction from* SVP *to* CVP.

Proof. The reduction works as follows. Given a lattice $\mathcal{L}$ with basis $B = (\vec{b}_1, \ldots, \vec{b}_n)$. Repeat the following steps for each basis vector $\vec{b}_i$:

1. Construct a new lattice $\mathcal{L}'$ with basis $(\vec{b}_1, \vec{b}_2, \ldots, 2\vec{b}_i \ldots, \vec{b}_n)$ ($\mathcal{L}' \subseteq \mathcal{L}$).
2. Use the CVP oracle on $\mathcal{L}'$ with target vector $\vec{b}_i$. Say the oracle returns a vector $\vec{v} = \alpha_1\vec{b}_1 + \alpha_2\vec{b}_2 + \ldots + 2\alpha_i\vec{b}_i + \ldots \alpha_n\vec{b}_n$.
3. Compute $\vec{v}' = \vec{v} - \vec{b}_i$. The coefficient of $\vec{b}_i$ in $\vec{v}'$ is always odd, and $\vec{v}' \neq \vec{0}$ since $\vec{b}_i \notin \mathcal{L}'$. Thus $\vec{v}'$ is the shortest non-zero lattice vector among all of the vectors in $\mathcal{L}$ whose coefficient for $\vec{b}_i$ is odd.

Note that the shortest vector in $\mathcal{L}$ must have at least one odd coefficient (otherwise dividing the vector by two generates a shorter non-zero lattice vector). Therefore it is in at least one of the sub-lattices considered above, and it will be found in at least one of these n CVP oracle calls. Return the minimum of the $\vec{v}'$s found. □

This reduction shows that if we can solve CVP in polynomial time, we can also solve SVP in polynomial time, proving that CVP is at least as hard as SVP.
The closest vector problem (CVP) is one of the hardest lattice problems, as most other lattice probelems like SVP reduce to it. We will see two algorithms to solve this problem, first a simple $2^{\mathcal{O}(n^2)}$ algorithm (based on ideas from Assignment 1 problem 1), followed by a $\mathcal{O}(4^n)$ time deterministic algorithm based on Voronoi cells [MV13].

3.2 CVP in $2^{\mathcal{O}(n^2)}$

We start with a simple example. Consider the following vectors

$$\begin{aligned}\vec{b_1} &= \vec{b}_1^* \\ \vec{b}_2 &= \vec{b}_2^* + 0.3\vec{b}_1^* \\ \vec{b}_3 &= \vec{b}_3^* + 0.2\vec{b}_2^* + 0.1\vec{b}_1^* \\ \vec{t} &= 2.8\vec{b}_1^* + 2.2\vec{b}_2^* + 6.3\vec{b}_3^*\end{aligned}$$

where each $\vec{b}_i$ is our basis vector for the lattice and $\vec{b}_i^*$ is the Gram-Schmidt vector obtained via Gram-Schmidt Orthogonalization. A straightforward idea is to apply the same idea as in the LLL algorithm: loop back from $i = n$ to 1 and subtract an integer multiple of $\vec{b}_i$, which we attain from rounding $\mu_i = \frac{\langle \vec{t}, \vec{b}_i^* \rangle}{\langle \vec{b}_i^*, \vec{b}_i^* \rangle}$. In this example, we end up with $\vec{t} - 6\vec{b}_3 - \vec{b}_2 - 2\vec{b}_1$. Similar to the LLL algorithm, this vector has the property that the coefficients of each $\vec{b}_i^*$ is within $\pm\frac{1}{2}$, giving us that

$$\text{dist}(\vec{t}, \vec{u} := 6\vec{b}_3 + \vec{b}_2 + 2\vec{b}_1)^2 \leq \sum_i \frac{1}{4}\|\vec{b}_i^*\|^2 \tag{3.2.1}$$

This with some modifications gives a polynomial time $2^{\frac{n}{2}}$ approximation algorithm for the CVP problem, known as Babai's Nearest Plane algorithm [Bab86]. We would like to use eq. (3.2.1) to bound the size of the coefficients of the closest vector to $\vec{t}$, written in the basis B. We can then *enumerate* over all possible coefficients in that range, and return the minimum. We start with the following theorem.

Theorem 3.2.1. *Suppose we are given a $\frac{3}{4}$-LLL reduced basis B. Let $\vec{v} \in \mathcal{L}$ be the closest lattice vector to $\vec{t} \in \mathbb{R}^n$ and the projection of $\vec{t} - \vec{v}$ in the direction of $\vec{b_n^*}$ is $\alpha \vec{b_n^*}$. Then $|\alpha| \leq 2^{\frac{n-2}{2}}$.*

Proof.

$$\begin{aligned}
\alpha^2 \|\vec{b}_n^*\|^2 &\leq \text{dist}(\vec{t}, \mathcal{L})^2 \\
&\leq \sum_i \frac{1}{4} \|\vec{b}_i^*\|^2 \quad \text{(Observe that in eq. (3.2.1), } u \text{ is such a vector)} \\
&\leq \sum_i \frac{1}{4} 2^{n-i} \|\vec{b}_n^*\|^2 \quad \text{(since the basis is LLL reduced.)} \\
&\leq \frac{1}{4} 2^n \|\vec{b}_n^*\|^2 \\
&\leq 2^{n-2} \|\vec{b}_n^*\|^2
\end{aligned}$$

which gives us $|\alpha| \leq 2^{(n-2)/2}$. □

This gives us a bound on the number of (integer) coefficients we need to choose from. Specifically, the number of coefficients to choose from is at most $2 \cdot 2^{(n-2)/2} = 2^{n/2}$.

Remark 3.2.2. *Thus, if we consider $\vec{v} = \sum_i u_i \vec{b_i}$ and $\vec{t} = \sum_i w_i \vec{b_i^*}$, the number of choices of $u_n \in [w_n - 2^{\frac{n-2}{2}}, w_n + 2^{\frac{n-2}{2}}]$, which has at most $2^{\frac{n}{2}}$ integer options.*

This gives us a reduction for $\mathsf{CVP}(n)$ (where n is the rank of B) to $\mathsf{CVP}(n-1)$ that makes $2^{\frac{n}{2}}$ calls to the $\mathsf{CVP}(n-1)$ oracle. The reduction works by iterating through all possible integer coefficients for $\vec{b_n}$ in the valid range. Formally, the following is the reduction algorithm.

Algorithm 6 $\mathsf{CVP}(B, n, t)$

Input: Lattice basis B, rank n, target t.
Output: The closest vector $\vec{v} \in \mathcal{L}(B)$ to the target t.

if n = 0 **then**
  Return $\vec{0}$
end if
Compute the Gram-Schmidt vectors: $[\vec{b}_1^*, \ldots, \vec{b}_n^*]$
Compute $w = \frac{\langle \vec{t}, \vec{b}_n^* \rangle}{\langle \vec{b}_n^*, \vec{b}_n^* \rangle}$
$\vec{x} = 0$
for $i = \lfloor w - 2^{\frac{n-2}{2}} \rfloor, \ldots \lceil w + 2^{\frac{n-2}{2}} \rceil$ **do**
  $\vec{y} = \mathsf{CVP}(\{\vec{b}_1, \ldots, \vec{b}_{n-1}\}, n-1, \vec{t} - i\vec{b}_n) + i\vec{b}_n$
  if $\|\vec{t} - \vec{y}\|_2 < \|\vec{t} - \vec{x}\|_2$ **then**
    $\vec{x} \leftarrow \vec{y}$
  end if
end for
Return $\vec{x}$

Remark 3.2.3. *Since there are $2^{\frac{n}{2}}$ calls to $\mathsf{CVP}(n-1)$, we have the following recurrence relation: $T(n) = 2^{\frac{n}{2}} T(n-1) + \mathsf{poly}(d)$. Which we can resolve to $T(n) = 2^{O(n^2)}$ complexity.*

3.3 Closest Vector Problem using Voronoi cell

We now describe a $\mathcal{O}(4^n)$ time deterministic algorithm based on Voronoi cell computation by Micciancio and Voulgaris [MV10]. Without loss of generality, we assume that the lattice is full rank and $\mathcal{L} \subseteq \mathbb{Z}^n$. Unless otherwise specified, we assume that all norms are ℓ_2 norms.

3.3.1 Introduction to Voronoi Cells and Voronoi Relevant Vectors

Definition 3.3.1 (Voronoi Cell)**.** *The Voronoi cell of a lattice $\mathcal{L}$ is defined as*

$$V = \{\vec{x} \in \mathbb{R}^n : \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|, \forall \vec{v} \in \mathcal{L} \setminus \{0\}\}.$$

That is, the set of points $\vec{x}$ in $\mathbb{R}^n$ such that $\forall \vec{v} \in \mathcal{L},\ dist(\vec{x}, 0) \leq dist(\vec{x}, \vec{v})$.

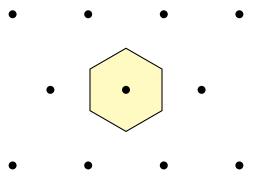


Figure 3.3: Voronoi Cell

Definition 3.3.2 (Half-spaces)**.** *Let a half-space defined by a vector $\vec{v}$ be defined as*

$$H_{\vec{v}} = \{\vec{x} \in \mathbb{R}^n : \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}.$$

That is, the set of points such that the distance from the origin is smaller than or equal to the distance from $\vec{v}$.

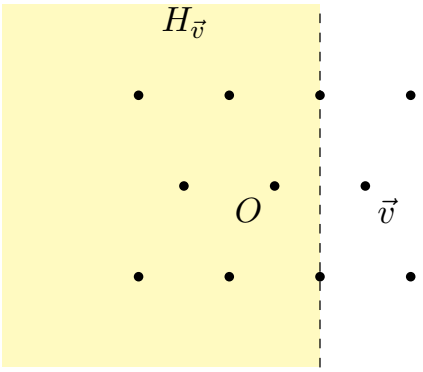


Figure 3.4: Half-space $H_{\vec{v}}$

Remark 3.3.3. *Note that*

$$V = \bigcap_{\vec{v} \in \mathcal{L} \setminus \{0\}} H_{\vec{v}}$$

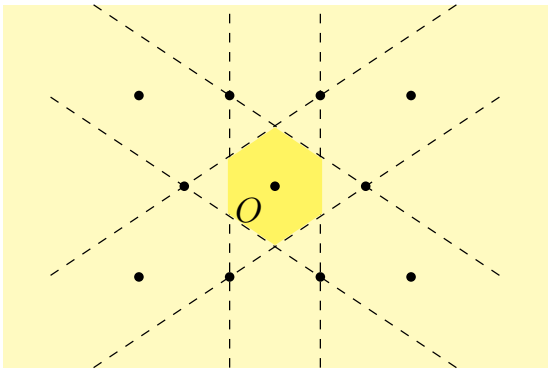


Figure 3.5: Voronoi cell as an intersection of half-spaces.

Remark 3.3.4. *Note that $H_{\vec{v}}$ is a convex set (by definition), and the intersection of convex sets is also a convex set. Therefore V is convex.*

Remark 3.3.5. *The shifts of V by a lattice point $\vec{v} \in \mathcal{L}$ is a tiling of $\mathbb{R}^n$. This is because each point in $\mathbb{R}^n$ will belong to the Voronoi cell of its closest point in the lattice. (Here the Voronoi cell of a non-zero lattice point x is defined similar to definition 3.3.1, the set of points in $\mathbb{R}^n$ whose distance from x is less than or equal to the distance from and other lattice point.)*

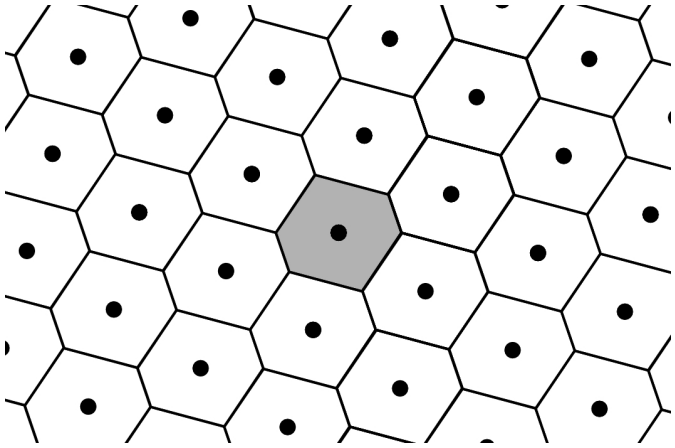


Figure 3.6: Voronoi cell of a lattice tiles $\mathbb{R}^n$ [MSW23]

Definition 3.3.6 (Voronoi Relevant Vectors)**.** *The set of vectors $\vec{v} \in \mathcal{L}$ such that $\mathsf{CVP}(\frac{\vec{v}}{2}, \mathcal{L}) = \{\vec{0}, \vec{v}\}$ are called Voronoi relevant vectors.*

Example 3.3.7. *In the example below, $\vec{v_1}, \vec{v_2}$ are both Voronoi relevant vectors since $\mathsf{CVP}(\vec{v_1}/2, \mathcal{L}) = \{\vec{v_1}, \vec{0}\}$. However, $\vec{v_1} + \vec{v_2}$ is not, as $\mathsf{CVP}((\vec{v_1} + \vec{v_2})/2, \mathcal{L}) = \{\vec{0}, \vec{v_1}, \vec{v_2}, \vec{v_1} + \vec{v_2}\}$.*

Lemma 3.3.8. *If R is the set of Voronoi relevant vectors,*

$$V = \bigcap_{\vec{v} \in R} H_{\vec{v}}$$

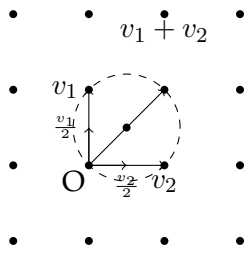


Figure 3.7: Voronoi Relevant vectors

Proof. Consider a $\vec{w} \notin R$, which means that $\exists \vec{u} \in \mathsf{CVP}(\frac{\vec{w}}{2}, \mathcal{L})$ such that $\vec{u} \notin \{\vec{w}, \vec{0}\}$. Note that $\left\|\vec{u} - \frac{\vec{w}}{2}\right\| \leq \left\|\frac{\vec{w}}{2}\right\|$ (since the RHS is the distance of $\vec{w}/2$ from $\vec{0}$). Now consider the lattice vector $\vec{w} - \vec{u}$. By triangle inequality (and since equality case does not hold as $\vec{u} \neq \vec{w}$),

$$\|\vec{w} - \vec{u}\| < \left\|\vec{w} - \vec{u} - \frac{\vec{w}}{2}\right\| + \left\|\frac{\vec{w}}{2}\right\| < \left\|\frac{\vec{w}}{2}\right\| + \left\|\frac{\vec{w}}{2}\right\| = \|\vec{w}\| \tag{3.3.1}$$

Now consider $H_{\vec{u}} \cap H_{\vec{w}-\vec{u}}$. Observe that $H_{\vec{u}} \cap H_{\vec{w}-\vec{u}} \subseteq H_{\vec{w}}$. [This is equivalent to prove that $\|\vec{x}\| \leq \|\vec{x} - \vec{u}\|, \|\vec{x}\| \leq \|\vec{x} - (\vec{w} - \vec{u})\|, \left\|\vec{u} - \frac{\vec{w}}{2}\right\| \leq \left\|\frac{\vec{w}}{2}\right\|$ together imply $\|\vec{x}\| \leq \|\vec{x} - \vec{w}\|$. Expansion of the norms' squares by inner products yields a simple proof.]
Thus, for all $\vec{w} \notin R$, we can remove $H_{\vec{w}}$ from the intersection of half-spaces due to the presence of $H_{\vec{u}} \cap H_{\vec{w}-\vec{u}}$ (note that $\vec{w} - \vec{u} \in \mathcal{L}$). We can keep repeating this for various such $\vec{w}$, but not infinitely, since the vector $\vec{u}$'s sizes are strictly decreasing (eq. (3.3.1)). Thus this process will terminate, and only when all the remaining vectors are in R. This proves the theorem. □

Lemma 3.3.9. *The number of Voronoi relevant vectors is at most* 2^{n+1}

Proof. Let the set of Voronoi relevant vectors of $\mathcal{L}(B)$ be R with $B = \{\vec{b_i}\}$.
Let $v \in \mathcal{L}(B)$ be a lattice vector at the boundary of the fundamental parallelepiped $\mathcal{P}(B)$. (Note that there are only 2^n such vectors.) Let $\vec{v^*} = \mathsf{argmin}\{\|\vec{x}\| : \vec{x} \in \vec{v} + 2\mathcal{L}, \vec{x} \neq \vec{0}\}$. Then we claim that the collection of $\vec{v^*}$ and $-\vec{v^*}$ for all v are the only possible Voronoi relevant vectors of $\mathcal{L}$.
To prove the claim, assume that there exists $\vec{w} \in R$, such that $\vec{w} \notin \{\vec{v^*}, -\vec{v^*}\}$. Note that, all coordinates of w in the given lattice basis cannot be even, as in that case $\frac{\vec{w}}{2}$ is a lattice vector, contradicting the fact that $\vec{w}$ is a Voronoi relevant vector. So, $\exists v \in \mathcal{L}(B)$ at the boundary of $\mathcal{P}(B)$ (Coordinates of v in the given lattice basis are either 0 or 1) such that $\frac{\vec{v}+\vec{w}}{2} \in \mathcal{L}$. So,

$$\frac{\vec{v^*} + \vec{w}}{2} = \frac{\vec{v} + 2\mathcal{L} + \vec{w}}{2} = \frac{\vec{v} + \vec{w}}{2} + \mathcal{L} \in \mathcal{L}$$

.
However, $\left\|\frac{\vec{v^*}+\vec{w}}{2} - \frac{\vec{w}}{2}\right\| = \left\|\frac{\vec{v^*}}{2}\right\| \leq \left\|\frac{\vec{w}}{2}\right\|$. The last inequality is because of the following reasoning. $\left\|\frac{\vec{v^*}}{2}\right\|$ is the length of the shortest vector in the lattice $\frac{\vec{v}}{2} + \mathcal{L}$. Also, $\frac{\vec{w}}{2} + \frac{\vec{v}}{2} \in \mathcal{L}$, so $\frac{\vec{w}}{2} \in \mathcal{L} - \frac{\vec{v}}{2} = (\mathcal{L} - \vec{v}) + \frac{\vec{v}}{2} = \frac{\vec{v}}{2} + \mathcal{L}$. This means that we have found a lattice vector $\frac{\vec{v^*}+\vec{w}}{2}$ whose distance from $\vec{w}/2$ is less than the distance of $\vec{0}$ from $\vec{w}/2$. This is a contradiction to the fact that $\vec{w}$ is Voronoi relevant. Thus $|R \cap (\vec{v} + 2\mathcal{L})| \leq 2$.
Now since there are 2^n points on the boundary of $\mathcal{P}(B)$, we get that $|R| \leq 2^{n+1}$.

□

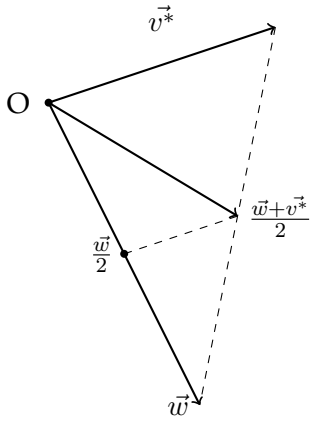


Figure 3.8: Voronoi Relevant vectors in $\vec{v} + 2\mathcal{L}$

3.3.2 CVP Algorithm given Voronoi Relevant Vectors

The closest vector problem is given a target vector $\vec{t}$, we want the lattice vector $\vec{x} \in \mathcal{L}$ that is closest to t. This is equivalently, we want to find a shift of $\vec{t}$ by a lattice vector $\vec{x}$ such that $\vec{t} - \vec{x}$ is the shortest possible. The following algorithm solves CVP given a set of Voronoi relevant vectors.

Algorithm 7 CVP given Voronoi relevant vectors

Input: Basis: $B \in \mathbb{R}^{n \times m}$, target $t \in \mathbb{R}^n$, set of Voronoi relevant vectors $R \subset \mathcal{L}(B) \setminus \{0\}$
Output: $\vec{x} \in \mathcal{L}$ such that $||\vec{t} - \vec{x}||$ is minimised.

1: Set $\vec{s} = \vec{t}$
2: **while** $\vec{s} \notin V$ **do**
3: Compute smallest $\delta > 1$ such that $\vec{s} \in \delta V$
4: Compute $1 < \alpha \leq 2 \in \mathbb{R}, k \in \mathbb{Z}_{\geq 0}$ such that $\delta = \alpha 2^k$
5: Find $\vec{v} \in R$ such that $\vec{s}$ lies on the boundary of $\delta H_{\vec{v}}$
6: $\vec{s} := \vec{s} - 2^k \vec{v}$
7: **end while**
8: **Return** $\vec{t} - \vec{s}$

Remark 3.3.10. *Note that always exists a $\vec{v} \in R$ such that $\vec{s}$ lies on the boundary of $\delta H_{\vec{v}}$ in line 5. This is because $\vec{s}$ lies on the boundary of δV due to line 3 which is made up of boundaries of $\delta H_{\vec{v}}$ by definition.*

We prove correctness for the case where $\delta \in (1, 2]$. The general case follows.

Lemma 3.3.11. *Let $1 < \delta \leq 2$, $\vec{s} \in \delta V$ and $\vec{v}$ be a Voronoi relevant vector such that $\vec{s}$ lies on the boundary of $\delta H_{\vec{v}}$. Then $\|\vec{s} - \vec{v}\| < \|\vec{s}\|$ and $\vec{s} - \vec{v} \in \delta V \subseteq 2V$*

Proof. First note that, since all vectors on the boundary of $\delta H_{\vec{v}}$ are equidistant from $\delta\vec{v}$ and $\vec{0}$, we have that $\left(\vec{s} - \frac{\delta\vec{v}}{2}\right) \perp \frac{\delta\vec{v}}{2}$ (in fact, perpendicular to any scaling of $\vec{v}$). Write $\vec{s} = \left(\vec{s} - \frac{\delta\vec{v}}{2}\right) + \frac{\delta\vec{v}}{2}$.

$$\|\vec{s}\|^2 = \left\|\vec{s} - \frac{\delta\vec{v}}{2}\right\|^2 + \left(\frac{\delta}{2}\right)^2 \|\vec{v}\|^2 \tag{3.3.2}$$

$$\|\vec{s}-\vec{v}\|^2 = \left\|\vec{s}-\frac{\delta\vec{v}}{2}-\left(1-\frac{\delta}{2}\right)\vec{v}\right\|^2 = \left\|\vec{s}-\frac{\delta\vec{v}}{2}\right\|^2 + \left(1-\frac{\delta}{2}\right)^2\|\vec{v}\|^2 \tag{3.3.3}$$

Observe that

$$1<\delta\leq 2 \implies 0\leq 1-\frac{\delta}{2}<\frac{1}{2}<\frac{\delta}{2} \implies \left(1-\frac{\delta}{2}\right)^2<\left(\frac{\delta}{2}\right)^2$$

Using this with eq. (3.3.2) and eq. (3.3.3), we get that

$$\|\vec{s}-\vec{v}\|<\|\vec{s}\| \tag{3.3.4}$$

To see that $\vec{s}-\vec{v}\in\delta V$, recall that V is convex. $\vec{s}$ is on the boundary of $\delta H_{\vec{v}}$, and so $\vec{s}\in\delta\vec{v}+\delta V$, which implies $\vec{s}-\delta\vec{v}\in\delta V$. By convexity, $\vec{s}-\vec{v}=\left(1-\frac{1}{\delta}\right)\vec{s}+\frac{1}{\delta}\left(\vec{s}-\delta\vec{v}\right)\in\delta V$.

□

Lemma 3.3.12. *Given a vector $\vec{s}$ and lattice $\mathcal{L}$ with Voronoi cell V, $|(\vec{s}-\mathcal{L})\cap 2V|\leq 4^n$*

Proof. We prove this by contradiction. Say the cardinality is more than 4^n. Then by pegionhole principle, we can find two distinct vectors with the same remainder modulo 4 for each coefficient of the basis vectors. Thus there exists distinct $\vec{x},\vec{y}\in(\vec{s}-\mathcal{L})\cap 2V$ such that $\vec{x}-\vec{y}\in 4\mathcal{L}$. Thus $\frac{\vec{x}-\vec{y}}{4}\in\mathcal{L}$.
Note that since $\vec{x},\vec{y}\in 2V$, $\frac{\vec{x}}{2},\frac{\vec{y}}{2}\in V$. Since V is symmetric and convex, we have that $-\frac{\vec{y}}{2}\in V$ and

$$\frac{\frac{\vec{x}}{2}-\frac{\vec{y}}{2}}{2}=\frac{\vec{x}-\vec{y}}{4}\in V.$$

However, $\frac{\vec{x}-\vec{y}}{4}\in\mathcal{L}$ and the only lattice point in V is $\vec{0}\implies\vec{x}=\vec{y}$ contradiction. □

Remark 3.3.13 (Termination of algorithm 7)**.** *We can extend lemma 3.3.11 by setting $V'=2^kV$ and $R'=2^kR$, as such $\vec{s}-2^k\vec{v}\in 2^{k+1}V$. Thus during the execution of Lines 2 - 6, the value of k is non-increasing. However, note that if k remains the same, $\vec{s}$ remains in $(\vec{s}-2^k\mathcal{L})\cap 2^{k+1}V$ as it only changes by $2^k\vec{v}$ and remains in $2^{k+1}V$. However, note that after each iteration, $||\vec{s}||$ strictly decreases as $||\vec{s}-2^k\vec{v}||<||\vec{s}||$ by lemma 3.3.11, thus each $\vec{s}$ after each iteration is distinct. If k remains the same, there are then infinite vectors in $(\vec{s}-2^k\mathcal{L})\cap 2^{k+1}V$, contradicting lemma 3.3.12. Thus, k must decrease until $k=0$, after which $\vec{s}\in V$ and the loop terminates.*

Remark 3.3.14 (Runtime Complexity)**.** *Note that the initial value of k is bounded by $O(\log||\vec{t}||)$. k will decrease by 1 in at most 4^n iterations. Thus, we get a complexity of $O(\log||\vec{t}||4^n)$*

Remark 3.3.15 (Correctness)**.** *$\vec{t}-\vec{x}\in V$, thus $\forall\vec{v}\in\mathcal{L}, ||\vec{t}-\vec{x}||\leq||\vec{t}-\vec{x}-\vec{v}||$, thus it is the shortest vector. As stated at the beginning of section 3.3.2, this solves CVP, given that we know the Voronoi relevant vectors.*

We now have an algorithm for CVP, given that we know the Voronoi relevant vectors. Let $T_{\mathsf{Vor}}(n)$ denote the time to compute the Voronoi cell (that is, find all the Voronoi relevant vectors) in n dimensions. Let $T_{\mathsf{CVP}}(n,k)$ denote the time to solve k instances of CVP in n dimensions. Observe that if we are given a CVP oracle, then we can compute the Voronoi cell. This can be done by solving a CVP instance for each of the 2^n cosets of the lattice. Precisely, let $b_1,\dots b_n$ be the basis of a lattice, and $v:=\sum_j\alpha_jb_j$, such that $\alpha_j\in\{0,1\}$. Note that there are 2^n such vectors, and they correspond to the vertices of the fundamental parallelepiped. One can solve $\mathsf{CVP}(2\mathcal{L},v)$ for each such v, which would give us at most two of the Voronoi relevant vectors each. Thus, we have the following reduction.

$$T_{\mathsf{Vor}}(n)\leq T_{\mathsf{CVP}}(n,2^n)$$

Now, recall algorithm 6. To solve CVP, one can guess the last coordinate of the closest vector, and recurse on the $n-1$ dimensional lattice. There are 2^{n+1} such choices for the last coefficient. This gives us the following reduction.

$$T_{\mathsf{CVP}}(n, 2^n) \leq T_{\mathsf{CVP}}(n-1, 2^{n+1} \cdot 2^n) = T_{\mathsf{CVP}}(n-1, 2^{2n+1})$$

From remark 3.3.14, we can solve CVP in 4^n time, if we are given the Voronoi relevant vectors. Hence, we also have the following reduction

$$T_{\mathsf{CVP}}(n-1, 2^{2n+1}) \leq T_{\mathsf{Vor}}(n-1) + 4^n \cdot 2^{2n+1} \cdot \mathsf{poly}(n).$$

Observe that the two terms above are additive because we only need to compute the Voronoi cell once, after which we can solve all the 2^{2n+1} instances of $\mathsf{CVP}(n-1)$. Now, observe that combining the above three equations, we have a reduction from computing the Voronoi cell in n dimensions to computing the Voronoi cell in $n-1$ dimensions, by paying an additional cost of $16^n \cdot \mathsf{poly}(n)$. Unfolding this recurrence $n-1$ times, we can see that

$$T_{\mathsf{Vor}}(n) \leq 16^n \cdot \mathsf{poly}(n).$$

Thus,

$$T_{\mathsf{CVP}}(n, 1) \leq 16^n \cdot \mathsf{poly}(n) + 4^n \mathsf{poly}(n).$$

We note that a better analysis gives us a time complexity of 4^n.

Chapter 4

Shortest Vector Problem

By now it is clear that SVP is an important problem. We will soon see an algorithm for the exact shortest vector problem. Table 4.1 gives a summary of the history of exact SVP.

Reference	Time Complexity	Remarks
Kannan [Kan83]	$n^{\frac{n}{2e}+o(n)}$	Enumeration; Also solves CVP
Ajtai, Kumar and Sivakumar [AKS01]	$2^{\mathcal{O}(n)}$	Randomized Sieve
Micciancio and Voulgaris [MV10]	$4^n\mathsf{poly}(n)$	Deterministic, also solves CVP
[ADRS15]	$2^n + o(n)$	A follow-up work also solves CVP [ADS15]

Table 4.1: A brief history of exact SVP

The best known algorithm solves n^k approximate SVP in time roughly $2^{\frac{n}{k+1}}$. The main idea follows the LLL algorithm, with reduction applied to a *block* of the basis vectors, instead of two basis vectors at once. We will first learn about dual lattices.

4.1 Dual Lattice

The *dual lattice* of any lattice is the set of all points $\vec{v}$ in the span on the lattice such that the inner product of $\vec{v}$ with *any* lattice vector is an integer. Formally,

Definition 4.1.1 (Dual Lattice)**.** *Given a lattice $\mathcal{L} \subset \mathbb{R}^d$ of rank n, the set*

$$\mathcal{L}^* := \{\vec{y} \in \mathrm{span}(\mathcal{L}) : \forall \vec{x} \in \mathcal{L}, \langle \vec{x}, \vec{y} \rangle \in \mathbb{Z}\}$$

is called the Dual Lattice *of $\mathcal{L}$.*

As an example, consider the integer lattice, $\mathbb{Z}^n$. Clearly, it's dual lattice is itself $(\mathbb{Z}^n)^* = \mathbb{Z}^n$, since all the vectors in the span of $\mathbb{Z}^n$ which have integer inner product with $\mathbb{Z}^n$ are in Z^n, and vice versa. Also observe that $(2\mathbb{Z}^n)^* = \frac{1}{2}\mathbb{Z}^n$.
For a full rank lattice $\mathcal{L}$ with basis $B := (\vec{b}_1, \dots \vec{b}_n)$, the basis of the dual lattice $\mathcal{L}^*$ is given by $(B^\top)^{-1}$. (Observe that the inner product of any vector in the integer span of $(B^\top)^{-1}$ with a vector in $\mathcal{L}(B)$ will be an integer.) For any lattice (not necessarily full-rank), the basis of the dual lattice is given as $B(B^\top B)^{-1}$, the pseudo-inverse of B.

This chapter is based on lectures 6, 7 and 8.

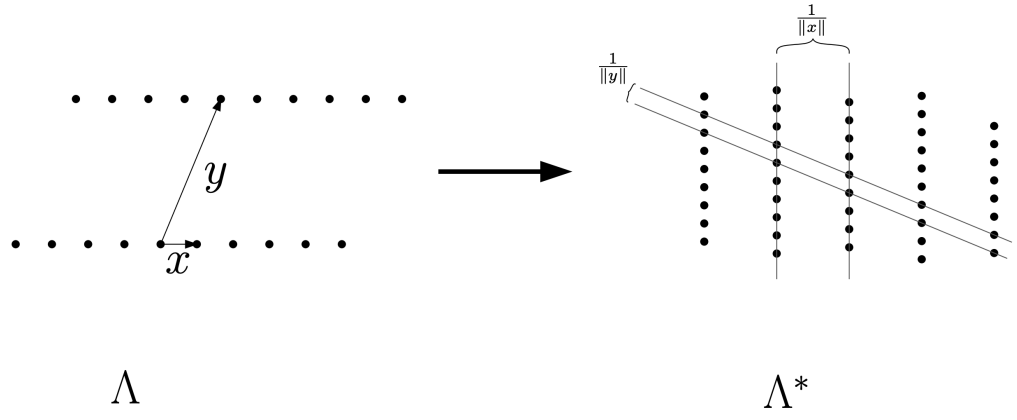


Figure 4.1: An example of the dual lattice [Reg04]

Lemma 4.1.2 (Dual Basis). $\mathcal{L}(B)^* = \mathcal{L}(B(B^\top B)^{-1})$

Proof. Let $D = B(B^\top B)^{-1}$. Observe that $B^\top D = I$. First we prove that $\mathcal{L}(D) \subseteq \mathcal{L}(B)^*$. Observe that $\forall i, j \in [n] : \left\langle \vec{b}_j, \vec{d}_i \right\rangle \in \{0, 1\}$. Then, for any $\vec{u} \in \mathcal{L}(D),\ \vec{v} \in \mathcal{L}(B)$, we have that $\vec{u} = \sum_i a_i \vec{d}_i$, $\vec{v} = \sum_j c_j \vec{b}_j$, such that $\forall i, j \in [n] : a_i, c_j \in \mathbb{Z}$. Then, $\langle \vec{u}, \vec{v} \rangle = \sum_{i,j} a_i c_j \left\langle \vec{d}_i, \vec{b}_j \right\rangle \in \mathbb{Z}$, and thus any such $\vec{u}$ is in the dual lattice.
Next, we prove that $\mathcal{L}(B)^* \subseteq \mathcal{L}(D)$. Let $v \in \mathcal{L}(B)^*$. Observe that since span $(\mathcal{L}(D))$ = span $(\mathcal{L}(B))$, $\exists \alpha_1, \ldots \alpha_n \in \mathbb{R}$ such that $\vec{v} = \alpha_1 \vec{d}_1 + \ldots \alpha_n \vec{d}_n$. Then $\forall i \in [n], \left\langle \vec{v}, \vec{b}_i \right\rangle = \alpha_i$. Since v was in the dual lattice, $\alpha_i \in \mathbb{Z}$. Thus, $\vec{v} \in \mathcal{L}(D)$. This concludes the proof. □

In what follows, we assume that the lattice is full rank. We mention the following corollaries without proof, which follow simply from the definition of the dual basis.

Claim 4.1.3. *For any lattice $\mathcal{L}$, we have that*

1. $(\mathcal{L}^*)^* = \mathcal{L}$
2. $\det(\mathcal{L}^*) = \frac{1}{\det(\mathcal{L})}$

Observe that the second point above gives us a clue about the *sparsity* of the lattice. If the primal lattice is very dense, the dual lattice will be very sparse, and vice versa. Formally, we have the following relation.

Claim 4.1.4. $\lambda_1(\mathcal{L}) \cdot \lambda_1(\mathcal{L}^*) \leq n$

We note that in fact, a stronger relation $\lambda_1(\mathcal{L}) \cdot \lambda_n(\mathcal{L}^*) \leq n$ holds true, which we will not prove.

Proof. From Minkowski's theorem we know that $\lambda_1(\mathcal{L}) \leq \sqrt{n} \det(\mathcal{L})^{1/n}$, and that $\lambda_1(\mathcal{L}^*) \leq \sqrt{n} \det(\mathcal{L}^*)^{1/n}$. Using the second item of claim 4.1.3 and these two inequalities, we get the desired result. □

Claim 4.1.5. $\lambda_1(\mathcal{L}) \cdot \lambda_n(\mathcal{L}^*) \geq 1$.

Proof. Let $\vec{u}$ be the shortest vector in $\mathcal{L}$. $\|\vec{u}\| = \lambda_1(\mathcal{L})$. Let $\vec{v}_1, \ldots \vec{v}_n$ be linearly independent vectors in $\mathcal{L}^*$, such that $\forall i \in [n], \|\vec{v}_i\| \leq \lambda_i(\mathcal{L}^*)$. Then since span $(\mathcal{L})$ = span $(\mathcal{L}^*)$, we have that $\exists i \in [n] : \langle \vec{u}, \vec{v}_i \rangle \neq 0$. Since, $\langle \vec{u}, \vec{v}_i \rangle \in \mathbb{Z}$, $|\langle \vec{u}, \vec{v}_i \rangle| \geq 1$. Therefore, $1 \leq |\langle \vec{u}, \vec{v}_i \rangle| \leq \|\vec{u}\| \|\vec{v}_i\| \leq \lambda_1(\mathcal{L}) \lambda_n(\mathcal{L}^*)$. □

Given a basis $\vec{b}_1, \dots \vec{b}_n$ for a lattice and some vector $\vec{v}$, let $\pi_i(\vec{v})$ denote the vector $\vec{v}$ projected onto the space span $\left(\vec{b}_1, \dots \vec{b}_{i-1}\right)^{\perp}$. Observe that the GSO vectors can be written using this notation as $\vec{b}_1, \pi_2(\vec{b}_2), \pi_3(\vec{b}_3), \dots \pi_n(\vec{b}_n)$.

Claim 4.1.6. *Let $\vec{d}_1, \dots \vec{d}_n$ be a basis of the dual lattice of $\mathcal{L}(\vec{b}_1, \dots \vec{b}_n)$. Then $\forall i \in [n], (\pi_i(\vec{b}_i), \dots \pi_i(\vec{b}_n))$ and $(\vec{d}_i, \dots \vec{d}_n)$ are dual basis.*

Proof. First note that span $\left(\pi_i(\vec{b}_i), \dots \pi_i(\vec{b}_n)\right) =$ span $\left(\vec{d}_i, \dots \vec{d}_n\right)$. This is because $\left\langle \vec{d}_j, \vec{b}_k \right\rangle = \delta_{j,k}$, which means that the space on the RHS is indeed orthogonal to $\vec{b}_1 \dots \vec{b}_{i-1}$, and has the same rank to the LHS. Next, observe that $\forall j, k \geq i, \left\langle \pi_i(\vec{b}_k), \vec{d}_j \right\rangle = \left\langle \vec{b}_k, \vec{d}_j \right\rangle + \alpha_1 \left\langle \vec{b}_1, \vec{d}_j \right\rangle + \dots \alpha_{i-1} \left\langle \vec{b}_{i-1}, \vec{d}_j \right\rangle = \left\langle \vec{b}_k, \vec{d}_j \right\rangle = \delta_{k,j}$ (note that all other inner product terms are 0, by the definition of dual basis). This means that the dual lattice of $\mathcal{L}(\pi_i(\vec{b}_i), \dots \pi_i(\vec{b}_n))$ has basis $(\vec{d}_i, \dots \vec{d}_n)$, and vice versa, as claimed. □

4.2 Korkin-Zolotarev Basis

Suppose our goal is to find a *nice* basis for a lattice. We saw that in polynomial time we can use LLL to get a reduced basis, such that the vectors are somewhat orthogonal and short (which is what we generally want from these *nice* basis). Now let's see what we can do if we have a budget of more than polynomial time. We begin by defining the Korkin-Zolotarev (KZ) basis.

Definition 4.2.1 (Korkin-Zolotarev (KZ) basis)**.** *We define it recursively. Given a rank n lattice $\mathcal{L}$, a basis $\vec{b}_1, \dots \vec{b}_n$ is called the KZ basis if the following two conditions hold:*

1. $\forall i \in [n] : \left\| \pi_i(\vec{b}_i) \right\| = \lambda_1(\pi_i(\mathcal{L}))$.
2. $\forall i, j \in [n], i < j : |\mu_{i,j}| \leq \frac{1}{2}$.

Note that the second condition guarantees that the KZ basis is length-reduced, same as the LLL basis (adding an integer multiple of one basis vector to another will not decrease its length.) Definition 4.2.1 can be equivalently stated as: $\vec{b}_1$ is a shortest non-zero lattice vector of $\mathcal{L}$, and $\left\{\pi_2(\vec{b}_2), \dots \pi_2(\vec{b}_n)\right\}$ is the KZ reduced basis of the projected lattice $\pi_2(\mathcal{L})$. To see why, let $\mathcal{L}' = \pi_2(\mathcal{L})$ be the lattice $\mathcal{L}$ projected orthogonal to $\vec{b}_1$. Let, $c_2, \dots c_n$ be the KZ basis of $\mathcal{L}'$. Define $\vec{b}_i = c_i + \alpha_i \vec{b}_1$ for $i = 2, \dots n$ such that $\alpha_i \in (-\frac{1}{2}, \frac{1}{2}]$ is the unique value such that $\vec{b}_i \in \mathcal{L}$.

Lemma 4.2.2. *For any lattice $\mathcal{L}$, there exists a basis $\vec{b}_1, \dots, \vec{b}_n$ such that*

$$\min(\|\vec{b}_1^*\|, \dots, \|\vec{b}_n^*\|) \geq \frac{\lambda_1(\mathcal{L})}{n},$$

where $\vec{b}_1^*, \dots, \vec{b}_n^*$ are the Gram-Schmidt orthogonalization (GSO) vectors of the basis.

Proof. Let $\vec{d}_1, \dots, \vec{d}_n$ be the dual basis obtained by $D = B(B^\top B)^{-1}$, where B corresponds to the original basis $\vec{b}_1, \dots, \vec{b}_n$. We first prove a technical claim:

Claim 4.2.3. *Let $\vec{d}_n^*, \dots, \vec{d}_1^*$ be the GSO vectors of $\vec{d}_n, \dots, \vec{d}_1$ in this reverse order. Then $\forall i$,*

$$\vec{d}_i^* = \frac{\vec{b}_i^*}{\|\vec{b}_i^*\|^2},$$

where $\vec{b}_i^*$'s are the GSO vectors obtained in the order of $\vec{b}_1, \dots, \vec{b}_n$.

Proof of Claim 4.2.3. We prove this by induction. For the base case $i = 1$, observe that:

1. $\vec{d}_1^*$ is orthogonal to $\vec{d}_n^*, \ldots, \vec{d}_2^*$ by definition.
2. $\vec{b}_1^*$ is orthogonal to $\vec{d}_n^*, \ldots, \vec{d}_2^*$ due to $B^\top D = I$.
3. Both B and D are full-rank, so $\vec{d}_1^*$ and $\vec{b}_1^*$ must be co-linear.

Therefore, $\vec{d}_1^* \in \text{Span}(\vec{b}_1^*) = \text{Span}(\vec{b}_1)$.
From $B^\top D = I$, we know that $\vec{b}_1$ is orthogonal to $\vec{d}_2, \ldots, \vec{d}_n$ and thus to $\vec{d}_2^*, \ldots, \vec{d}_n^*$. This implies:

$$\langle \vec{b}_1, \vec{d}_1 \rangle = \langle \vec{b}_1, \vec{d}_1^* \rangle = \|\vec{b}_1^*\| \cdot \|\vec{d}_1^*\| = 1 \Rightarrow \|\vec{d}_1^*\| = \frac{1}{\|\vec{b}_1^*\|}.$$

Combining these results proves the claim for $i = 1$.
For the inductive step, we use the fact that $\pi_i(\vec{b}_i), \ldots, \pi_i(\vec{b}_n)$ is the dual basis of $\vec{d}_i, \ldots, \vec{d}_n$ (claim 4.1.6), where π_i is the projection function defined as:

$$\pi_i(\vec{x}) = \sum_{j \geq i} \frac{\langle \vec{x}, \vec{b}_j^* \rangle}{\langle \vec{b}_j^*, \vec{b}_j^* \rangle} \vec{b}_j^*.$$

The GSO vectors of $\pi_i(\vec{b}_i), \ldots, \pi_i(\vec{b}_n)$ are $\vec{b}_i^*, \ldots, \vec{b}_n^*$. We can then apply the same argument as for $i = 1$ to prove the claim for all i. □

Now, we return to the proof of Lemma 4.2.2. Let $\vec{d}_1, \ldots, \vec{d}_n$ be the Korkine-Zolotarev (K.Z.) basis of $\mathcal{L}^*$ and $\vec{b}_n, \ldots, \vec{b}_1$ be the dual basis of $\vec{d}_1, \ldots, \vec{d}_n$ in reverse order. From Claim 4.2.3, we have:

$$\|\vec{d}_i^*\| = \frac{1}{\|\vec{b}_i^*\|}.$$

Thus, the following statements are equivalent:

$$\min\left(\|\vec{b}_1^*\|, \ldots, \|\vec{b}_n^*\|\right) \geq \frac{1}{n}\lambda_1(\mathcal{L}) \Leftrightarrow \max\left(\|\vec{d}_1^*\|, \ldots, \|\vec{d}_n^*\|\right) \leq \frac{n}{\lambda_1(\mathcal{L})}.$$

We know that $\lambda_1(\mathcal{L}) \cdot \lambda_1(\mathcal{L}^*) \leq n$ (claim 4.1.4), and by the definition of the K.Z. basis, $\|\vec{d}_1^*\| = \lambda_1(\mathcal{L}^*)$. This immediately implies:

$$\|\vec{d}_1^*\| \leq \frac{n}{\lambda_1(\mathcal{L})}.$$

For $i > 1$, we use the fact that $\pi_i(\vec{d}_i), \ldots, \pi_i(\vec{d}_n)$ is the dual basis of $\vec{b}_i, \ldots, \vec{b}_n$. The same argument implies:

$$\max\left(\|\vec{d}_i^*\|, \ldots, \|\vec{d}_n^*\|\right) \leq \frac{n - i + 1}{\lambda_1(\mathcal{L}(\vec{b}_i, \ldots, \vec{b}_n))} \leq \frac{n}{\lambda_1(\mathcal{L})}.$$

The last inequality holds because removing any vectors from a given basis results in a new lattice whose shortest vector is at least as long as the previous shortest vector.
Combining these results completes the proof of Lemma 4.2.2. □

4.3 $\mathsf{GapSVP}_n(d) \in \mathsf{coNP}$

Recall the GapSVP problem.

Definition 4.3.1 (GapSVP_γ)**.** *Given a lattice basis B, and $d > 0$, decide if*

- *YES:* $\lambda_1(\mathcal{L}(B)) \leq d$
- *NO:* $\lambda_1(\mathcal{L}(B)) > \gamma \cdot d$

Following corollary follows from lemma 4.2.2.

Corollary 4.3.2. $\mathsf{GapSVP}_n(d) \in \mathsf{coNP}$

Proof. We show that $\mathsf{coGapSVP}_n(d) \in \mathsf{NP}$. That is, we want a witness for the case $\lambda_1(\mathcal{L}(B)) > n \cdot d$, which will simply the basis $\vec{v}_1, \ldots \vec{v}_n$ for $\mathcal{L}$, as defined in lemma 4.2.2. The verifier checks if

1. $\vec{v}_1, \ldots \vec{v}_n$ is a basis of $\mathcal{L}$.
2. $\min(\|\vec{v}_1^*\|, \ldots \|\vec{v}_n^*\|) > d$.

Observe that if we were in the YES case, that is, $\lambda_1(\mathcal{L}(B)) > n \cdot d$, it follows that

$$\min(\|\vec{v}_1^*\|, \ldots \|\vec{v}_n^*\|) \geq \frac{\lambda_1}{n} > d.$$

Otherwise (that is, $\lambda_1(\mathcal{L}(B)) \leq d$ case), we use Theorem 1.4 from lecture two, which says that for all basis of $\mathcal{L}$, $\min(\left\|\vec{b}_1^*\right\|, \ldots \left\|\vec{b}_n^*\right\|) \leq \lambda_1(\mathcal{L})$. This implies that for any witness (basis) that Merlin returns, we have that

$$\min(\left\|\vec{b}_1^*\right\|, \ldots \left\|\vec{b}_n^*\right\|) \leq \lambda_1(\mathcal{L}) \leq d$$

□

4.4 Algorithm for Approximate SVP

The best known algorithms for exact SVP include a deterministic 4^n time algorithm [MV10] (the computation of Voronoi cells we have seen in chapter 3 also allows us to solve SVP,) and a randomized 2^n-time algorithm [ADRS15]. Recall that we also know a polynomial time algorithm for $2^{n/2}$-approximate SVP, the LLL algorithm from chapter 2. Lattices problems are heavily used in modern cryptography. In cryptanalysis, we often have a budget exceeding polynomial time, say $2^{k+o(n)}$ operations. We may ask: *what is the best possible approximation factor for the* SVP *problem given a compute budget of* $2^{k+o(n)}$? We will answer this question. We start with the concept of a primitive vector.

Definition 4.4.1 (Primitive Vector)**.** *A primitive vector $\vec{v} \in \mathcal{L}$ is a vector such that $\forall c \in \mathbb{Z} \setminus \{-1, 0, 1\}$, $\frac{\vec{v}}{c} \notin \mathcal{L}$.*

Equivalently, a primitive vector is the shortest non-zero lattice vector within its own integer span. Note that for an integer lattice, these are all the points whose coordinates have GCD unity. We state an important property of primitive vectors:

Lemma 4.4.2. *For any primitive vector $\vec{v} \in \mathcal{L}$, there exists a basis of $\mathcal{L}$ of the form $\{\vec{v}, \vec{b}_2, \ldots, \vec{b}_n\}$.*

Proof. Part of the next problem set. □

Corollary 4.4.3. *For any primitive vector $\vec{v} \in \mathcal{L}$,*

$$\det(\mathcal{L}_{\perp\vec{v}}) = \frac{\det(\mathcal{L})}{\|\vec{v}\|}.$$

Corollary 4.4.4. *Given a lattice $\mathcal{L}$ whose dual lattice is $\mathcal{L}^*$, for any primitive vector $\vec{v} \in \mathcal{L}^*$,*

$$\det(\mathcal{L}_{\perp\vec{v}}) = \frac{1}{\det(\mathcal{L}^*_{\perp\vec{v}})} = \frac{\|\vec{v}\|}{\det(\mathcal{L}^*)} = \det(\mathcal{L}) \cdot \|\vec{v}\|.$$

Lemma 4.4.5. *Let A be an algorithm that given a lattice $\mathcal{L}$ of rank n, finds a vector $\vec{v} \in \mathcal{L}$ of length at most $\gamma \cdot \det(\mathcal{L})^{1/n}$, where γ can be a function of n. Then there is a polynomial time algorithm $\mathcal{A}$ that takes A as an oracle and finds a non-zero vector $\vec{v} \in \mathcal{L}$ of length at most $\gamma^2 \cdot \lambda_1(\mathcal{L})$.*

Proof. $\mathcal{A}$ works as follows. It first uses the oracle to find $\vec{z}_1 \in \mathcal{L}$ such that $\|\vec{z}_1\| \leq \gamma \cdot \det(\mathcal{L})^{\frac{1}{n}}$. Then it uses the oracle to find $\vec{w} \in \mathcal{L}^*$ such that $\|\vec{w}\| \leq \gamma \cdot \det(\mathcal{L}^*)^{\frac{1}{n}}$. Then it recursively calls itself with the (lower dimensional) lattice $\mathcal{L} \cap \vec{w}^\perp$ to find a vector $\vec{z}_2$ with $\|\vec{z}_2\| \leq \gamma^2 \cdot \lambda_1(\mathcal{L} \cap \vec{w}^\perp)$. It outputs the shorter of $\vec{z}_1, \vec{z}_2$.

Correctness. It is easy to see that $\mathcal{L} \cap \vec{w}^\perp$ is a lattice of lower dimension. A nonzero vector $\vec{w} \in \mathcal{L}^*$ can be interpreted as a linear function $f \colon \mathcal{L} \to \mathbb{Z}$, defined as $f(\vec{v}) = \langle \vec{v}, \vec{w} \rangle$. For a given $\vec{w}$, the lattice $\mathcal{L}$ is divided into infinitely many equivalence classes based on the value of f. Each class will lie on a hyperplane. Each such hyperplane is perpendicular to $\vec{w}$. Consider any such hyperplane $\vec{w}^\perp$. Consider a vector $\vec{x} \in \mathcal{L}$ such that it has a non-zero component on $\vec{w}$, $\vec{x} \notin \vec{w}^\perp$. Then we have

$$1 \leq |\langle \vec{x}, \vec{w} \rangle| \leq \|\vec{x}\| \, \|\vec{w}\| \implies \|\vec{x}\| \geq \frac{1}{\|\vec{w}\|}. \tag{4.4.1}$$

This shows that the distance between any consecutive hyperplane is $\frac{1}{\|\vec{w}\|}$. The correctness follows by induction. Let $\vec{v}^* \in \mathcal{L}$ be a shortest vector.

Case 1: $\vec{v}^* \in (\mathcal{L} \cap \vec{w}^\perp)$. Then, $\lambda_1(\mathcal{L}) = \lambda_1(\mathcal{L} \cap \vec{w}^\perp)$ so $\vec{z}_2$ is of the desired length.

Case 2: $\vec{v}^* \notin (\mathcal{L} \cap \vec{w}^\perp)$. From eq. (4.4.1), $\|\vec{v}^*\| \geq \frac{1}{\|\vec{w}\|}$. Then, from the promise of the oracle, we have that

$$\begin{aligned}
\lambda_1(\mathcal{L}) &\geq \frac{1}{\|\vec{w}\|} \\
&\geq \frac{1}{\gamma \cdot \det(\mathcal{L}^*)^{\frac{1}{n}}} \quad \text{(oracle's promise)} \\
&= \frac{1}{\gamma} \cdot \det(\mathcal{L})^{\frac{1}{n}} \\
\implies \det(\mathcal{L})^{\frac{1}{n}} &\leq \gamma \cdot \lambda_1(\mathcal{L}) \\
\therefore \|\vec{z}_1\| &\leq \gamma \cdot \det(\mathcal{L})^{\frac{1}{n}} \quad \text{(oracle's promise)} \\
&\leq \gamma^2 \cdot \lambda_1(\mathcal{L}).
\end{aligned}$$

Hence $\vec{z}_1$ is of the desired length. Since $\vec{z}_1$ or $\vec{z}_2$ is a desired length, the shorter of the two will be of the desired length.

Complexity. $\mathcal{A}$ clearly calls the oracle at most n times, and performs $\mathsf{poly}(n)$ auxiliary computations. □

We are now ready to present an algorithm that returns a vector which is within $k^{\frac{n}{2k}}$ factor of $\det(\mathcal{L})^{1/n}$. Algorithm 8, together with lemma 4.4.5 achieves a $k^{\frac{n}{k}}$-approximation for SVP.

Algorithm 8 Algorithm $\mathcal{A}(\mathcal{L}, \tau)$

Input: Basis B of lattice $\mathcal{L}$, depth parameter τ, parameter k.
if $\text{rank}(\mathcal{L}(B)) \leq k$ **then**
    Use an exact SVP oracle on B to obtain $\vec{y} \leftarrow \mathsf{SVP}(B)$
    return $\vec{y}$
else if $\tau = 0$ **then**
    Run LLL on B to obtain $\vec{y} \leftarrow \mathsf{LLL}(B)$
    return $\vec{y}$
else
    $\vec{\omega} \leftarrow \mathcal{A}(\mathcal{L}^*, \tau - 1)$, where $\mathcal{L}^*$ is the dual of $\mathcal{L}(B)$
    $\vec{y} \leftarrow \mathcal{A}(\mathcal{L} \cap \vec{\omega}^{\perp}, \tau)$
    return $\vec{y}$
end if

This recursive algorithm uses a depth parameter τ to control the recursion depth when calling the algorithm on the dual lattice. This ensures that the repeated calls eventually terminate when $\tau = 0$. Note that $\mathcal{L} \cap \vec{\omega}^{\perp}$ is well defined: if $\vec{v} \in \vec{w}^{\perp}$ then $\langle \vec{v}, \vec{w} \rangle = 0 \in \mathbb{Z}$; $\vec{w} \in \mathcal{L}^*$ and $\langle v, w \rangle \in \mathbb{Z}$ imply $\vec{v} \in \mathcal{L}$.

4.4.1 Analysis of Approximation Factor

Let $\gamma(n, \tau)$ denote the approximation factor (wrt $\det(\mathcal{L})^{1/n}$) achieved by Algorithm 8 on an n rank lattice with depth parameter τ. So, if the algorithm return a vector $\vec{v}$, we know that $\|\vec{v}\| \leq \gamma(n, \tau) \det(\mathcal{L})^{1/n}$. We aim to find an expression for $\gamma(n, \tau)$ that satisfies the recursive structure of the algorithm.
We start with the base cases:

$$\gamma(n, \tau) \leq \begin{cases} 2^{n/2} & \text{when } \tau = 0 \text{ (LLL)}, \\ k^{1/2} & \text{when } n = k \text{ (Minkowski's First Theorem)}. \end{cases}$$

For the recursive case, we have:

$$\|\vec{\omega}\| \leq \gamma(n, \tau - 1) \cdot \det(\mathcal{L}^*)^{1/n}, \tag{4.4.2}$$
$$\|\vec{y}\| \leq \gamma(n - 1, \tau) \cdot \det(\mathcal{L}_{\perp \vec{\omega}})^{1/(n-1)}. \tag{4.4.3}$$

Note that without loss of generality, we can assume that w is primitive, as $\mathcal{L} \cap w^{\perp}$ remains the same if w is scaled up. Combining equations (4.4.2), (4.4.3), and Corollary 4.4.4, we get:

$$\begin{aligned} \|\vec{y}\| &\leq \gamma(n - 1, \tau) \cdot (\det(\mathcal{L}) \cdot \|\vec{\omega}\|)^{1/(n-1)} \\ &\leq \gamma(n - 1, \tau) \cdot (\det(\mathcal{L}) \cdot \gamma(n, \tau - 1) \cdot \det(\mathcal{L}^*)^{1/n})^{1/(n-1)} \\ &= \gamma(n - 1, \tau) \cdot \gamma(n, \tau - 1)^{1/(n-1)} \cdot \det(\mathcal{L})^{1/n}. \end{aligned} \tag{4.4.4}$$

This implies:

$$\gamma(n, \tau) \leq \gamma(n - 1, \tau) \cdot \gamma(n, \tau - 1)^{1/(n-1)}. \tag{4.4.5}$$

To find a closed-form expression for $\gamma(n, \tau)$, we make an educated guess based on the recurrence relation. Let's consider the simplified case where we ignore the depth parameter τ:

$$\gamma(n) \leq \gamma(n-1) \cdot \gamma(n)^{1/(n-1)} \tag{4.4.6}$$
$$\Rightarrow \gamma(n)^{(n-2)/(n-1)} \leq \gamma(n-1) \tag{4.4.7}$$
$$\Rightarrow \gamma(n)^{1/(n-1)} \leq \gamma(n-1)^{1/(n-2)} \tag{4.4.8}$$
$$\Rightarrow \gamma(n)^{1/(n-1)} \leq \gamma(k)^{1/(k-1)} \tag{4.4.9}$$
$$\Rightarrow \gamma(n)^{1/(n-1)} \leq k^{1/(2(k-1))} \tag{4.4.10}$$
$$\Rightarrow \gamma(n) \leq k^{(n-1)/(2(k-1))}. \tag{4.4.11}$$

To account for the depth parameter τ, we introduce an error term above.

Claim 4.4.6.

$$\gamma(n, \tau) \leq k^{(n-1)/(2(k-1))} \cdot 2^{n^3/2^\tau}.$$

Proof. We can verify this bound by substituting it into the recurrence relation (4.4.5):

$$\begin{aligned}
\gamma(n, \tau) &\leq \gamma(n-1, \tau) \cdot \gamma(n, \tau-1)^{1/(n-1)} \\
&= k^{(n-2)/(2(k-1))} \cdot k^{1/(2(k-1))} \cdot 2^{((n-1)^3 + n^3/(n-1))/2^\tau} \\
&\approx k^{(n-1)/(2(k-1))} \cdot 2^{n^3/2^\tau}.
\end{aligned}$$

For the base case $\tau = 0$, we have $2^{n^3/2^\tau} = 2^{n^3}$, which upper bounds the approximation factor given by LLL. □

By setting $\tau = 4 \log n$, we get $2^{n^3/2^\tau} = 2^{1/n}$. This implies that our final approximation factor becomes:

$$\gamma(n, \tau) \leq k^{(n-1)/(2(k-1))} \cdot 2^{1/n} \approx k^{n/(2k)}.$$

4.4.2 Analysis of Running Time

Theorem 4.4.7. *Given inputs B, which is the basis of an n-rank lattice ($n > k$), and depth parameter $\tau = 4 \log n$, the running time of Algorithm 8 is $2^{k + O(\log^2 n)}$.*

Proof. Let $T(n, \tau)$ denote the number of calls to the SVP oracle when running *algorithm* 8 with an n-dimensional lattice and depth parameter τ. We have the following base cases:

$$T(n, \tau) = \begin{cases} 1 & \text{if } n = k, \\ 0 & \text{if } \tau = 0. \end{cases}$$

For the recursive case, we have:

$$T(n, \tau) = T(n, \tau - 1) + T(n - 1, \tau). \tag{4.4.12}$$

We claim that:

$$T(n, \tau) = \binom{n + \tau - k - 1}{\tau - 1}$$

satisfies the the base cases, which is immediate, and the recurrence relation (4.4.12).

To verify this, observe that:

$$T(n, \tau - 1) = \binom{n + (\tau - 1) - k - 1}{(\tau - 1) - 1} = \binom{n + \tau - k - 2}{\tau - 2},$$
$$T(n - 1, \tau) = \binom{(n - 1) + \tau - k - 1}{\tau - 1} = \binom{n + \tau - k - 2}{\tau - 1}.$$

Recall Pascal's Identity:

$$\binom{m}{r} = \binom{m - 1}{r} + \binom{m - 1}{r - 1}.$$

Let $m = n + \tau - k - 2$. Then:

$$\binom{m}{\tau - 2} + \binom{m}{\tau - 1} = \binom{m + 1}{\tau - 1} = \binom{n + \tau - k - 1}{\tau - 1} = T(n, \tau).$$

This verifies that our proposed formula for $T(n, \tau)$ satisfies the recurrence relation. Now, we can bound the running time:

$$T(n, \tau) \cdot 2^k = \binom{n + \tau - k - 1}{\tau - 1} \cdot 2^k \leq (n + \tau)^\tau \cdot 2^k \approx 2^{k + O(\log^2 n)}.$$

□

Chapter 5

Integer Programming

Now we will see how concepts from the theory of lattices can be used to solve the integer programming problem.

Definition 5.0.1 (Integer Programming Problem)**.** *Given a polytope $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ where $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m$, determine if P contains an integer point.*

Theorem 5.0.2. *Integer Programming (IP) problem is* NPComplete*.*

Proof. It is clearly in NP: given an integer point we can check if it lies in P by checking if it satisfies all the inequalities. We will now reduce 3-SAT to IP.

For a given 3-SAT instance over the variables $\{x_i\}_{i \in [n]}$ we will construct an IP instance with the same variables. We first impose the inequalities $\forall i \in [n] : 0 \leq x_i \leq 1$ in the IP instance. Then we convert each clause C_j into an inequality constraint as follows. Add up each variable x_i appearing in C_j, and add up $(1 - x_i)$ each negation $\overline{x_i}$ appearing in the clause, and require that the sum is ≥ 1. For example, given a clause $x_i \vee x_j \vee \overline{x_k}$, define an inequality constraint as $x_i + x_j + (1 - x_k) \geq 1$ in the IP instance. This completes the reduction. We now show that this is correct.
Suppose the 3-SAT instance has a solution (a $\{0, 1\}$ assignment). Clearly, the same assignment will satisfy the IP instance (notice that at least one assignment in each clause will be 1, and hence each inequality will be satisfied.) This point is clearly an integer point. Thus, if the 3-SAT instance has a solution, the IP instance has a solution.
Now suppose the 3-SAT instance has no solution. This means that there is no $\{0, 1\}$ assignment that satisfies the constraints. Clearly in this case there is no integer point that satisfies the constraints. Suppose for contradiction that there was. Since the x_i in the IP instance are integers, they must be 0 or 1 due to the constraint $0 \leq x_i \leq 1$. Then the same variables would satisfy the 3-SAT instance (sum being at least one).
Hence the 3-SAT instance has a solution iff the IP instance has a solution, and the reduction is obviously poly time. □

5.1 IP $\in \mathcal{O}(n^{O(n)})$

We will now show a $O(n^{O(n)})$ time algorithm for integer programming in n varibales. We use the following result without proof.

This chapter is based on lectures 8 and 9.

Lemma 5.1.1 (Algorithmic John Ellipsoids)**.** *Given a polytope P, one can efficiently compute an Ellipsoid E such that $E \subseteq P \subseteq 2n^{\frac{3}{2}}E$. Equivalently, one can find a linear transformation T, a vector $\vec{p}$, and a radius r such that $\mathcal{B}(\vec{p}, r) \subseteq T(P) \subseteq \mathcal{B}(\vec{p}, R)$, where $R = 2n^{\frac{3}{2}}r$.*

Here $\mathcal{B}(\vec{p}, r)$ is a ℓ_2 ball of radius r centered at $\vec{p}$. We want to determine if $P \cap \mathbb{Z}^n$ is empty, which is equivalent to determining if $T(P) \cap T(\mathbb{Z}^n)$ is empty. Notice that $T(\mathbb{Z}^n)$ is the lattice generated by the basis corresponding to the linear transformation T. Let $B = \{\vec{b}_1, \ldots, \vec{b}_n\}$ be the basis for $T(\mathbb{Z}^n) = \mathcal{L}(B)$. It is easy to see that if $\mathsf{CVP}(\mathcal{L}, \vec{p}) \leq r$ then $\mathcal{L} \cap T(P)$ is non-empty and if $\mathsf{CVP}(\mathcal{L}, \vec{p}) \geq R$ then it is empty. Let B be a KZ basis (definition 4.2.1) of the lattice. This can be computed in 2^n time. Let i be such that $\left\|\vec{b}_i^*\right\| = \max_j \left\|\vec{b}_j^*\right\|$. Now consider the following two cases:

Case 1: $r > \frac{\sqrt{n}}{2} \cdot \left\|\vec{b}_i^*\right\|$. Recall from eq. (3.2.1) given a lattice $\mathcal{L}$ and a target t, one can always find a lattice point $\vec{x} \in \mathcal{L}$ such that $\mathrm{dist}(\vec{x}, \vec{t}) \leq \frac{1}{2}\left(\sum_{i=1}^n \left\|\vec{b}_i^*\right\|^2\right)^{\frac{1}{2}}$. Here we have that $\frac{1}{2}\left(\sum_{i=1}^n \left\|\vec{b}_i^*\right\|^2\right)^{\frac{1}{2}} \leq \frac{1}{2}(n\left\|\vec{b}_i^*\right\|^2)^{\frac{1}{2}} = \frac{\sqrt{n}}{2}\left\|\vec{b}_i^*\right\| < r$. Therefore, we can always find a lattice point $x \in B(\vec{p}, r) \subseteq T(P)$. So P contains an integer vector.

Case 2: $r \leq \frac{\sqrt{n}}{2} \cdot \left\|\vec{b}_i^*\right\|$. This tells us that $\left\|\vec{b}_i^*\right\| \geq \frac{2r}{\sqrt{n}}$, and we learn something about this lattice. We will make use of this structure to bound the number of relevant vectors. We decompose the lattice $\mathcal{L}$ into a union of lattice shifts

$$\mathcal{L} = \bigcup_{(x_i,\ldots,x_n)\in\mathbb{Z}^{n-i+1}} \left(\mathcal{L}(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_{i-1}) + \sum_{j=i}^n x_j \vec{b}_j\right).$$

Suppose $\vec{y} \in T(P) \subseteq \mathcal{B}(\vec{p}, R)$, and let $\vec{y} = \sum_{j=1}^n x_j \vec{b}_j$. We will fix the coefficients $x_n, \ldots, x_i$, then recurse into the sublattice $\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{i-1})$. Suppose that for some j we have fixed the coordinates $x_n, \ldots, x_{j+1}$. Then we can get the following bound on the possible x_j:

$$\begin{aligned}
R &\geq \|\vec{y} - \vec{p}\| \\
&\geq \left\|\text{projection of } \vec{y} - \vec{p} \text{ onto } \vec{b}_j^*\right\| \\
&= \left|x_j + \sum_{k=j+1}^n \left(x_k \frac{\left\langle b_k, \vec{b}_j^*\right\rangle}{\left\langle \vec{b}_j^*, \vec{b}_j^*\right\rangle}\right) - \frac{\left\langle p, \vec{b}_j^*\right\rangle}{\left\langle \vec{b}_j^*, \vec{b}_j^*\right\rangle}\right| \cdot \left\|\vec{b}_j^*\right\| \\
&= |x_j + c| \cdot \left\|\vec{b}_j^*\right\|
\end{aligned}$$

for some constant c determined by $x_{j+1}, \ldots, x_n$. So there are at most $\frac{2R}{\|\vec{b}_j^*\|} + 1$ possible values of x_j. We then bound this:

$$\frac{R}{\left\|\vec{b}_j^*\right\|} = \frac{2n^{\frac{3}{2}}r}{\left\|\vec{b}_j^*\right\|} < \frac{n^2\left\|\vec{b}_i^*\right\|}{\left\|\vec{b}_j^*\right\|}$$

Hence there are at most

$$\frac{2n^2\left\|\vec{b}_i^*\right\|}{\left\|\vec{b}_j^*\right\|} + 1 \leq \frac{3n^2\left\|\vec{b}_i^*\right\|}{\left\|\vec{b}_j^*\right\|}$$

choices for x_j.

Let $\mathcal{L}_i$ be the usual: projection of $\mathcal{L}$ onto $\text{span}(\vec{b}_i^*, \ldots, \vec{b}_n^*)$. So by definition of KZ basis we have $\left\|\vec{b}_i^*\right\| = \lambda_1(\mathcal{L}_i)$. We can now bound the total number of choices for $x_i, \ldots, x_n$:

$$\begin{aligned}
\text{Total choices} &\leq \prod_{j=i}^{n} \frac{3n^2 \left\|\vec{b}_i^*\right\|}{\left\|\vec{b}_j^*\right\|} \\
&= (3n^2)^{n-i+1} \cdot \frac{\lambda_1(\mathcal{L}_i)^{n-i+1}}{\det(\mathcal{L}_i)} \\
&\leq (3n^2)^{n-i+1} \cdot (n-i+1)^{\frac{n-i+1}{2}} \qquad \text{(by Minkowski's Theorem)} \\
&\leq (3n)^{\frac{5}{2}(n-i+1)}
\end{aligned}$$

Hence if we let $T(n)$ be the total number of choices for $x_1, \ldots, x_n$, we have $T(n) \leq (3n)^{\frac{5}{2}(n-i+1)} \cdots T(i-1)$ by recursing into the lattice $\mathcal{L}(\vec{b}_1, \ldots, \vec{b}_{i-1})$ to compute the number of choices for $x_1, \ldots, x_{i-1}$. It's easy to see that this implies that $T(n) \leq n^{\frac{5}{2}n+o(n)}$, so we can solve IP in $O(n^{O(n)})$ by enumerating these vectors and checking if they lie in $T(P)$.

5.2 Integer Programming in Polynomial Time in Total Regime

5.2.1 1-dimension: Unbounded Subset Sum

Definition 5.2.1. *In the Unbounded Subset Sum (USS) problem, we are given n distinct positive integers $a_1 < \ldots < a_n$ and a target positive integer b with $\gcd(a_1, \ldots, a_n) \mid b$. The task is to find non-negative integers $x_1, \ldots, x_n$ such that*

$$x_1 a_1 + \cdots + x_n a_n = b, \tag{5.2.1}$$

whenever such an n-tuple $(x_1, \ldots, x_n) \in \mathbb{Z}_{\geq 0}^n$ exists.

The decision version of the problem is a variant of integer Knapsack, which is well known to be NP-complete, making USS NP-hard. The following existence result is due to Erdős and Graham [EG72] and Dixmier [Dix90]:

Theorem 5.2.2. *The USS instance $(a_1, \ldots, a_n, b)$ has a solution if $\gcd(a_1, ..., a_n)$ divides b and $b \geq \frac{a_n a_{n-1}}{n-1}$.*

Example 5.2.3. *We give an example illustrating the tightness of the condition $b \geq \frac{a_n a_{n-1}}{n-1}$. Let $n = 2$, and $a_1 < a_2$ are co-prime. Take*

$$b = a_1 a_2 - a_1 - a_2 < a_1 a_2 = \frac{a_n a_{n-1}}{n-1}. \tag{5.2.2}$$

Seeking contradiction, suppose there exists non-negative solution x_1, x_2 such that $b = x_1 a_1 + x_2 a_2$. Then we have $-a_2 \equiv x_2 a_2 \pmod{a_1}$. Since a_1 and a_2 are coprime, we have $a_2^{-1} \pmod{a_1}$ exists, and so $x_2 \equiv -1 \pmod{a_1}$. Since x_2 is non-negative, we have $x_2 \geq a_1 - 1$. Using a similar argument, we have also $x_1 \geq a_2 - 1$. But then we have

$$x_1 a_1 + x_2 a_2 \geq (a_2 - 1)a_1 + (a_1 - 1)a_2 = 2a_1 a_2 - a_1 - a_2 > a_1 a_2 - a_1 - a_2 = b, \tag{5.2.3}$$

which is a contradiction.

We now give a corresponding algorithmic result:

Theorem 5.2.4. *There is an algorithm that finds solutions $(x_1, \ldots, x_n)$ for the USS instance $(a_1, \ldots, a_n, b)$ if*

- $\gcd(a_1, ..., a_n)$ *divides* b
- $b \geq \frac{a_i a_{i-1}}{i-1}$ *for all* $i = 2, \ldots, n$.

Proof. We assume without loss of generality that $\gcd(a_1, \ldots, a_n) = 1$ since this is equivalent to finding a solution to the instance

$$\left(\frac{a_1}{\gcd(a_1, \ldots, a_n)}, \ldots, \frac{a_n}{\gcd(a_1, \ldots, a_n)}, \frac{b}{\gcd(a_1, \ldots, a_n)} \right), \tag{5.2.4}$$

where the assumption is still valid since $b \geq \frac{a_i a_{i-1}}{i-1}$ implies

$$\frac{b}{\gcd(a_1, \ldots, a_n)} \geq \frac{a_i a_{i-1}}{(i-1)\gcd(a_1, \ldots, a_n)} \geq \frac{a_i a_{i-1}}{(i-1)\gcd(a_1, \ldots, a_n)^2} = \frac{\frac{a_i}{\gcd(a_1, \ldots, a_n)} \cdot \frac{a_{i-1}}{\gcd(a_1, \ldots, a_n)}}{(i-1)}. \tag{5.2.5}$$

Let $d = \gcd(a_1, \ldots, a_{n-1})$. The algorithm recurses as follows:

- If $n = 1$, then we pick $x_1 = b/a_1$, which exists since $a_1 = \gcd(a_1)$ divides b.
- Set $x_n \equiv a_n^{-1} b \pmod{d}$, i.e., the unique integer in $\{0, 1, \ldots, d-1\}$ such that d divides $b - x_n a_n$.
- Recurse on instance $\left(\frac{a_1}{d}, \ldots, \frac{a_{n-1}}{d}, \frac{b - x_n a_n}{d} \right)$ to obtain $(x_1, \ldots, x_{n-1})$.

Note that in the second step, $a_n^{-1} \pmod{d}$ exists because $\gcd(d, a_n) = \gcd(a_1, \ldots, a_n) = 1$, i.e., d is relatively prime to a_n. We now prove the correctness of the algorithm by induction on the n. The base case $n = 1$ holds trivially. For the induction step, it is sufficient to prove

$$\frac{b}{a_i \cdot a_{i-1}} \leq \frac{(b - x_n a_n)/d}{a_i/d \cdot a_{i-1}/d} \tag{5.2.6}$$

since this implies

$$\frac{b - x_n a_n}{d} \geq \frac{a_i}{d} \cdot \frac{a_{i-1}}{d} \cdot \frac{b}{a_i \cdot a_{i-1}} \geq \frac{\frac{a_i}{d} \cdot \frac{a_{i-1}}{d} \cdot}{i-1} \quad \text{for all } i = 2, \ldots n-1, \tag{5.2.7}$$

and so the instance $\left(\frac{a_1}{d}, \ldots, \frac{a_{n-1}}{d}, \frac{b - x_n a_n}{d} \right)$ in the recursive step has a solution $(x_1, \ldots, x_{n-1})$ by the induction hypothesis. We now prove (5.2.6), which holds if and only if $b \leq (b - x_n a_n)d$, if and only if

$$x_n(a_n d) \leq (d-1)b. \tag{5.2.8}$$

Since $x_n \in \{0, 1, \ldots, d-1\}$, it is sufficient to prove that $a_n d \leq b$. Note that since $a_{n-1}, \ldots, a_1$ are distinct positive integer multiples of d, and $a_{n-1} > \ldots > a_1$, we have $a_{n-1} \geq d(n-1)$. It follows that

$$a_n d \leq \frac{a_n a_{n-1}}{n-1} \leq b \tag{5.2.9}$$

as desired. □

5.2.2 Integer Linear Programming with Equalities

Definition 5.2.5. *In the Integer Linear Programming (ILP) problem, we are given n distinct vectors $\vec{a}_1, \ldots, \vec{a}_n \in \mathbb{Z}^d$ and a target vector $\vec{b} \in \mathbb{Z}^d$. The task is to find non-negative integers $x_1, \ldots, x_n$ such that*

$$x_1\vec{a}_1 + \cdots + x_n\vec{a}_n \leq \vec{b}, \tag{5.2.10}$$

whenever such an n-tuple $(x_1, \ldots, x_n) \in \mathbb{Z}^n_{\geq 0}$ exists. By denoting $A = [\vec{a}_1, \ldots, \vec{a}_n] \in \mathbb{Z}^{d\times n}$, the ILP problem can equivalently be stated as finding $\vec{x} \in \mathbb{Z}^n_{\geq 0}$ such that $A\vec{x} \leq \vec{b}$.

Definition 5.2.6. *In the Integer Linear Programming with Equalities (ILPE) problem, we are given n distinct vectors $\vec{a}_1, \ldots, \vec{a}_n \in \mathbb{Z}^d$ and a target vector $\vec{b} \in \mathbb{Z}^d$. The task is to find non-negative integers $x_1, \ldots, x_n$ such that*

$$x_1\vec{a}_1 + \cdots + x_n\vec{a}_n = \vec{b}, \tag{5.2.11}$$

whenever such an n-tuple $(x_1, \ldots, x_n) \in \mathbb{Z}^n_{\geq 0}$ exists. By denoting $A = [\vec{a}_1, \ldots, \vec{a}_n] \in \mathbb{Z}^{d\times n}$, the ILPE problem can equivalently be stated as finding $\vec{x} \in \mathbb{Z}^n_{\geq 0}$ such that $A\vec{x} = \vec{b}$.

Remark 5.2.7. *Notice that the two variants of ILP mentioned above are computationally equivalent.*

- *An ILPE $Ax = b$ with d equality constraints can be reduced to an ILP with $2d$ inequality constraints $Ax \leq b$ and $-Ax \leq -b$.*
- *An ILP with d inequality constraints on n variables $x = (x_1, \ldots, x_n)$ given by $Ax \leq b$ can be reduced to an ILP with d equality constraints on $n + d$ variables $(x, y) = (x_1, \ldots, x_n, y_1, \ldots, y_d)$ given by $Ax + y = b$.*

Remark 5.2.8. *Since ILPE problem can be interpreted as a generalization of USS problem, it is natural to consider how to generalize the two sufficient conditions in Theorem 5.2.4. For the condition $\gcd$ divides b, we generalize it to $\vec{b} \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_n)$. However, an example below shows that the condition b being sufficiently large cannot be generalized to $\vec{b}$ being sufficiently large. Consider*

$$\vec{a}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \vec{a}_2 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} M \\ M \end{bmatrix} \tag{5.2.12}$$

for some integer M. Since $3\vec{a}_1 - 2\vec{a}_2 = [1, 0]$ and $\vec{a}_2 - \vec{a}_1 = [0, 1]$, we have $\vec{b} \in \mathcal{L}(\vec{a}_1, \vec{a}_2) = \mathbb{Z}_2$. However, there is no nonnegative integers x_1, x_2 satisfying $x_1\vec{a}_1 + x_2\vec{a}_2 = \vec{b}$, no matter how large M is.

Remark 5.2.9. *We will assume that A is a rank d matrix. This is without loss of generality, since if A has rank $d' < d$, then we can use Gaussian elimination to find a subset of d' linearly independent rows that forms a matrix $A' \in \mathbb{Z}^{d'\times n}$. Taking $\vec{b}' \in \mathbb{Z}^{d'}$ to be $\vec{b}$ restricted to the same rows, it suffices to find a solution to $A'\vec{x} = b'$. Furthermore, we will assume that $\vec{a}_1, \ldots, \vec{a}_d$ are linearly independent, i.e., the first d columns are linearly independent, which can be achieved by appropriately permuting the columns.*

Theorem 5.2.10. *Algorithm 9 finds solutions $(x_1, \ldots, x_n)$ for the ILPE instance $(\vec{a}_1, \ldots, \vec{a}_n, \vec{b})$ if the following conditions hold:*

- *$\vec{a}_1, \ldots, \vec{a}_d \in \mathbb{Z}^d$ are linearly independent;*
- *$\vec{b} \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_n)$;*

- *there exist* $y_1, \ldots, y_n \in \mathbb{R}_{\geq 0}$ *such that*

$$\sum_{i=1}^{n} y_i \vec{a}_i = \vec{b} \quad \text{and} \quad y_1, \ldots, y_d > (n-d) \left(\max_{1 \leq i \leq n} \|\vec{a}_i\| \right)^d. \tag{5.2.13}$$

Proof. We first show that $\sum_{i=1}^{n} x_i \vec{a}_i = \vec{b}$, where x_i is as defined in Line 14. By our assumption, $y_1, \ldots, y_n \in \mathbb{R}_{\geq 0}$ satisfying constraints (5.2.13) can be found in Line 1. Let

$$\vec{b}' = \vec{b} - \sum_{i=1}^{n} \lfloor y_i \rfloor \vec{a}_i \tag{5.2.14}$$

be as defined in Line 5, and let $x'_{n+1}, x'_n, \ldots, x'_{d+1}$ be the non-negative integers found Line 10. We will show in Lemma 5.2.11 a polynomial time procedure to find $x'_{n+1}, x'_n, \ldots, x'_{d+1}$ satisfying (5.2.16), but we do not directly use this property in this proof. Finally, let $x'_1, \ldots, x'_d$ be the numbers found in Line 12. By Lines 14 and 5 and (5.2.17), we have

$$\sum_{i=1}^{n} x_i \vec{a}_i = \sum_{i=1}^{n} \left(\lfloor y_i \rfloor + x'_i \right) \vec{a}_i = \sum_{i=1}^{n} \lfloor y_i \rfloor \vec{a}_i + \sum_{i=1}^{n} x'_i \vec{a}_i = \left(\vec{b} - \vec{b}' \right) + \vec{b}' = \vec{b}. \tag{5.2.15}$$

To ensure that $x_i = \lfloor y_i \rfloor + x'_i$ is a non-negative integer for each $i = 1, \ldots n$, it is sufficient to establish that $x'_1, \ldots, x'_d$ are all integers and $x'_1, \ldots, x'_d \geq -(n-d) \left(\max_{1 \leq i \leq n} \|\vec{a}_i\| \right)^d$. The proof for these properties is deferred to Lemma 5.2.12. The proof uses the guarantee that $x'_{n+1}, x'_n, \ldots, x'_{d+1}$ satisfy (5.2.16) - see the second last inequality in (5.2.29). □

Lemma 5.2.11. *There exists a polynomial time algorithm that finds* $x'_{n+1}, x'_n, \ldots, x'_{d+1}$ *satisfying* (5.2.16).

Proof. We prove this claim by induction. For the base case $k = n+1$,[1] we take $x'_{n+1} = 0$ and we have $\vec{b}' - \sum_{j=n+1}^{n} x'_j \vec{a}_j = \vec{b}' \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_n)$. Fix an arbitrary positive integer $k \in \{d+1, \ldots, n\}$. Suppose we have found an x'_{k+1} satisfying condition (5.2.16) for $i = k+1$. Specifically, we have

$$\vec{w}' \coloneqq \vec{b}' - \sum_{j=k+1}^{n} x'_j \vec{a}_j \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_k). \tag{5.2.18}$$

To finish the induction step, we need to find an $x'_k \in \mathbb{Z}_{\geq 0}$ such that

$$x'_k \in \{0, 1, \ldots, V_{k-1} - 1\} \quad \text{and} \quad \vec{w}' - x'_k \vec{a}_k \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_{k-1}). \tag{5.2.19}$$

We first find a lattice basis $B = [\vec{b}_1, \ldots, \vec{b}_d] \in \mathbb{Z}^{d \times d}$ of $\mathcal{L}(\vec{a}_1, \ldots, \vec{a}_{k-1})$ using any polynomial-time algorithm such as the LLL algorithm. Note that B is invertible and by Lecture 1, we have $\det(B) = \det(\mathcal{L}_{k-1}) = V_{k-1}$. Since $\vec{w}' \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_k)$, there must exist some $\gamma \in \mathbb{Z}$ such that

$$\vec{w}' - \gamma \vec{a}_k \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_{k-1}). \tag{5.2.20}$$

Since B is a lattice basis of $\mathcal{L}(\vec{a}_1, \ldots, \vec{a}_{k-1})$, we have

$$B^{-1}(\vec{w}' - \gamma \vec{a}_k) = \frac{\text{adj}(B)}{\det(B)} (\vec{w}' - \gamma \vec{a}_k) \in \mathbb{Z}^d \implies \text{adj}(B)(\vec{w}' - \gamma \vec{a}_k) \in \mathbb{Z}^d \det(B), \tag{5.2.21}$$

[1] We use x'_{n+1} only for the sake of induction, it has no other uses in the algorithm.

Algorithm 9 ILPE solver

Input: $\vec{a}_1, \ldots, \vec{a}_n \in \mathbb{Z}^d$ and a target vector $\vec{b} \in \mathbb{Z}^d$.

Output: $x_1, \ldots, x_n \in \mathbb{Z}_{\geq 0}$ satisfying $x_1\vec{a}_1 + \cdots + x_n\vec{a}_n = \vec{b}$.

Find $y_1, \ldots, y_n \in \mathbb{R}_{\geq 0}$ satisfying constraints (5.2.13) using an LP solver, which can be found in poly(n) time.
for $i = 1$ to n **do**
    Set $z_i = y_i - \lfloor y_i \rfloor \in [0, 1)$
end for
Set $\vec{b}' = \sum_{i=1}^n z_i\vec{a}_i = \vec{b} - \sum_{i=1}^n \lfloor y_i \rfloor \vec{a}_i$.
for $i = n$ to $d + 1$ **do**
    Let $\mathcal{L}_i = \mathcal{L}(\vec{a_1}, \ldots, \vec{a_i})$ and set $V_i = \det(\mathcal{L}_i)$.
end for
for $i = n + 1$ to $d + 1$ **do**
    Find x'_i such that

$$x'_i \in \{0, 1, \ldots, V_{i-1} - 1\} \quad \text{and} \quad \vec{b}' - \sum_{j=i}^{n} x'_j\vec{a_j} \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_{i-1}) \tag{5.2.16}$$

end for
Use Gaussian elimination to find $x'_1, \ldots, x'_d$ such that

$$\sum_{i=1}^{d} x'_i\vec{a}_i = \vec{b}' - \sum_{i=d+1}^{n} x'_i\vec{a}_i \tag{5.2.17}$$

for $i = 1$ to n **do**
    Set $x_i = \lfloor y_i \rfloor + x'_i$
end for

i.e., each entry of $\mathrm{adj}(B)(\vec{w}' - \gamma\vec{a}_k)$ is a multiple of $\det(B) = V_{k-1}$. Since $\vec{w}'$ and $\gamma\vec{a}_k$ are integer vectors, this gives us d modular equations of the form

$$\begin{aligned} \alpha_1 \equiv & \gamma\beta_1 \pmod{V_{k-1}} \\ & \vdots \\ \alpha_d \equiv & \gamma\beta_d \pmod{V_{k-1}} \end{aligned} \tag{5.2.22}$$

where α_i and β_i are the i-th coordinates of $\mathrm{adj}(B)\vec{w}'$ and $\mathrm{adj}(B)\vec{a}_k$ respectively. This implies that $\gcd(\beta_i, V_{k-1})$ divides α_i. Therefore, we can divide α_i, β_i, and V_{k-1} by $\gcd(\beta_i, V_{k-1})$ to obtain

$$\begin{aligned} \alpha_1' \equiv & \gamma\beta_1' \pmod{v_1'} \\ & \vdots \\ \alpha_d' \equiv & \gamma\beta_d' \pmod{v_d'} \end{aligned} \tag{5.2.23}$$

where each v_i' is a factor of V_{k-1}, and β_i' and v_i' are coprime. This gives

$$\begin{aligned} \gamma \equiv & \beta_1'^{-1}\alpha_1' \pmod{v_1'}, \\ & \vdots \\ \gamma \equiv & \beta_d'^{-1}\alpha_d' \pmod{v_d'} \end{aligned} \tag{5.2.24}$$

This gives exactly one solution γ modulo $\mathrm{lcm}(v_1', \ldots, v_d')$, i.e., $\gamma \in \{0, 1, \ldots, \mathrm{lcm}(v_1', \ldots, v_d') - 1\}$ which can be found using Chinese Remainder Theorem. By picking $x_k' = \gamma \le \mathrm{lcm}(v_1', \ldots, v_d') - 1 \le V_{k-1} - 1$, we see that x_k' satisfies (5.2.19) as desired. □

Lemma 5.2.12. *Let $x_1', \ldots, x_d'$ be the numbers found in Line 12. For each $i = 1, \ldots, d$, we have $x_i \in \mathbb{Z}$ and $x_i' \ge -(n-d)\left(\max_{1\le i\le n} \|\vec{a}_i\|\right)^d$.*

Proof. Write

$$\vec{b}^* := \vec{b}' - \sum_{j=d+1}^{n} x_j'\vec{a}_j = \sum_{i=1}^{d} z_i\vec{a}_i + \sum_{j=d+1}^{n} (z_j - x_j')\vec{a}_j. \tag{5.2.25}$$

By Lemma 5.2.11, we have $\vec{b}^* \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_d)$. Since $\vec{a}_1, \ldots, \vec{a}_d$ are linearly independent, there is a unique $(x_1', \ldots, x_d') \in \mathbb{R}^d$ such that

$$b^* = \sum_{i=1}^{d} x_i'\vec{a}_i. \tag{5.2.26}$$

We have $(x_1', \ldots, x_d') \in \mathbb{Z}^d$ since $\vec{b}^* \in \mathcal{L}(a_1, \ldots, \vec{a}_d)$. Furthermore, it is straightforward to compute $(x_1', \ldots, x_d')$ using Gaussian elimination. It remains to prove that $x_i' \ge -(n-d)\left(\max_{1\le i\le n} \|\vec{a}_i\|\right)^d$ for $i = 1, \ldots, d$. By symmetry, it is sufficient to prove that $x_d' \ge -(n-d)\left(\max_{1\le i\le d} \|\vec{a}_i\|\right)^d$.
We will make use of the following equation

$$\sum_{i=1}^{d}(x_i' - z_i)\vec{a}_i = \sum_{j=d+1}^{n} (z_j - x_j')\vec{a}_j, \tag{5.2.27}$$

which can be obtained by combining (5.2.25) and (5.2.26). Let the projection of $\vec{a}_d$ orthogonal to $\vec{a}_1, \ldots, \vec{a}_{d-1}$ to be $\tilde{a}_d$, and let $\pi_{\tilde{a}_d}(u)$ be the projection of any vector $u \in \mathbb{R}^d$ in the direction of $\tilde{a}_d$. We now project both sides of (5.2.27) in the direction of $\tilde{a}_d$. On the LHS, we obtain

$$\left\| \pi_{\tilde{a}_d} \left(\sum_{i=1}^{d} (x'_i - z_i)\vec{a}_i \right) \right\| = |x'_d - z_d| \|\tilde{a}_d\| = |x'_d - z_d| \frac{V_d}{V_{d-1}}, \tag{5.2.28}$$

where the first equality follows from $\tilde{a}_d$ being orthogonal to $\vec{a}_1, \ldots, \vec{a}_{d-1}$, and the second equality follows from properties of lattice determinant: $\det(\mathcal{L}_d) = \det(\mathcal{L}_{d-1})|\tilde{a}_d|$. On the RHS, we obtain

$$\begin{aligned}
\left\| \pi_{\tilde{a}_d} \left(\sum_{j=d+1}^{n} (z_j - x'_j)\vec{a}_j \right) \right\| &\leq \left\| \sum_{j=d+1}^{n} (z_j - x'_j)\vec{a}_j \right\| \\
&\leq \sum_{j=d+1}^{n} \left\| (z_j - x'_j)\vec{a}_j \right\| \\
&\leq \sum_{j=d+1}^{n} |z_j - x'_j| \left(\max_{1 \leq i \leq n} \|\vec{a}_i\| \right) \\
&\leq \sum_{j=d+1}^{n} V_{j-1} \left(\max_{1 \leq i \leq n} \|\vec{a}_i\| \right) \\
&\leq (n-d) V_d \left(\max_{1 \leq i \leq n} \|\vec{a}_i\| \right),
\end{aligned} \tag{5.2.29}$$

where the second last inequality follows from $x'_j \in \{0, 1, \ldots V_{j-1} - 1\}$ for $j \geq d+1$ and $z_j \in [0, 1)$, and the last inequality follows from $\mathcal{L}(\vec{a}_1, \ldots, \vec{a}_{j-1})$ is a superlattice of $\mathcal{L}(\vec{a}_1, \ldots, \vec{a}_d)$ for each $j \geq d+1$, which implies $V_{j-1} = \det(\mathcal{L}_{j-1}) \leq \det(\mathcal{L}_d) = V_d$. Combining (5.2.27)–(5.2.29), we have

$$|x'_d - z_d| \leq (n-d) \left(\max_{1 \leq i \leq n} \|\vec{a}_i\| \right) V_{d-1} \leq (n-d) \left(\max_{1 \leq i \leq n} \|\vec{a}_i\| \right)^d, \tag{5.2.30}$$

where the last inequality follows from $\det(\mathcal{L}_{d-1}) \leq \prod_{i=1}^{d-1} \|\vec{a}_i\| \leq \left(\max_{1 \leq i \leq n} \|\vec{a}_i\| \right)^{d-1}$. Since $z_d \geq 0$, we have $x'_d \geq -(n-d) \left(\max_{1 \leq i \leq d} \|\vec{a}_i\| \right)^d$ as desired. □

5.3 Lower bound for the Sufficient Condition

We showed a sufficient condition under which an ILPE instance has a solution that can be computed in polynomial time. We now show that the condition is almost tight.

Theorem 5.3.1. *For any $d \geq 2$, there exists $\vec{a}_1, \ldots, \vec{a}_{d+1} \in \mathbb{Z}^d$, and $\vec{b} \in \mathbb{Z}^d$ such that $\vec{b} \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_{d+1})$ and there exist $y_1, \ldots, y_{d+1} \in \mathbb{R}_{\geq 0}$ satisfying*

$$\sum_{i=1}^{d+1} y_i \vec{a}_i = \vec{b} \quad \text{and} \quad y_1, \ldots, y_d > \frac{\left(\max_{1 \leq i \leq d+1} \|\vec{a}_i\| \right)^d}{20\sqrt{d}}, \tag{5.3.1}$$

but there do not exist non-negative integers $x_1, \ldots, x_{d+1} \in \mathbb{Z}_{\geq 0}$ satisfying $\sum_{i=1}^{d+1} x_i \vec{a}_i = \vec{b}$.

Proof. By [HB88], there exists $c \geq 5$ such that there are at least d distinct primes between cd^2 and $cd^2(1-1/d)$. Let $p_1, \ldots, p_d$ be d distinct primes between $cd^2 - cd$ and cd^2 and $P = p_1 \cdots p_d$. Then we have

$$\Delta := \max_{1 \leq i \leq d} p_i \in (cd^2 - cd, cd^2], \quad p_i \geq cd^2 - cd = cd^2(1-1/d) \geq \Delta(1-1/d) \quad \text{and} \quad \frac{P}{p_i} \geq \Delta^{d-1}(1-1/d)^{d-1} \tag{5.3.2}$$

and for each $i \in \{1, \ldots d\}$. Also, let p_{d+1} be a prime such that

$$\frac{\Delta}{2\sqrt{d}} \leq p_{d+1} \leq \frac{\Delta}{\sqrt{d}}. \tag{5.3.3}$$

Let $\vec{a}_i$ for $i = 1, \ldots, d$, and let $\vec{a}_{d+1}$, and let $\vec{b}$ be defined as follows:

$$\vec{a}_1 = \begin{bmatrix} p_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \vec{a}_d = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ p_d \end{bmatrix}, \vec{a}_{d+1} = \begin{bmatrix} p_{d+1} \\ p_{d+1} \\ \vdots \\ p_{d+1} \end{bmatrix}, \vec{b} = \begin{bmatrix} p_1 \cdot p_2 \cdots p_{d+1} - p_1 - p_{d+1} \\ p_1 \cdot p_2 \cdots p_{d+1} - p_2 - p_{d+1} \\ \vdots \\ p_1 \cdot p_2 \cdots p_{d+1} - p_d - p_{d+1} \end{bmatrix} = \begin{bmatrix} p_{d+1}(P-1) - p_1 \\ p_{d+1}(P-1) - p_2 \\ \vdots \\ p_{d+1}(P-1) - p_d \end{bmatrix} \tag{5.3.4}$$

Note that we have $\|\vec{a}_i\| \leq \Delta$ for each $i \in [d+1]$.
We first show that $\vec{b} \in \mathcal{L}(\vec{a}_1, \ldots, \vec{a}_{d+1})$ by showing that $\mathcal{L}(a_1, \ldots, a_{d+1}) = \mathbb{Z}^d$. To see this, consider any vector $\vec{c} = (c_1, \ldots, c_d) \in \mathbb{Z}^d$. By the Chinese Remainder Theorem, there is a unique integer $\gamma_{d+1} \in \{0, 1, \ldots, P-1\}$ such that for each $i \in \{1, \ldots, d\}$

$$\gamma_{d+1} \equiv p_{d+1}^{-1} c_i \pmod{p_i}. \tag{5.3.5}$$

This implies $c_i - p_{d+1}\gamma_{d+1}$ is an integer multiple of p_i. By letting

$$\gamma_i = \frac{c_i - p_{d+1}\gamma_{d+1}}{p_i} \in \mathbb{Z}, \tag{5.3.6}$$

we have

$$\sum_{i=1}^{d+1} \gamma_i \vec{a}_i = \sum_{i=1}^{d} \frac{c_i - p_{d+1}\gamma_{d+1}}{p_i} p_i \vec{e}_i + \gamma_{d+1} \begin{bmatrix} p_{d+1} \\ \vdots \\ p_{d+1} \end{bmatrix} = \sum_{i=1}^{d} c_i \vec{e}_i = \vec{c}. \tag{5.3.7}$$

Next, we show that there exist large real $y_1, \ldots, y_{d+1} \in \mathbb{R}_{\geq 0}$ satisfying (5.3.1) Let $y_{d+1} = \frac{P}{2}$, and

$$y_i = p_{d+1} \cdot \frac{P}{2p_i} - 1 - \frac{p_{d+1}}{p_i} = \frac{p_{d+1}}{p_i}\left(\frac{P}{2} - 1\right) - 1 \tag{5.3.8}$$

for $1 \leq i \leq d$. Then $\sum_{i=1}^{d+1} y_i \vec{a}_i = \vec{b}$. Furthermore, for all $i \in [d]$, using (5.3.8), (5.3.3), (5.3.2)

$$y_i \geq \frac{\Delta}{2\sqrt{d}} \cdot \frac{\Delta^{d-1}(1-1/d)^{d-1}}{2} - 1 - \frac{\Delta}{\sqrt{d}(cd^2 - cd)} \tag{5.3.9}$$

$$\geq \frac{\Delta^d}{4e\sqrt{d}} - 2 \tag{5.3.10}$$

$$\geq \frac{\Delta^d}{20\sqrt{d}}, \tag{5.3.11}$$

where we use the fact that for $d \geq 2$, we have $(1-1/d)^{d-1} \geq \frac{1}{e}$ and $\Delta \geq cd^2 - cd \geq 5d^2 - 5d \geq 10$.

It remains to show that there does not exist non-negative integers $x_1, \ldots, x_{d+1}$ such that

$$\sum_{i=1}^{d+1} x_i \vec{a}_i = x_1 \begin{bmatrix} p_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \cdots + x_d \begin{bmatrix} 0 \\ 0 \\ \vdots \\ p_d \end{bmatrix} + x_{d+1} \begin{bmatrix} p_{d+1} \\ p_{d+1} \\ \vdots \\ p_{d+1} \end{bmatrix} = \begin{bmatrix} p_{d+1}(P-1) - p_1 \\ p_{d+1}(P-1) - p_2 \\ \vdots \\ p_{d+1}(P-1) - p_d \end{bmatrix} = \vec{b}. \tag{5.3.12}$$

Seeking contradiction, suppose there exist non-negative integers satisfying (5.3.12). Then, we must have

$$x_i p_i + x_{d+1} p_{d+1} = P p_{d+1} - p_i - p_{d+1} = (p_1 \cdots p_d) p_{d+1} - p_i - p_{d+1}, \tag{5.3.13}$$

for each $i = 1, \ldots, d$, which implies that $x_{d+1} = -1 \pmod{p_i}$ for all $i \in [d]$. By Chinese Remainder Theorem, this implies $x_{d+1} = -1 \pmod{P}$. Since x_{d+1} is non-negative, we must have $x_{d+1} \geq P - 1$. This implies that for $1 \leq i \leq d$,

$$x_i p_i = P p_{d+1} - p_i - p_{d+1} - x_{d+1} p_{d+1} \leq P p_{d+1} - p_i - p_{d+1} - (P-1) p_{d+1} = -p_i < 0, \tag{5.3.14}$$

which is a contradiction. □

Chapter 6

Lattice Based Cryptography

In this chapter, we explore two related problems which are used in cryptography. The proofs of their average-case hardness will be deferred to the next chapter.

Average-Case Hardness vs Worst-Case Hardness. Recall the RSA cryptosystem: two humongous primes p and q are chosen in secret; their product $N := pq$ is published as a public key. The security of RSA stems from the believed difficulty in factoring large numbers. In this case, even though N is known to the public, p and q are difficult to obtain. We do not go into the details for how RSA works, but the paradigm is that some problem is believed to be hard, and based on this belief, we design cryptosystems.

One key point to note is that we not only need the worst-case hardness of a problem: if we view the generation of a problem instance as sampling from some distribution of problems, we need that the *average* instance coming from this distribution is hard, not just the worst-case. Ergo, we cannot directly build cryptosystems from worst-case hardness; we need the hardness of the average case, with respect to some distribution. For some lattice problems, we can find nice average case instance distribution $\mathcal{D}$ such that

- It is as hard as a worst-case problem; and
- It is "useful", say in cryptography.

6.1 Learning with Errors

For a set S, denote by $v \leftarrow S$ an element sampled uniformly at random from S. For a distribution $\mathcal{D}$, denote by $v \leftarrow \mathcal{D}$ a sample from the distribution. Also, by writing $\vec{x}$, we mean $\vec{x}$ to be column vector. Consider the following problem.

Definition 6.1.1 (Learning With Errors (LWE) [Reg05])**.** *Let q be a fixed large prime. Let $\vec{s} \in \mathbb{Z}_q^n$ be a secret, and $e_1, \ldots, e_m \in \mathbb{Z}$ be (small) unknown errors. Now given $\vec{a}_1, \ldots, \vec{a}_m \in \mathbb{Z}_q^n$, as well as the values $\langle \vec{a}_i, \vec{s} \rangle + e_i \pmod q$ for each $1 \leq i \leq m$, find $\vec{s}$.*

Remark 6.1.2. LWE *is easy without errors using: Gaussian Elimination.*

In the above problem, view the $\vec{a}_i$ as uniformly sampled from $\mathbb{Z}_q^n$, i.e. $\vec{a} \leftarrow \mathbb{Z}_q^n$; and view the e_i as samples from a distribution χ on $\mathbb{Z}$ which outputs integers of small magnitude, e.g. uniform on $\{-B, -B+1, \ldots, B-1, B\}$, where $B \ll q$; or a discrete Gaussian with standard deviation $\sigma \ll q$.

This chapter is based on lecture 10.

We can stack the row vectors $\vec{a}_i^\top$ into a matrix $A^\top$, and the errors into a vector $\vec{e}$, and then rewrite the instance compactly as

$$A^\top \vec{s} + \vec{e} = \underbrace{\begin{bmatrix} — & a_1^\top & — \\ — & a_2^\top & — \\ & \vdots & \\ — & a_m^\top & — \end{bmatrix}}_{A^\top} \begin{bmatrix} | \\ \vec{s} \\ | \end{bmatrix} + \underbrace{\begin{bmatrix} e_1 \\ \vdots \\ e_m \end{bmatrix}}_{\vec{e}}.$$

And now given $A^\top$, $A^\top \vec{s} + \vec{e}$, find $\vec{s}$.
The problem comes with parameters for which we have not decided their scale. For concreteness, think of q as a large prime in the order of n^4; m polynomially small in n, say $\mathcal{O}(n^2)$; and $|e_i| < \frac{q}{n^2}$ with high probability.
LWE **as a lattice problem.** We can view LWE as a lattice problem using the following formulation: Consider the lattice $\mathcal{L}$ spanned by the columns of the block matrix $\begin{bmatrix} A^\top & qI_m \end{bmatrix}$. The motivation behind adding the qI_m matrix is to capture the component wise modulus $\pmod q$ operation. Now, if $\vec{e}$ is small enough, then $\vec{s}$ is the first n entries in the coefficient vector of the closest vector to $A^\top \vec{s} + \vec{e} \pmod q$ in $\mathcal{L}$.
Before we use the LWE problem to build cryptographic primitives, let's explore an adjacent problem.

6.2 Short Integer Solution

Definition 6.2.1 (Short Integer Solution (SIS) [Ajt96])**.** *Suppose given $A \in \mathbb{Z}_q^{n \times m}$ and $\vec{b} \in \mathbb{Z}_q^n$, find* short *$\vec{e} \in \mathbb{Z}^m$ such that $A\vec{e} = \vec{b} \pmod q$.*

"Short" can take on the following definitions:

1. $\vec{e} \in \{0,1\}^m$;
2. $\vec{e} \in \{-B, -B+1, \ldots, B-1, B\}^m$ for some $B \ll q$; or
3. $\vec{e}$ in a Euclidean ball of small radius centered on the origin.

For now, let's take $A \leftarrow \mathbb{Z}_q^{n \times m}$ and use the second definition of "short" from above.

Remark 6.2.2. *If we do not insist that the solution needs to be short, the problem is trivial via Gaussian Elimination.*

Definition 6.2.3 (Total Problems)**.** *A worst-case computational search problem is* total *if a solution always exists. An average-case computational search problem is* total *if a solution exists with high probability.*

Theorem 6.2.4. *If $\vec{b} = \vec{0}$ and $(B+1)^m > q^n$ then SIS is total in the worst case.*

Proof. For any $\vec{z} \in \{0, 1, \ldots, B\}^m$, note that $A\vec{z} \pmod q \in \mathbb{Z}_q^n$; if $(B+1)^m > q^n$, by pigeonhole principle, A cannot be injective so we may find $\vec{z}_1, \vec{z}_2 \in \{0, 1, \ldots, B\}^m$ such that $A(\vec{z}_1 - \vec{z}_2) = 0$. Then $\vec{e} := \vec{z}_1 - \vec{z}_2 \in \{-B, \ldots, B\}^m$ is a solution to SIS. □

Theorem 6.2.5. *If $\vec{b} \leftarrow \mathbb{Z}_q^n$ and $(2B+1)^m \gg q^n$, then* SIS *is total in the average case. That is, a solution exists with high probability.*

Proof Sketch. For any $A \in \mathbb{Z}_q^{n \times m}$, define $h_A : \mathbb{Z}_q^m \to \mathbb{Z}_q^n$ by $\vec{z} \mapsto A\vec{z}$. Recall that $\{h_A : A \in \mathbb{Z}_q^{n \times m}\}$ forms a universal hash family. In particular, we are hashing $(2B+1)^m$ items into q^n buckets; with high probability there will be no empty buckets, i.e. a randomly selected $\vec{b} \in \mathbb{Z}_q^n$ is the image of some element in $\mathbb{Z}_q^m$ under A. □

On the other hand, if $m \ll \frac{n \log q}{\log B}$, then with high probability there is no solution for a random $\vec{b}$. However, we can still plant a solution by sampling a secret $\vec{e} \leftarrow \{-B, \ldots, B\}^m$ and taking $\vec{b} = A\vec{e}$.

6.3 Cryptomania

Recall that the SIS problem is of the form: given $A, \vec{b}$, find $\vec{e}$ such that $A\vec{e} = \vec{b}$. We can also rewrite LWE similarly: Given $A^\top \in \mathbb{Z}_q^{m \times n}$ and $\vec{y} := A^\top \vec{s} + \vec{e}$, define $A^\perp \in \mathbb{Z}_q^{(m-n) \times m}$ to be a full rank matrix such that $A^\perp A^\top = 0 \pmod q$. In our setting, we can view $A^\perp$ as a random matrix. We can now write

$$A^\perp \vec{y} = A^\perp (A^\top \vec{s} + \vec{e}) = A^\perp \vec{e}$$

which is equivalent to SIS with $\vec{b} = A^\perp \vec{y}$. With that, we have the following remarks:

1. Planted SIS is the same as LWE;
2. The difference between SIS and LWE is mainly whether we are in the "planted" setting or in the "total" setting;
3. Algorithmically, based on our current knowledge, the "planted" setting and "total" setting are kind of similar;
4. Cryptographically, applications differ: SIS gives us minicrypt (one way functions, collision-resistant hash functions, symmetric key encryption, etc), while LWE gives cryptomania (public key encryption, fully homomorphic encryption etc.).

Theorem 6.3.1. *We can build a collision-resistant hash function from SIS.*

Proof. We have an SIS instance (A, b, q). Let $m \gg \frac{n \log q}{\log B}$. For $\vec{z} \in \{0, \ldots, B\}^m$, define $h_A(\vec{z}) := A\vec{b} \pmod q$. Then finding distinct $\vec{z}_1, \vec{z}_2 \in \{0, \ldots, B\}^m$ such that $h_A(\vec{z}_1) = h_A(\vec{z}_2)$ is equivalent to finding $\vec{e} \in \{-B, \ldots B\}^m$ such that $A\vec{e} = \vec{0} \pmod q$. □

Remark 6.3.2. *One can build the following from collision-resistant hash functions:*

1. *One-way functions;*
2. *Symmetric key encryption;*
3. *Digital signatures.*

Theorem 6.3.3. *We can build public key encryption from* LWE.

Proof. We can construct a protocol achieving the following:

1. Key Generation: produce a public key p_k and a secret key s_k;
2. Encryption: given a message a and the public key p_k, we need to define $c = \mathsf{Enc}(m, p_k)$;

3. Decryption: given an encrypted c and the secret key s_k, we need to define $a^* = \mathsf{Dec}(c, s_k)$;
4. Correctness: we need $\mathsf{Dec}(\mathsf{Enc}(a, p_k), s_k) = a$ with probability 1 or with high probability;
5. Security: for any $a \neq a'$, we need $\mathsf{Enc}(a, p_k) \approx \mathsf{Enc}(a', p_k)$.

We will use the following decision variant of LWE, which has been shown to be as hard as the search variant by Regev [Reg05].

Definition 6.3.4 (Decision-LWE). *Distinguish between the following distributions*

- $(A^\top, A^\top \vec{s} + \vec{e})$ *where* $A^\top \leftarrow \mathbb{Z}_q^{m \times n}$ *and* $\vec{e} \leftarrow \chi$*; and*
- $(A^\top, \vec{b})$ *where* $A^\top \leftarrow \mathbb{Z}_q^{m \times n}$ *and* $\vec{b} \leftarrow \mathbb{Z}_q^n$.

Theorem 6.3.5 (Regev, 2005 [Reg05]). *Search-*LWE *is equivalent to Decision-*LWE.

We now construct the required components, using an instance of LWE, $(A^\top, e)$ chosen as in the above definition, where $e_i \ll \frac{q}{n}$ with high probability.

1. Key Generation: Take the secret key $s_k = \vec{s} \in \mathbb{Z}_q^n$, sampled uniformly. The public key will be the LWE input, $p_k = (A^\top, \vec{y})$ where $\vec{y} := A^\top \vec{s} + \vec{e}$.
2. Encryption: for simplicity, we describe only the encoding of $a \in \{0, 1\}$. Sample $\vec{r} \leftarrow \{0, 1\}^m$ and define $\mathsf{Enc}(a) := (\vec{r}^\top A^\top, \vec{r}^\top \vec{y} + a\lfloor \frac{q}{2} \rceil)$.

   Concretely, if we define $\vec{v} := \vec{r}^\top A^\top$, we see that $\vec{v}$ is just the sum of a random subset of rows of $A^\top$, which looks like a random vector. On the other hand, $\vec{r}^\top \vec{y} = \vec{r}^\top (A^\top \vec{s} + \vec{e}) = \vec{v}^\top \vec{s} + \vec{r}^\top \vec{e}$, where $\vec{r}^\top \vec{e}$ is small because $\vec{r}$ is a binary vector and $\vec{e}$ is small.

   With this notation, we now have

$$\mathsf{Enc}(0) = (\vec{r}^\top A^\top, \vec{r}^\top \vec{y}) = (\vec{v}, \vec{v}^\top \vec{s} + \vec{r}^\top \vec{e})$$
$$\mathsf{Enc}(1) = (\vec{r}^\top A^\top, \vec{r}^\top \vec{y} + \lfloor \frac{q}{2} \rceil) = (\vec{v}, \vec{v}^\top \vec{s} + \vec{r}^\top \vec{e} + \lfloor \frac{q}{2} \rceil).$$

3. Decryption: check if $\left|\vec{r}^\top A^\top \vec{s} - (\vec{r}^\top \vec{y} + a\lfloor \frac{q}{2} \rceil)\right|$ is close to 0 or close to $\lfloor \frac{q}{2} \rceil$. If it is the first case, output 0; otherwise output 1. To see why this works, simply substitute:

$$\begin{aligned}
\left|\vec{r}^\top A^\top \vec{s} - \vec{r}^\top \vec{y} - a\lfloor \frac{q}{2} \rceil\right| &= \left|\vec{r}^\top A^\top \vec{s} - \vec{r}^\top (A^\top \vec{s} + \vec{e}) - a\lfloor \frac{q}{2} \rceil\right| \\
&= \left|-\vec{r}^\top \vec{e} - a\lfloor \frac{q}{2} \rceil\right| \\
&= \left|\vec{r}^\top \vec{e} + a\lfloor \frac{q}{2} \rceil\right| \\
&\approx a\lfloor \frac{q}{2} \rceil.
\end{aligned}$$

4. Correctness is guaranteed with high probability, since a solution was "planted" initially.
5. Security: By decision LWE assumption, $(\vec{v}, \vec{v}^\top \vec{s} + e)$ *looks like* $(\vec{v}, b)$, where b is uniform on $\mathbb{Z}_q$. Also, $(\vec{v}, \vec{v}^\top \vec{s} + e + \lfloor \frac{q}{2} \rceil) \approx (\vec{v}, b + \lfloor \frac{q}{2} \rceil) \approx (\vec{v}, b)$ as well. So being able to figure out a is equivalent to solving decision LWE.

The above protocol can easily be extended to work on longer messages.

□

Chapter 7

Average Case Hardness of SIS

We introduced SIS and LWE in the previous lecture, and we discussed that these problems show the much desired property of average case hardness which make them useful in constructing cryptographic schemes. By reducing some (believed to be) worst-case lattice problems, such as CVP, SVP and SIVP, to SIS or LWE in the average case, we demonstrate the average-case hardness of SIS and LWE. In this lecture, we show a reduction from worst-case SIVP to average-case SIS. This reduction was originally shown by Ajtai [Ajt96], but we follow the reduction from the work of Micciancio and Regev [MR04]. Recall the definition of SIVP and SIS.

Definition 7.0.1 (SIVP_γ)**.** *Given a basis $B \in \mathbb{Z}^{n\times n}$, find a set of n linearly independent vectors in $\mathcal{L}(B)$ each of length at most $\gamma\lambda_n(\mathcal{L}(B))$.*

Definition 7.0.2 (SIS)**.** *Given $A \in \mathbb{Z}_p^{n\times m}$, find a short vector $\vec{e} \in \mathbb{Z}^m$, where $\forall i \in [m],\ |e_i| < B$ for a small constant B such that $A\vec{e} = 0 \pmod p$.*

We present a reduction R such that

$$B \xrightarrow{R} A \leftarrow \mathbb{Z}_p^{n\times m},$$

where $B \in \mathbb{Z}^{n\times n}$ is an instance of SIVP_γ, and A is uniformly distributed over $\mathbb{Z}_p^{n\times m}$. For any instance B, the reduction R appends some randomness to B and turns it into a uniform distribution. Although the reduction always produces a uniform distribution, the randomness generated by R helps us recover the short vector of $\mathcal{L}(B)$ from the solution of the SIS instance. We begin by introducing the notion of a distribution on the lattice vectors.

7.1 Discrete Gaussian Distribution

The probability density function (PDF) of Gaussian distribution over $\mathbb{R}$ with zero mean and parameter s is

$$\rho_s(x) := \frac{1}{s}e^{-\pi x^2/s^2}.$$

This can be generalized to $\mathbb{R}^n$ as the PDF

$$\rho_s(\vec{x}) := \frac{1}{s^n}e^{-\pi\left(x_1^2+x_2^2+\cdots+x_n^2\right)/s^2} = \frac{1}{s^n}e^{-\pi\|\vec{x}\|^2/s^2}. \tag{7.1.1}$$

This chapter is based on lecture 11.

Remark 7.1.1. *Note that a centered (at 0) Gaussian distribution has the following properties (easy exercise).*

1. *It is directionally invariant (the function depends on the length of the vector, and is independent of the direction).*
2. *The expected length of vectors sampled according ρ is $s\sqrt{n}$.*

We can extend this notion to a discrete Gaussian distribution $D_{\mathcal{L},s}$ over a lattice $\mathcal{L}$ with parameter s. For a set S of vectors, we define $\rho(S) := \sum_{\vec{x}\in S} \rho(\vec{x})$. Then the discrete Gaussian distribution is defined as follows.

$$\Pr[X = \vec{x}] = \frac{\rho_s(\vec{x})}{\rho_s(\mathcal{L})} = \frac{e^{-\pi\|\vec{x}\|^2/s^2}}{\sum_{\vec{x}'\in\mathcal{L}} e^{-\pi\|\vec{x}'\|^2/s^2}}, \tag{7.1.2}$$

where ρ is as defined in eq. (7.1.1).

7.1.1 Expected length of lattice vectors sampled from the Discrete Gaussian

It is an easy exercise to show the expected length of vectors sampled from a Gaussian distribution over $\mathbb{R}^n$. What about the expected length of lattice vectors sampled from the discrete Gaussian distribution? In this section, we wish to give an informal exposition of this result. We first wish to study the number of vectors of a particular length in the lattice. Let R be a real number larger than 1. Denote the number of lattice vectors of length at most $R\lambda_1(\mathcal{L})$ by N. To bound N, consider Euclidean balls of radius $\lambda_1(\mathcal{L})/2$ around all lattice vectors of length at most $R\lambda_1(\mathcal{L})$. Such balls do not overlap. The volume of each ball is proportional to $\lambda_1(\mathcal{L})/2)^n V_n$, where V_n is the volume of a ball of unit radius. The total volume of these N balls is $N(\lambda_1(\mathcal{L})/2)^n V_n$. Consider also another ball of radius $R\lambda_1(\mathcal{L}) + \lambda_1(\mathcal{L})/2$, centered at the origin. This picture looks like fig. 7.1. By construction, these smaller balls around the lattice points are contained in the larger ball.

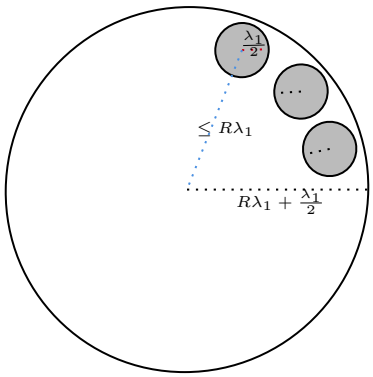


Figure 7.1: A ball covering all lattice vectors of length at most $R\lambda_1$.

Therefore, we have

$$N\left(\frac{\lambda_1}{2}\right)^n V_n \leq \left(R + \frac{1}{2}\right)^n \lambda_1^n V_n$$
$$\implies N \leq (2R+1)^n.$$

We note that this is only an informal description of the result. For a complete description, see notes on Fourier analysis from [Reg04], or [MR04]

Now consider the case when R is very large, say $R\lambda_1(\mathcal{L}) \gg \lambda_n(\mathcal{L})$. Draw the fundamental parallelepiped shifted by lattice vectors with length at most $R\lambda_1(\mathcal{L})$. Similarly, they do not intersect, and they are (almost) covered by the ball centered at 0 of radius $R\lambda_1(\mathcal{L})$ illustrated in Figure 7.2. Because $R\lambda_1(\mathcal{L}) \gg \lambda_n(\mathcal{L})$, many such parallelipipeds will be contained in this ball.

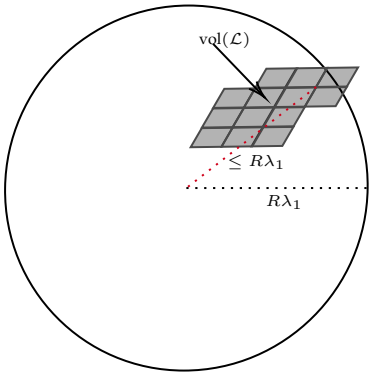


Figure 7.2: A ball covering parallelepipeds shifted by lattice vectors of length at most $R\lambda_1$, when $R\lambda_1(\mathcal{L}) \gg \lambda_n(\mathcal{L})$.

Using a similar argument as before, it can be shown that

$$N = \left(\frac{(R\lambda_1)^n V_n}{\text{vol}(\mathcal{L})}\right)^{1+o(1)}. \tag{7.1.3}$$

Thus, number of lattice vectors up to a particular length R is proportional to R^n. Observe that the total Gaussian mass of all the vectors of a particular length is equal to the product of the mass and the number of such vectors. In the remainder of this subsection, suppose that $s \geq n\lambda_n(\mathcal{L})$. In this parameter regime, the discrete Gaussian distribution *looks like* a continuous Gaussian distribution. As an example, the mass of lattice vectors looks as in table 7.1.

Length of Vector	No. of vectors	Mass of 1 vector	Total mass
s	$C(s)$	$e^{-\pi s^2/s^2} = e^{-\pi}$	$C(s)e^{-\pi}$
$10s$	$C(s)10^n$	$e^{-\pi 100s^2/s^2} = e^{-100\pi}$	$10^n C(s)e^{-100\pi}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\sqrt{n}s$	$C(s)n^{n/2}$	$e^{-\pi n}$	$C(s)n^{n/2}e^{-\pi n}$
$10\sqrt{n}s$	$C(s)n^{n/2}10^n$	$e^{-100\pi n}$	$C(s)n^{n/2}10^n e^{-100\pi n}$

Table 7.1: Total mass of lattice vectors of a particular length.

Note that, $e^{-\pi} < 10^n e^{-100\pi} < n^{n/2}e^{-\pi n} > n^{n/2}10^n e^{-100\pi n}$. This reveals that the mass is concentrated somewhere between the vectors of length $s\sqrt{n}$ and $10s\sqrt{n}$. The key is to observe that as R increases, even though the number of vectors grows much larger, the mass of each vectors gets smaller much faster. Hence, the mass of the lattice is roughly

$$\rho_s(\mathcal{L}) \approx \sum_{\substack{\vec{x}\in\mathcal{L}, \\ s\sqrt{n}\leq\|\vec{x}\|\leq 10s\sqrt{n}}} \rho_s(\vec{x}).$$

Now for some $\vec{t} \in \mathbb{R}^n$, consider the mass of $\vec{t} + \mathcal{L}$. Note that for any $\vec{t} \in \mathbb{R}^n$, we can efficiently find a shift vector $\vec{t}' \in \vec{t} + \mathcal{L}$ such that $\|\vec{t}'\| \leq n\lambda_n \leq s$. This can be done by shifting it to one corner of the fundamental parallelepiped, as the total mass remains the same. Since $\vec{t} + \mathcal{L} = \vec{t}' + \mathcal{L}$, we have

$$\sum_{\substack{\vec{x} \in \mathcal{L}, \\ s\sqrt{n} \leq \|\vec{x}\| \leq 10s\sqrt{n}}} \rho_s\left(\vec{x} + \vec{t}\right) \approx \sum_{\substack{\vec{x} \in \mathcal{L}, \\ s\sqrt{n} \leq \|\vec{x}\| \leq 10s\sqrt{n}}} \rho_s\left(\vec{x} + \vec{t}'\right) \approx \sum_{\substack{\vec{x} \in \mathcal{L}, \\ s\sqrt{n} \leq \|\vec{x}\| \leq 10s\sqrt{n}}} \rho_s(\vec{x}).$$

Formally the following result can be shown, using some Fourier analysis.

Fact 7.1.2. *Say $s \geq \sqrt{n}\lambda_n(\mathcal{L})$. For any $\vec{t} \in \mathbb{R}^n$, $\rho_s(\vec{t} + \mathcal{L}) = \rho_s(\mathcal{L})(1 \pm 2^{-n})$.*

7.2 Reduction from SIVP to SIS

Micciancio and Regev [MR04] presented a reduction from $\mathsf{SIVP}_{\mathcal{O}(n^{3.5})}$ to $\mathsf{SIS}_{m,n,p}$, where $m = 4n^2$, $2^{2n-1} \leq p \leq 2^{2n}$. (In fact a more careful analysis reveals a reduction from $\mathsf{SIVP}_{\mathcal{O}(n)}$). Denote by ν_s the continuous Gaussian distribution over $\mathbb{R}^n$. We will show that, given access to a SIS oracle, repeated application of algorithm 10 solves $\mathsf{SIVP}_{\mathcal{O}(n^{3.5})}$ with high probability.

Algorithm 10 FINDVECTOR($\mathcal{L}$)

Input: A lattice $\mathcal{L}$.

Output: $\vec{v} \in \mathcal{L} : \|\vec{v}\| \leq \lambda_n(\mathcal{L})$.

1: Find an LLL-reduced basis B of $\mathcal{L}$
2: Choose $s \in [n\lambda_n, 2n\lambda_n]$
3: $\forall i \in [m]$ sample IID $\vec{x}_i \leftarrow \nu_s$
4: $\forall i \in [m]$, let $\vec{y}_i = \vec{x}_i \bmod \mathcal{P}(B)$ ▷ Note that $\vec{y}_i - \vec{x}_i \in \mathcal{L}$.
5: $\forall i \in [m]$, let $\vec{z}_i = \lfloor p\vec{y}_i \rfloor / p$. [See fig. 7.3].
6: Let $\vec{a}_i = B^{-1}\vec{z}_i p \in \mathbb{Z}_p^n$
7: Use SIS oracle to find $c_1, \ldots, c_m \in \{-1, 0, 1\}$ such that $\sum_{i=1}^m c_i \vec{a}_i = 0 \pmod p$
8: **return** $\vec{v} = \sum_{i=1}^m c_i(\vec{x}_i - \vec{y}_i + \vec{z}_i)$

Before analyzing Algorithm 10, we need to address how we can perform Line 2 of Algorithm 10. In particular, we need an estimate of $\lambda_n(\mathcal{L})$. Recall that B is LLL reduced, so we know $\tilde{\lambda} \in [\lambda_n(\mathcal{L}), 2^n\lambda_n(\mathcal{L})]$. This gives a guarantee that a least one of $\tilde{\lambda}, \frac{\tilde{\lambda}}{2}, \ldots, \frac{\tilde{\lambda}}{2^{n-1}}$ must lie in the range $[n\lambda_n, 2n\lambda_n]$. So we can run n parallel copies of lines 3 - 8 and return the shortest of these n vectors returned.

Claim 7.2.1. *The output of algorithm 10 $\vec{v}$ is a lattice vector such that $\|\vec{v}\| \leq \mathcal{O}(n^{3.5})\lambda_n(\mathcal{L})$*

Proof. From line 6 of algorithm 10, we know that $\sum_{i=1}^m c_i \vec{a}_i = 0 \pmod p$. Therefore, $\sum_{i=1}^m c_i \vec{a}_i / p \in \mathbb{Z}^n$. Therefore,

$$\sum_{i=1}^m c_i \vec{z}_i = \sum_{i=1}^m c_i \frac{B\vec{a}_i}{p} \in \mathcal{L}.$$

Moreover, the vector $\vec{x}_i - \vec{y}_i$ is a lattice vector. Therefore, $\vec{v} \in \mathcal{L}$. Since $\vec{x}_i$'s are independently chosen from the continuous Gaussian distribution ν_s, with high probability $\|\vec{x}_i\| \leq 10\sqrt{n}s = \mathcal{O}(n^{1.5})\lambda_n(\mathcal{L})$. Since $|c_i| \in \{0, 1\}$, we have that $\sum_{i=1}^m |c_i| \|\vec{x}_i\| \leq \mathcal{O}(n^{1.5})\lambda_n(\mathcal{L}) \cdot m = \mathcal{O}(n^{3.5})\lambda_n(\mathcal{L})$ (with high probability). To

illustrate the relation between $\vec{y}_i$ and $\vec{z}_i$, consider packing the fundamental parallelepiped with smaller (identically shrunk down) parallelepipeds $\mathcal{P}'$, obtained by dividing each basis vector by p.

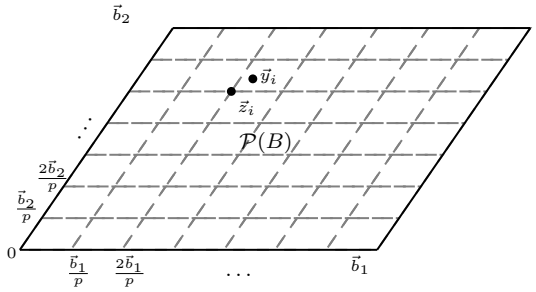


Figure 7.3: Divide $\mathcal{P}(B)$ into p^n subparallelepipeds

That is, $\mathcal{P}' := \mathcal{P}(b_1/p, \dots b_n/p)$. We obtain p^n sub-parallelepipeds, and $\vec{z}_i$ is selected such that $\vec{y}_i$ is in the sub-parallelepiped $\mathcal{P}'$ shifted by $\vec{z}_i$ as in Figure 7.3. Therefore, $\|\vec{z}_i - \vec{y}_i\|$ is bounded by the diameter of sub-parallelepiped $= \operatorname{diam}(\mathcal{P}(B))/p$. Since B is a LLL-reduced basis, we have $\operatorname{diam}(\mathcal{P}(B)) \leq 2^n \lambda_n \cdot n$. Then,

$$\begin{aligned}\sum_{i=1}^{m} |c_i| \, \|\vec{z}_i - \vec{y}_i\| &\leq \frac{\operatorname{diam}(\mathcal{P}(B))}{p} \cdot m \\ &\leq \frac{2^n \lambda_n \cdot n \cdot 4n^2}{2^{2n-1}} \\ &\ll \lambda_n(\mathcal{L}).\end{aligned}$$

It follows that $\|\vec{v}\| = \|\sum_{i=1}^{m} c_i(\vec{x}_i - \vec{y}_i + \vec{z}_i)\| \leq \mathcal{O}(n^{3.5})\lambda_n(\mathcal{L})$, with high probability. □

Claim 7.2.2. *$\vec{a}_i$'s as in line 6 of algorithm 10 are uniformly distributed in $\mathbb{Z}_p^n$, thus fulfilling the promise of the* SIS *oracle.*

Proof. We will analyze how $\vec{y}$ is distributed in $\mathcal{P}(B)$. In line 3-4, we pick a $\vec{x}$ from continuous Gaussian and move it by a lattice vector to obtain $\vec{y}$. For any $\vec{y} \in \mathcal{P}(B)$, the probability of obtaining $\vec{y}$ is proportional to $\rho_s(\vec{y} + \mathcal{L})$. Since $s \geq n\lambda_n(\mathcal{L})$, by fact 7.1.2, we know that

$$\rho_s(\vec{y} + \mathcal{L}) = \rho_s(\mathcal{L})(1 \pm 2^{-n}).$$

Crucially, the probability of obtaining any particular $\vec{y}$ is the same. Therefore we have that for any $\vec{a} \in \mathbb{Z}_p^n$ and $\vec{z} \in \mathcal{P} \cap (\mathcal{L}/p)$,

$$\Pr[\vec{a}_i = \vec{a}] = \Pr[\vec{z}_i = \vec{z}] = \frac{1}{p^n}(1 \pm 2^{-n}),$$

where $\vec{a} \in \mathbb{Z}_p^n$. The first equality is due to the fact that there is a one-one correspondence between $\vec{z}_i$ and $\vec{a}_i$. The second equality is due to the fact that there is a one-one correspondence between a sub-parallelepiped of volume $\frac{1}{p^n}$ and a $\vec{z}_i$. Hence, $\vec{a}_i$ is uniformly distributed in $\mathbb{Z}_p^n$. □

Note that Algorithm 10 only produces one short lattice vector. However, we want n linearly independent short lattice vectors. We use the following fact without proof.

Fact 7.2.3. *For any vector space V of dimension $\leq n-1$, the probability that* FINDVECTOR($\mathcal{L}$) *outputs a vector in V is ≤ 0.9.*

Crucially, there is a sufficient probability that we land outside any subspace.

Corollary 7.2.4. *Based on Fact 7.2.3, after n^2 times executions of* FINDVECTOR($\mathcal{L}$)*, we get n linearly independent lattice vector $\vec{v}_i$'s such that $\|\vec{v}_i\| \leq \mathcal{O}(n^{3.5})\lambda_n(\mathcal{L})$ w.h.p..*

This corollary can be proved using Azuma's inequality.

Chapter 8

Average Case Hardness of LWE

In this lecture we will discuss the reduction from the worst case *bounded distance decoding* problem to the average case *learning with errors* (LWE) problem, which is the basis for many lattice-based public-key encryption. This reduction is due to [Reg05], improvements from [Pei08]. We will reduce from a problem called *bounded distance decoding* BDD. We begin with an algorithm for sampling from discrete Gaussian when the parameter s is large.

8.1 DGS when $s \geq \max_i \|\vec{b_i}\| \cdot 2^n$

Given a lattice basis B and a parameter $s \geq \max_i \|\vec{b_i}\| \cdot 2^n$, output samples from the discrete Gaussian distribution over the lattice $\Pr[X = \vec{x}] = \frac{\rho_s(\vec{x})}{\rho_s(\mathcal{L})}$.

Algorithm 11 Discrete Gaussian Sampling with Large Width

Input: Basis B of a lattice $\mathcal{L}$ and a parameter $s \geq \max_i \|\vec{b_i}\| \cdot 2^n$
Output: Point $\vec{x} \in \mathcal{L}$ such that $\Pr[X = \vec{x}] = \frac{\rho_s(\vec{x})}{\rho_s(\mathcal{L})}$.

1: Sample a vector $\vec{w}$ from the continuous gaussian distribution of width s.
2: Express $\vec{w}$ as following: $\vec{w} = \sum_i \alpha_i \vec{b_i}$ for $\alpha \in \mathbb{R}^n$.
3: Output $\vec{x} := \sum_i \lfloor \alpha_i \rfloor \vec{b_i}$

Claim 8.1.1. *If a random variable X denotes the output of algorithm 11 then $\forall v \in \mathcal{L} : \Pr[X = \vec{v}] \approx \frac{\rho_s(\vec{v})}{f(\mathcal{L})}$ for some f.*

Proof. We know that for some $\vec{v} \in \mathcal{L}$,

$$Pr[X = \vec{v}] \propto \int_{\vec{x} \in \mathcal{P}(B)} \rho_s(\vec{x} + \vec{v}) dx. \tag{8.1.1}$$

Note that $\|\vec{x}\| \leq n \max_i \|\vec{b_i}\|$, since it is in the parallepiped. Now consider the ratio

This chapter is based on lecture 12.

$$\begin{aligned}
\frac{\rho_s(\vec{v}+\vec{x})}{\rho_s(\vec{v})} &= \frac{e^{-(\pi\frac{||\vec{v}||^2}{s^2}+\pi\frac{||\vec{x}||^2}{s^2}+\frac{2\pi\langle\vec{v},\vec{x}\rangle}{s^2})}}{e^{-\pi\frac{||\vec{v}||^2}{s^2}}} \\
&\approx e^{-\pi\frac{||\vec{x}||^2}{s^2}}\cdot e^{\pm\frac{2||\vec{v}||||\vec{x}||}{s^2}} && \text{(Assume largest variance)} \\
&\approx e^{-\pi\frac{||\vec{x}||^2}{s^2}}\cdot e^{\pm\frac{2n^{1.5}\max_i||\vec{b_i}||}{s}} && (\because ||\vec{v}||||\vec{x}|| \le n^{1.5}\max_i||\vec{b_i}||s) \\
&\approx e^{-\pi\frac{||\vec{x}||^2}{s^2}}\cdot e^{\pm\frac{2n^{1.5}}{2^n}} && (\because s \ge \max_i||\vec{b_i}||\cdot 2^n)
\end{aligned}$$

Note that for $e^{-\pi\frac{\|\vec{x}\|^2}{s^2}}$, since $\frac{\|\vec{x}\|^2}{s^2} \le \frac{n}{2^{2n}}$, it is a significantly smaller term, we can ignore it and we get

$$\frac{\rho_s(\vec{v}+\vec{x})}{\rho_s(\vec{v})} \approx e^{\pm\frac{2n^{1.5}}{2^n}} \approx 1 \pm \frac{\mathsf{poly}(n)}{2^n}$$
$$\therefore \rho_s(\vec{v}+\vec{x}) \approx \rho_s(\vec{v})\left(1 \pm \frac{\mathsf{poly}(n)}{2^n}\right)$$

We can now evaluate the earlier integral eq. (8.1.1) as

$$\begin{aligned}
\Pr[X=\vec{v}] &\propto \int_{\vec{x}\in\mathcal{P}(B)} \rho_s(\vec{x}+\vec{v})dx \\
&\approx \rho_s(\vec{v})\left(1 \pm \frac{\mathsf{poly}(n)}{2^n}\right)\int_{\vec{x}\in\mathcal{P}(B)} dx \\
&= \rho_s(\vec{v})\left(1 \pm \frac{\mathsf{poly}(n)}{2^n}\right)\det(\mathcal{L})
\end{aligned}$$

Therefore, the discrete Gaussian distribution is exponentially close to the continuous Gaussian distribution for the chosen parameter s. □

8.2 Bounded Distance Decoding

Bounded distance decoding is a variant of CVP with an additional promise that the target is not too far from lattice, such that there is a unique solution to the closest vector problem. See Figure 8.1.

Definition 8.2.1 (Bounded Distance Decoding (BDD_α))**.** *Given $(0 < \alpha < 1/2)$, $B \in \mathbb{Z}^{n\times n}, \vec{t} \in \mathbb{Z}^n$ such that $dist(\vec{t}, \mathcal{L}(B) \le \alpha\lambda_1(\mathcal{L}(B))$, find the closest vector to $\vec{t}$ in $\mathcal{L}(B)$.*

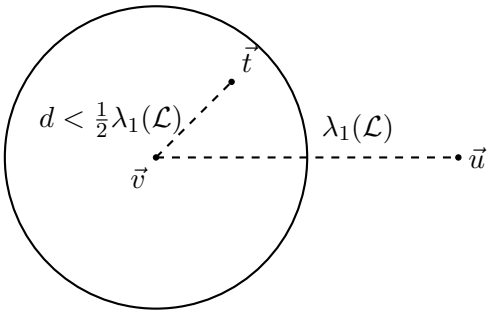


Figure 8.1: Let $\vec{v}, \vec{u}$ have distance $\lambda_1(\mathcal{L})$. The target is $\vec{t}$. $\vec{v}$ is the closest lattice vector to $\vec{t}$ a distance at most $\frac{1}{2}\lambda_1(\mathcal{L})$ away. Thus, $\vec{t}$ must have a unique CVP solution: $\vec{v}$.

8.3 BDD reduces to LWE

We will be showing reduction from BDD_α with $\alpha = 1/\mathsf{poly}(n)$ to LWE. The reduction works as follows (algorithm 12). As in the previous lecture, the goal is to get the correct distributions over the instances of LWE.

Algorithm 12 Reduction from BDD to LWE.

Input: BDD instance: $B \in \mathbb{Z}^{n\times m}$, $t = B^\top \vec{x} + \vec{e} \in \mathbb{Z}^n$, with promise $\|\vec{e}\| \le \alpha\lambda_1(\mathcal{L}(B))$
Output: LWE instance $A \in \mathbb{Z}_q^{m\times n}$ and $\vec{t'} = A\vec{s} + \vec{e'}$.

1: Pick $s \ge q \cdot n \cdot \lambda_n(\mathcal{L}(B)^*)$ where $q \ge 2^{2n}$
2: **for** $i \in 1 \dots m$ **do**
3: Sample $\vec{v}_i \leftarrow \mathcal{D}_{\mathcal{L}^*,s}$ ▷ Use algorithm 11
4: Compute the coefficients $\vec{a}_i = (B^*)^{-1}\vec{v}_i = B^\top \vec{v}_i \pmod q$
5: Sample $e'_i \leftarrow \mathcal{N}(0, \frac{q}{n^2})$
6: Compute $t'_i = \langle \vec{t}, \vec{v}_i \rangle + e'_i$
7: **end for**
8: Sample $\vec{y} \leftarrow \mathbb{Z}_q^n$
9: Let $\vec{t'} := \vec{t'} + A\vec{y}$
10: Output $(A, \vec{t'})$

We note that e'_i could be picked from any other distribution, such that its magnitude satisfies the analysis of claim 8.3.3 (dominates the already presenting error which comes from the inner product term in line 6 and converts the unknown error distribution into a known one). For correctness, we need to show that A and $\vec{t'}$ are uniformly distributed. We recall fact 7.1.2.

Fact 8.3.1. $\forall s \ge \sqrt{n}\lambda_n(\mathcal{L}), \vec{c} \in \mathbb{R}^n$

$$\rho_s(\mathcal{L} + \vec{c}) = \left(1 \pm \frac{1}{2^n}\right)\rho_s(\mathcal{L})$$

Claim 8.3.2. *The distribution of $\vec{a}_i$ in line 4 of algorithm 12 is statistically close to the uniform distribution on $\mathbb{Z}^n$.*

Proof. $\vec{a}_i$ are the coefficients of $\vec{v}_i$ in $\mathcal{L}(B^*)$. Note that $\vec{a}_i = \vec{a} \in \mathbb{Z}_q^n \iff \vec{v}_i \in q\mathcal{L}^* + B^*\vec{a}$. Therefore,

$$\begin{aligned}
\Pr[\vec{a}_i = \vec{a} \in \mathbb{Z}_q^n] &= \frac{\rho_s(q\mathcal{L}^* + B^*\vec{a})}{\sum_{\vec{c}\in\mathbb{Z}_q^n}\rho_s(q\mathcal{L}^* + B^*\vec{c})} \\
&= \frac{(1 \pm 2^{-n})\rho_s(q\mathcal{L}^*)}{\sum_{\vec{c}\in\mathbb{Z}_q^n}(1 \pm 2^{-n})\rho_s(q\mathcal{L}^*)} \qquad (\because s \ge qn\lambda_n(\mathcal{L}^*) = n\lambda_n(q\mathcal{L}^*), \text{fact 7.1.2}) \\
&= \left(1 \pm \frac{2}{2^n}\right)\frac{1}{q^n}
\end{aligned}$$

□

Claim 8.3.3. *The distribution of the errors $\vec{e'}$ is statistically close to normal distribution.*

Proof. Note that in Line 6, when we calculate $\vec{t'}$, the error is $\langle \vec{e}, \vec{v}_i \rangle + e'_i$, since $\vec{t} = B\vec{x} + \vec{e}$. Since s is large, we have $\|\vec{v}_i\| \le s\sqrt{n} \approx qn\lambda_n(\mathcal{L}^*)\sqrt{n}$ and $\|\vec{e}\| \le \alpha\lambda_1(\mathcal{L})$ by the BDD promise.

$$\begin{aligned}\langle \vec{e}, \vec{v}_i \rangle &\leq \|\vec{e}\| \|\vec{v}_i\| \\ &\leq qn\sqrt{n}\alpha\lambda_1(\mathcal{L})\lambda_n(\mathcal{L}^*) \\ &\leq qn^{2.5}\alpha \quad \text{(using claim 4.1.4)}\end{aligned}$$

We can assume $\alpha = \frac{1}{\mathsf{poly}(n)}$ is of some sufficient large polynomial for example: $\alpha \leq \frac{1}{n^5}$, thus the error from $\vec{t}$ is $\leq \frac{q}{n^{2.5}}$. Since we choose e'_i from a distribution with expected absolute value of $\frac{q}{n^2}$, the distribution of the total error $\langle \vec{e}, \vec{v}_i \rangle + e'_i$ is dominated by e'_i. □

Claim 8.3.4. *Finally in algorithm 12, $\vec{t}' = A\vec{s} + \vec{e}'$ is such that $\vec{s}$ is uniformly distributed over $\mathbb{Z}_q^n$.*

Proof. By adding $A\vec{y}$ in line 9, where $\vec{y}$ is uniformly distributed over $\mathbb{Z}_q^n$, we get $\vec{t}' = A(\vec{x} + \vec{y}) + \vec{e}'$, where $\vec{x}$ came from some unknown distribution. Thus, $\vec{s} = \vec{x} + \vec{y}$ is uniformly distributed. □

Remark 8.3.5. *Note that we can retrieve the closest vector (*BDD *solution) from the* LWE *output by subtracting $\vec{y}$ and then mapping the vector back from A to B.*

Thus we have reduced an instance of BDD, to a uniformly distributed instance of LWE.

8.4 $\mathsf{GapSVP}_{2\gamma(n)\sqrt{n}}$ Reduces to $\mathsf{BDD}_{\frac{1}{\gamma(n)}}$

To prove that BDD is indeed a hard problem, we will reduce $\mathsf{GapSVP}_{2\gamma(n)\sqrt{n}}$ to $\mathsf{BDD}_{\frac{1}{\gamma(n)}}$.

Definition 8.4.1 (GapSVP_γ)**.** *Given $B \in \mathbb{R}^{n \times m}$ and $d \in \mathbb{R}$, output:*

- *YES if $\lambda_1(\mathcal{L}(B)) \leq d$*
- *NO if $\lambda_1(\mathcal{L}(B)) > \gamma d$*

The reduction proceeds as follows.

Algorithm 13 $\mathsf{GapSVP}_{2\gamma(n)\sqrt{n}}$ to $\mathsf{BDD}_{\frac{1}{\gamma(n)}}$

Input: Basis: $B \in \mathbb{R}^{n \times m}$, $d \in \mathbb{R}$
Output: YES/NO

1: ans := YES
2: **for** $\mathsf{poly}(n)$ times **do**
3: Sample $\vec{y} \leftarrow \mathcal{B}(\vec{0}, d\sqrt{n})$
4: $\vec{z} = \vec{y} \pmod{\mathcal{P}(B)}$
5: **if** $\mathsf{BDD}(B, \vec{z}) = \vec{z} - \vec{y}$ **then**
6: ans := NO
7: **end if**
8: **end for**
9: Output ans

Claim 8.4.2. *Algorithm 13 maps NO instance of $\mathsf{GapSVP}_{2\gamma(n)\sqrt{n}}$ to NO instance of $\mathsf{BDD}_{\frac{1}{\gamma(n)}}$ with probability 1.*

Proof. Since the input is a NO instance, we have

$$\lambda_1(\mathcal{L}) > 2\gamma(n)\sqrt{n}d \implies d\sqrt{n} < \frac{\lambda_1(\mathcal{L})}{2\gamma(n)} \implies \|\vec{y}\| < \frac{\lambda_1(\mathcal{L})}{2\gamma(n)},$$

exactly what we need for a valid $\mathsf{BDD}_{\frac{1}{\gamma(n)}}$ instance. Therefore the BDD oracle will return $\vec{0}$, when queried with the vector $\vec{y}$. To see that, assume for contradiction that there is a closer vector $\vec{x}$. Then,

$$\|\vec{x}\| \leq \|\vec{x} - \vec{y}\| + \|\vec{y}\| < \frac{\lambda_1(\mathcal{L})}{2\gamma(n)} + d\sqrt{n} < \frac{\lambda_1(\mathcal{L})}{\gamma(n)} \leq \lambda_1(\mathcal{L}).$$

This is a contradiction. Also note that $\mathrm{dist}(\vec{y}, \mathcal{L}) = \mathrm{dist}(\vec{z}, \mathcal{L})$. Therefore $\mathsf{BDD}(B, \vec{y}) = \vec{0}$ and $\mathsf{BDD}(B, \vec{z}) = \vec{z} - \vec{y}$. Thus the algorithm will always output NO. □

Claim 8.4.3. *Algorithm 13 maps YES instance of* $\mathsf{GapSVP}_{2\gamma(n)\sqrt{n}}$ *to YES instance of* $\mathsf{BDD}_{\frac{1}{\gamma(n)}}$.

Proof. Since the input is a YES instance, we have $\lambda_1(\mathcal{L}) \leq d \implies \|\vec{y}\| \leq \lambda_1(\mathcal{L})\sqrt{n}$. Consider the worse case where an adversarial BDD oracle tries to make the algorithm output NO. However, the oracle only knows the value of $\vec{z}$ and not $\vec{y}$. Thus, if will try to guess a $\vec{y} \in \mathcal{B}(\vec{0}, d\sqrt{n})$ such that $\vec{y} \pmod{\mathcal{P}(B)} = \vec{z}$. Let $\vec{x} \in \mathcal{L}$ be an lattice vector such that $\|\vec{x}\| = \lambda_1(\mathcal{L}(B))$. As d is large, with high probability $\vec{y}$ and $\vec{y} + \vec{x}$ are both in $\mathcal{B}(\vec{0}, d\sqrt{n})$.

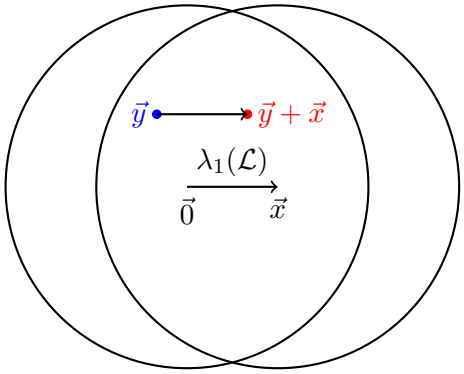


Figure 8.2: Consider balls of radius $d\sqrt{n}$ centered at the origin and $\vec{x}$. Since the radius is much larger than $\lambda_1(\mathcal{L})$, these balls are almost completely intersecting (in high dimensions.)

The following lemma formalizes this intuition.

Lemma 8.4.4 (Symmetric difference between close spheres [GG00])**.** *Let $\vec{x} \in \mathbb{R}^n$ such that $\|\vec{x}\| \leq d$. If $\vec{s}$ is a point chosen uniformly at random from $B(0, d\sqrt{n})$ then with probability $\delta > 1 - \frac{1}{n^c}$ for some position constant c, $\|s - x\| \leq d\sqrt{n}$*

We will use this lemma without proof, and with a exponentially high probability that we can get a $\vec{y}$ such that $\vec{y} + \vec{x}$ is also present in the sphere. Note that since $\vec{x} \in \mathcal{L}$, $\vec{x} + \vec{y} \pmod{\mathcal{P}(B)} = \vec{z}$ as well. Thus given the input of $\vec{z}$, both $\vec{y}$ and $\vec{y} + \vec{x}$ are valid answers. Thus the
bdd oracle has at most a $\frac{1}{2}$ probability of guessing the right one, and thus to output $\vec{z} - \vec{y}$. Thus running this BDD check a polynomial number of times, the probability of guessing right for each one is exponentially small. □

A careful analysis reveals that the above reduction works as long as γ is $\mathcal{O}(n)$, which means that we can base the hardness of LWE on the hardness of $\mathsf{GapSVP}_{n^{1.5}}$.

Bibliography

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2ˆ n time using discrete gaussian sampling: Extended abstract. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 733–742, New York, NY, USA, 2015. Association for Computing Machinery.

[ADS15] Divesh Aggarwal, Daniel Dadush, and Noah Stephens-Davidowitz. Solving the closest vector problem in 2^n time–the discrete gaussian strikes again! In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 563–582. IEEE, 2015.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, STOC '01, page 601–610, New York, NY, USA, 2001. Association for Computing Machinery.

[Bab86] László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

[BGPS23] Huck Bennett, Atul Ganju, Pura Peetathawatchai, and Noah Stephens-Davidowitz. Just how hard are rotations of z n? algorithms and cryptography with the simplest lattice. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 252–281. Springer, 2023.

[Dix90] Jacques Dixmier. Proof of a conjecture by erdős and graham concerning the problem of frobenius. *Journal of Number Theory*, 34(2):198–209, 1990.

[dt23] The FPLLL development team. fplll, a lattice reduction library, Version: 5.4.5. Available at https://github.com/fplll/fplll, 2023.

[EG72] Paul Erdös and Ronald Graham. On a linear diophantine problem of frobenius. *Acta Arithmetica*, 1(21):399–408, 1972.

[GG00] Oded Goldreich and Shafi Goldwasser. On the limits of nonapproximability of lattice problems. *Journal of Computer and System Sciences*, 60(3):540–563, 2000.

[Goo22] Michael Goodrich. More np complete and np hard problems, 2022. https://ics.uci.edu/~goodrich/teach/cs162/notes/pnp3.pdf.

[HB88] D.R. Heath-Brown. The number of primes in a short interval. *Journal für die reine und angewandte Mathematik*, 1988(389):22–63, 1988.

[Imp95] R. Impagliazzo. A personal view of average-case complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 134–147, 1995.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 193–206, New York, NY, USA, 1983. Association for Computing Machinery.

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982.

[MR04] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 372–381, 2004.

[MSW23] Ina Seidel Matthias Schymura and Stefan Weltge. Lifts for voronoi cells of lattices. *Mathematische annalen*, 70:845—-865, 2023.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, pages 351–358, 2010.

[MV13] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. *SIAM Journal on Computing*, 42(3):1364–1391, 2013.

[Pei08] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. Cryptology ePrint Archive, Paper 2008/481, 2008.

[Reg04] Oded Regev. Lattices in Computer Science. https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/, 2004. [Online; accessed 11-Jan-2025].

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

[RH23] Keegan Ryan and Nadia Heninger. Fast practical lattice reduction through iterated compression. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 3–36, Cham, 2023. Springer Nature Switzerland.