Cours 5 : application des réseaux euclidiens à la cryptanalyse

Charles Bouillaguet (charles.bouillaguet@lip6.fr)

3 décembre 2020

Plan

Présentation de quelques mécanismes cryptographiques

- RSA avec Petits exposants secrets
- Une fonction à sens-unique « post-quantique »
- Générateurs congruentiels linéaires tronqués

Introduction aux réseaux euclidiens

- Propriétés de base, volume, plus court vecteur
- Problèmes algorithmiques dans les réseaux

Cryptanalyse avec les algorithmes exacts pour SVP/CVP

- Attaque de Wiener sur RSA avec petit exposant secret

Plan

Présentation de quelques mécanismes cryptographiques
- RSA avec Petits exposants secrets
- Une fonction à sens-unique « post-quantique »
- Générateurs congruentiels linéaires tronqués

Introduction aux réseaux euclidiens
- Propriétés de base, volume, plus court vecteur
- Problèmes algorithmiques dans les réseaux

Cryptanalyse avec les algorithmes exacts pour SVP/CVP
- Attaque de Wiener sur RSA avec petit exposant secret

Plan

Signature RSA

Génération de clef classique

1. Choisir $p, q \approx 2^{n/2}$ aléatoires, premiers.
2. Calculer $N = pq$ et $\phi(N) = (p-1)(q-1)$
3. Choisir $e = 3$
4. Si e n'est pas inversible modulo $\phi(N)$, retourner en 1.
5. Calculer $d \leftarrow e^{-1} \bmod \phi(N)$

À la fin, $ed \equiv 1 \bmod N$

Bilan : $N \approx 2^n$, $e = 3$, $d \approx 2^n$.

- Signature = $2n$ multiplications $\bmod N \rightsquigarrow \mathcal{O}\left(n^3\right)$.
- Vérification = 2 multiplications $\bmod N \rightsquigarrow \mathcal{O}\left(n^2\right)$.

Signer est plus lent que vérifier

La CB signe, le terminal vérifie... dommage !
Peut-on faire l'inverse ?

Signature RSA

Génération de clef avec petit exposant secret

1. Choisir $p, q \approx 2^{n/2}$ aléatoires, premiers
2. Calculer $N = pq$ et $\phi(N) = (p-1)(q-1)$
3. Choisir ~~$d = 3$~~ $d \approx 2^{128}$ aléatoire
4. Si d n'est pas inversible modulo $\phi(N)$, retourner en 1.
5. Calculer $e \leftarrow d^{-1} \bmod \phi(N)$ À la fin, $ed \equiv 1 \bmod N$

Bilan : $N \approx 2^n$, $e \approx 2^n$, $d \approx 2^{128}$.

- Signature = 256 multiplications $\bmod N \rightsquigarrow \mathcal{O}\left(n^2\right)$.
- Vérification = $2n$ multiplications $\bmod N \rightsquigarrow \mathcal{O}\left(n^3\right)$.

C'est 16× plus rapide, mais...

Est-ce sûr ?

Plan

Fonctions à sens-unique

Intuition et exemples

- « *Facile à évaluer, difficile à inverser* ».
- $x \mapsto x^e \bmod N$ avec $N = pq$ (factorisation inconnue).
- $x \mapsto g^x \bmod p$ (p premier)

- Familles de fonction (on peut choisir N, p, g, ...)
- $\mathcal{F} = \{\mathcal{F}_i\}$, ($i =$ indice de la fonction = paramètres).

Syntaxe d'une famille de fonctions $\mathcal{F}$:

Deux algorithmes polynomiaux probabilistes :

- I (**I**ndexation) : $i \leftarrow I(1^n)$.
- V (é**V**aluation) : $\mathcal{F}_i(x) = V(i, x)$.

I tire au hasard une fonction de $\mathcal{F}$ de « paramètre de sécurité » n.
Le domaine d'entrée/sortie de $\mathcal{F}_i$ dépend de i. Taille $\approx 2^n$.
Pas de trappe (=inversion efficace en connaissant un secret).

Fonctions à sens-unique : sécurité

Définition avec un « jeu »

1. Le challenger génère un indice $i \leftarrow I(1^n)$ ainsi qu'un élément aléatoire x dans le domaine de $\mathcal{F}_i$. Il calcule $y \leftarrow V(i, x)$.
2. Le challenger transmet (i, y) à l'adversaire.
3. L'adversaire renvoie une chaîne de bits $\hat{x}$ et il gagne si $\hat{x} = x$.

L'avantage de $\mathcal{A}$ est :

$$\mathbf{AdvOW}[\mathcal{F}, \mathcal{A}] : n \mapsto |\mathbb{P}[\mathcal{A} \text{ gagne}] - 1/2^n|$$

(capacité de faire mieux que répondre au hasard)

Sécurité

La famille $\mathcal{F}$ est **à sens unique** (*One-Way*) si tout adversaire polynomial n'a qu'un avantage négligeable (= plus petit que l'inverse de n'importe quel polynôme en n).

Commentaires sur la définition

1. Jeux : permet de capturer des scénarios complexes.
   L'adversaire peut faire chiffrer/déchiffrer/signer tout ce qu'il veut par le challenger.
2. « négligeable » $\rightsquigarrow$ définition asymptotique de la sécurité.
   - En fonction du paramètre de sécurité n
   - Comment la capacité de l'adversaire évolue avec n
   - Sûr = pas d'adversaire *polynomial* qui casse le schéma
3. Point de vue théorique : « rien de concret » sur RSA-2048...
4. Définir la sécurité de fonctions fixées (`SHA-256`) est dur.

Pourquoi s'intéresser aux fonctions à sens-unique ?

- Pas de crypto sinon...
- On peut faire du chiffrement à clef secrète avec !

Exemples

- $x \mapsto x^e \bmod N$ avec $N = pq$ (factorisation inconnue).
- $x \mapsto g^x \bmod p$ (p premier)

Factorisation / log discret = polynomial sur ordi quantique
Problèmes NP-dur = pas polynomial sur ordi quantique

Alternative #1 : systèmes polynomiaux

I : génère aléatoirement n polynômes quadratique en n variables
V : évalue les polynômes sur l'entrée x

$$y_1 = x_1 + x_1x_3 + x_2x_3 + x_2x_4 + x_3 + x_3x_4 + 1$$
$$y_2 = x_1 + x_1x_2 + x_1x_3 + x_2 + x_2x_4 + x_3 + x_4 + 1$$
$$y_3 = x_1x_2 + x_1x_4 + x_2x_3 + x_2x_4 + x_3 + x_3x_4 + x_4$$
$$y_4 = x_1x_2 + x_1x_3 + x_2 + x_2x_3 + x_3x_4$$

Sens-unique : résoudre le système est **NP-dur** (même modulo 2)

Exemples

- $x \mapsto x^e \bmod N$ avec $N = pq$ (factorisation inconnue).
- $x \mapsto g^x \bmod p$ (p premier)

Factorisation / log discret = polynomial sur ordi quantique
Problèmes NP-dur = pas polynomial sur ordi quantique

Alternative #2 : Subset Sum

I : génère aléatoirement n grands entiers N_i
V : Sur une entrée de n bits $(x_1, x_2, \ldots, x_n)$, calcule :

$$\mathcal{F}_{(N_i)}(x) = \sum_{i=1}^{n} x_i N_i$$

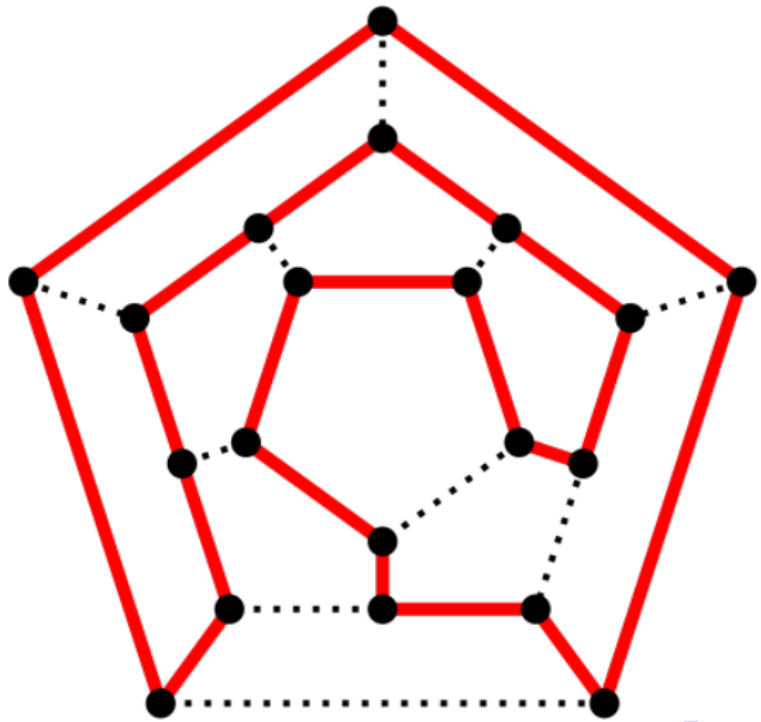
Sens-unique : retrouver les x_i à partir de la somme est **NP-dur**.

Théorie vs Pratique

- Factorisation et log. discrets ne sont pas NP-durs
  - Comment formuler un problème de décision raisonnable ?
  - Largement utilisés et « sûrs » pour l'instant
- NP-dur = sûr ? (si P $\neq$ NP...)
  - NP-dur = « pas d'algo polynomial dans le pire des cas »
  - Il existe des instances difficiles
  - NP-dur $\neq$ « impossible tout le temps »
  - Il se peut que presque toutes les instances soient faciles !
- Pour chaque problème, il faut voir concrètement.

Théorie vs Pratique : CIRCUIT HAMILTONIEN

- Prouvé NP-complet en 1972 (un des 21 problèmes de Karp)
- Mais…
- …se résout en temps linéaire pour des graphes « aléatoires ».



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS†

Richard M. Karp

University of California at Berkeley

Abstract: A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and elements of other countable domains. Through simple encodings

Main Theorem. All the problems on the following list are complete.

10. UNDIRECTED HAMILTON CIRCUIT
INPUT: graph G
PROPERTY: G has a cycle which includes each node exactly once.

Discrete Mathematics 75 (1989) 373–379
North-Holland

A SIMPLE LINEAR EXPECTED TIME ALGORITHM FOR FINDING A HAMILTON PATH

Andrew THOMASON

Department of Pure Mathematics and Mathematical Statistics, 16 Mill Lane, Cambridge CB2 1SD, England

We give a simple algorithm which either finds a hamilton path between two specified vertices of a graph G of order n, or shows that no such path exists. If the probability of an edge in G is $p \geq 12n^{-\frac{1}{3}}$ the algorithm runs in expected time cnp^{-1} and uses storage cn, where c is an absolute constant.

Retour sur un cas concret

Alternative #2 : Subset Sum

I : génère aléatoirement n entiers A_i
V : Sur une entrée de n bits $(x_1, x_2, \ldots, x_n)$, calcule :

$$\mathcal{F}_{(N_i)}(x) = \sum_{i=1}^{n} x_i A_i$$

Exemple : $n = 128$, $A_i \approx 1024$ bits.

Très tentant !

- Simple, facile à comprendre et à programmer.
- Les problèmes suivants sont **NP-durs** :
  - Inversion : retrouver les x_i à partir de la somme.
  - Collision : trouver $x \neq y$ tels que $F(x) = F(y)$.

Question

Est-ce sûr ?

Plan

Norme POSIX : spécification (obligatoire) de `mrand48()`

```c
uint64_t rand48_state;

void srand48(uint32_t seed) {
  rand48_state = seed;
  rand48_state = 0x330e + (rand48_state << 16);
}

uint32_t mrand48() { /* renvoie 32 bits aléatoires */
  rand48_state = (0x5deece66d * rand48_state + 11) & 0xffffffffffff;
  return (rand48_state >> 16);
}
```

Spécification de C : suggestion d'implantation de `rand()`

```c
static unsigned long int next = 1;

void srand(unsigned int seed) {
    next = seed;
}

int rand(void) { /* renvoie 15 bits aléatoires */
    next = next * 1103515245 + 12345;
    return ((unsigned)(next/65536) % 32768);
}
```

`Math.rand()` dans JavaScript (Chrome, Firefox, Safari)

```
uint64_t 64 a, b;

/* PAS un générateur linéaire congruentiel (plutôt une sorte de LFSR) */
uint64_t xorshift128plus()  /* renvoie 64 bits aléatoires */
{
    uint64_t s1 = a
    uint64_t s0 = b;
    a = s0;
    s1 ^= s1 << 23;
    s1 ^= s1 >> 17;
    s1 ^= s0;
    s1 ^= s0 >> 26;
    b = s1;
    return a + b;
}
```

Générateurs **pseudo**-aléatoires

- Pas du « vrai hasard », mais résultat d'un calcul.
- Très utile (génération de clefs cryptographiques…)

Générateurs conguentiels linéaires

Histoire

Inventés par Derrick Lehmer (1905–1991) pour utilisation sur l'ENIAC ! Proposition de 1949 :

$$u_0 = 47\,594\,118,$$
$$u_{i+1} = 23u_i \bmod 10^8 + 1$$

- Utiliser X_i comme source de bits « aléatoires » avec

$$X_{i+1} = aX_i + b \bmod m$$

- X_0 est la graine

Question

Est-il raisonnable d'utiliser ça pour générer des cartes bleues ?

Définition des Générateurs Pseudo-Aléatoires

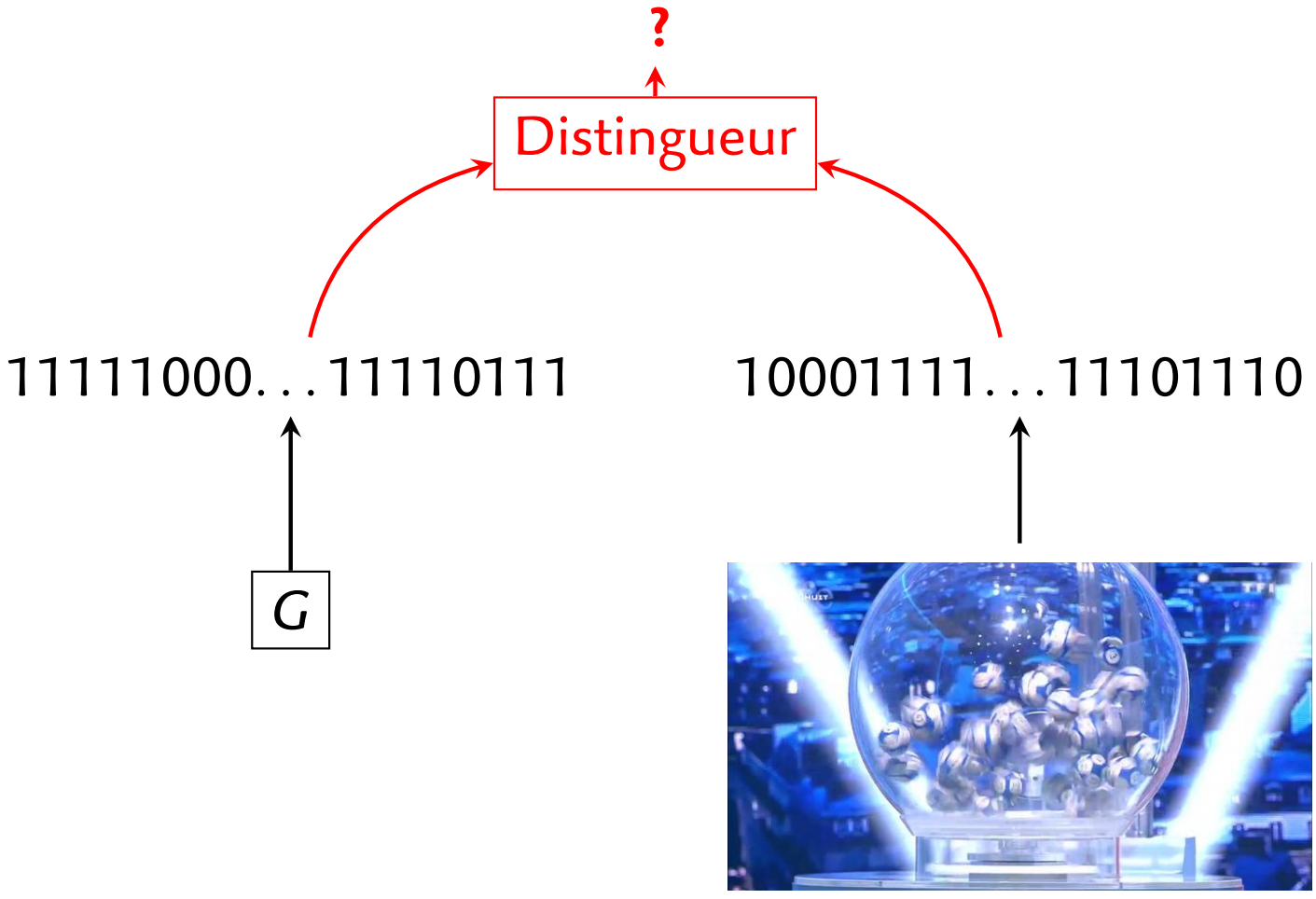
Un **générateur pseudo-aléatoire** G est un algorithme déterministe qui produit une *longue* séquence de bits pseudo-aléatoires, de taille t, à partir d'une *courte* graine.

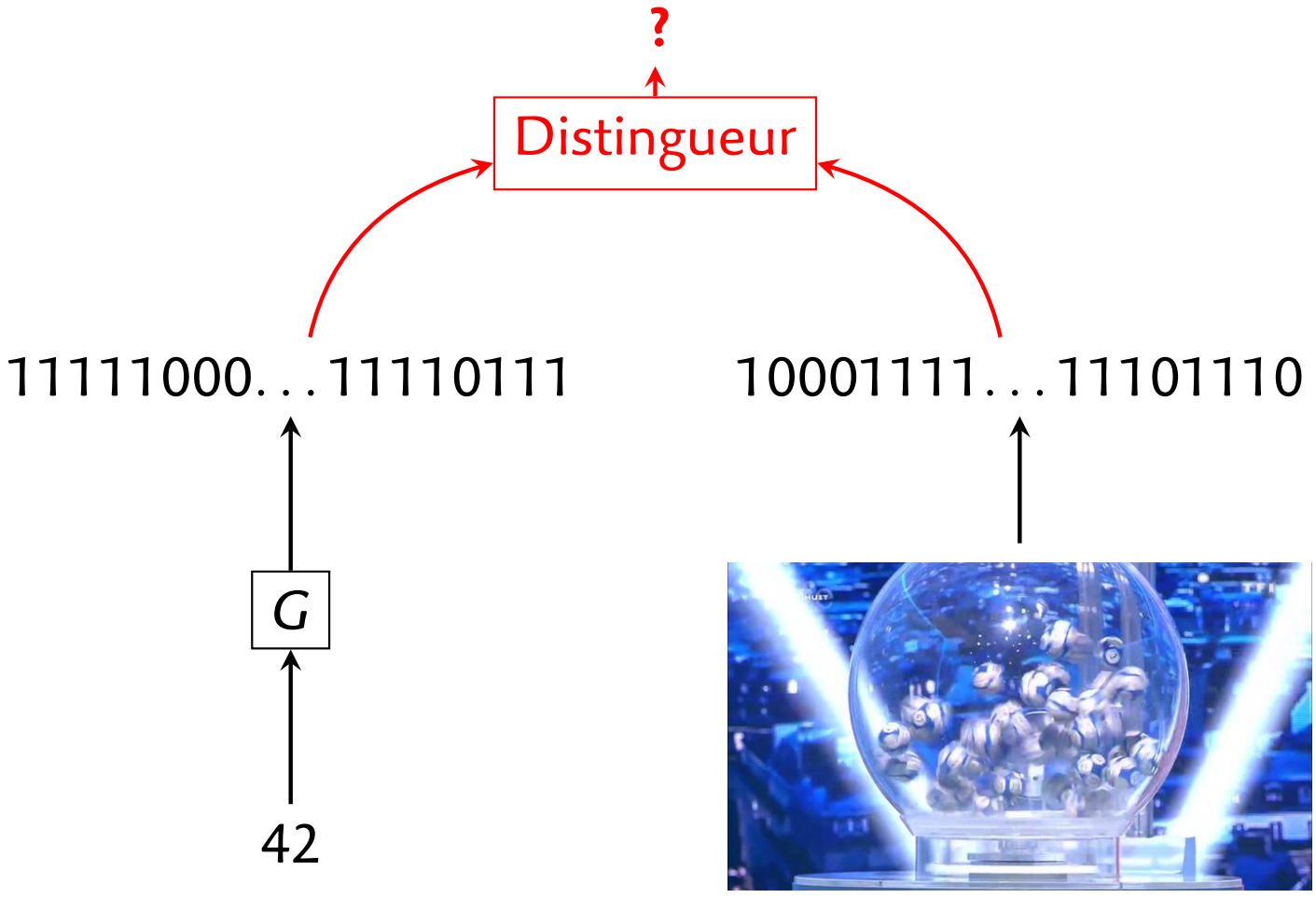
Informellement

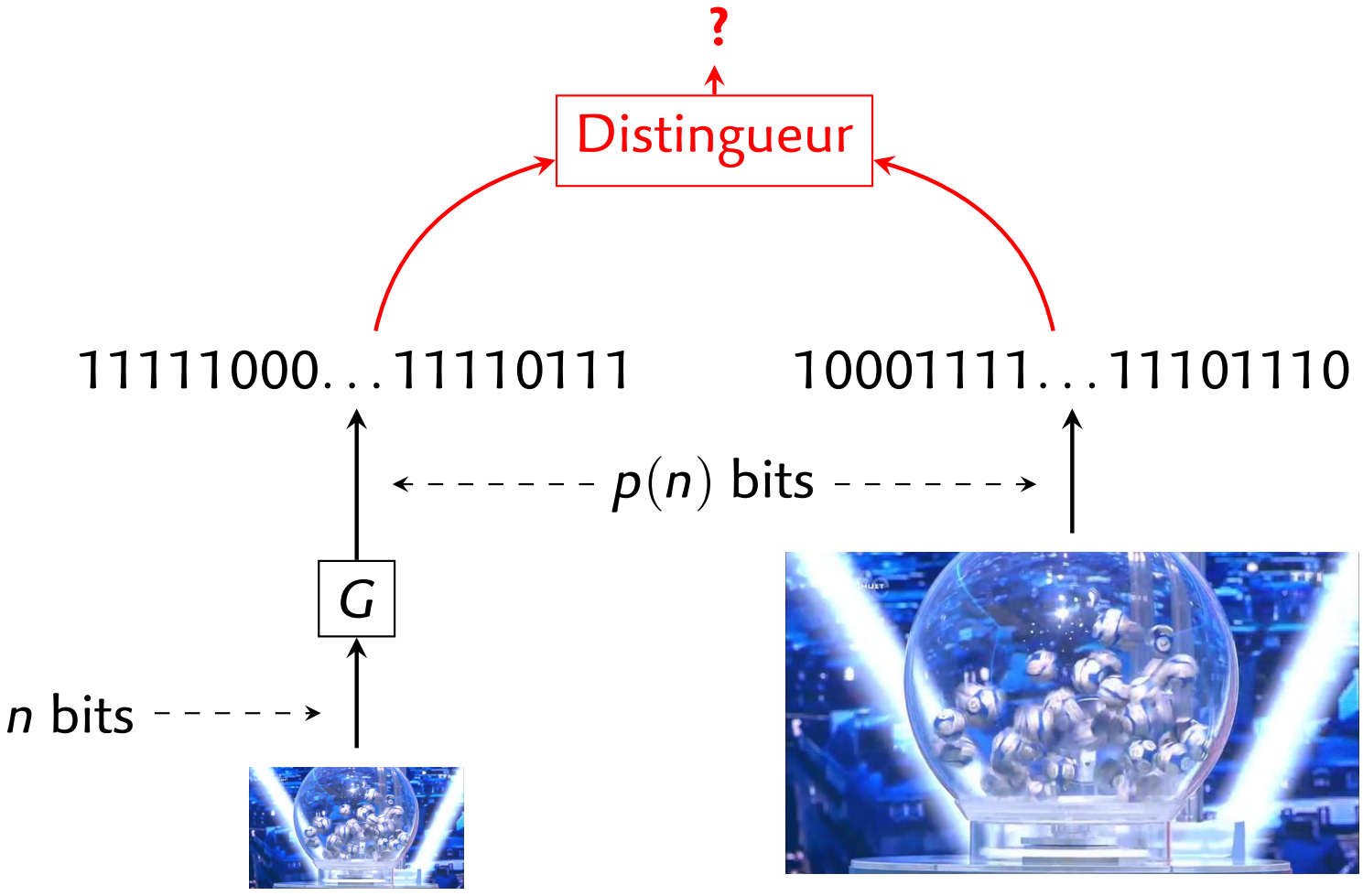
séquence de bits « Pseudo-aléatoire » = indistinguable de bits vraiment aléatoires par un algorithme polynomial.

Distingueur

Algorithme $\mathcal{A}$ polynomial qui tente de faire la différence entre la sortie du PRG et des bits vraiment aléatoires.
Par ex. $\mathcal{A}(x) = 1$ si x vient du PRG, et 0 sinon


?
Distingueur
11111000...11110111
10001111...11101110
G


?
Distingueur
11111000...11110111
10001111...11101110
G
42



En fait le distingueur doit faire la différence entre deux distributions sur les chaines de bits : la distribution uniforme d'un côté et la sortie de G de l'autre (avec graine aléatoire).

Indistinguabilité calculatoire

- Si X_n et Y_n sont deux variables aléatoires dans $\{0,1\}^n$ (c.a.d. des chaines de bits tirées selon deux distributions différentes), l'avantage de l'algorithme $\mathcal{A}$ pour les distinguer est :

$$\mathbf{Adv}(\mathcal{A}) = \left|\mathbb{P}\left[\mathcal{A}(1^n, X_n) = 1\right] - \mathbb{P}\left[\mathcal{A}(1^n, Y_n) = 1\right]\right|.$$

- Deux ensembles de variables aléatoires $\{X_n\}_{n\geq 0}$ et $\{Y_n\}_{n\geq 0}$ sont calculatoirement indistinguables si tout distingueur en temps polynomial n'a qu'un avantage négligeable (plus petit que l'inverse de n'importe quel polynôme en n)

DDH : $\left(g^{X_n}, g^{Y_n}, g^{X_n Y_n}\right) \overset{?}{\longleftrightarrow} \left(g^{X_n}, g^{Y_n}, g^{Z_n}\right)$

Pseudo-aléatoire

- $\{X_n\}_{n\geq 0}$ est pseudo-aléatoire s'il est calculatoirement indistinguable de l'ensemble uniforme $\{U_n\}_{n\geq 0}$ (U_n est uniformément distribuée sur $\{0,1\}^n$)

Définition des Générateurs Pseudo-Aléatoires

Commentaire

- « negligeable » $\rightsquigarrow$ définition asymptotique.
  - Capacité du distingueur baisse fortement quand $|s|$ grandit.

Enfin ! Définition d'un PRG

Un algorithme polynomial déterministe G est un générateur pseudo-aléatoire (à sortie arbitrairement grande) si :

1. $|G(s, 1^t)| = t$ et $G(s, 1^t)$ est un préfixe de $G(s, 1^{t+1})$.
2. Pour tout polynôme p, l'ensemble $\{G(U_n, 1^{p(n)})\}_{n \in \mathbb{N}}$ est pseudo-aléatoire.

Question

`mrand48()` ? Rentre pas dans le cadre (graine de taille fixe).

Retour sur les générateurs congruentiels linéaires

Générateurs congruentiels linéaires

- Graine = (X_0, a, b, m)
- Sortie = $X_0, X_1, X_2, \ldots$ avec $X_{i+1} = aX_i + b \bmod m$

a, b et m sont inconnus du distingueur !

En fait, c'est facile à casser (on le fera en TD).

Générateurs congruentiels linéaires tronqués (Knuth, 1980)

- Graine = X_0
- $X_{i+1} = aX_i + b \bmod m$ et $Y_i = \lfloor X_i/k \rfloor$
- Sortie = $Y_0, Y_1, Y_2, \ldots$

Les bits de poids faibles des X_i sont masqués.
a, b et m peuvent être connus... ou pas.

Générateurs congruentiels linéaires tronqués (Knuth, 1980)

- Graine = (X_0, a, b, m)
- $X_{i+1} = aX_i + b \bmod m$ et $Y_i = \lfloor X_i/k \rfloor$
- Sortie = $Y_0, Y_1, Y_2, \ldots$

Les bits de poids faibles des X_i sont masqués.

`mrand48()`

$a = 25214903917, \quad b = 11, \quad m = 2^{48}, \quad k = 2^{16}$

Générateurs congruentiels linéaires tronqués (Knuth, 1980)

- Graine = (X_0, a, b, m)
- $X_{i+1} = aX_i + b \bmod m$ et $Y_i = \lfloor X_i/k \rfloor$
- Sortie = $Y_0, Y_1, Y_2, \ldots$

Les bits de poids faibles des X_i sont masqués.

`mrand48()`

$a = 25214903917, \quad b = 11, \quad m = 2^{48}, \quad k = 2^{16}$

Faible ! Il suffit de « deviner » les 16 bits de poids faible de X_0.

$a = [\textit{quelconque}]$, $b = 0$, $m = 2^{128}$, $k = 2^{120}$

- État interne 128 bits, sortie 8 bits.
- Les 120 bits de poids faibles sont cachés à chaque fois.
- Question : sûr ?

Plan

Présentation de quelques mécanismes cryptographiques
- RSA avec Petits exposants secrets
- Une fonction à sens-unique « post-quantique »
- Générateurs congruentiels linéaires tronqués

Introduction aux réseaux euclidiens
- Propriétés de base, volume, plus court vecteur
- Problèmes algorithmiques dans les réseaux

Cryptanalyse avec les algorithmes exacts pour SVP/CVP
- Attaque de Wiener sur RSA avec petit exposant secret

Plan

Rappels

- Vecteur $\mathbf{x} = (x_1, \ldots, x_n)$.
- Norme (euclidienne) (« norme 2 », « longueur ») :

$$\|\mathbf{x}\| = \sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}$$

$\mathbb{R}$-Espace vectoriel V engendré par $\mathbf{b}_1, \ldots, \mathbf{b}_m$

- Ensemble des combinaisons linéaires de $\mathbf{b}_1, \ldots, \mathbf{b}_m$

$$V = \left\{ \sum_{i=1}^{m} \lambda_i \mathbf{b}_i : (\lambda_1, \ldots, \lambda_m) \in \mathbb{R}^m \right\}$$

- $\mathbf{x}, \mathbf{y} \in V \Longrightarrow \lambda\mathbf{x} + \mu\mathbf{y}$ pour tout $\lambda, \mu \in \mathbb{R}$.

Réseau euclidien

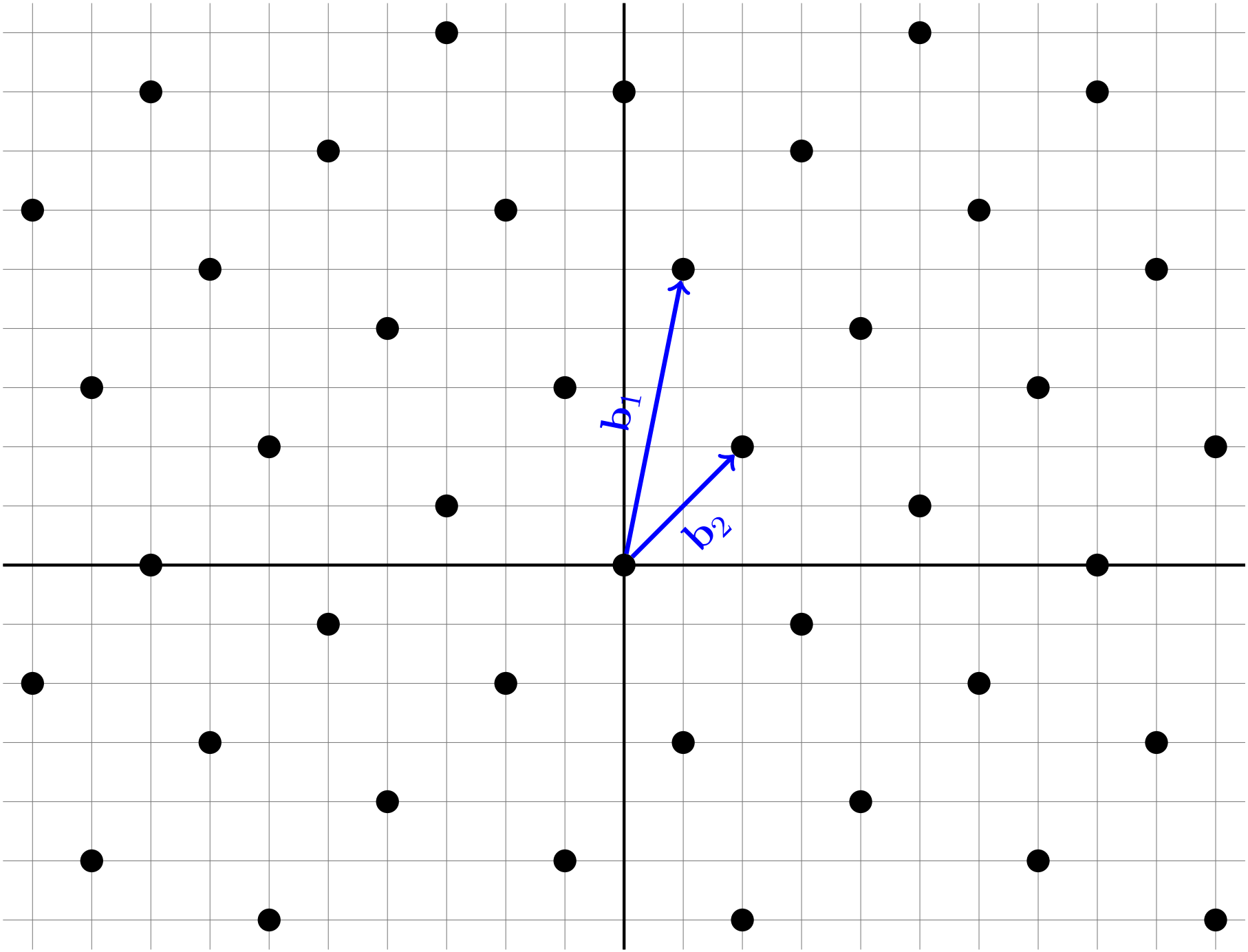
Un réseau euclidien (« *Euclidean lattice* »), c'est la même chose, mais avec des entiers. Il suffit de remplacer $\mathbb{R}$ par $\mathbb{Z}$ dans la diapo précédente.

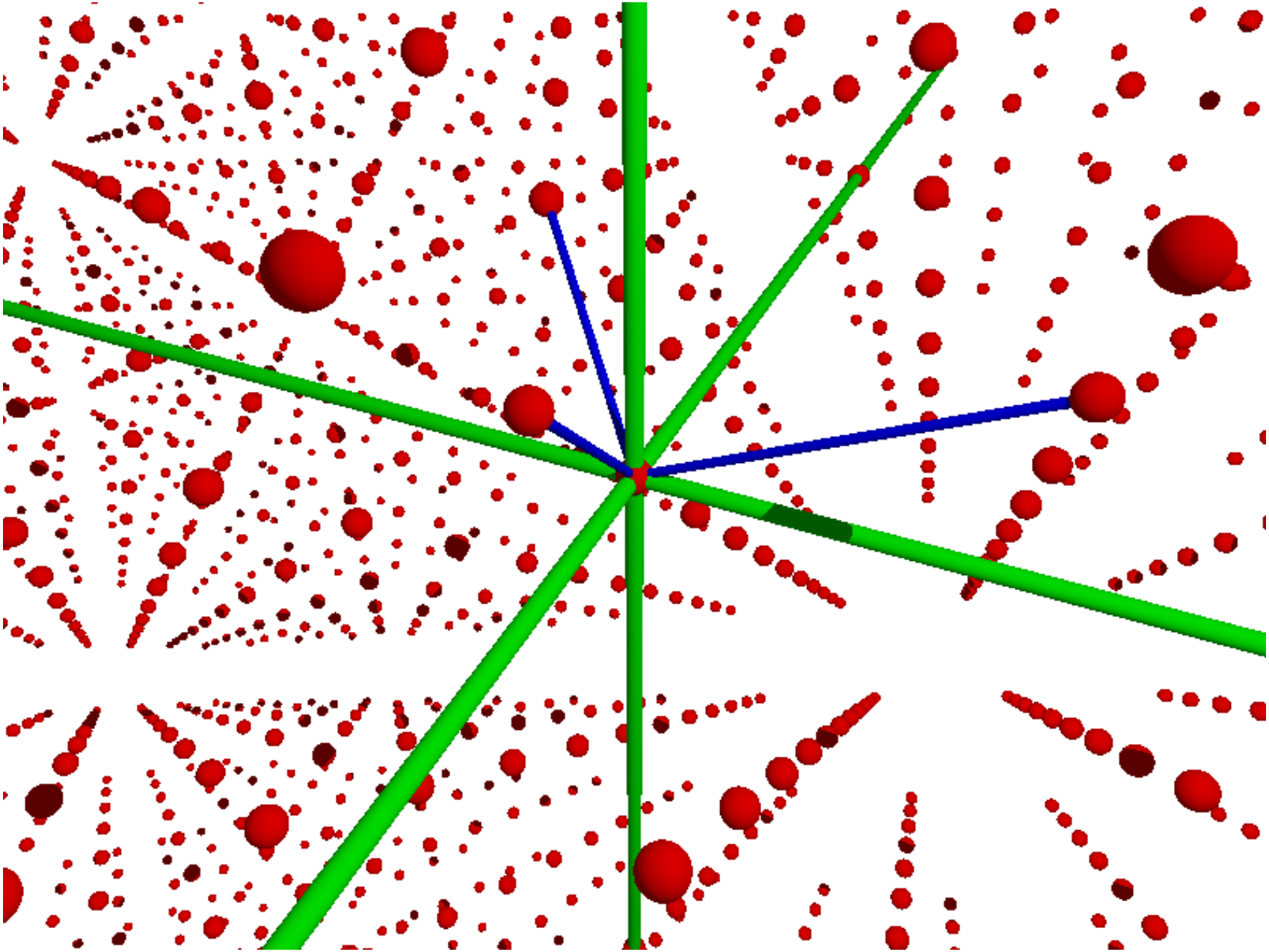
Definition (Réseau euclidien)

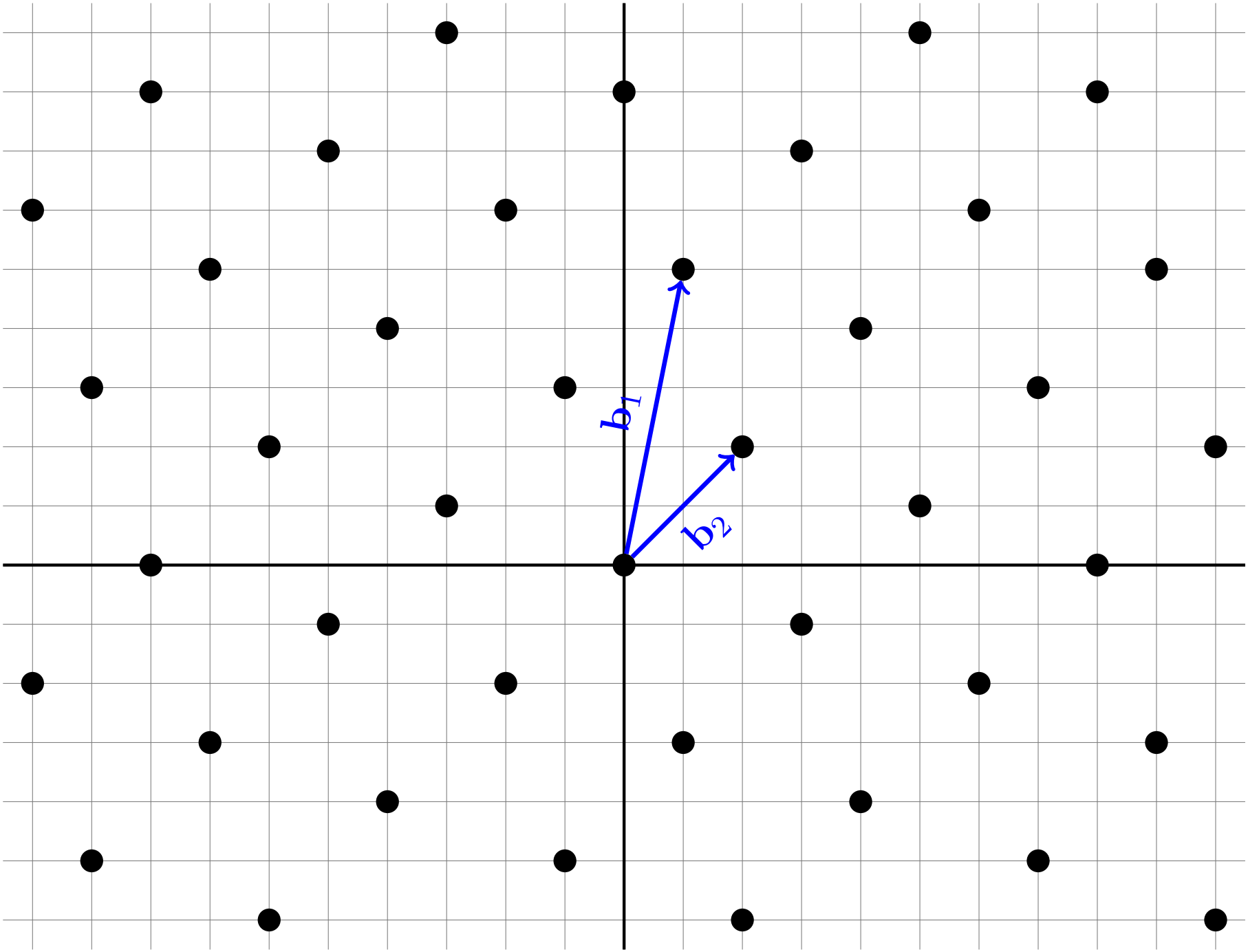
$\mathbf{b}_1, \ldots, \mathbf{b}_d$: vecteurs de $\mathbb{Z}^n$. Ils *engendrent* le réseau euclidien :

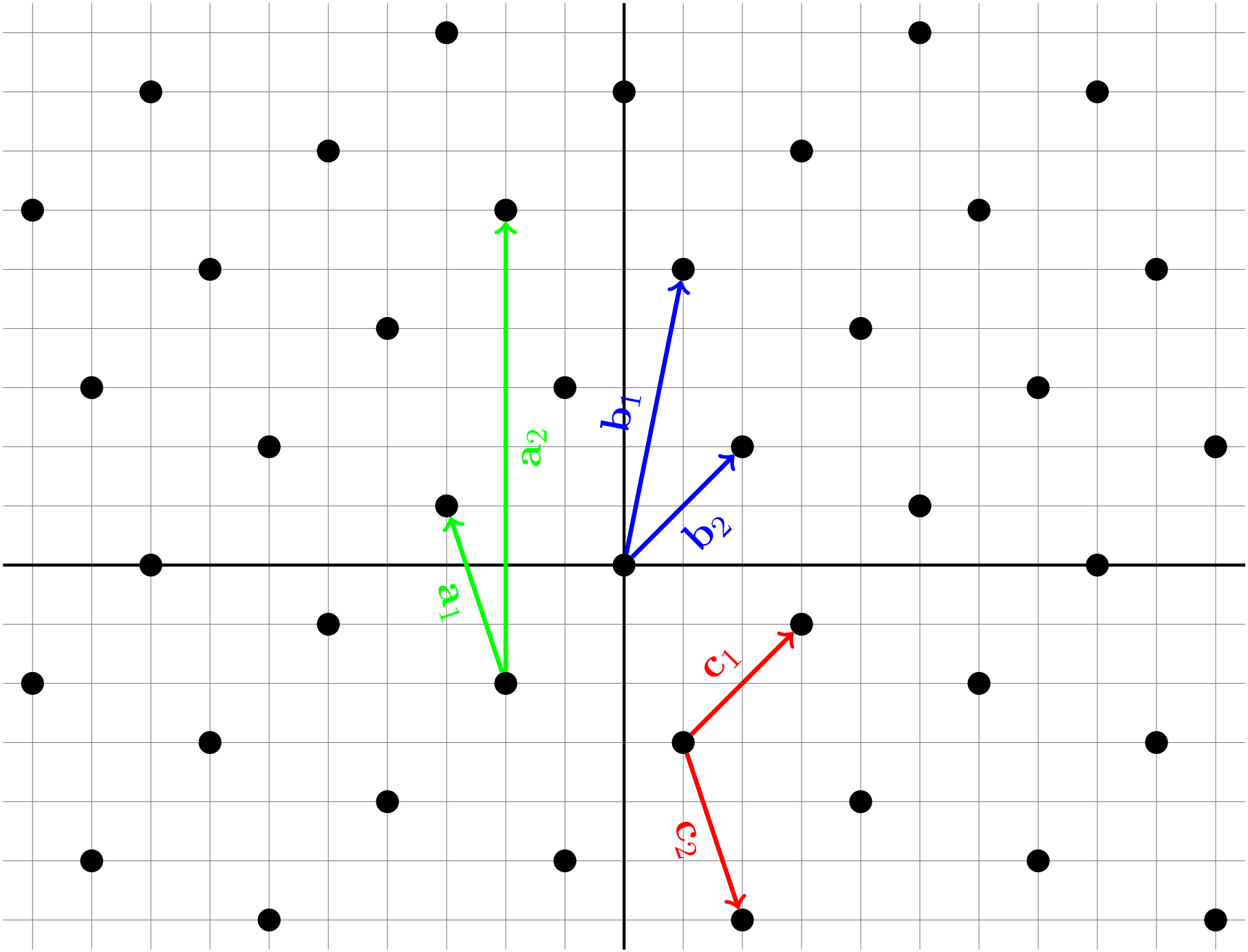
$$\mathcal{L} = \left\{ \sum_{i=1}^{d} \mu_i \mathbf{b}_i : \mu_1, \ldots, \mu_d \in \mathbb{Z} \right\}.$$

$\mathbf{b}_i$ linéairement indépendants $\rightsquigarrow$ *base* de $\mathcal{L}$ (d = dimension).
Si $d = n$, on dit que le réseau est *de rang plein*.


b_1
b_2




b_1
b_2


$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{c}_1$
$\mathbf{c}_2$

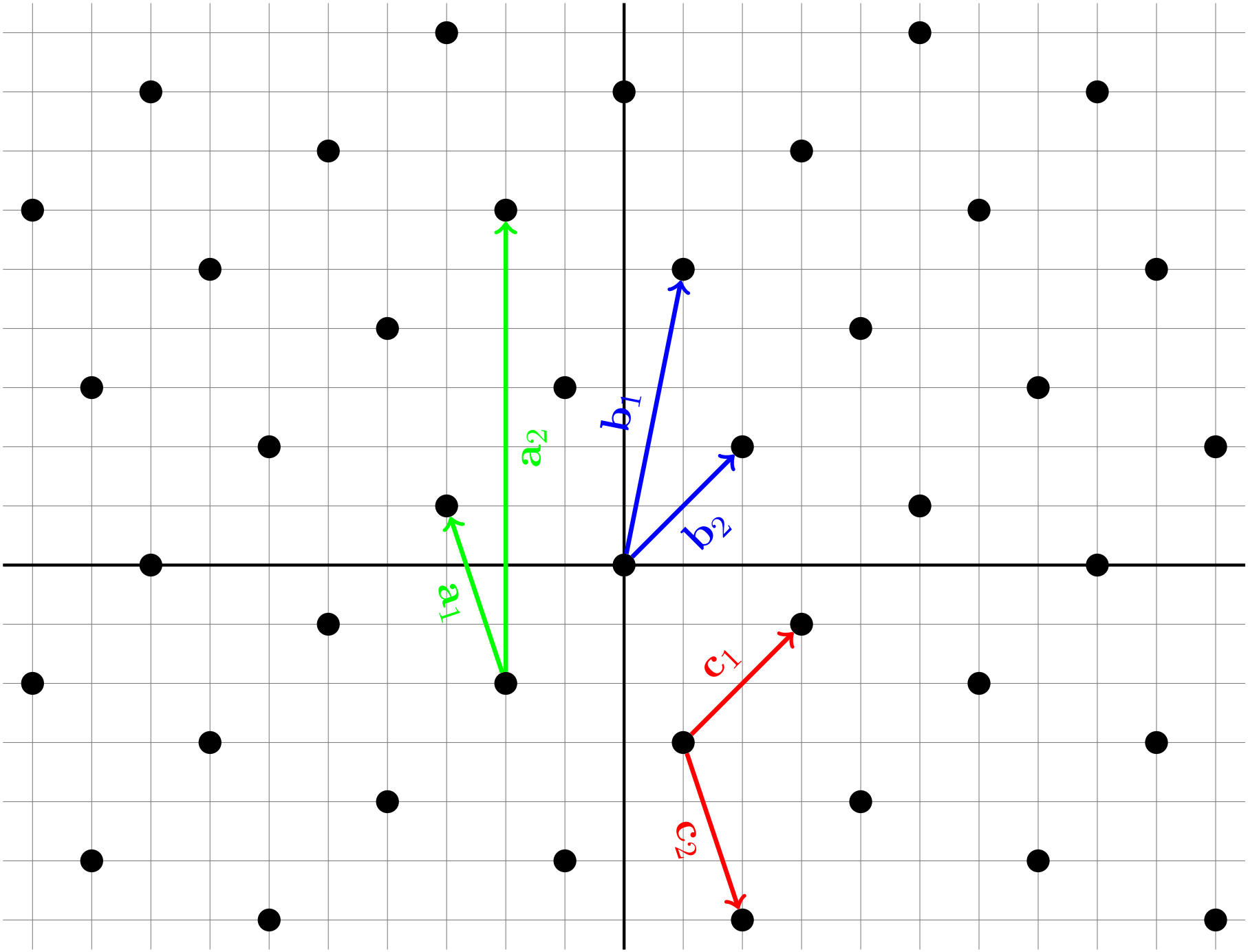
bases

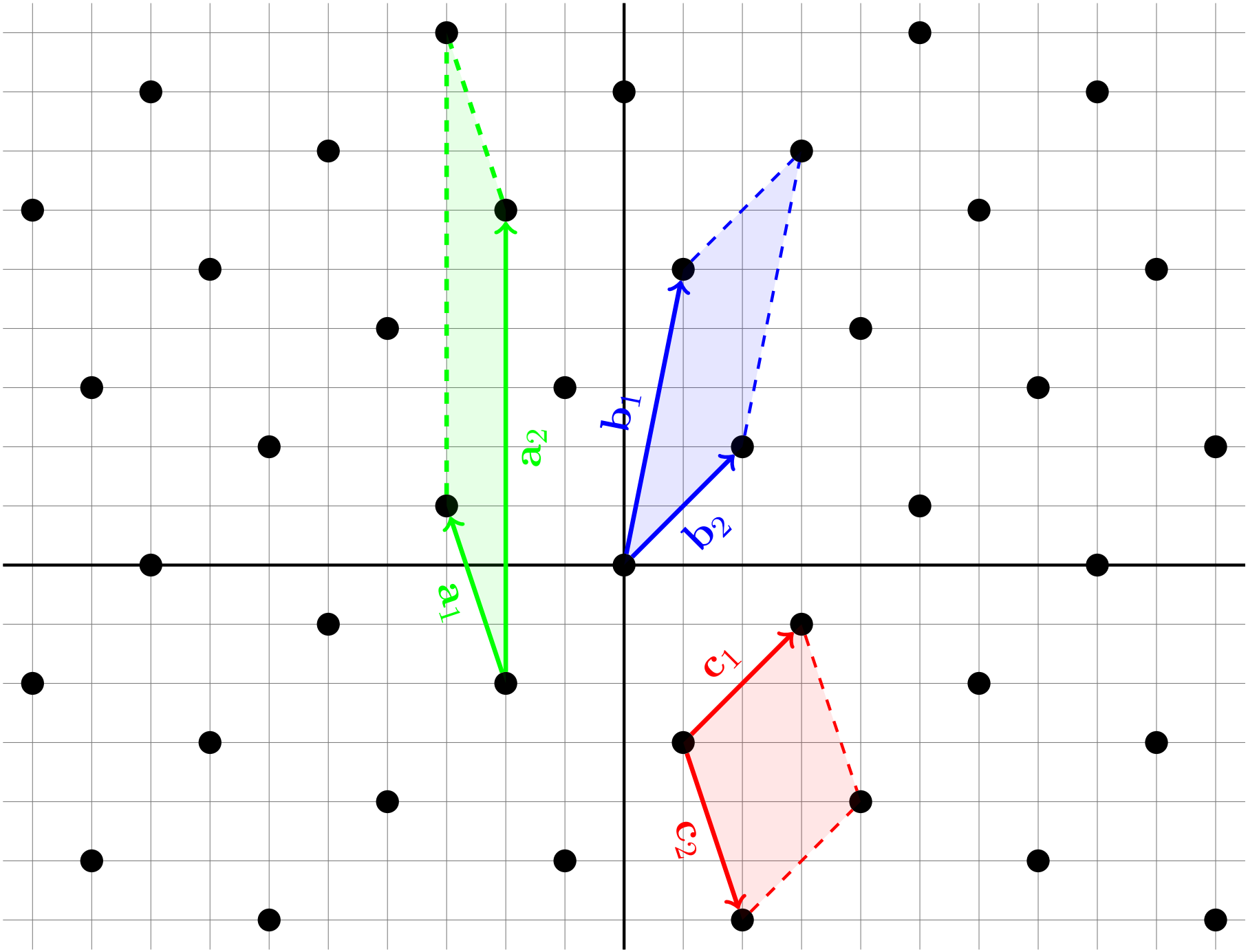
- Réseau précédent engendré par $\mathbf{b_1} = (1, 5), \mathbf{b_2} = (2, 2)$.
- Représenté de manière commode par :

$$B = \begin{pmatrix} 1 & 5 \\ 2 & 2 \end{pmatrix}$$

  Alors $\mathcal{L} = \{\mathbf{x} \cdot B \text{ avec } \mathbf{x} \in \mathbb{Z}^d\}$.
- Plusieurs bases possibles. Par ex. :

$$A = \begin{pmatrix} -1 & 3 \\ 0 & 8 \end{pmatrix} \quad \text{et} \quad C = \begin{pmatrix} 2 & 2 \\ 1 & -3 \end{pmatrix}$$


$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{c}_1$
$\mathbf{c}_2$


$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{c}_1$
$\mathbf{c}_2$

On fixe une base $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.

- parallélepipède fondamental :

$$\left\{ \sum_{i=1}^{d} x_i \mathbf{b}_i : x_1, \ldots, x_d \in [0; 1[\right\}.$$

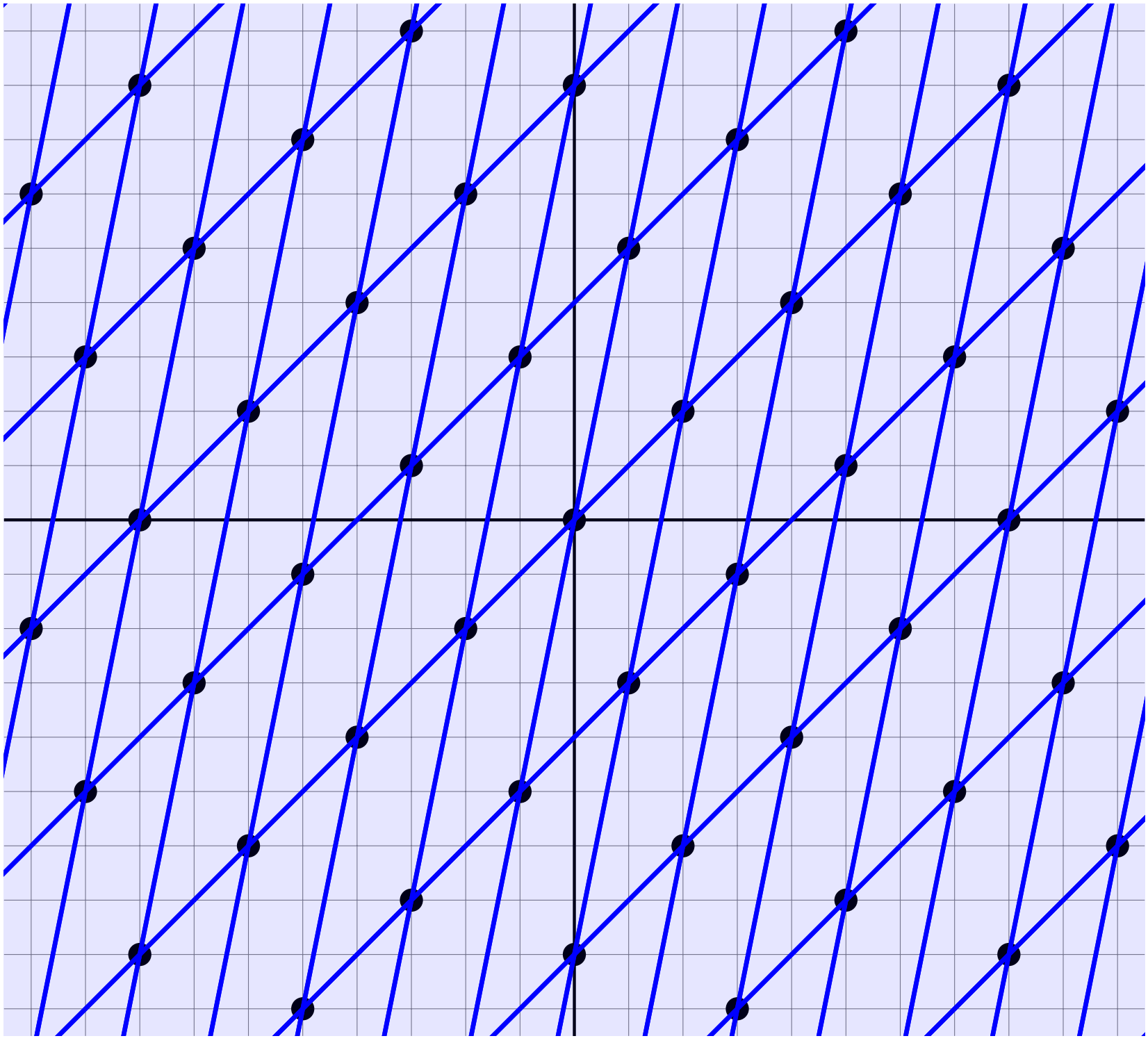
- son volume ne dépend pas du choix de la base.

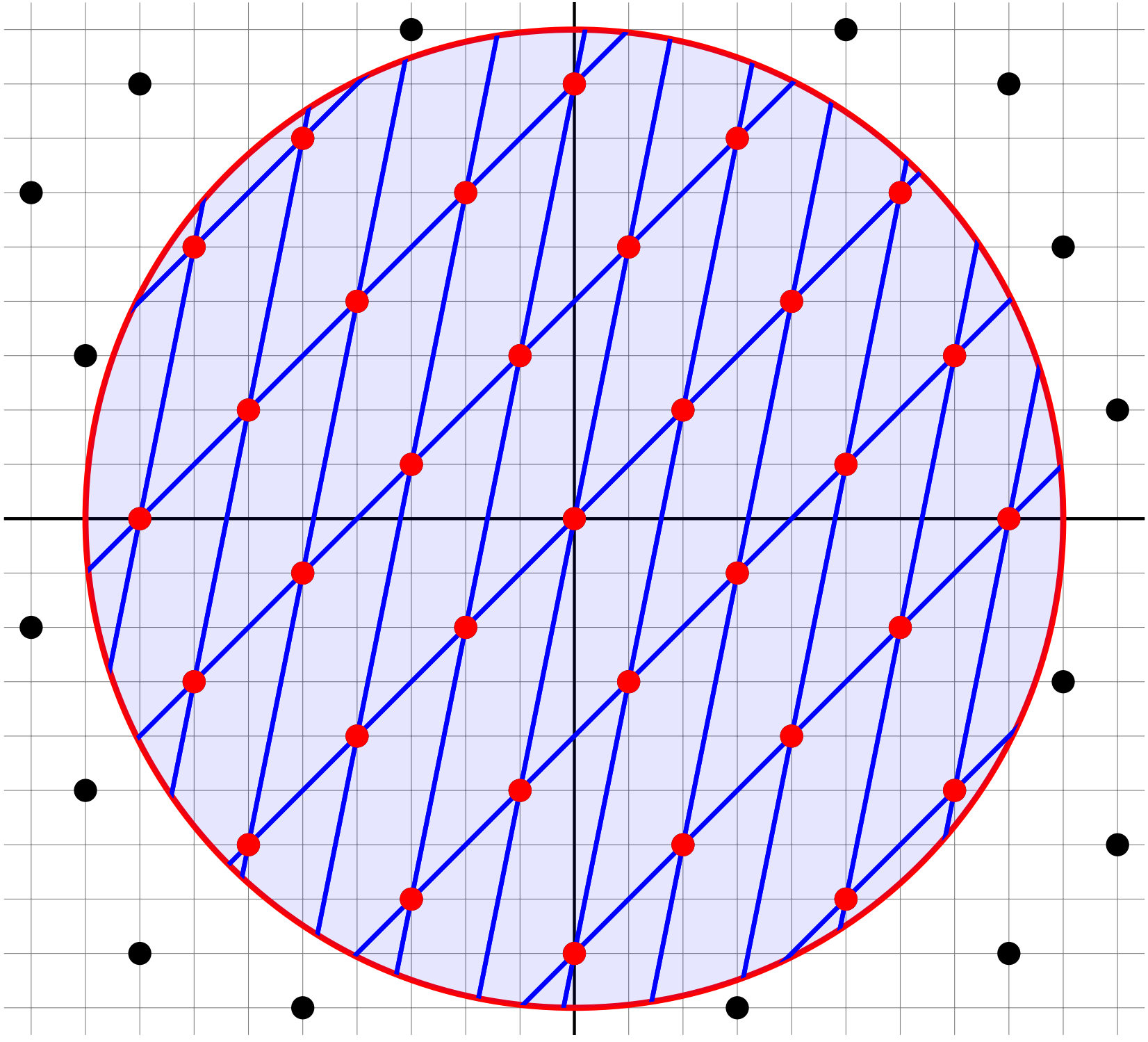
$\Rightarrow$ caractéristique (importante) du réseau.

Theorem

Soit $\mathcal{L}$ un réseau de base B. On a $\mathrm{Vol}(\mathcal{L}) = \sqrt{\det BB^t}$.

Si $\mathcal{L}$ est de rang plein, ça se simplifie en $\mathrm{Vol}(\mathcal{L}) = |\det B|$.



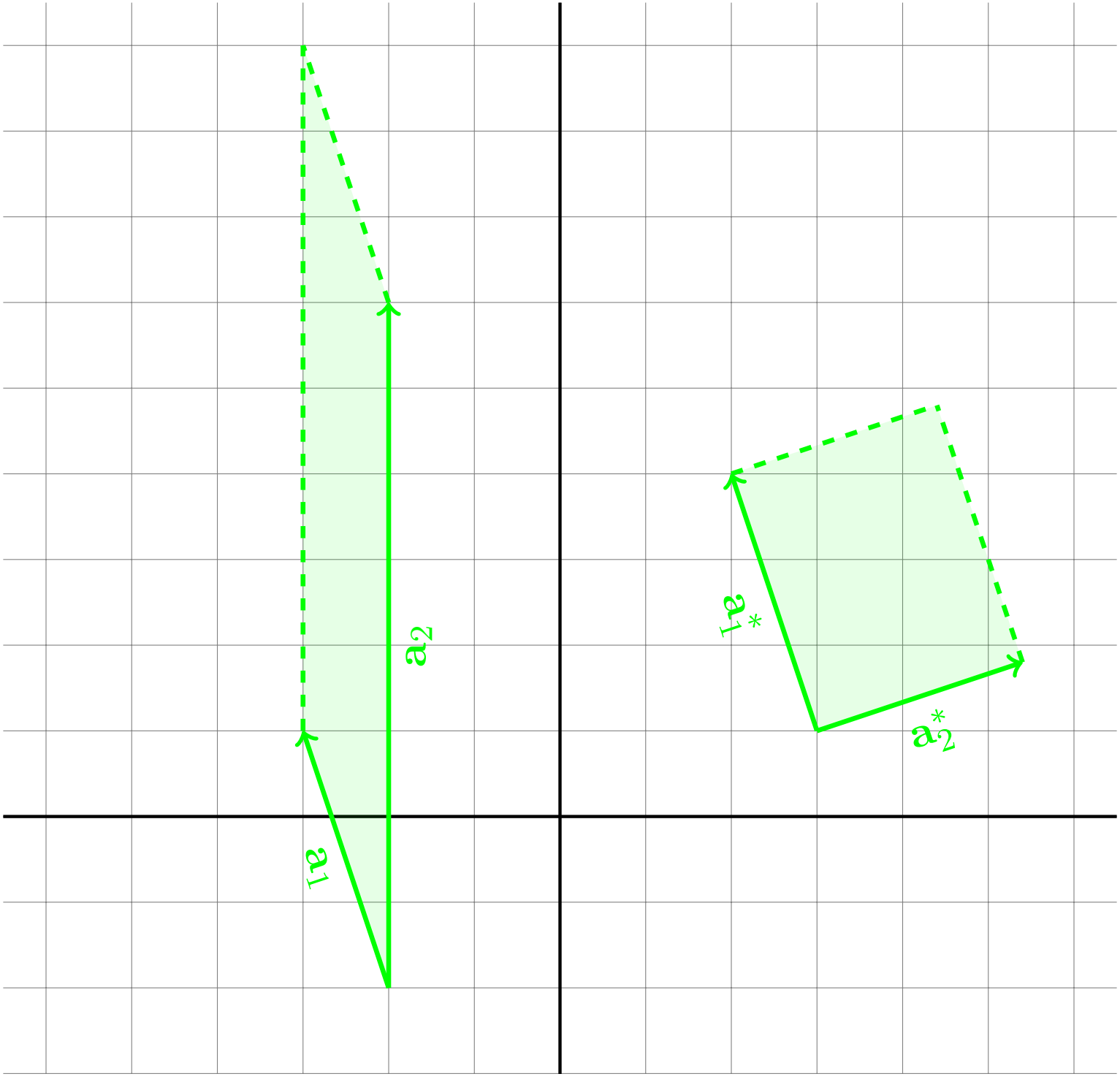


Heuristique gaussienne

Un des principaux intérêts du concept de volume

$$\#\text{ points dans la boule} \approx \frac{\text{Volume de la boule}}{\text{Volume du réseau}}.$$

« Boule » = ensembles des points à distance r de l'origine.


$\mathbf{a}_2$
$\mathbf{a}_1$
$\mathbf{a}_1{}^*$
$\mathbf{a}_2^*$

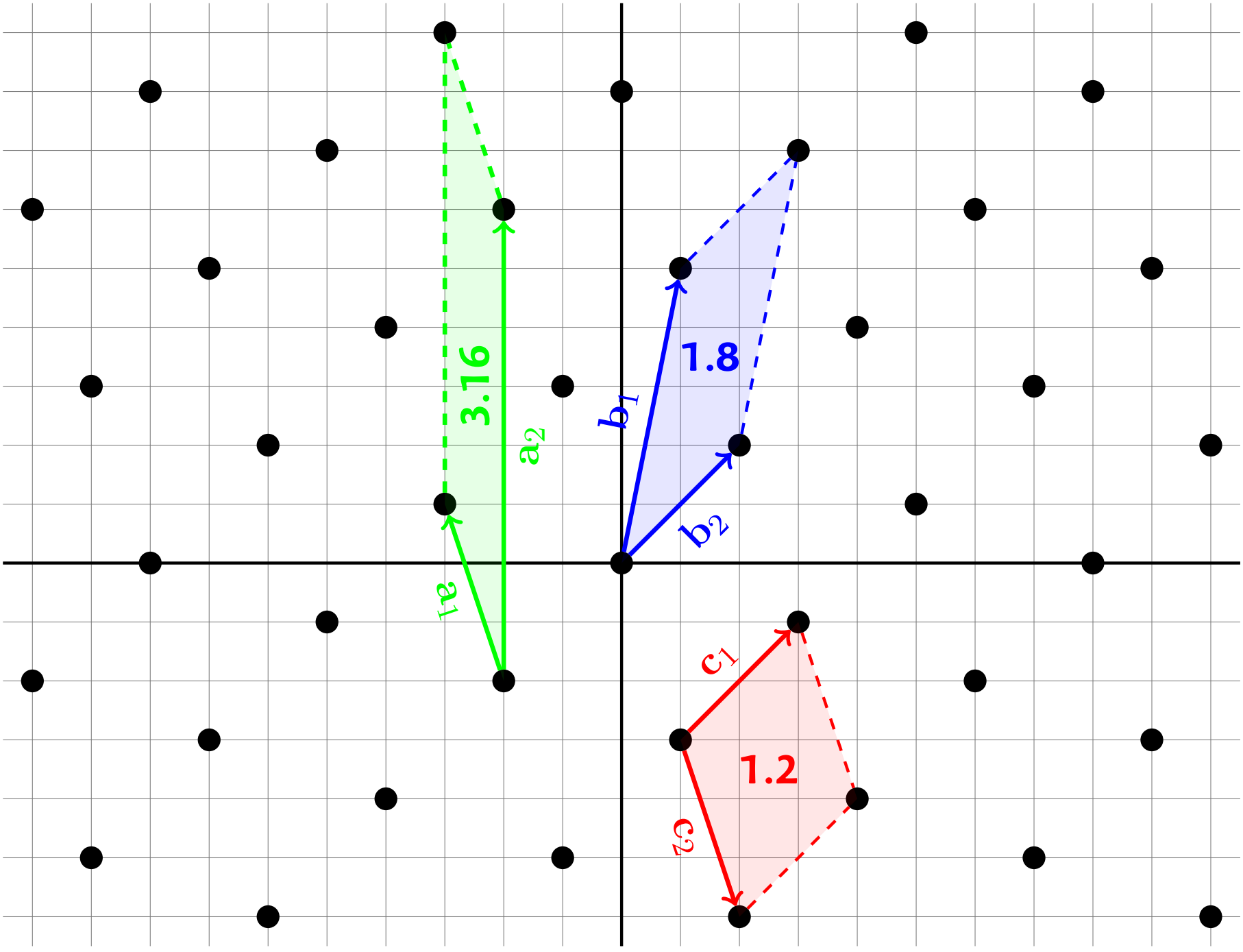
Orthogonalité

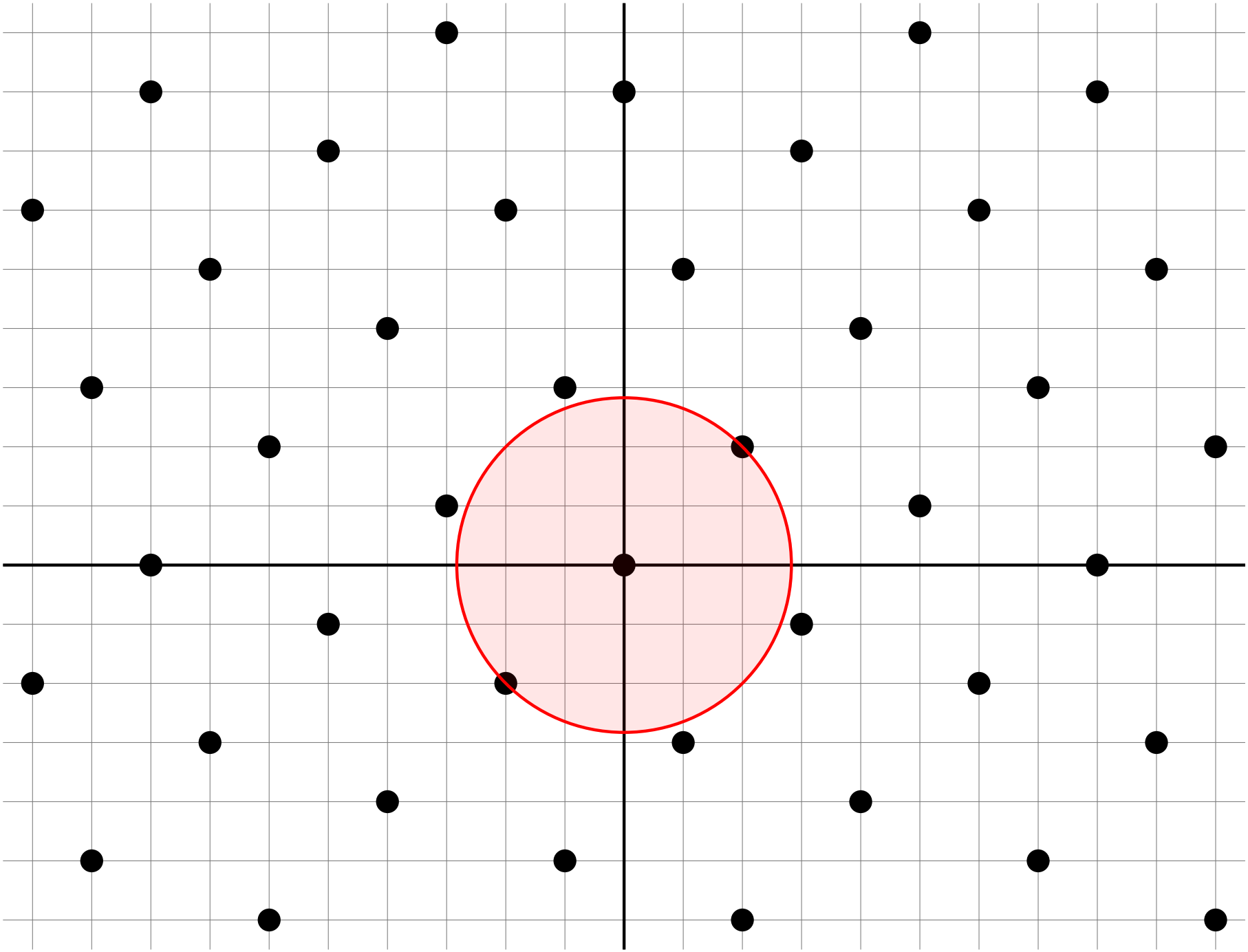
- Orthogonal = « perdpendiculaire »
- Deux vecteurs $\mathbf{x}$ et $\mathbf{y}$ sont orthogonaux ssi $\mathbf{x} \cdot \mathbf{y} = 0$
- Un réseau n'admet pas forcément (et même rarement) une base orthogonale

défaut d'orthogonalité

$$\frac{\|\mathbf{b}_1\| \times \cdots \times \|\mathbf{b}_d\|}{\mathrm{Vol}(\mathcal{L})}$$

Plus il est grand, moins la base est orthogonale / courte.


3.16
a2
a1
1.8
b1
b2
c1
1.2
c2



Plus court vecteur

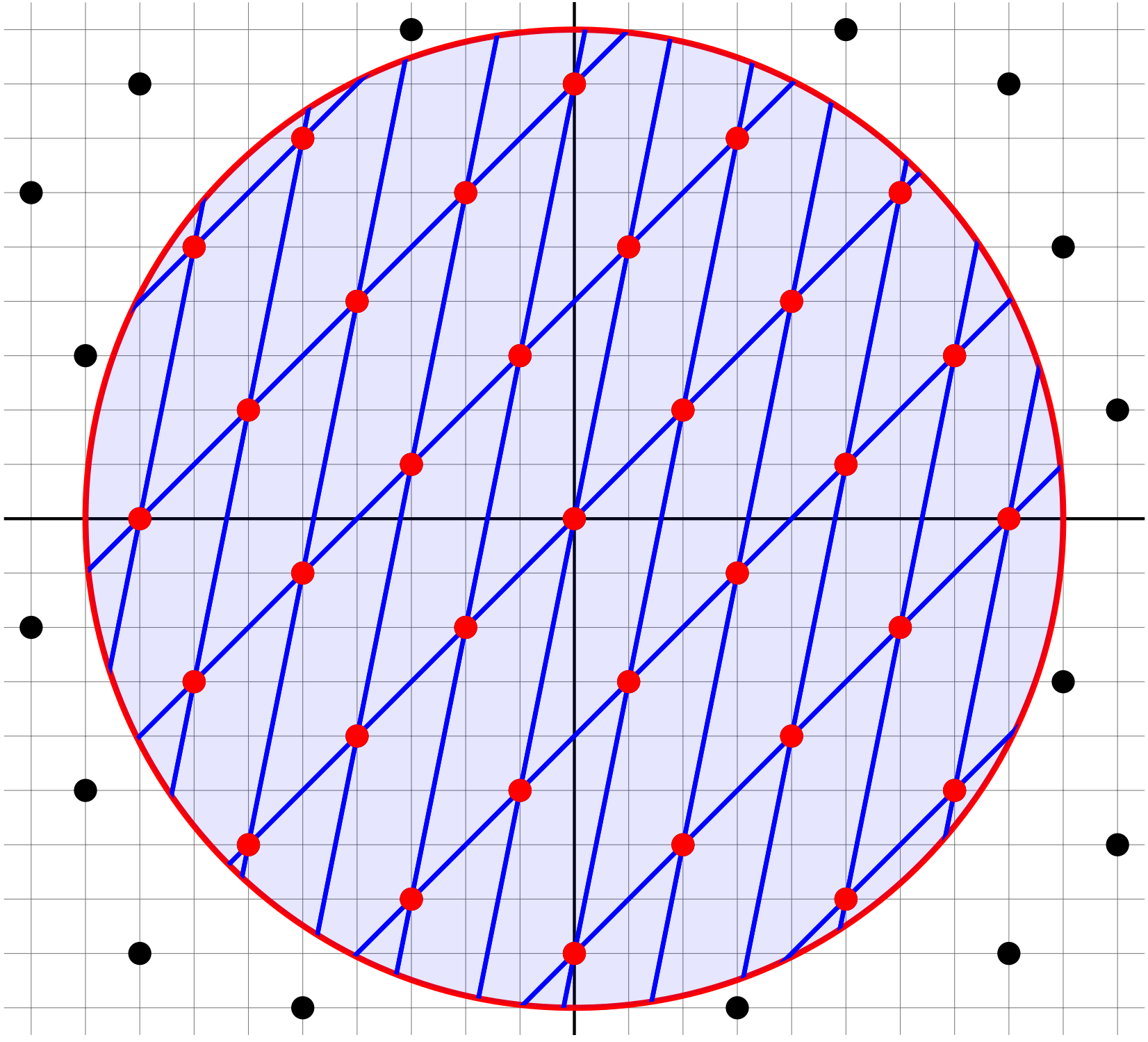
- Un réseau possède un plus court vecteur (non-nul).
- Sa longueur : $\lambda_1(\mathcal{L})$ — caractéristique importante de $\mathcal{L}$

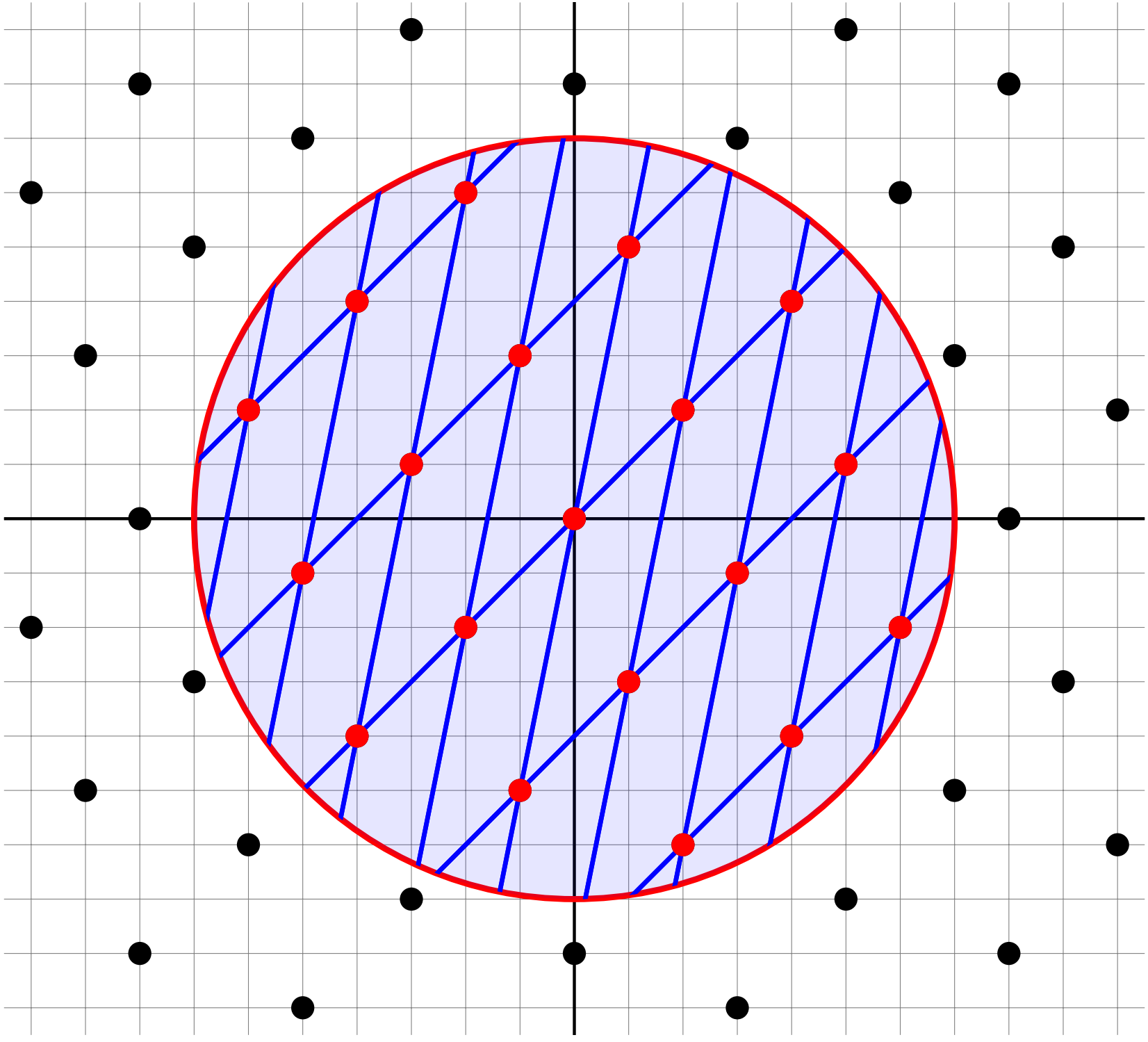
Heuristique gaussienne

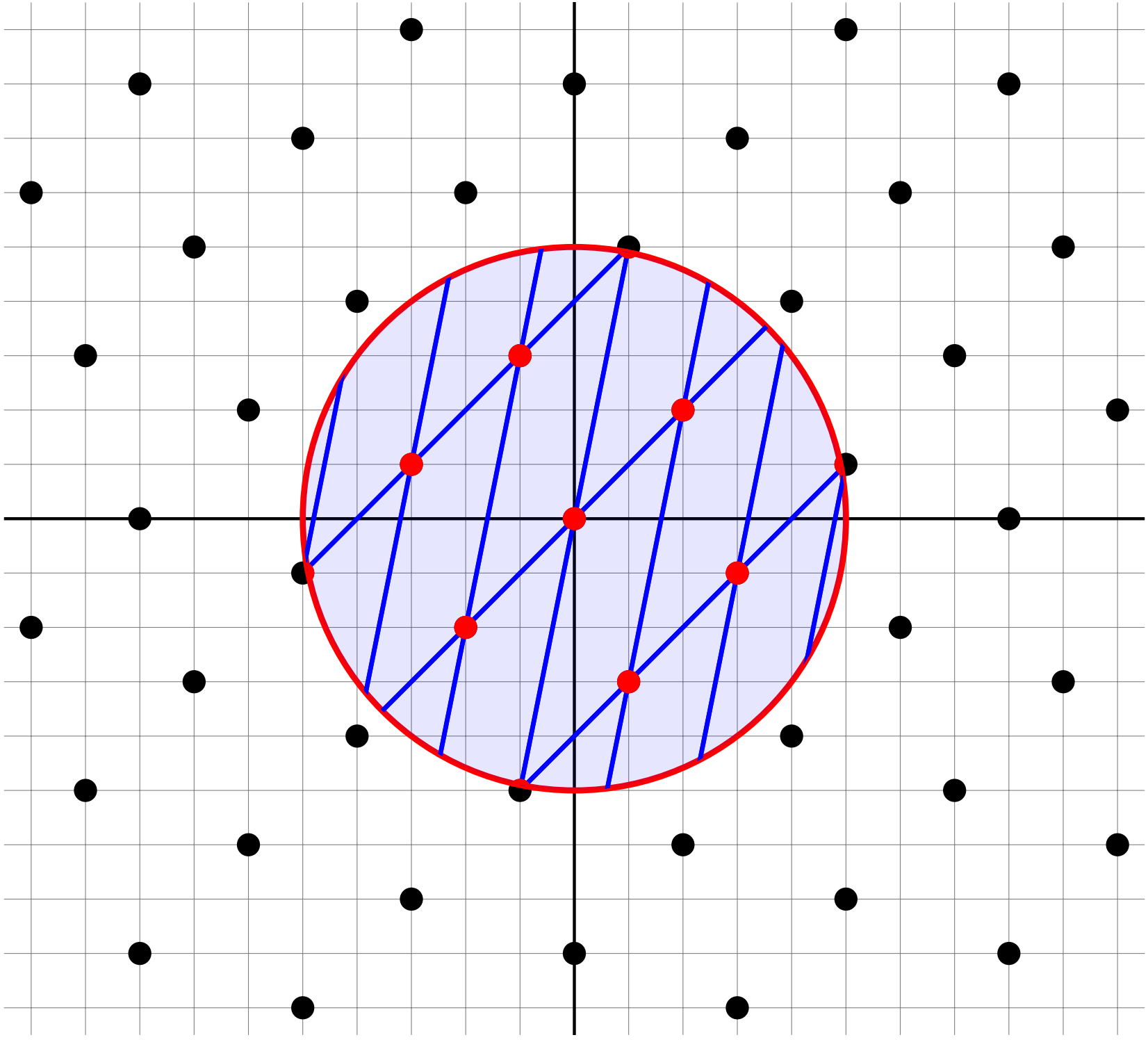
$$\#\text{ points dans la boule} \approx \frac{\text{Volume de la boule}}{\text{Volume du réseau}}$$

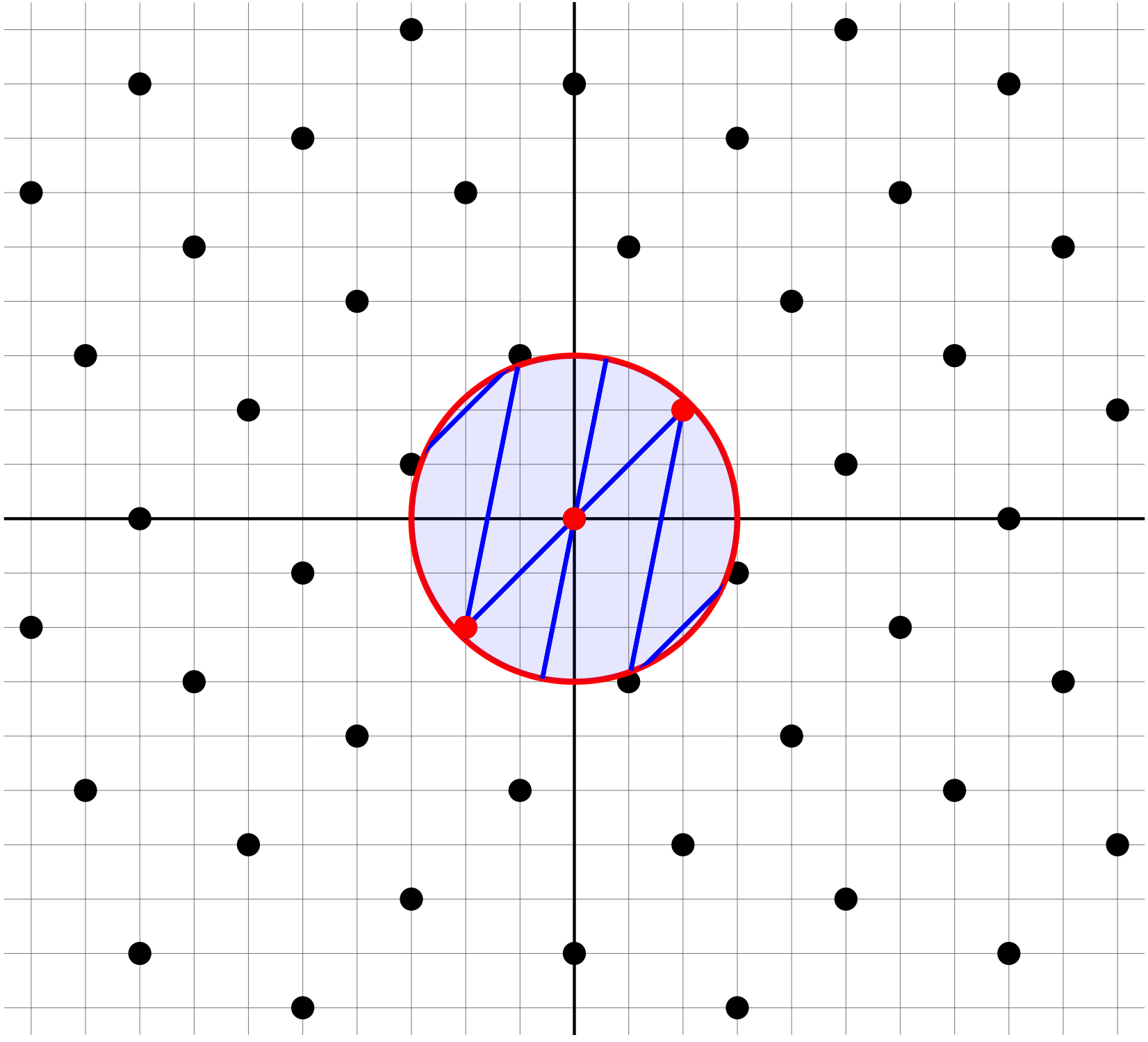
Le + grand rayon pour lequel la boule de rayon r ne contient plus qu'un point est (à peu près) la taille du plus court vecteur.

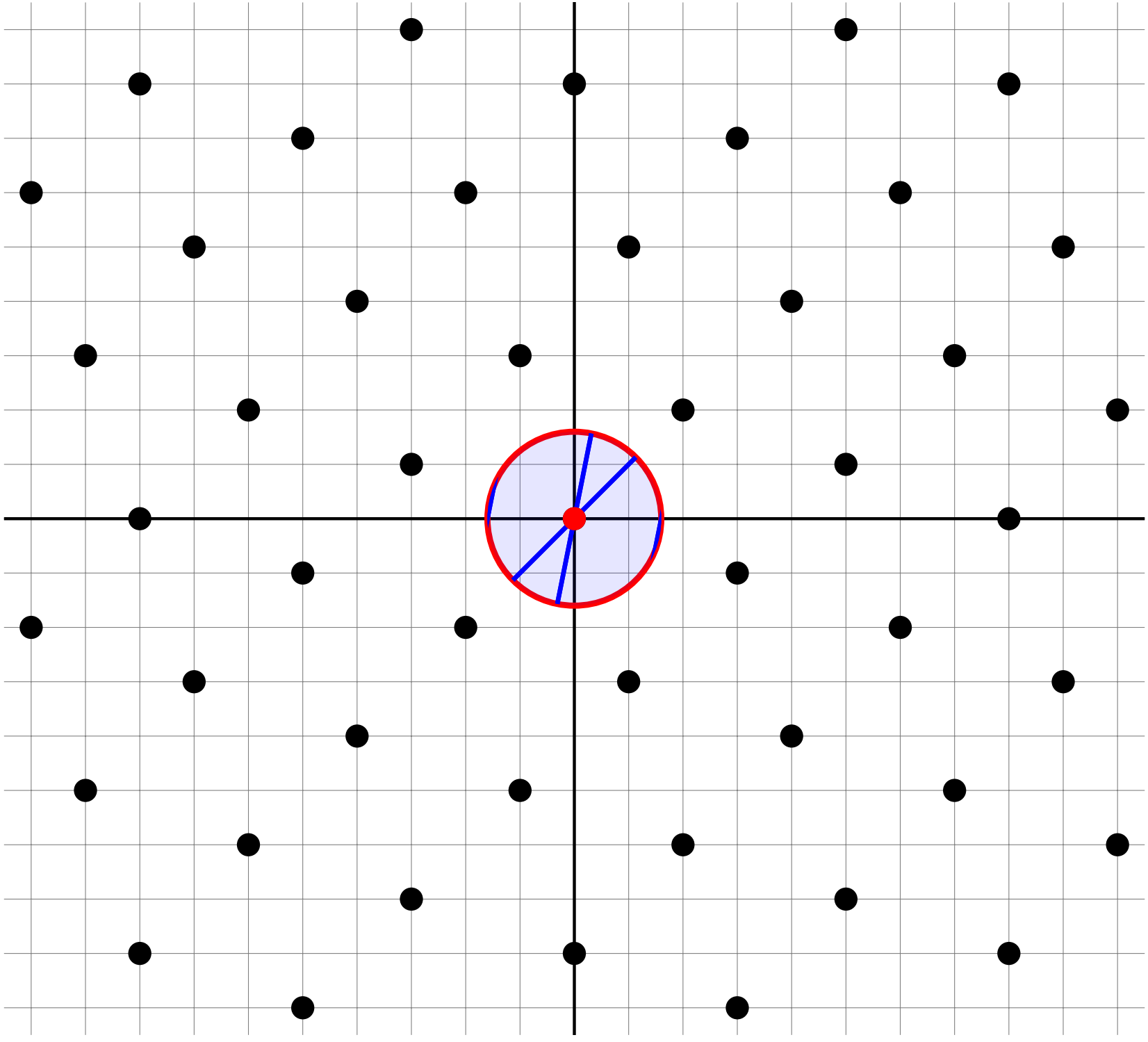
$$\lambda_1(\mathcal{L}) \approx \sqrt{n}\,(\mathrm{Vol}(\mathcal{L}))^{\frac{1}{n}}\,.$$











Plan

Les classiques

Shortest Vector Problem (SVP) — **NP-dur**

Étant donné une base d'un réseau $\mathcal{L}$, calculer un vecteur non-nul $\mathbf{x}$ tel que $\|\mathbf{x}\| = \lambda_1(\mathcal{L})$.

Closest Vector Problem (CVP) — **NP-dur**

Étant donné une base d'un réseau $\mathcal{L}$ et un vecteur $\mathbf{y}$, calculer un vecteur $\mathbf{x}$ tel que $\|\mathbf{y} - \mathbf{x}\| \leq \|\mathbf{y} - \mathbf{z}\|$ pour tout $\mathbf{z} \in \mathcal{L}$.

Les classiques

Approximate Shortest Vector Problem (SVP_γ)

Étant donné une base d'un réseau $\mathcal{L}$, calculer un vecteur non-nul $\mathbf{x}$ tel que $\|\mathbf{x}\| \leq \gamma \times \lambda_1(\mathcal{L})$.

Approximate Closest Vector Problem (CVP_γ)

Étant donné une base d'un réseau $\mathcal{L}$ et un vecteur $\mathbf{y}$, calculer un vecteur $\mathbf{x}$ tel que $\|\mathbf{y} - \mathbf{x}\| \leq \gamma \times \|\mathbf{y} - \mathbf{z}\|$ pour tout $\mathbf{z} \in \mathcal{L}$.

Les classiques

Approximate Shortest Vector Problem (SVP_γ)

Étant donné une base d'un réseau $\mathcal{L}$, calculer un vecteur non-nul $\mathbf{x}$ tel que $\|\mathbf{x}\| \leq \gamma \times \lambda_1(\mathcal{L})$.

Hermite Shortest Vector Problem (HSVP_γ)

Étant donné une base d'un réseau $\mathcal{L}$, calculer un vecteur non-nul $\mathbf{x}$ tel que $\|\mathbf{x}\| \leq \gamma \times \mathrm{Vol}(\mathcal{L})^{1/n}$.

Approximate Closest Vector Problem (CVP_γ)

Étant donné une base d'un réseau $\mathcal{L}$ et un vecteur $\mathbf{y}$, calculer un vecteur $\mathbf{x}$ tel que $\|\mathbf{y} - \mathbf{x}\| \leq \gamma \times \|\mathbf{y} - \mathbf{z}\|$ pour tout $\mathbf{z} \in \mathcal{L}$.

CVP / SVP **exact**

- Complexité : $\mathcal{O}\left(2^{d^2}\right)$ ou $\mathcal{O}\left(2^{d \log d}\right)$
- Facile en petite dimension (≤ 50 disons)
  - Algorithmes disponibles dans `fpylll`
- Impossible en grande dimension (≥ 300 disons)

CVP / SVP **approché** en grande dimension

- « Facile » d'obtenir γ exponentiel en la dimension
  - Algorithmes disponibles dans `fpylll`, `SageMath`, etc.
- Difficile d'obtenir γ faible

Utilisation typique des réseaux en cryptanalyse

Procédure habituelle #1

1. Infos publiques $\rightsquigarrow$ base d'un réseau
2. vecteur spécial $\mathbf{x}$ est « anormalement court »
3. Résoudre SVP $\rightsquigarrow$ obtenir $\mathbf{x}$
4. Extraire secrets de $\mathbf{x}$

Procédure habituelle #2

1. Infos publiques $\rightsquigarrow$ base d'un réseau et $\mathbf{y}$
2. vecteur spécial $\mathbf{x}$ est « anormalement proche » de $\mathbf{y}$
3. Résoudre CVP sur $\mathbf{y}$ $\rightsquigarrow$ obtenir $\mathbf{x}$
4. Extraire secrets de $\mathbf{x}$

- PB #1 : être sûr que $\mathbf{x}$ est bien le plus court/proche
- PB #2 : résoudre SVP/CVP (potentiellement difficile)

Utilisation typique des réseaux en cryptanalyse

Problème #1

Comment être sûr que $\mathbf{x}$ est bien le plus **court** ?

Heuristique gaussienne

$$\lambda_1(\mathcal{L}) \approx \sqrt{n}\,(\mathrm{Vol}(\mathcal{L}))^{\frac{1}{n}}\,.$$

Si on sait d'avance que le réseau contient un vecteur spécial $\mathbf{x}$ sensiblement plus court que ce que « prédit » l'heuristique gaussienne, alors c'est **probablement** le plus court du réseau.

Plan

Présentation de quelques mécanismes cryptographiques
- RSA avec Petits exposants secrets
- Une fonction à sens-unique « post-quantique »
- Générateurs congruentiels linéaires tronqués

Introduction aux réseaux euclidiens
- Propriétés de base, volume, plus court vecteur
- Problèmes algorithmiques dans les réseaux

Cryptanalyse avec les algorithmes exacts pour SVP/CVP
- Attaque de Wiener sur RSA avec petit exposant secret

Application #1

Plan

Présentation de quelques mécanismes cryptographiques
- RSA avec Petits exposants secrets
- Une fonction à sens-unique « post-quantique »
- Générateurs congruentiels linéaires tronqués

Introduction aux réseaux euclidiens
- Propriétés de base, volume, plus court vecteur
- Problèmes algorithmiques dans les réseaux

Cryptanalyse avec les algorithmes exacts pour SVP/CVP
- Attaque de Wiener sur RSA avec petit exposant secret

RSA avec petit exposant secret

Michael J. Wiener, 1989

Situation

- ENTRÉE : e, N
- PROMESSES : $ed = 1 + k \cdot \phi(N)$ et $d \leq N^{\alpha}$ (« petit »)
- BUT : retrouver d (ou $\phi(N)$).

$$B = \begin{pmatrix} \sqrt{N} & -e \\ 0 & N \end{pmatrix}$$

[Pourquoi ? → poly]

Vecteur (inconnu) « intéressant » : $\mathbf{x} = (d, k)B = (d\sqrt{N}, kN - ed)$

Applications des diapos précédentes

- Volume du réseau : $\mathrm{Vol}(\mathcal{L}) = |\det B| = N\sqrt{N} = N^{3/2}$.
- Heuristique gaussienne suggère $\lambda_1(\mathcal{L}) \approx \sqrt{\mathrm{Vol}(\mathcal{L})} \approx N^{3/4}$.

$\Longrightarrow$ Si $\|\mathbf{x}\| \lll N^{3/4}$, on retrouve $\mathbf{x}$ avec SVP $\Longrightarrow$ d

RSA avec petit exposant secret (suite & fin)

Michael J. Wiener, 1989

Situation

- $ed = 1 + k \cdot \phi(N)$
- $\mathbf{x} = (d, k)B = (d\sqrt{N}, kN - ed)$
- $B = \begin{pmatrix} \sqrt{N} & -e \\ 0 & N \end{pmatrix}$

CLAIM : $\|\mathbf{x}\| \approx d\sqrt{N}$

Il suffit de montrer $kN - ed \approx d\sqrt{N}$.

1. $kN - ed = k(N - \phi(N)) - 1$.
2. $k = (ed - 1)/\phi(n) < ed/\phi(N) < d$
3. $N - \phi(N) = N - (p-1)(q-1) = p + q - 1 \approx \sqrt{N}$. CQFD

Conclusion : si $d < N^{1/4}$, alors...

- $\|\mathbf{x}\| < N^{3/4} \rightsquigarrow \mathbf{x}$ = plus court vecteur
- Résoudre SVP en dim. 2 révèle d

Debriefing et retour sur les réseaux

- Paramètres proposés ($N \approx 2048$ bits, $d \approx 128$ bits) cassés !

```
import fpylll                # cf. https://github.com/fplll/fpylll
def attack(N, e):            # temps d'exécution : 1.5ms
    M = fpylll.IntegerMatrix(2, 2)
    s = int(sqrt(N))
    M[0, 0] = s
    M[0, 1] = -e
    M[1, 1] = N
    x = fpylll.SVP.shortest_vector(M)
    return abs(x[0] // s)
```

- On peut construire une mauvaise base du réseau avec les informations publiques (vecteurs de taille $\approx N$)

$\Rightarrow$ Le plus court vecteur est de taille $\lll N^{3/4}$

- Une super base contient directement les vecteurs courts...

Partir sur de bonnes bases (SVP)