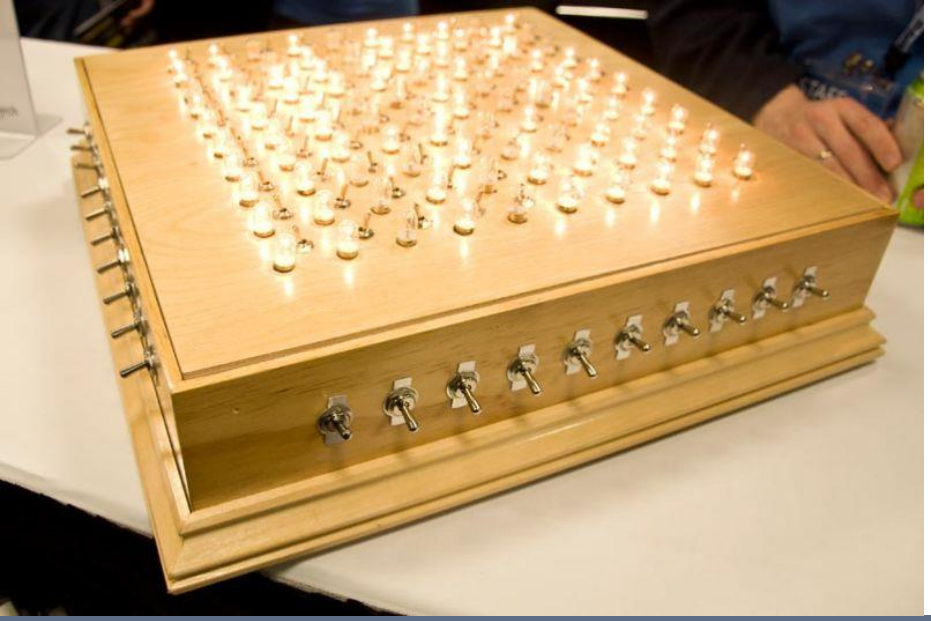
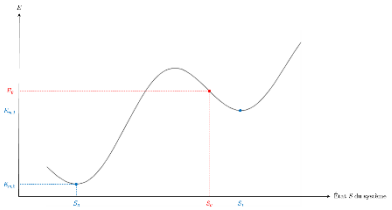




```
def Minimize(pos, data, nbsteps=10000, maxcounter=30,
    earlyabort=0, coeffA = 1.0, coeffB = 1.0, coeffC = 2.0) :

    def T(i) :
        return (1-i/nbsteps)

    coeffC *= (data.nbc + data.nbl)
    def P(Efrom, Eto, Ebest, T) :
        diff = coeffA * (Eto - Ebest) + coeffB * (Eto -
            Efrom)
        return exp(-diff / (coeffC*T))

    score = Score(pos)
    best, bestscore, counter = pos, score, 0
    lst = [ score ]

    for i in range(nbsteps) :
        set = randint(0, data.nbl + data.nbc - 1)
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Le problème de Berlekamp : résolution théorique, résolution pratique

Théorie des codes et recuit simulé

Clémentine Laurens

Année scolaire 2016-2017

1 Position du problème : principe du *Berlekamp's Switching Game*

1 Position du problème : principe du *Berlekamp's Switching Game*

1 tableau d'ampoules à n lignes et n colonnes
1 interrupteur au bout de chaque ligne et de chaque colonne

1 Position du problème : principe du *Berlekamp's Switching Game*

1 tableau d'ampoules à n lignes et n colonnes
1 interrupteur au bout de chaque ligne et de chaque colonne

Joueur 1

- Cherche à maximiser le nombre d'ampoules allumées.
- Choisit la configuration initiale.

Joueur 2

- Cherche à minimiser le nombre d'ampoules allumées.
- Peut actionner les interrupteurs à l'infini et dans n'importe quel ordre.

1 Position du problème : principe du *Berlekamp's Switching Game*

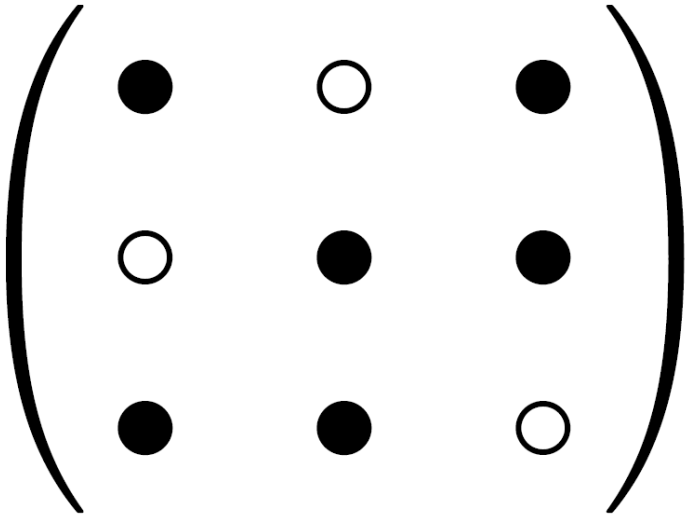
1 tableau d'ampoules à n lignes et n colonnes
1 interrupteur au bout de chaque ligne et de chaque colonne

Joueur 1

- Cherche à maximiser le nombre d'ampoules allumées.
- Choisit la configuration initiale.

Joueur 2

- Cherche à minimiser le nombre d'ampoules allumées.
- Peut actionner les interrupteurs à l'infini et dans n'importe quel ordre.



1 Position du problème : principe du *Berlekamp's Switching Game*

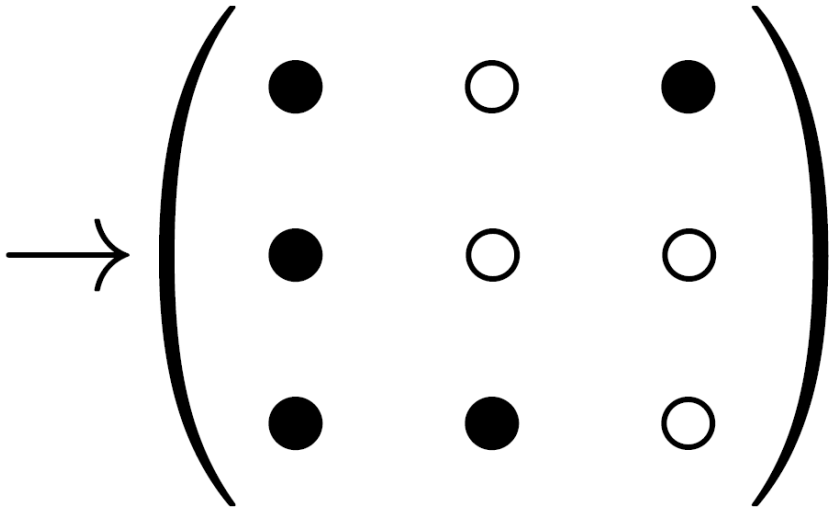
1 tableau d'ampoules à n lignes et n colonnes
1 interrupteur au bout de chaque ligne et de chaque colonne

Joueur 1

- Cherche à maximiser le nombre d'ampoules allumées.
- Choisit la configuration initiale.

Joueur 2

- Cherche à minimiser le nombre d'ampoules allumées.
- Peut actionner les interrupteurs à l'infini et dans n'importe quel ordre.



1 Position du problème : principe du *Berlekamp's Switching Game*

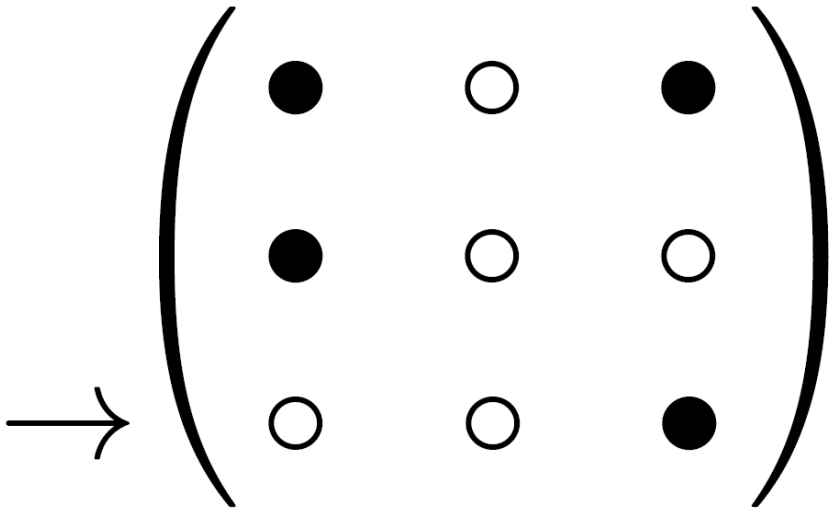
1 tableau d'ampoules à n lignes et n colonnes
1 interrupteur au bout de chaque ligne et de chaque colonne

Joueur 1

- Cherche à maximiser le nombre d'ampoules allumées.
- Choisit la configuration initiale.

Joueur 2

- Cherche à minimiser le nombre d'ampoules allumées.
- Peut actionner les interrupteurs à l'infini et dans n'importe quel ordre.



1 Position du problème : principe du *Berlekamp's Switching Game*

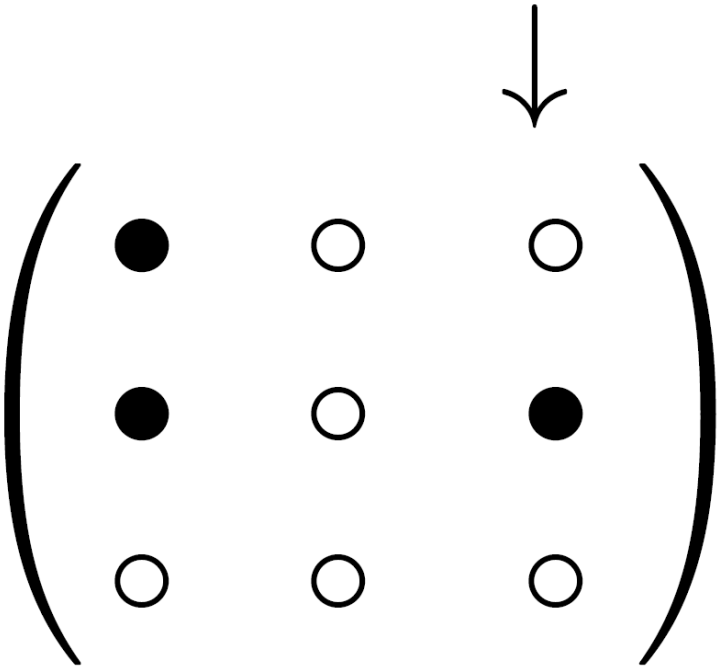
1 tableau d'ampoules à n lignes et n colonnes
1 interrupteur au bout de chaque ligne et de chaque colonne

Joueur 1

- Cherche à maximiser le nombre d'ampoules allumées.
- Choisit la configuration initiale.

Joueur 2

- Cherche à minimiser le nombre d'ampoules allumées.
- Peut actionner les interrupteurs à l'infini et dans n'importe quel ordre.



1 Position du problème : principe du *Berlekamp's Switching Game*

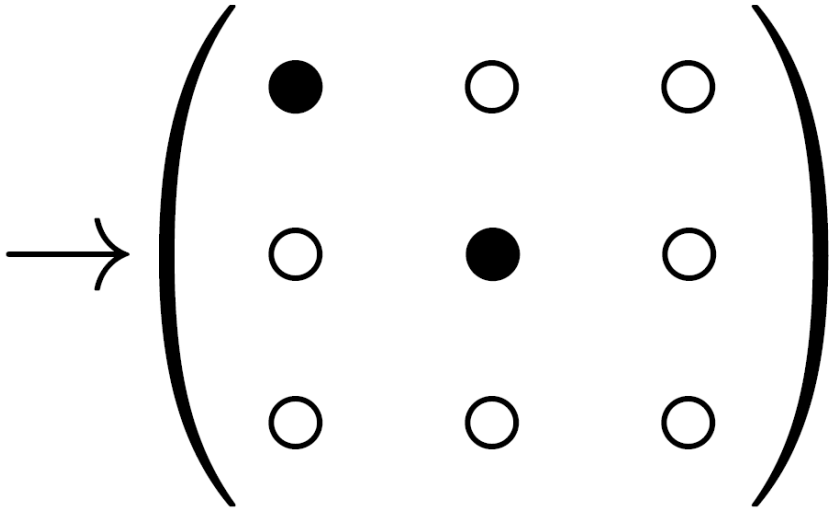
1 tableau d'ampoules à n lignes et n colonnes
1 interrupteur au bout de chaque ligne et de chaque colonne

Joueur 1

- Cherche à maximiser le nombre d'ampoules allumées.
- Choisit la configuration initiale.

Joueur 2

- Cherche à minimiser le nombre d'ampoules allumées.
- Peut actionner les interrupteurs à l'infini et dans n'importe quel ordre.



1 Position du problème : principe du *Berlekamp's Switching Game*

1 Position du problème : principe du *Berlekamp's Switching Game*

Problème 1 (problème intermédiaire).

Configuration de départ quelconque

↳ Configurations accessibles, à partir de cette configuration de départ, qui minimisent le nombre d'ampoules allumées ?

⇨ M**éthode algorithmique** de recherche des configurations optimales pour le second joueur, à partir d'une situation de départ donnée.

1 Position du problème : principe du *Berlekamp's Switching Game*

Problème 1 (problème intermédiaire).

Configuration de départ quelconque

↳ Configurations accessibles, à partir de cette configuration de départ, qui minimisent le nombre d'ampoules allumées ?

⇨ M**éthode algorithmique** de recherche des configurations optimales pour le second joueur, à partir d'une situation de départ donnée.

Problème 2 (*Problème de Berlekamp*).

Configurations initiales qui maximisent le nombre minimal d'ampoules allumées que peut obtenir le second joueur ?

1 Position du problème : principe du *Berlekamp's Switching Game*

Problème 1 (problème intermédiaire).

Configuration de départ quelconque

↳ Configurations accessibles, à partir de cette configuration de départ, qui minimisent le nombre d'ampoules allumées ?

⇨ M**éthode algorithmique** de recherche des configurations optimales pour le second joueur, à partir d'une situation de départ donnée.

Problème 2 (*Problème de Berlekamp*).

Configurations initiales qui maximisent le nombre minimal d'ampoules allumées que peut obtenir le second joueur ?

Etude théorique → Théorie des codes

1 Position du problème : principe du *Berlekamp's Switching Game*

Problème 1 (problème intermédiaire).

Configuration de départ quelconque

↳ Configurations accessibles, à partir de cette configuration de départ, qui minimisent le nombre d'ampoules allumées ?

⇨ M**éthode algorithmique** de recherche des configurations optimales pour le second joueur, à partir d'une situation de départ donnée.

Problème 2 (*Problème de Berlekamp*).

Configurations initiales qui maximisent le nombre minimal d'ampoules allumées que peut obtenir le second joueur ?

Etude théorique → Théorie des codes

Etude pratique → Adaptation de l'algorithme du « recuit simulé »

2 Quelques notions de théorie des codes

2.1 Approche intuitive

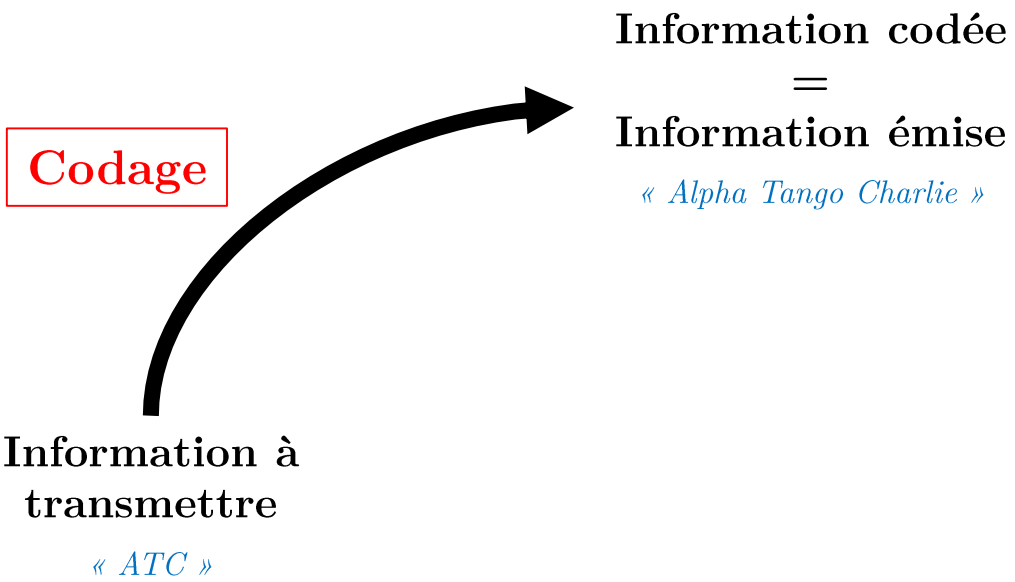
2 Quelques notions de théorie des codes

2.1 Approche intuitive

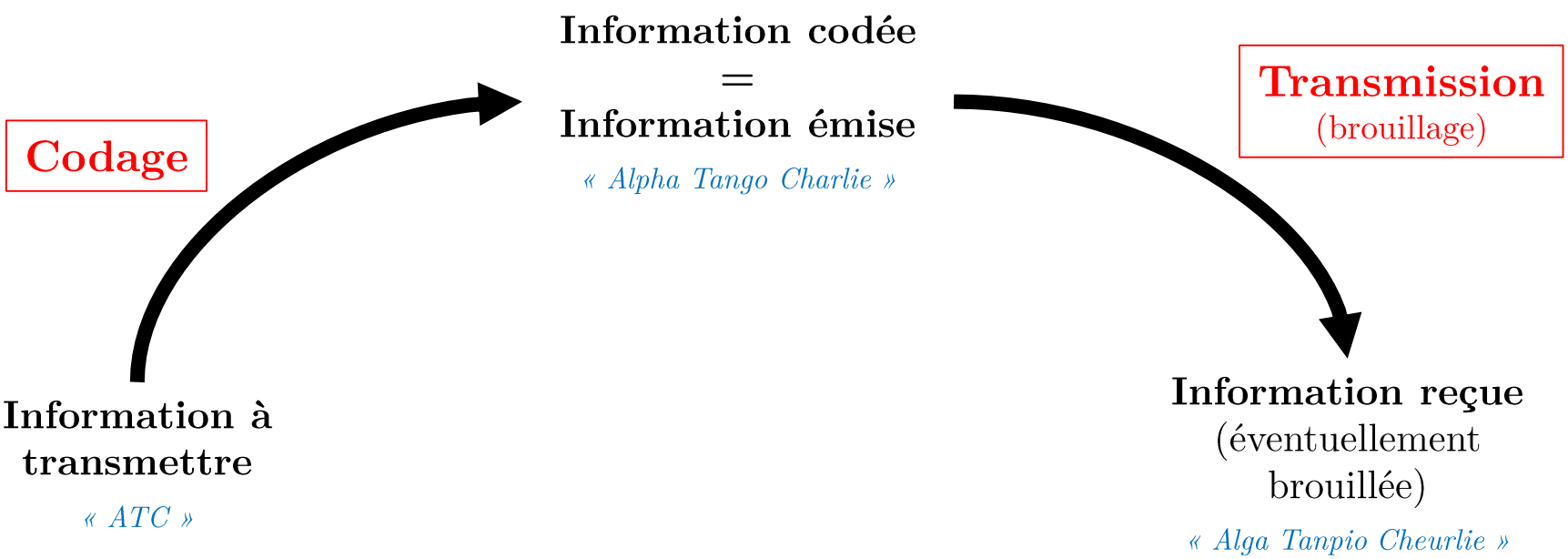
Information à transmettre

« ATC »

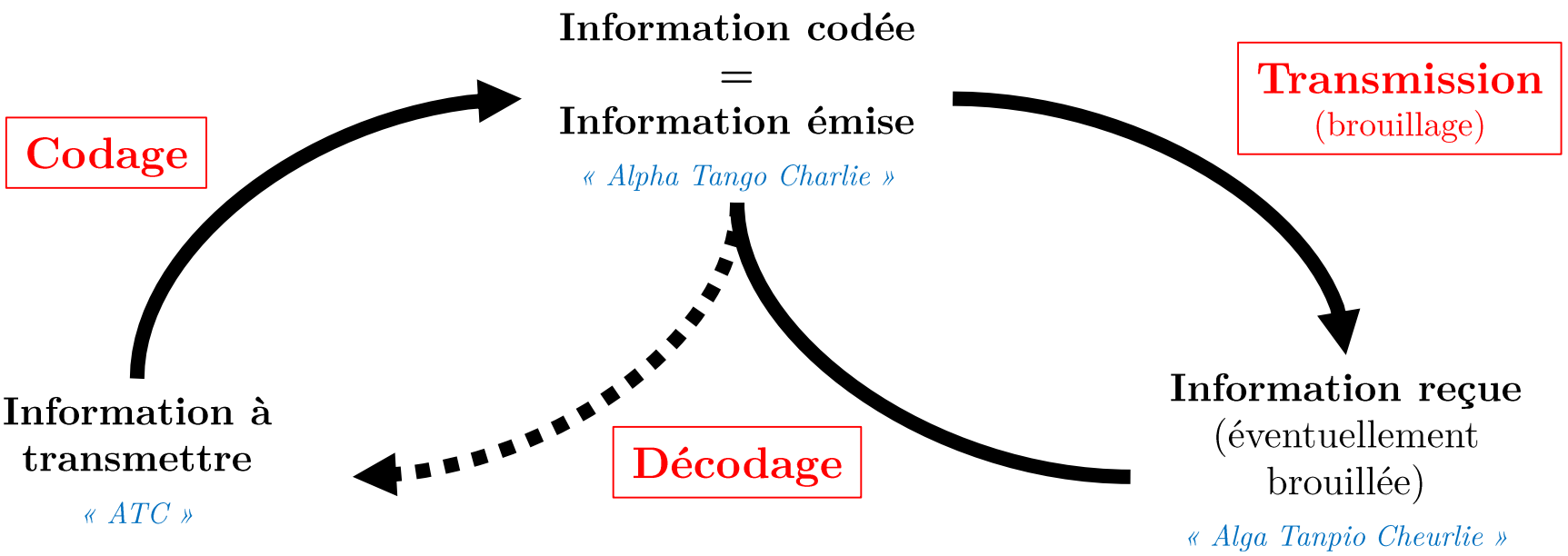
2.1 Approche intuitive



2.1 Approche intuitive



2.1 Approche intuitive



2.2 Formalisation : définitions fondamentales

2.2 Formalisation : définitions fondamentales

$\mathbb{F}$= corps fini de cardinal q
$n \in \mathbb{N}$
$k \in [\![0, n]\!]$

Code linéaire $C(n, k)$ = sev de $\mathbb{F}^n$ de dimension k

2.2 Formalisation : définitions fondamentales

$\mathbb{F}$= corps fini de cardinal q
$n \in \mathbb{N}$
$k \in [\![0, n]\!]$

Code linéaire $C(n, k)$ = sev de $\mathbb{F}^n$ de dimension k

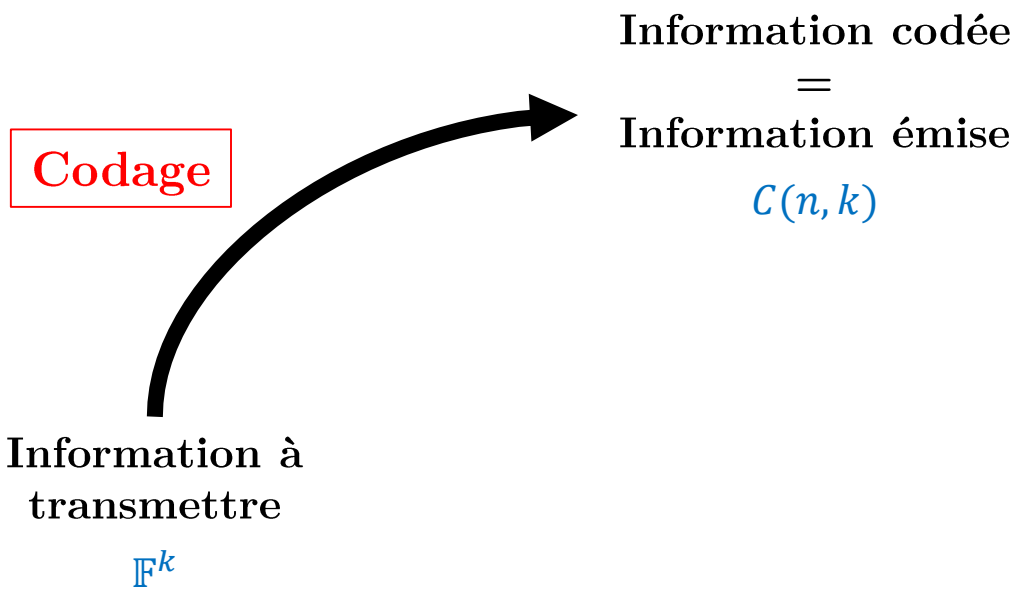
Information à transmettre

$\mathbb{F}^k$

2.2 Formalisation : définitions fondamentales

$\mathbb{F}$= corps fini de cardinal q
$n \in \mathbb{N}$
$k \in [\![0, n]\!]$

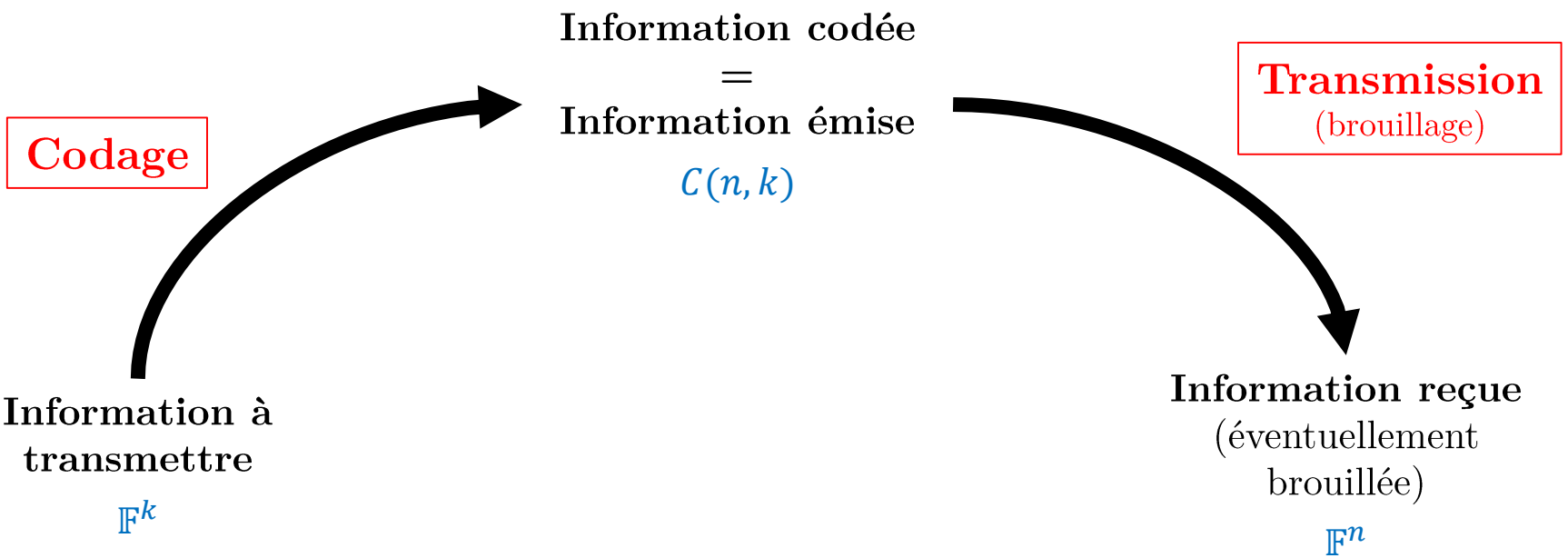
Code linéaire $C(n,k)$ = sev de $\mathbb{F}^n$ de dimension k



2.2 Formalisation : définitions fondamentales

$\mathbb{F}$= corps fini de cardinal q
$n \in \mathbb{N}$
$k \in [\![0, n]\!]$

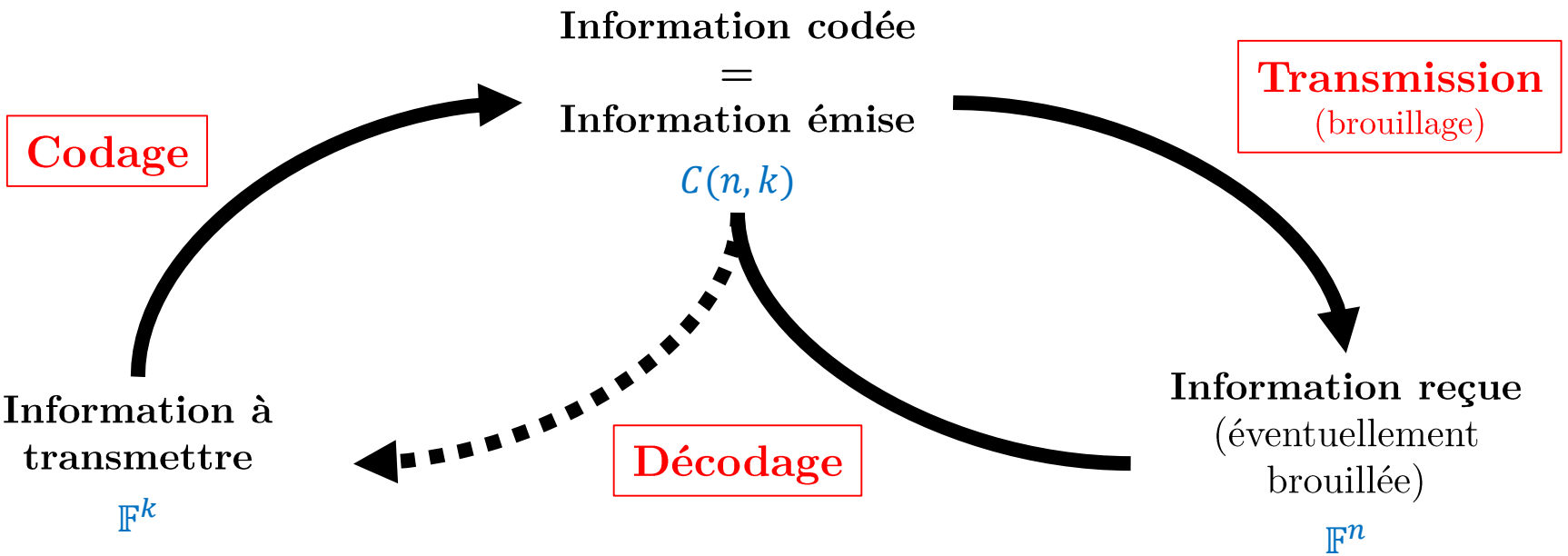
Code linéaire $C(n,k)$ = sev de $\mathbb{F}^n$ de dimension k



2.2 Formalisation : définitions fondamentales

$\mathbb{F}$= corps fini de cardinal q
$n \in \mathbb{N}$
$k \in [\![0, n]\!]$

Code linéaire $C(n,k)$ = sev de $\mathbb{F}^n$ de dimension k



3 Du *Berlekamp's Switching Game* à la théorie des codes

3.1 Modélisation mathématique du problème

3 Du *Berlekamp's Switching Game* à la théorie des codes

3.1 Modélisation mathématique du problème

$n \in \mathbb{N}$
$\mathcal{M}(n) = \mathcal{M}_n(\mathbb{F}_2)$

$$\begin{pmatrix} \bullet & \circ & \bullet \\ \circ & \bullet & \bullet \\ \bullet & \bullet & \circ \end{pmatrix} \longmapsto \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

3 Du *Berlekamp's Switching Game* à la théorie des codes

3.1 Modélisation mathématique du problème

$n \in \mathbb{N}$
$\mathcal{M}(n) = \mathcal{M}_n(\mathbb{F}_2)$

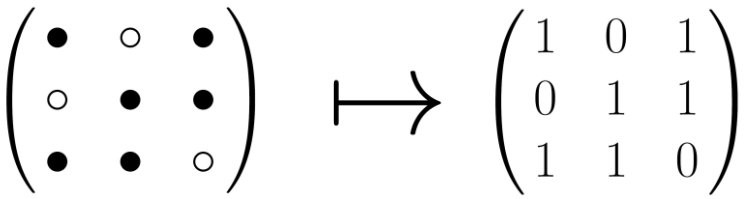
$$\begin{pmatrix} \bullet & \circ & \bullet \\ \circ & \bullet & \bullet \\ \bullet & \bullet & \circ \end{pmatrix} \longmapsto \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

$\mathcal{C} = Vect\ (\{C \in \mathcal{M}(n) \mid C$ comporte exactement une ligne ou une colonne de 1$\})$

3 Du *Berlekamp's Switching Game* à la théorie des codes

3.1 Modélisation mathématique du problème

$n \in \mathbb{N}$
$\mathcal{M}(n) = \mathcal{M}_n(\mathbb{F}_2)$



$\mathcal{C} = Vect\ (\{C \in \mathcal{M}(n) \mid C$ comporte exactement une ligne ou une colonne de 1$\})$

Configuration initiale : $S \in \mathcal{M}(n)$

↳ Configurations accessibles :

$$\bar{S} = S + \mathcal{C} = \{S + C \mid\ C \in \mathcal{C}\}$$

3.2 Une distance adaptée au problème : la distance de Hamming

3.2 Une distance adaptée au problème : la distance de Hamming

Définition (Distance de Hamming)**.**

$\forall (M_1, M_2) \in \left(\mathcal{M}(n)\right)^2, \quad d_H(M_1, M_2) =$ nombre de composantes dont diffèrent M_1 et M_2

3.2 Une distance adaptée au problème : la distance de Hamming

Définition (Distance de Hamming)**.**

$\forall (M_1, M_2) \in \left(\mathcal{M}(n)\right)^2, \quad d_H(M_1, M_2) =$ nombre de composantes dont diffèrent M_1 et M_2

Définition (Poids d'un vecteur)**.**

$$\forall M \in \mathcal{M}(n)\,, \quad w_H(M) = d_H\left(M, 0_{\mathcal{M}(n)}\right)$$

3.3 Réduction du *Problème de Berlekamp* en problème de théorie des codes

3.3 Réduction du *Problème de Berlekamp* en problème de théorie des codes

Problème 1 (problème intermédiaire).

Etant donnée une matrice S de $\mathcal{M}(n)$, on recherche les matrices S' de $\mathcal{M}(n)$ vérifiant :

- $\exists C \in \mathcal{C}$ telle que $S' = S + C$
- $w_H(S') = \min_{C \in \mathcal{C}} \{w_H(M + C)\}$

3.3 Réduction du *Problème de Berlekamp* en problème de théorie des codes

Problème 1 (problème intermédiaire).

Etant donnée une matrice S de $\mathcal{M}(n)$, on recherche les matrices S' de $\mathcal{M}(n)$ vérifiant :

- $\exists C \in \mathcal{C}$ telle que $S' = S + C$
- $w_H(S') = \min_{C \in \mathcal{C}}\{w_H(M + C)\}$

Problème 2 (*Problème de Berlekamp*).

On recherche l'ensemble des matrices S de $\mathcal{M}(n)$ vérifiant :

$$\min_{C \in \mathcal{C}}\{w_H(S + C)\} = \max_{M \in \mathcal{M}(n)}\left\{\min_{C \in \mathcal{C}}\{w_H(M + C)\}\right\}$$

4 Résolution théorique : groupe quotient et tableau standard

4.1 Résolution théorique

4 Résolution théorique : groupe quotient et tableau standard

4.1 Résolution théorique

Définition (Relation d'équivalence $\mathcal{R}$)**.**

$\forall (M_1, M_2) \in \left(\mathcal{M}(n)\right)^2$,

$$M_1 \,\mathcal{R}\, M_2 \Leftrightarrow \exists C \in \mathcal{C}\,, M_1 = M_2 + C$$

4 Résolution théorique : groupe quotient et tableau standard

4.1 Résolution théorique

Définition (Relation d'équivalence $\mathcal{R}$)**.**

$\forall (M_1, M_2) \in \left(\mathcal{M}(n)\right)^2$,

$$M_1 \,\mathcal{R}\, M_2 \Leftrightarrow \exists C \in \mathcal{C}\,, M_1 = M_2 + C$$

Définition (Chef de classe)**.**

Elément de poids minimal d'une classe d'équivalence de $\mathcal{R}$.

4 Résolution théorique : groupe quotient et tableau standard

4.1 Résolution théorique

4.1 Résolution théorique

Définition / Méthode de résolution (Tableau standard)**.**

$$\Big(\quad M_i + C_j \quad \Big)_{(i,j)\in[\![1,2^{n^2-2n+1}]\!]\times[\![1,2^{2n-1}]\!]}$$

où :

- $\{ M_i \mid i \in [\![1, 2^{n^2-2n+1}]\!] \}$ = chefs de classe (avec $M_0 = 0_{\mathcal{M}(n)}$)
- $\mathcal{C} = \{ C_j \mid j \in [\![1, 2^{2n-1}]\!] \}$ (avec $C_0 = 0_{\mathcal{M}(n)}$)

4.2 Limites de cette résolution : problèmes de complexité

4.2 Limites de cette résolution : problèmes de complexité

- Construction du tableau : 2^{n^2} vecteurs

4.2 Limites de cette résolution : problèmes de complexité

- Construction du tableau : 2^{n^2} vecteurs

- Lecture du tableau

5 Une solution algorithmique efficace

5.1 Modélisation informatique

5.1 Modélisation informatique

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

5.1 Modélisation informatique

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\overline{101111001}^{2}$$

5.1 Modélisation informatique

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\overline{101111001}^{2}$$

377

5.2 Principe du « recuit simulé »

5.2 Principe du « recuit simulé »

Fonction numérique E à minimiser

Paramètre numérique T à valeurs dans $\mathbb{R}_+^*$

5.2 Principe du « recuit simulé »

Fonction numérique E à minimiser

Paramètre numérique T à valeurs dans $\mathbb{R}_+^*$

Initialisation :

- $E = E_0$
- $T = T_0$

5.2 Principe du « recuit simulé »

Fonction numérique E à minimiser

Paramètre numérique T à valeurs dans $\mathbb{R}_+^*$

Initialisation :

- $E = E_0$
- $T = T_0$

Récursion :

Modification élémentaire de l'état du système

Calcul des nouvelles valeurs de E et T

5.2 Principe du « recuit simulé »

Fonction numérique E à minimiser

Paramètre numérique T à valeurs dans $\mathbb{R}_+^*$

Initialisation :

- $E = E_0$
- $T = T_0$

Récursion :

Modification élémentaire de l'état du système

Calcul des nouvelles valeurs de E et T

↳ Si la nouvelle valeur de E est inférieure à la précédente : nouvel état conservé

5.2 Principe du « recuit simulé »

Fonction numérique E à minimiser

Paramètre numérique T à valeurs dans $\mathbb{R}_+^*$

Initialisation :

- $E = E_0$
- $T = T_0$

Récursion :

Modification élémentaire de l'état du système

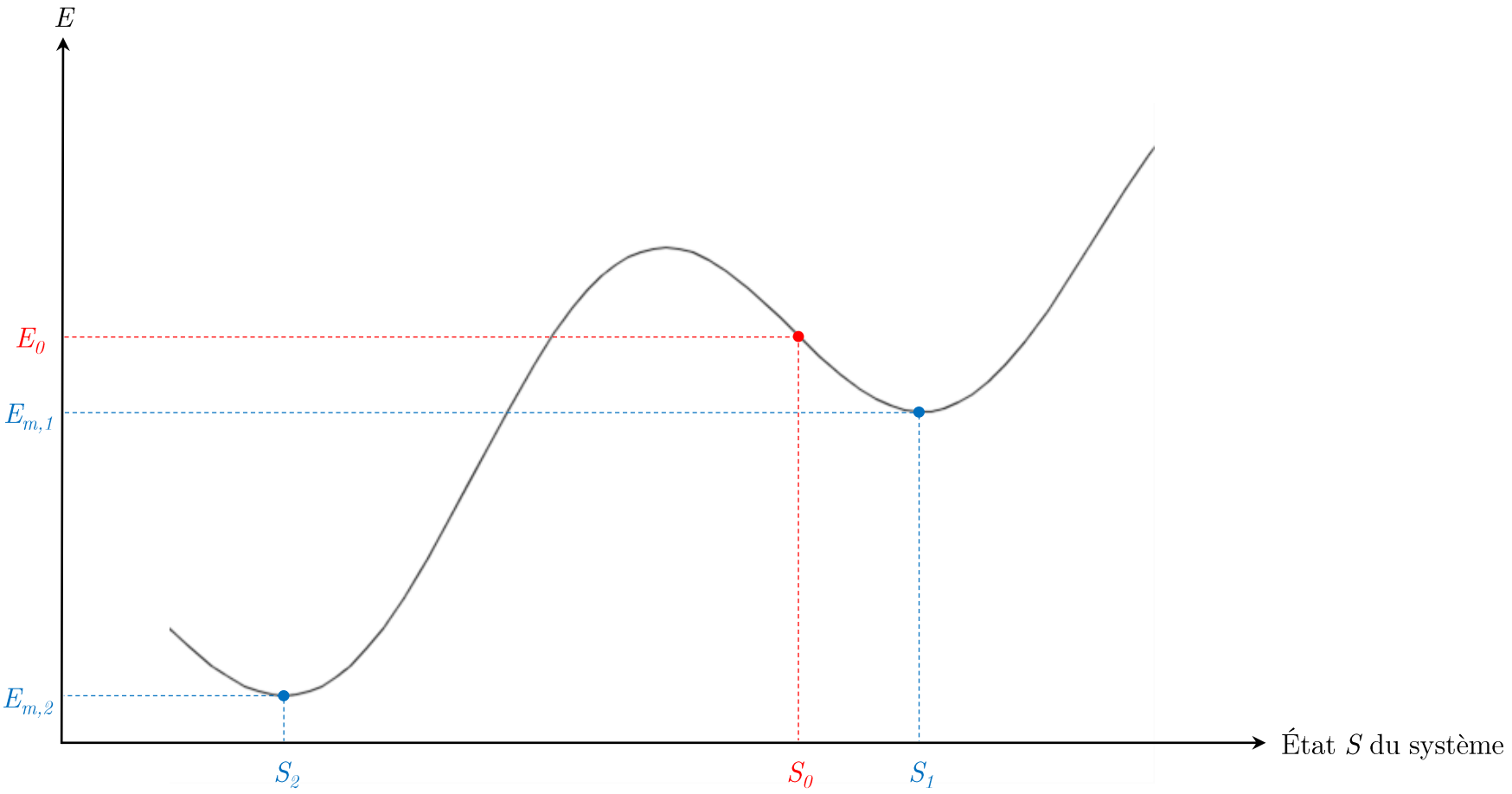
Calcul des nouvelles valeurs de E et T

↳ Si la nouvelle valeur de E est inférieure à la précédente : nouvel état conservé.

↳ Si la nouvelle valeur de E est supérieure de $\Delta E > 0$ à la précédente : nouvel état conservé avec une probabilité $p = \exp\left(-\frac{\Delta E}{T}\right)$.

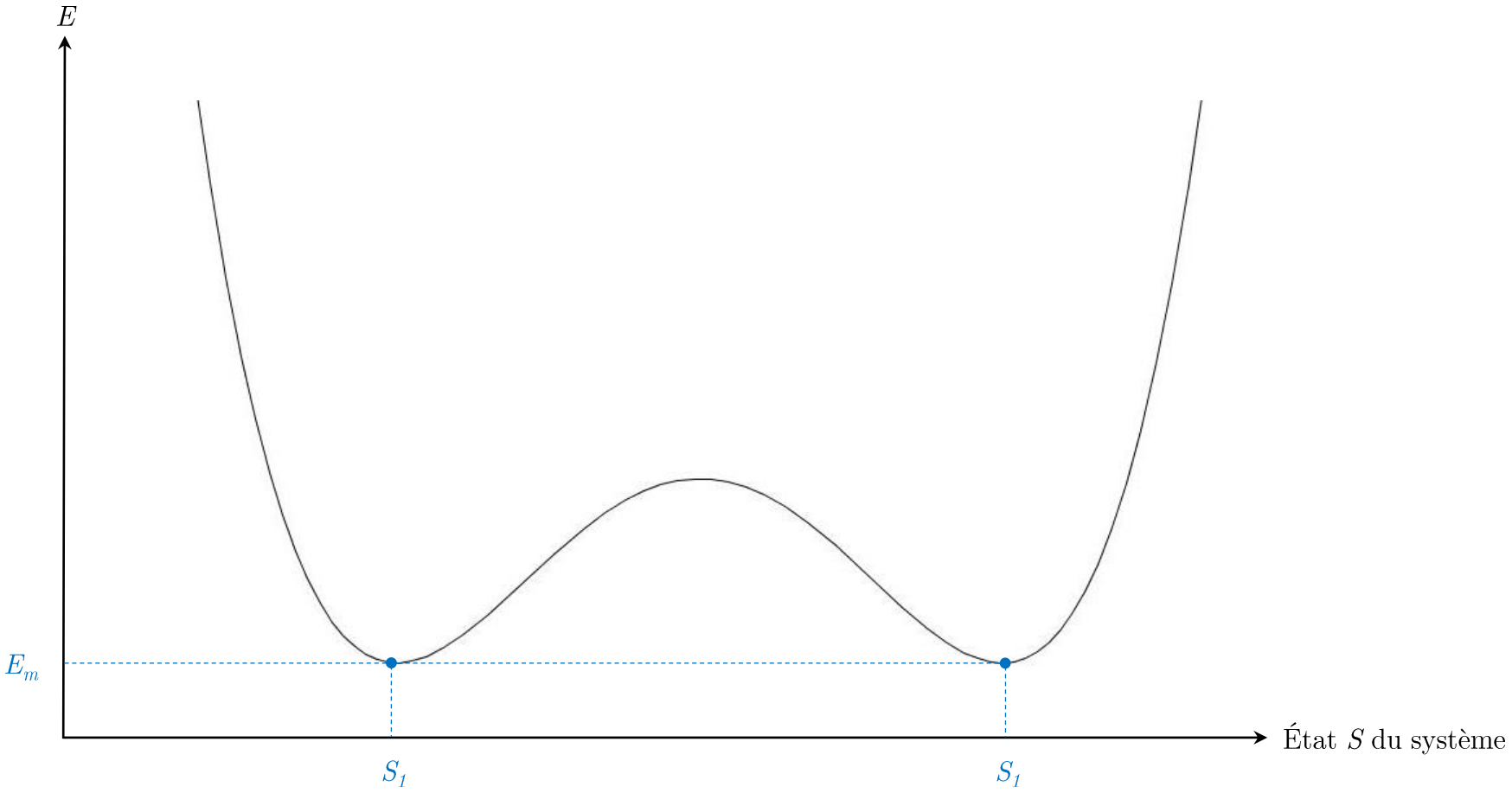
5.2 Principe du « recuit simulé »

5.2 Principe du « recuit simulé »



5.2 Principe du « recuit simulé »

5.2 Principe du « recuit simulé »



5.3 Améliorations

5.3 Améliorations

- Sortie anticipée de l'algorithme en cas de détection d'une configuration mathématiquement optimale.

5.3 Améliorations

- Sortie anticipée de l'algorithme en cas de détection d'une configuration mathématiquement optimale.

- Limitation de l'éloignement du « meilleur » minimum local rencontré.

5.3 Améliorations

- Sortie anticipée de l'algorithme en cas de détection d'une configuration mathématiquement optimale.

- Limitation de l'éloignement du « meilleur » minimum local rencontré.

- Affinement de la définition de la probabilité p pour prendre en compte le meilleur résultat obtenu jusqu'alors.

5.3 Améliorations

- Sortie anticipée de l'algorithme en cas de détection d'une configuration mathématiquement optimale.

- Limitation de l'éloignement du « meilleur » minimum local rencontré.

- Affinement de la définition de la probabilité p pour prendre en compte le meilleur résultat obtenu jusqu'alors.

- Optimisation du mode de décroissance de la fonction T.

5.4 Adaptation au *problème de Berlekamp*

5.4 Adaptation au *problème de Berlekamp*

Algorithme 1 (recuit simulé) : résolution du problème 1 (problème intermédiaire)

Fonction à minimiser	Poids de la matrice correspondant à l'état courant
Initialisation	Configuration initiale imposée par le joueur 1
« Modification élémentaire » de l'état du système	Basculement d'un interrupteur choisi au hasard

5.4 Adaptation au *problème de Berlekamp*

Algorithme 1 (recuit simulé) : résolution du problème 1 (problème intermédiaire)

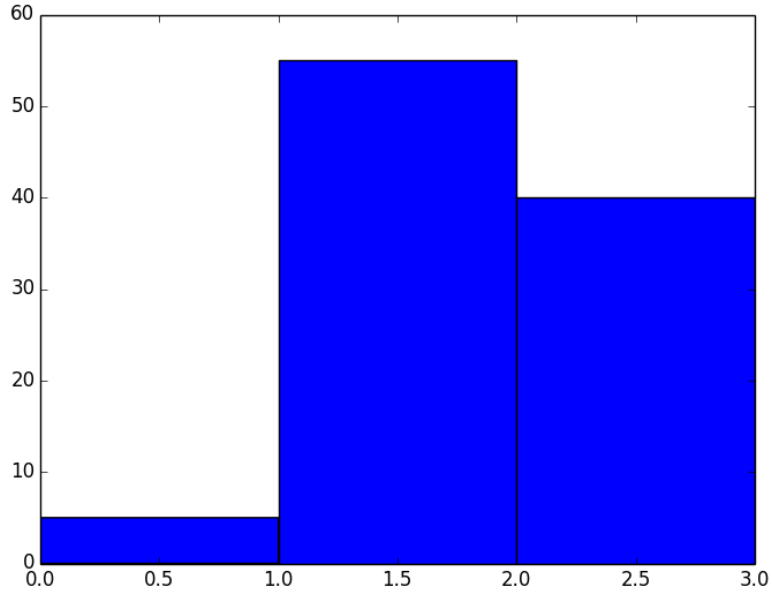
Fonction à minimiser	Poids de la matrice correspondant à l'état courant
Initialisation	Configuration initiale imposée par le joueur 1
« Modification élémentaire » de l'état du système	Basculement d'un interrupteur choisi au hasard

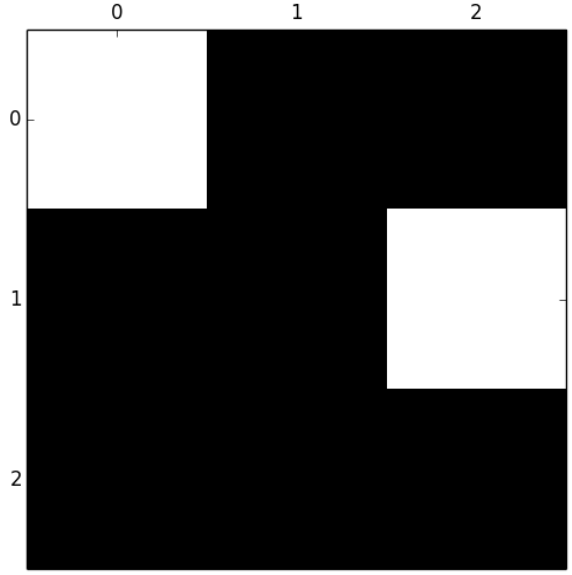
Algorithme 2 : résolution du problème 2 (*Problème de Berlekamp*)

- Tirage au sort d'une configuration de départ.
- Application de l'algorithme 1.
- Détection et mémorisation des configurations initiales optimales pour le premier joueur.

6 Résultats

6.1 Cas $n = 3$

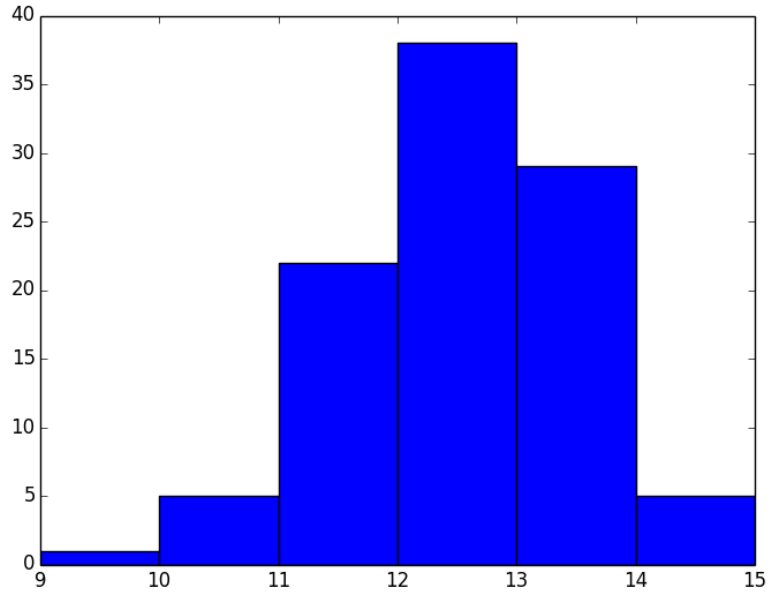


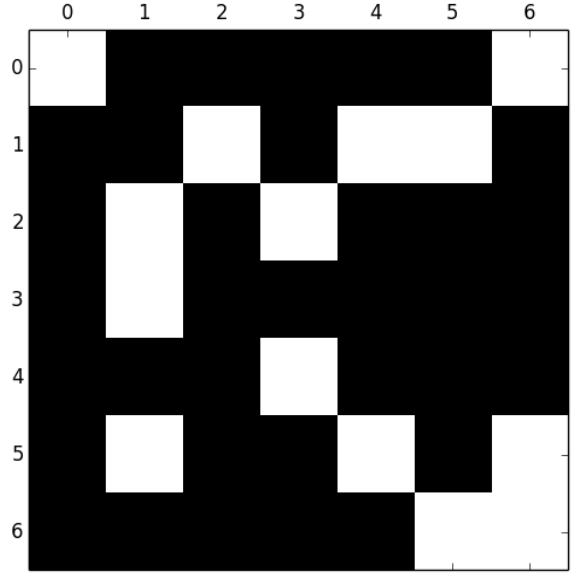


2 ampoules allumées

6 Résultats

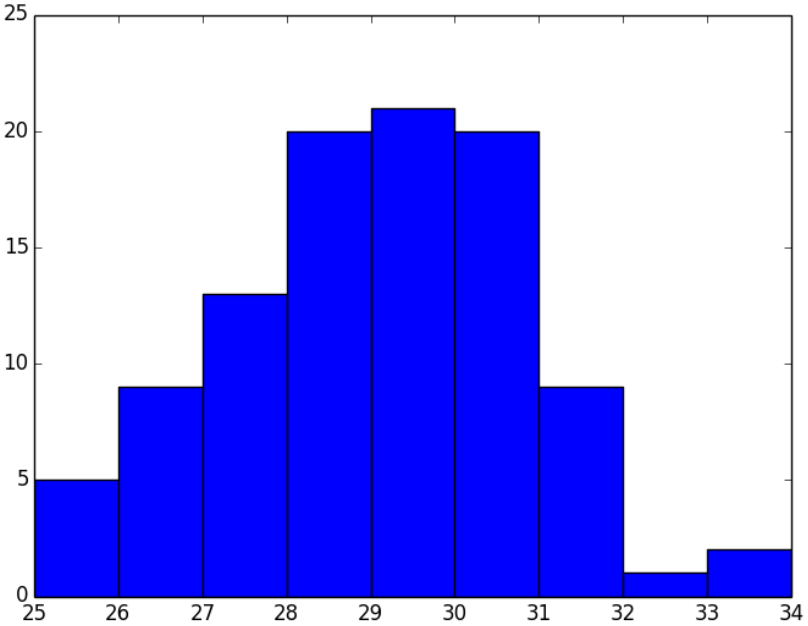
6.1 Cas $n = 7$

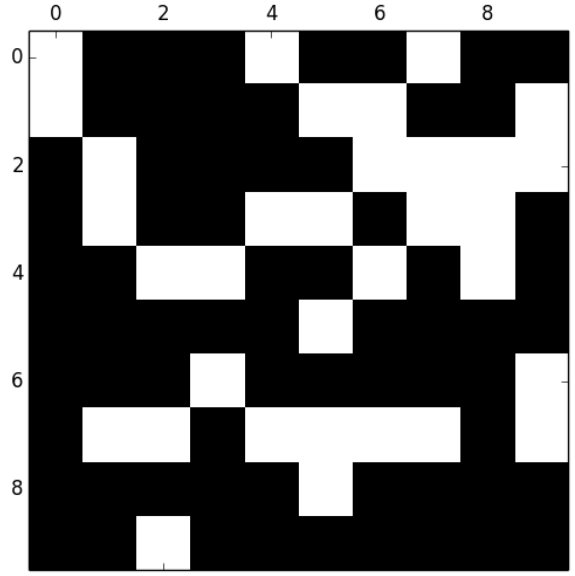




14 ampoules allumées

6.1 Cas $n = 10$

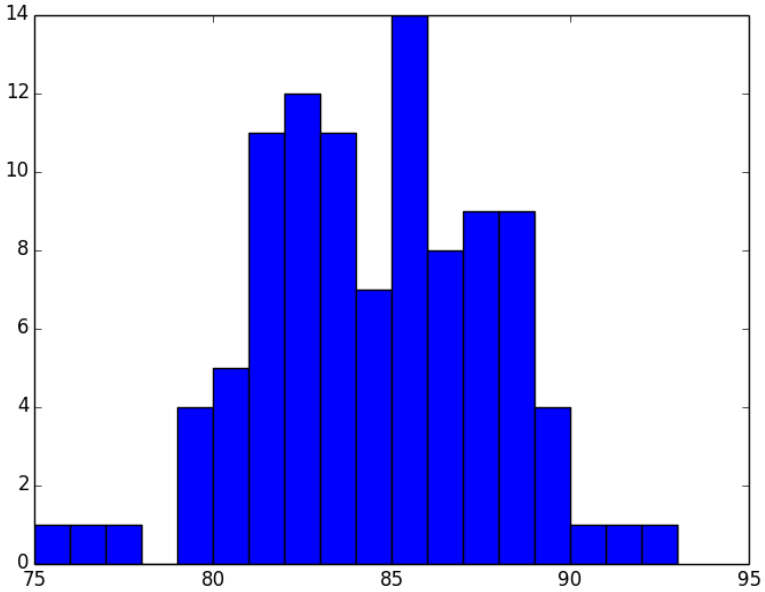


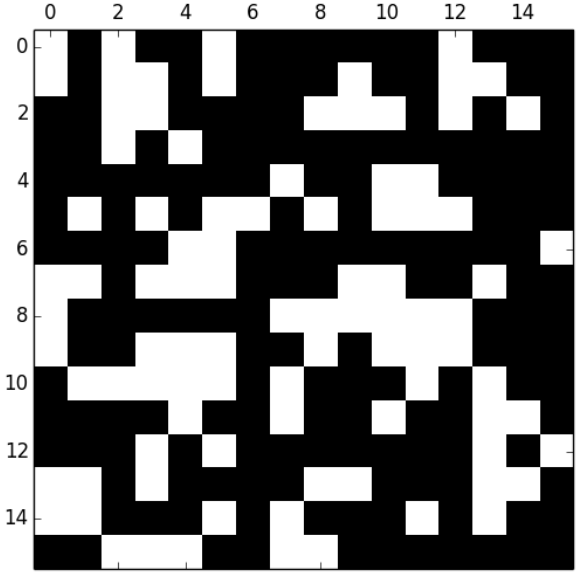


33 ampoules allumées

6 Résultats

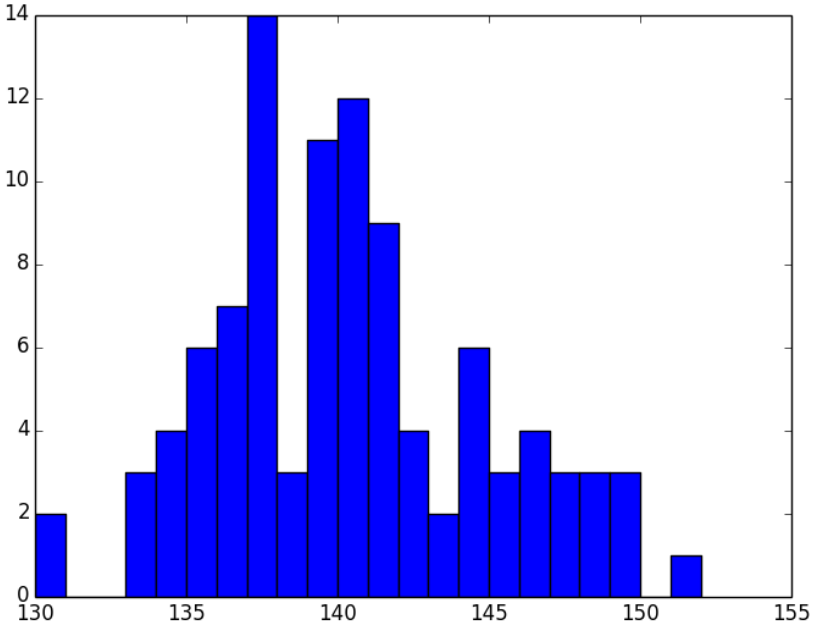
6.1 Cas n =16

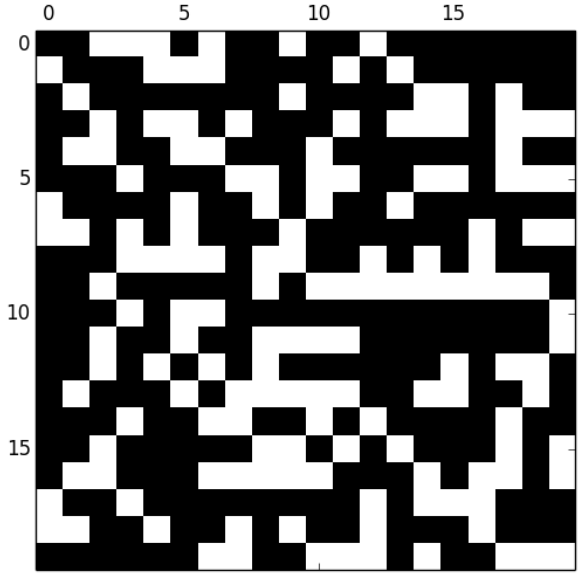




92 ampoules allumées

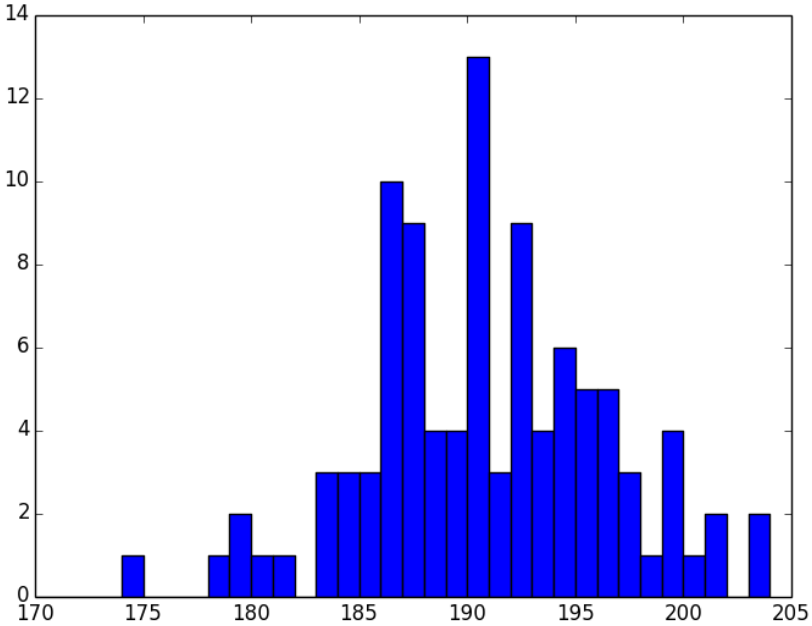
6.1 Cas $n = 20$

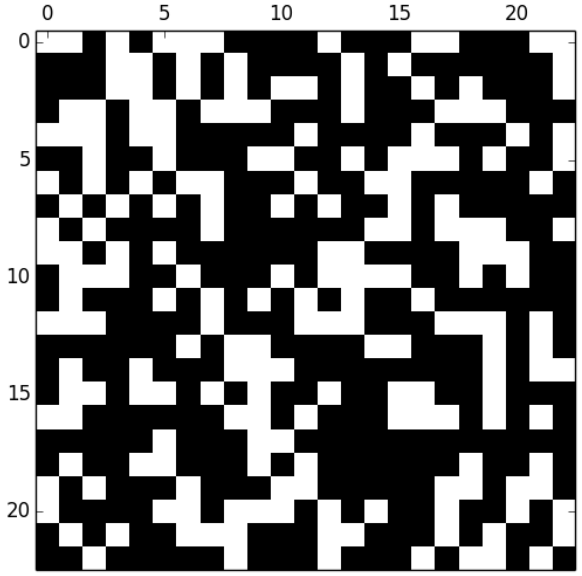




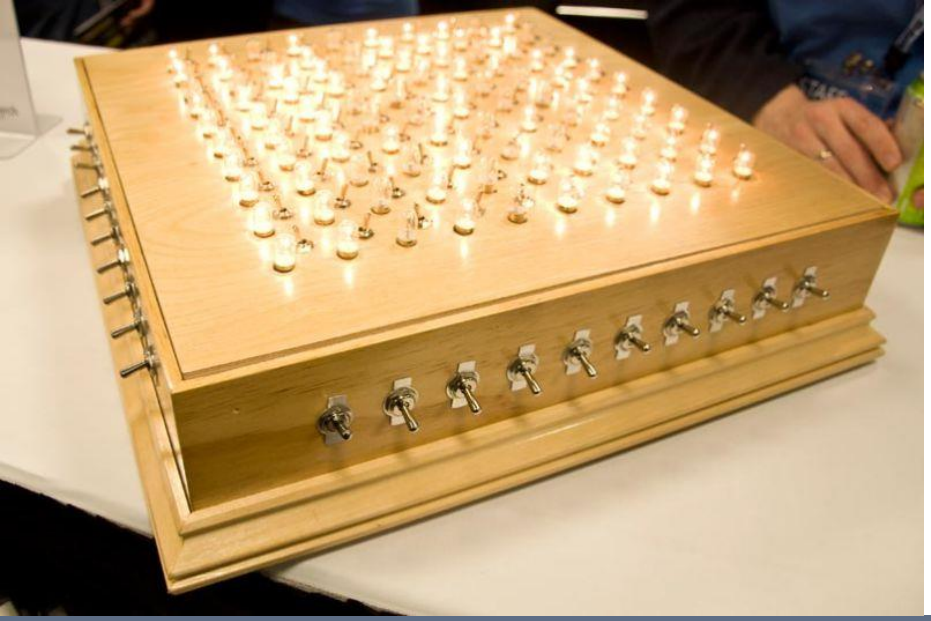
151 ampoules allumées

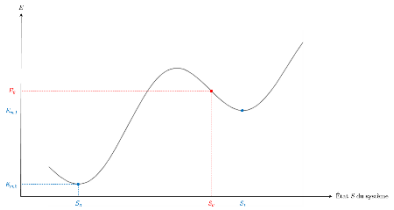
6.1 Cas $n = 23$





204 ampoules allumées





```
def Minimize(pos, data, nbsteps=10000, maxcounter=30,
    earlyabort=0, coeffA = 1.0, coeffB = 1.0, coeffC = 2.0) :

    def T(i) :
        return (1-i/nbsteps)

    coeffC *= (data.nbc + data.nbl)
    def P(Efrom, Eto, Ebest, T) :
        diff = coeffA * (Eto - Ebest) + coeffB * (Eto -
            Efrom)
        return exp(-diff / (coeffC*T))

    score = Score(pos)
    best, bestscore, counter = pos, score, 0
    lst = [ score ]

    for i in range(nbsteps) :
        set = randint(0, data.nbl + data.nbc - 1)
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Annexes

Construction du tableau standard

Lemme.

Les classes d'équivalence de $\mathcal{R}$ sont toutes de même cardinal que $\mathcal{C}$, à savoir 2^{2n-1}.

Preuve.

$$\forall\, M \in \mathcal{M}(n)\,,\quad \bar{M} = \{M + C \mid C \in \mathcal{C}\} \implies \begin{aligned} Card(\bar{M}) &= Card(\mathcal{C}) \\ &= Card(\mathbb{F}_2) \times dim(\mathcal{C}) \\ &= 2^{2n-1} \end{aligned}$$

□

Lemme.

$\mathcal{R}$ admet exactement 2^{n^2-2n+1} classes d'équivalence.

Preuve.

$$Card\big(\mathcal{M}(n)\big) = Card(\mathbb{F}_2) \times dim\big(\mathcal{M}(n)\big) = 2^{n^2}$$

Donc $\mathcal{R}$ admet exactement $\frac{Card(\mathcal{M}(n))}{Card(\mathcal{C})} = 2^{n^2-2n+1}$ classes d'équivalence.

□

Tableau standard (rappel).

$$\left(M_i + C_j \right)_{(i,j)\in[\![1,2^{n^2-2n+1}]\!]\times[\![1,2^{2n-1}]\!]}$$

Compléments de théorie des codes : décodage par tableau standard

Proposition (Invariance par translation)**.**

La distance de Hamming est invariante par translation :

$$\forall (M1, M2, M) \in \mathcal{M}(n)^3, \qquad d_H(M1, M2) = d_H(M1 + M, M2 + M)$$

Méthode (Décodage au maximum de vraisemblance par tableau standard)**.**

$$\Big(\quad M_i + C_j \quad \Big)_{(i,j) \in [\![1, 2^{n^2 - 2n + 1}]\!] \times [\![1, 2^{2n-1}]\!]}$$

$$d_H(M_i + C_j\,,\, C_j) = d_H(M_i\,, 0_{\mathcal{M}(n)}) = w_H(M_i) = \min_{C \in \mathcal{C}}(w_H(M_i + C)) = \min_{C \in \mathcal{C}}\left(d_H(M_i + C, 0_{\mathcal{M}(n)})\right) = \min_{C \in \mathcal{C}}(d_H(M_i, C))$$

Complément de théorie des codes : rayon de recouvrement

Définition (Rayon de recouvrement du code $\mathcal{C}$).

$$\rho(\mathcal{C}) = \text{ plus grande distance entre un vecteur de } \mathcal{M}(n) \text{ et un mot de } \mathcal{C}.$$

$$= \max_{M \in \mathcal{M}(n)} \left(\min_{C \in \mathcal{C}} (d(M, C)) \right)$$

Proposition (Invariance par translation)**.**

La distance de Hamming est invariante par translation :

$$\forall (M1, M2, M) \in \mathcal{M}(n)^3, \qquad d_H(M1, M2) = d_H(M1 + M, M2 + M)$$

Problème 2 (*Problème de Berlekamp*).

On recherche l'ensemble des matrices S de $\mathcal{M}(n)$ vérifiant :

$$\min_{C \in \mathcal{C}} \{w_H(S + C)\} = \max_{M \in \mathcal{M}(n)} \left\{ \min_{C \in \mathcal{C}} \{w_H(M + C)\} \right\}$$

$$= \max_{M \in \mathcal{M}(n)} \left\{ \min_{C \in \mathcal{C}} \{d_H(M + C, 0_{\mathcal{M}(n)})\} \right\}$$

$$= \max_{M \in \mathcal{M}(n)} \left\{ \min_{C \in \mathcal{C}} \{d_H(M, C)\} \right\}$$

Programme Python : définition d'une classe adaptée

```
class panel :
    def __init__(self, nbl, nbc) :
        self.nbl = nbl
        self.nbc = nbc
        self.n = nbl*nbc
        self.max = 2**self.n-1


        self.shift = 2**nbc
        self.lastlgn = self.shift-1
        self.lastcol = sum(2**(nbc*i) for i in range(nbl))

        lgn = self.lastlgn
        res = []
        for _ in range(nbl) :
            res.append(lgn)
            lgn <<= nbc
        self.lgns = list(reversed(res))

        col = self.lastcol
        res = []
        for _ in range(nbc) :
            res.append(col)
            col <<= 1
        self.cols = list(reversed(res))
```

```
self.nbones = { 0:0 }
for i in range(nbc) :
    tmp = {}
    for elem in self.nbones :
        tmp[elem] = self.nbones[elem]
        tmp[elem+(1<<i)] = self.nbones[elem] + 1
    self.nbones = tmp

dic2 = { 0:0 }
for i in range(nbl) :
    tmp = {}
    for elem in dic2 :
        tmp[elem] = dic2[elem]
        tmp[elem+(1<<(i*nbc))] = dic2[elem] + 1
    dic2 = tmp

self.nbones.update(dic2)
```

Programme Python : algorithme de résolution

```
def Minimize(pos, data, nbsteps=10000, maxcounter=30, earlyabort=0, coeffA = 1.0, coeffB = 1.0,
coeffC = 2.0) :

    # Fonction température : décroissance linéaire de 1.0 à 0.0+
    def T(i) :
        return (1-i/nbsteps)

    # Fonction P : D'autant plus proche de 0.0 que l'évolution est mauvaise et T est faible
    coeffC *= (data.nbc + data.nbl)
    def P(Efrom, Eto, Ebest, T) :
        diff = coeffA * (Eto - Ebest) + coeffB * (Eto - Efrom)
        return exp(-diff / (coeffC*T))

    # Initialisation
    score = Score(pos)
    best, bestscore, counter = pos, score, 0
    lst = [ score ]

    # On effectue nbsteps pas aléatoires
    for i in range(nbsteps) :
        # Calcul de la nouvelle position et du nouveau score
        set = randint(0, data.nbl + data.nbc - 1)
        nextscore = score + ChangeSet(pos, set, data)

        # Evolution si meilleur ou si random() < P
        if nextscore < score or random() < P(score, nextscore, bestscore, T(i)) :
            pos = SwitchSet(pos, set, data)
            score = nextscore
```

(…)

Programme Python : algorithme de résolution

(…)

```
        # Mémorisation du meilleur état
        if score < bestscore :
            best, bestscore = pos, score
            counter = 0

        # Retour en arrière : après un certain nombre de pas peu intéressants,
        # on revient sur le meilleur état observé
        if score > bestscore :
            counter += 1
            if counter > maxcounter :
                counter, pos, score = 0, best, bestscore

        # Mémorisation du score (pour tracer le résultat)
        lst.append(score)

        # Arrête la recherche si l'on a atteint la limite earlyabort
        if score <= earlyabort :
            break

    return best, bestscore, lst
```

Programme Python : algorithme de résolution

```
def Simu_1(nbl, nbc, nbtests=100) :
    data, res = panel(nbl, nbc), []

    for i in range(nbtests) :
        pos = randint(0, 2**(data.nbl*data.nbc))
        _, score, _ = Minimize(pos, data)
        res.append(score)

    hist(res, max(res)+1, (0, max(res)+1))
    show()
```

```
def Simu_3(nbl, nbc, nbtests=100) :
    maxi = 0
    configuration = 0

    data = panel(nbl, nbc)

    for i in range(nbtests) :
        pos = randint(0, 2**(data.nbl*data.nbc))
        config, score, _ = Minimize(pos, data, earlyabort = maxi)
        if max(maxi, score) == score :
            maxi = score
            configuration = config

    Disp(configuration, data)
    return maxi
```

Démonstration : $\boldsymbol{dim(\mathcal{C}) = 2n - 1}$

$\forall (i,j) \in [\![1,n]\!]^2$:

$$A_i = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & 0 \\ 1 & \ldots & 1 \\ 0 & 0 & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} (i\text{-ème ligne}) \qquad B_j = \begin{pmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & 0 & \vdots & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{pmatrix} (j\text{-ème colonne})$$

$\boldsymbol{\mathcal{B} = (A_i, B_j)_{(i,j)\in[\![1,n]\!]\times[\![1,n-1]\!]}}$ **est une base de** $\boldsymbol{\mathcal{C}}$ (et $Card(\mathcal{B}) = 2n - 1$)

↳ **Famille génératrice :**

$$B_n = \sum_{i=1}^{n} A_i + \sum_{j=1}^{n-1} B_j$$

↳ **Famille libre :**

$\forall (\alpha_i, \beta_j)_{(i,j)\in[\![1,n]\!]\times[\![1,n-1]\!]} \in (\mathbb{F}_2)^{2n-1}$,

$$\sum_{i=1}^{n} \alpha_i \cdot A_i + \sum_{j=1}^{n-1} \beta_j \cdot B_j = \begin{pmatrix} \alpha_1 + \beta_1 & \cdots & \alpha_1 + \beta_{n-1} & \alpha_1 \\ \vdots & \ddots & \vdots & \vdots \\ \alpha_n + \beta_1 & \cdots & \alpha_n + \beta_{n-1} & \alpha_n \end{pmatrix}$$

Démonstration : groupe quotient

Proposition.

L'ensemble des classes d'équivalence pour la relation $\mathcal{R}$ forme le groupe quotient ${}^{\mathcal{M}(n)}/_{\mathcal{C}}$.

Preuve.

- $(\mathcal{M}(n), +)$ est un groupe abélien (structure d'espace vectoriel).
- $(\mathcal{C}, +)$ est un sous-groupe distingué de $(\mathcal{M}(n), +)$.

- Définissons sur ${}^{\mathcal{M}(n)}/_{\mathcal{C}}$ la loi interne $+_q$ par :

$$\forall (M_1, M_2) \in \mathcal{M}(n)^2 \ , \qquad \overline{M_1} +_q \overline{M_2} = \overline{M_1 + M_2}$$

Cette loi induite sur ${}^{\mathcal{M}(n)}/_{\mathcal{C}}$ par $\mathcal{M}(n)$ est bien définie, car $\mathcal{C}$ étant un sous-groupe distingué de $\mathcal{M}(n)$, on a :

$\forall (M_1, M_2, M_1', M_2') \in \mathcal{M}(n)^4$,

$$\begin{cases} \overline{M_1} = \overline{M_1'} \\ \overline{M_2} = \overline{M_2'} \end{cases} \quad \Longrightarrow \quad \overline{M_1 + M_2} = \overline{M_1' + M_2'}$$

Alors :

- Neutre pour $+_q$: $\overline{0_{\mathcal{M}(n)}}$
- Symétrique de $\bar{M} \in \left({}^{\mathcal{M}(n)}/_{\mathcal{C}}\right)$: $\overline{-M}$
- Associativité