

Université Paris 8

Année 2022–2023

Master Mathématiques et applications
Parcours ACC (Arithmétique, Codage et Cryptologie)

Théorie de l'information

Notes de cours

Julien Lavauzelle
julien.lavauzelle@univ-paris8.fr

2 décembre 2022

Table des matières

1	Introduction	7
1.1	Systèmes de communication	7
1.2	Autres applications	8
2	Grandeurs en théorie de l'information	11
2.1	Rappels sur les probabilités discrètes	11
2.1.1	Premières définitions	11
2.1.2	Lois dérivées	13
2.2	Entropie	15
2.3	Information mutuelle	21
3	Codage de source	25
3.1	Encodage d'un alphabet	25
3.2	Codes à longueur fixe	26
3.3	Codes à longueur variable	28
3.4	Code de Huffman	34
4	Codage de canal	39
4.1	Canaux	39
4.1.1	Définitions et exemples	39
4.1.2	Capacité	41
4.2	Codes en blocs	42
4.3	Codes convolutifs	45
4.3.1	Définition et représentations	45
4.3.2	Algorithme de décodage de Viterbi	49
5	Modèles de sources stochastiques	53
5.1	Processus stochastiques	53
5.1.1	Premières définitions	53
5.1.2	Taux d'entropie et codage universel	55
5.1.3	Le cas des processus markoviens	57
5.2	Algorithmes de Lempel et Ziv	60
5.2.1	Présentation	60
5.2.2	Algorithme de Lempel–Ziv–Welch	61
5.2.3	Analyse	62
6	Autour de la compression de données	65
6.1	Compression sans perte	65
6.1.1	Un premier algorithme de compression : RLE	65
6.1.2	Autour de bzip2 : la transformée de Burrows–Wheeler	66

Quelques références utiles

Théorie de l'information et codage

[Rio07] Olivier Rioul. *Théorie de l'information et du codage*. 2007

[CT01] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2001

[DRVET18] Jean-Guillaume Dumas, Jean-Louis Roch, Sébastien Varrette, and Éric Tannier. *Théorie des codes : compression, cryptage, correction*. Dunod, 2018

[CL22] Antoine Chambert-Loir. *Théorie de l'information — trois théorèmes de Claude Shannon*. Calvage et Mounet, 2022

Pour aller plus loin sur le codage canal

[HP10] William Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2010

Théorie de l'information et apprentissage

[Mac03] David JC MacKay. *Information Theory, Inference, and Learning Algorithms*. 2003

Chapitre 1

Introduction

1.1 Systèmes de communication

La théorie des communications a pour but d'étudier les moyens de transmission de l'information à travers un canal, depuis une source jusqu'à un récepteur. Cette théorie a été introduite par Claude Shannon en 1948 dans un article précurseur [Sha48] qui pose les fondements de la théorie de l'information.

Shannon propose un modèle simple et générique dans lequel une *source* d'information produit un message. Ce message est ensuite encodé par un *codeur*, puis transite à travers un *canal* sujet à un bruit. Le message potentiellement altéré est fourni à un *décodeur* qui retourne l'information obtenue au *récepteur*. Voir Figure 1.1 pour une illustration.

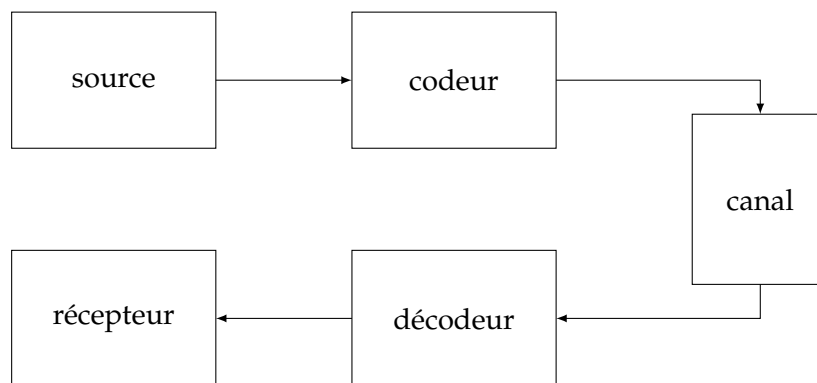


FIGURE 1.1 – Modèle générique d'un système de communication

Les codeurs et décodeurs mentionnés plus haut entrent dans le domaine du *codage de source*. Le plus souvent, ils ont pour but d'interpréter le message comme une suite numérique.

On peut raffiner le modèle présenté ci-dessus en intégrant des *codeur et décodeur de canal*. Leur objectif est de corriger de potentielles erreurs induites par bruit du canal. Ce modèle est illustré en Figure 1.2.

Enfin, dans le domaine d'application des communications, le canal est concrètement vu comme un opérateur aléatoire agissant sur des signaux continus (par exemple, une interférence entre ondes électro-magnétiques). Il faut donc transformer le signal numérique issu du codeur de canal en signal analogique continu. On ajoute donc aux schémas précédents des *modulateur et démodulateur* de données, voir Figure 1.3.

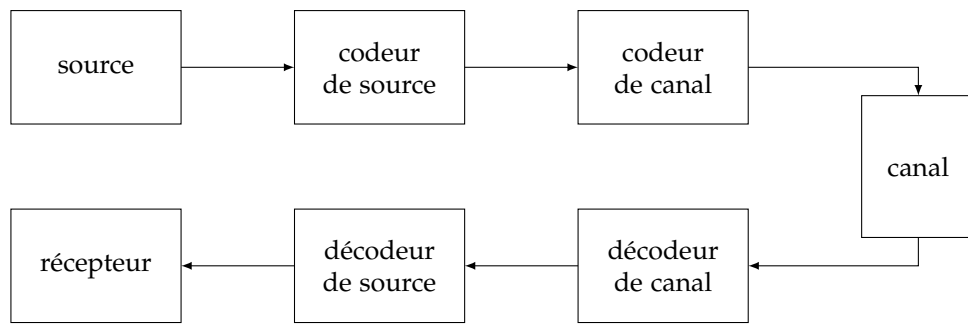


FIGURE 1.2 – Modèle d'un système de communication avec distinction des codeurs/décodeurs

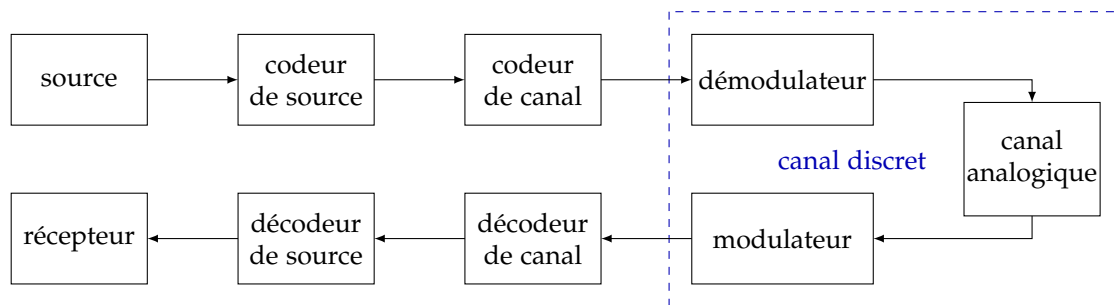


FIGURE 1.3 – Modèle raffiné d'un système de communication

La *théorie de l'information* a alors pour but de proposer un ensemble de définitions formelles et de résultats théoriques qui, d'une part permettent de modéliser, de concevoir et d'analyser de nouveaux systèmes de communication, et d'autre part d'en poser les limites fondamentales.

On s'intéressera particulièrement aux questions suivantes :

1. Comment peut-on quantifier une information, et la comparer à une autre ?
2. Comment peut-on encoder une information de manière efficace ?
3. Est-il possible de créer, détruire ou transmettre (sans perte) de l'information ?

1.2 Autres applications

Au-delà de son intérêt naturel pour l'étude des communications, la théorie de l'information trouve également des applications en cryptographie, en stockage ou dans la génération d'aléa.

Secret, cryptographie. Historiquement, Shannon [Sha48] a également introduit la théorie de l'information pour étudier la sécurité des communications *chiffrées*. L'idée présente dans la conception de la plupart des systèmes de chiffrement est la suivante : on incorpore des éléments aléatoires au message à envoyer afin de le rendre incompréhensible à toute personne n'ayant pas accès à un secret connu seulement de l'émetteur et/ou du récepteur. Dans ce contexte, la théorie de l'information permet de valider ou de contester la sécurité d'un système de chiffrement, en étudiant les corrélations entre l'information contenue dans le message, dans le secret et dans le chiffré.

Par des moyens théoriques, Shannon a notamment démontré que le chiffrement de Vernam (aussi appelé *masque jetable*, dans lequel une clé aléatoire de taille aussi grande que le message lui est combinée pour former le chiffré) est inconditionnellement sûr. Le problème de l'accès confidentiel à une information stockée à distance (*private information retrieval*) fournit une autre application moderne de la théorie de l'information en cryptographie.

Compression de données. La théorie de l'information donne aussi un cadre à la compression de données numériques. Elle en pose les limites théoriques (jusqu'à quel point puis-je compresser un fichier?) et fournit des méthodes pour approcher ou atteindre cet optimum, en fonction de la structure du fichier à compresser. En particulier, l'outil de compression `lzip`, disponible sous licence libre et utilisé par de nombreux logiciels de compression et d'archivage, repose sur l'algorithme LZMA (pour *Lempel-Ziv-Markov chain algorithm*), prouvé optimal en moyenne grâce à la théorie de l'information.

Génération et extraction d'aléa. De nombreux algorithmes (en cryptographie, intelligence artificielle, méthodes numériques, etc.) nécessitent une source de nombres aléatoires dont la distribution est contrôlée, par exemple uniforme. En pratique, il est rare d'avoir à disposition une telle source. La théorie de l'information donne des outils pour transformer une source d'aléa quelconque en source uniforme. Elle permet également de quantifier les ressources nécessaires à cette transformation, appelée extraction d'aléa.

Chapitre 2

Grandeurs en théorie de l'information

Dans ce chapitre, nous introduisons des grandeurs permettant de modéliser et de quantifier le concept d'information. Dans le sens commun, une information permet de lever une incertitude concernant un phénomène, un évènement. Cet évènement peut être vu comme la réalisation d'une variable aléatoire ; ainsi nous commencerons par rappeler certaines notions élémentaires de probabilités.

2.1 Rappels sur les probabilités discrètes

2.1.1 Premières définitions

Dans ce cours, le contexte d'application est celui des communications et de la cryptographie. Ainsi, il est naturel de considérer un ensemble d'évènements élémentaires qui est discret (dénombrable, le plus souvent fini). Cet ensemble est appelé l'univers Ω , et les éléments composés sont alors les parties de Ω . Une loi de probabilité $p : \Omega \rightarrow [0,1]$ telle que $\sum_{\omega} p(\omega) = 1$ est attachée à cet univers, formant un espace probabilisé discret (Ω, p) .

Sauf indication contraire, les espaces probabilisés, lois, distributions et variables aléatoires sont supposés discrets.

Exemple 2.1

La loi de probabilité uniforme définie sur un espace Ω fini est par définition :

$$p_{\text{unif}}(\omega) := \frac{1}{|\Omega|}, \quad \forall \omega \in \Omega.$$

Tous les évènements élémentaires ont une probabilité d'occurrence égale.

Définition 2.2

La distribution d'une loi p d'un espace probabilisé (Ω, p) est la donnée de $(p(\omega))_{\omega \in \Omega}$. Dans le cas où ω est fini, la distribution peut être vue comme un vecteur de longueur $|\Omega|$ dont les coefficients sont dans $[0, 1]$.

Par exemple, la distribution de la loi uniforme sur un univers Ω de cardinal n est $\underbrace{\left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}\right)}_{n \text{ fois}}$.

Si p est une loi de probabilité d'un espace probabilisé (Ω, p) , alors on peut prolonger p sur les évènements composés de Ω (i.e. les parties de Ω). Pour cela on définit :

$$p(A) := \sum_{\omega \in A} p(\omega), \quad \forall A \subseteq \Omega.$$

Observons alors que $p(\emptyset) = 0$ et que $p(\Omega) = 1$. On obtient alors les résultats suivants.

Proposition 2.3 (Formule d'inclusion-exclusion)

Soit (Ω, p) un espace probabilisé et A_1, A_2, \dots, A_n des parties de Ω . Alors on a :

$$p(A_1 \cup A_2) = p(A_1) + p(A_2) - p(A_1 \cap A_2).$$

Plus généralement :

$$p\left(\bigcup_{i=1}^n A_i\right) = \sum_{I \subseteq [1, n]} (-1)^{|I|} p\left(\bigcap_{i \in I} A_i\right).$$

Démonstration : Pour le premier point, on a

$$p(A_1 \cup A_2) = \sum_{\omega \in A_1 \cup A_2} p(\omega) = \sum_{\omega_1 \in A_1} p(\omega_1) + \sum_{\omega_2 \in A_2} p(\omega_2) - \sum_{\omega \in A_1 \cap A_2} p(\omega) = p(A_1) + p(A_2) - p(A_1 \cap A_2).$$

En effet, il faut prendre garde de ne pas compter deux fois les évènements élémentaires présents à la fois dans A_1 et dans A_2 .

Le second point se démontre par récurrence. ■

Le cas où les parties sont disjointes forme un corollaire important.

Corollaire 2.4

Si A et B sont deux parties disjointes de Ω , alors $p(A \cap B) = 0$ et $p(A \cup B) = p(A) + p(B)$.

Définition 2.5 (variable aléatoire)

Une variable aléatoire est une application $X : \Omega \rightarrow \mathcal{X}$, où \mathcal{X} est un ensemble discret quelconque, que l'on peut supposer fini si Ω l'est. Pour $x \in \mathcal{X}$, on note alors $\{X = x\} := \{\omega \in \Omega, X(\omega) = x\}$. La variable aléatoire est dite réelle si $\mathcal{X} \subset \mathbb{R}$.

Une variable aléatoire « transporte » donc sur \mathcal{X} une loi de probabilité originellement définie sur Ω . On appelle donc loi d'une variable aléatoire X l'application $p_X : \mathcal{X} \rightarrow [0, 1]$ définie par

$$p_X(x) := p(\{X = x\}) = \sum_{\omega | X(\omega) = x} p(\omega)$$

On peut vérifier que (\mathcal{X}, p_X) forme également un espace probabilisé discret.

La variable X est dite finie si $\mathcal{X}_0 = \{x \in \mathcal{X} \mid p_X(x) \neq 0\}$ est fini. remarquons qu'il est coutumier de restreindre l'image de X à \mathcal{X}_0 lorsqu'on en a besoin. Enfin, on dit que X est binaire si $|\mathcal{X}| = 2$.

Définition 2.6

L'espérance d'une variable aléatoire réelle $X : \Omega \rightarrow \mathcal{X} \subset \mathbb{R}$ est le nombre réel

$$\mathbb{E}(X) := \sum_{x \in \mathcal{X}} p_X(x)x.$$

La variance de X est la quantité

$$\mathbb{V}(X) := \mathbb{E}((X - \mathbb{E}(X))^2).$$

L'espérance représente donc la valeur moyenne prise par X sur Ω , pondérée par sa loi de probabilité. La variance mesure quant à elle la dispersion de X autour de sa moyenne. Notons également que la variance est toujours positive ou nulle (d'ailleurs, on appelle *écart-type* sa racine carrée positive), et qu'elle satisfait $\mathbb{V}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$.

Exemple 2.7 (Loi de Bernoulli)

Une variable aléatoire $X : \Omega \rightarrow \mathbb{R}$ suit une loi de Bernoulli de paramètre $\lambda \in [0, 1]$ si elle admet la distribution de probabilité suivante :

$$p_X(x) = \begin{cases} \lambda & \text{si } x = 1, \\ 1 - \lambda & \text{si } x = 0, \\ 0 & \text{sinon.} \end{cases}$$

Une telle variable est alors appelée *variable de Bernoulli*. Son espérance est λ et sa variance est $\lambda(1 - \lambda)$.

2.1.2 Lois dérivées

À partir de lois et de variables existantes sur un univers Ω , il est naturel d'essayer d'en dériver d'autres. Une première manière consiste à s'intéresser à un sous-ensemble $B \subseteq \Omega$, et d'observer le comportement de la loi sur B . On parle alors de loi conditionnelle à B .

Définition 2.8 (Loi conditionnelle)

Soit (Ω, p) un espace probabilisé et $B \subseteq \Omega$ un évènement tel que $p(B) \neq 0$. La probabilité conditionnelle de $\omega \in \Omega$ sachant B est

$$p(\omega | B) := \begin{cases} \frac{p(\omega)}{p(B)} & \text{si } \omega \in B \\ 0 & \text{sinon.} \end{cases}$$

On montre aisément que pour tout $B \subseteq \Omega$, l'application $p_{\cdot|B} : \omega \mapsto p(\omega | B)$ est une loi de probabilité ; autrement dit, $(\Omega, p_{\cdot|B})$ est un espace probabilisé.

Comme dans la partie précédente, on peut étendre $p(\cdot | B)$ aux parties de Ω . Définissons donc, pour $A, B \subseteq \Omega$, la quantité $p(A | B) := \sum_{\alpha \in A} p(\alpha | B)$. On déduit alors les résultats suivants.

Proposition 2.9

Soit (Ω, p) un espace probabilisé et $A, B \subseteq \Omega$. Alors,

1. on a $p(A | B) = \frac{p(A \cap B)}{p(B)}$,
2. les probabilités conditionnelles $p(B | A)$ et $p(A | B)$ sont liées par la formule de Bayes :

$$p(B | A) = p(A | B) \frac{p(B)}{p(A)}.$$

Deux évènements sont dits indépendants si la probabilité d'occurrence de l'un ne dépend pas de l'autre. Plus formellement :

Définition 2.10

Deux évènements $A, B \subseteq \Omega$ sont dits indépendants si $p(A \cap B) = p(A)p(B)$.

Remarquons que cette définition rejoint l'intuition donnée à la probabilité conditionnelle : si A et B sont indépendants, alors on a

$$p(A | B) = \frac{p(A \cap B)}{p(B)} = p(A).$$

Autrement dit, la restriction aux évènements de B ne modifie pas la probabilité d'occurrence de A .

Remarque 2.11

Si X et Y sont deux variables aléatoires sur Ω et à images respectives dans \mathcal{X} et \mathcal{Y} , alors pour tout $y \in \mathcal{Y}$ tel que $p_Y(y) \neq 0$, on peut définir une variable aléatoire conditionnelle de loi

$$p_{X|Y=y} : x \mapsto p_X(\{X = x\} | \{Y = y\}) = \frac{p_{X,Y}(x,y)}{p_Y(y)}$$

On peut aisément vérifier que $(\mathcal{X}, p_{X|Y=y})$ est un espace probabilisé, car

$$\sum_{x \in \mathcal{X}} p_{X|Y=y}(x) = \sum_{x \in \mathcal{X}} \frac{p_{X,Y}(x,y)}{p_Y(y)} = 1.$$

Par abus de langage, on notera parfois rapidement $p_{X|Y=y}(x) = p_{X|Y}(x,y)$, mais on prendra garde au fait que $X | Y$ n'est pas une variable aléatoire à proprement parler.

Il est en revanche possible de regrouper deux variables aléatoires en une seule variable. On parle alors de variable *couple* ou *conjointe* et de loi conjointe.

Définition 2.12 (Loi et variable conjointes)

Soient (Ω, p) un espace probabilisé fini, et $X : \Omega \rightarrow \mathcal{X}$ et $Y : \Omega \rightarrow \mathcal{Y}$ deux variables aléatoires. La loi conjointe de X et Y est $p_{X,Y} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ définie par

$$p_{X,Y}(x,y) := p(\{\omega \in \Omega, X(\omega) = x \text{ et } Y(\omega) = y\}).$$

La variable aléatoire $\Omega \rightarrow \mathcal{X} \times \mathcal{Y}$ ayant pour loi $p_{X,Y}$ est appelée variable conjointe de X et Y , et est notée X,Y .

En d'autres termes, la loi conjointe $p_{X,Y}$ est la loi représentant l'occurrence simultanée d'évènements associés aux variables aléatoires X et Y .

Proposition 2.13

Soit X et Y deux variables aléatoires. Alors, pour tout $x \in \mathcal{X}$ on a :

$$p_X(x) = \sum_{y \in \mathcal{Y}} p_{X,Y}(x,y) = \sum_{y \in \mathcal{Y}} p_Y(y)p_{X|Y}(x | y).$$

Exemple 2.14

Un ensemble de 50 étudiants (univers Ω) doit choisir une mineure à son entrée à l'université. Deux choix sont possibles : informatique (I) et mathématiques (M). On note $X : \omega \rightarrow \{I, M\}$ la

variable qui assigne à chaque étudiant sa mineure. Par ailleurs, on note $Y(\omega)$ la date de naissance de l'étudiant $\omega \in \Omega$, et on remarque que $\mathcal{Y} = Y(\Omega) = \{2005, 2004, 2003\}$.

On observe alors les effectifs suivants :

	$x = I$	$x = M$
$y = 2005$	0	4
$y = 2004$	20	12
$y = 2003$	7	7

Ici, la loi conjointe X, Y se calcule en divisant les effectifs de chaque groupe par le nombre total d'étudiants. À savoir :

$p_{X,Y}(x,y)$	$x = I$	$x = M$
$y = 2005$	0	0.08
$y = 2004$	0.4	0.24
$y = 2003$	0.14	0.14

Pour obtenir la loi conditionnelle $X | Y$ (loi du nombre d'étudiants par date de naissance), il faut alors effectuer le calcul $p_{X|Y}(x,y) = \frac{p_{X,Y}(x,y)}{p_Y(y)}$. On obtient alors :

$p_{X Y}(x,y)$	$x = I$	$x = M$
$y = 2005$	0	1
$y = 2004$	0.625	0.375
$y = 2003$	0.5	0.5

2.2 Entropie

On souhaite maintenant proposer une définition quantitative de l'information. Rappelons que l'on pense l'information comme une levée d'incertitude sur une variable aléatoire. Ainsi, il faut que l'information apportée par un évènement soit grande lorsque l'évènement est improbable, et faible lorsqu'il est presque certain. Pour quantifier l'information obtenue, on doit donc choisir une fonction **décroissante** de la probabilité d'occurrence de l'évènement.

Par ailleurs, il est souhaitable que l'information apportée par des évènements portant sur deux variables *indépendantes* soit la *somme* des informations apportées séparément par ces évènements. Ainsi, il est naturel de choisir pour définition de l'information une fonction **logarithmique** en la probabilité d'occurrence de l'évènement.

Finalement, on souhaite qu'un évènement provenant d'une variable binaire uniforme (exemple : tirage pile ou face) apporte exactement une unité d'information. Cela nous mène à la définition suivante.

Définition 2.15

Soit X une variable aléatoire, et $x \in \mathcal{X}$ un évènement tel que $p_X(x) \neq 0$. L'information propre de x , aussi appelée incertitude de x , est la quantité

$$I(x) = -\log_2(p_X(x)).$$

L'unité de mesure de l'information propre est le bit.

Remarque 2.16

Dans la suite, pour alléger la notation on écrira $p(x)$ au lieu de $p_X(x)$ lorsque le contexte sera clair.

Notons une nouvelle fois que l'incertitude, ou information propre, représente la quantité d'information apportée par l'occurrence de l'évènement $\{X = x\}$: si la probabilité $p(x)$ est faible, alors l'occurrence de x apporte beaucoup d'information. Au contraire, si l'évènement est certain (par exemple $p(x) = 1$), alors son occurrence n'apporte aucune information.

Pour quantifier l'information susceptible d'être apportée par une variable aléatoire, on considère la moyenne des informations portées par les réalisations de la variable. Cette valeur moyenne est appelée *entropie de Shannon*.

Définition 2.17 (Entropie de Shannon)

L'entropie de Shannon (ou plus simplement entropie) d'une variable aléatoire X de loi de probabilité p_X est :

$$H(X) := \mathbb{E}(I(X)) = - \sum_{x \in \mathcal{X}} p_X(x) \log_2(p_X(x)).$$

Son unité d'expression est le bit.

Notons que l'on prolonge en 0 la fonction $t \mapsto -t \log_2(t)$, de sorte que l'entropie soit bien définie. C'est-à-dire, on pose « $0 \log_2 0 = 0$ ». Le graphe de cette fonction est tracé en Figure 2.1.

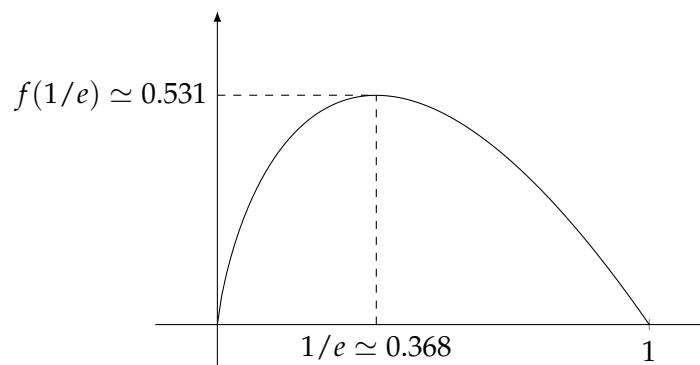


FIGURE 2.1 – La fonction $f : t \mapsto -t \log_2 t$ sur $[0,1]$.

Remarque 2.18

On peut voir l'entropie de Shannon comme une fonction de la distribution $(p_i)_{i=1..n}$, où les p_i sont les probabilités d'occurrence des $x_i \in \mathcal{X}$. Alors, la fonction entropie $h : (p_1, \dots, p_n) \mapsto - \sum_{i=1}^n p_i \log_2 p_i$ satisfait les axiomes suivants :

- A1. l'application $h : (p_1, \dots, p_n) \mapsto - \sum_{i=1}^n p_i \log_2 p_i$ est symétrique en ses variables,
- A2. l'application h est continue sur $]0,1]^n$ et peut être prolongée en sa frontière,
- A3. la suite $(h(1/n, \dots, 1/n))_{n \geq 1}$ est strictement croissante,
- A4. si (p_1, \dots, p_n) et (q_1, \dots, q_m) sont deux distributions, alors

$$h(p_1 q_1, \dots, p_1 q_m, p_2, \dots, p_n) = h(p_1, \dots, p_n) + p_1 h(q_1, \dots, q_m).$$

En réalité, la fonction d'entropie de Shannon est, à un facteur multiplicatif près, la seule fonction vérifiant les quatre axiomes précédents.

L'entropie mesure le degré d'incertitude d'une variable aléatoire. Plus l'entropie est grande, moins on a d'information *a priori* sur la valeur que prendra la variable aléatoire. Par exemple, l'entropie d'une variable aléatoire X pour laquelle l'un des évènements est certain (et par conséquent, dont les autres évènements ont probabilité d'occurrence 0) est nulle.

Exemple 2.19

L'entropie d'un lancer de pièce idéal est de 1 bit. Celle de N lancers de pièce indépendants est de N bits. L'entropie d'un lancer de dé idéal à 6 faces est $\log_2(6) \simeq 2.58$ nits. Il y a donc plus d'aléa (ou plus d'information) dans 3 lancers de pièces que dans un lancer de dé.

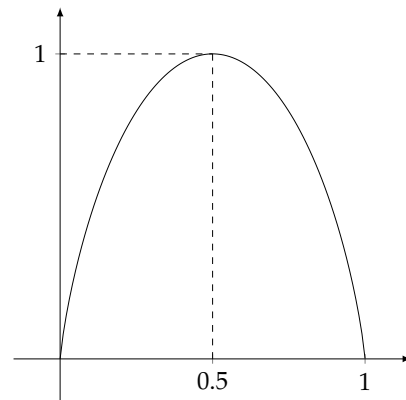
Exemple 2.20

L'entropie d'une variable de Bernoulli de paramètre λ est :

$$h_2(\lambda) := -\lambda \log_2 \lambda - (1 - \lambda) \log_2(1 - \lambda).$$

Cette fonction est communément appelée fonction d'entropie binaire, et elle est fondamentale en théorie de l'information. Un graphe de cette fonction est tracé ci-contre.

Observons que h_2 est minimale pour $\lambda = 0$ et $\lambda = 1$ (dans ces cas, la variable aléatoire prend presque sûrement la même valeur), et maximale pour $\lambda = 1/2$, auquel cas la variable de Bernoulli représente un lancer de pièce idéal.

**Proposition 2.21**

Soit $X : \Omega \rightarrow \mathcal{X}$ une variable aléatoire de loi p . Supposons que \mathcal{X} est fini de cardinal n . Alors l'entropie de Shannon de X vérifie :

$$0 \leq H(X) \leq \log_2(n).$$

La borne supérieure est atteinte si et seulement si la distribution de probabilité est uniforme sur \mathcal{X} , c'est-à-dire si $p(x_1) = \dots = p(x_n) = 1/n$.

Démonstration : Pour la première inégalité, observons que la fonction $t \mapsto -t \log_2 t$ est positive ou nulle sur l'intervalle $[0, 1]$ (vérification laissée en exercice, voir aussi Figure 2.1). Le résultat vient donc du fait que $p(x) \in [0, 1]$ pour tout $x \in \mathcal{X}$.

Pour la seconde inégalité, on utilise la concavité (voir ci-dessous) de la fonction \log_2 :

$$\sum_{x \in \mathcal{X}} p(x) \log_2 \left(\frac{1}{p(x)} \right) \leq \log_2 \left(\sum_{x \in \mathcal{X}} \frac{p(x)}{p(x)} \right) = \log_2(|\mathcal{X}|).$$

Enfin, concernant le cas d'égalité, on utilise le critère d'égalité des fonctions strictement concaves. Ce critère impose que les $p(x_i)$ soient égaux, c'est-à-dire que X suit une loi de probabilité uniforme. ■

Concavité. Dans la preuve de la Proposition 2.21, on utilise la propriété de concavité (stricte) de la fonction \log_2 . Par la suite, nous aurons besoin de cette notion à plusieurs occasions ; donnons-en une définition.

Définition 2.22

Une fonction réelle continue f , définie sur un segment $[a, b]$, est dite strictement concave si, pour tout $a \leq x < y \leq b$ et tout $\lambda \in [0, 1]$, on a

$$f(\lambda x + (1 - \lambda)y) > \lambda f(x) + (1 - \lambda)f(y).$$

Dans le cas où f est dérivable sur $]a, b[$, cela signifie que le graphe de f est en-dessous de ses tangentes en tout point de l'intervalle. Si f est dérivable deux fois, cela correspond à dire que $f''(x) < 0$ pour tout $x \in]a, b[$.

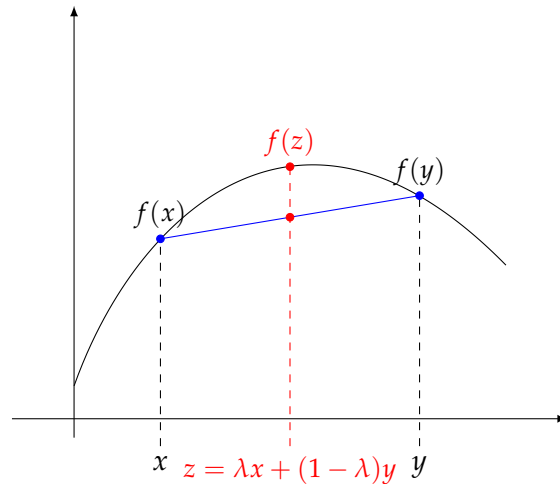


FIGURE 2.2 – Exemple de fonction concave

Dans le cas d'une fonction réelle à n variables f définie sur $\mathcal{A} := [a_1, b_1] \times \cdots \times [a_n, b_n]$, on dit que f est strictement concave si pour tous $(x_1, \dots, x_n), (y_1, \dots, y_n)$ distincts dans \mathcal{A} et pour tout $\lambda \in]0, 1[$, on a

$$f(\lambda a_1 + (1 - \lambda)b_1, \dots, \lambda a_n + (1 - \lambda)b_n) > \lambda f(a_1, \dots, a_n) + (1 - \lambda)f(b_1, \dots, b_n).$$

Lemme 2.23

Toute fonction strictement concave $f : \mathcal{A} \rightarrow \mathbb{R}$ vérifie alors les deux propriétés suivantes.

— Pour tout $m \geq 1$, pour tous points x_1, \dots, x_m de \mathcal{A} , et pour tous coefficients $\lambda_1, \dots, \lambda_m > 0$ tels que $\sum_{i=1}^m \lambda_i = 1$, on a

$$f\left(\sum_{i=1}^m \lambda_i x_i\right) \geq \sum_{i=1}^m \lambda_i f(x_i). \quad (2.1)$$

— Il y a égalité dans (2.1) si et seulement si les points sont égaux : $x_1 = \cdots = x_m$.

On a vu précédemment que la fonction \log_2 est concave. Il en est de même de la fonction d'entropie de Shannon, dans le sens précisé dans la proposition suivante.

Proposition 2.24

La fonction d'entropie de Shannon $h : (x_1, \dots, x_n) \mapsto -\sum_{i=1}^n x_i \log_2 x_i$ est strictement concave. Autrement dit, pour tout $m \geq 2$, pour tous points $(x_{i,j})_{i=1..n, j=1..m}$ tels que pour tout j , la famille $(x_{i,j})_{i=1..n}$ est une distribution de probabilité, et pour tous coefficients $\lambda_1, \dots, \lambda_m \in]0, 1[^n$ tels que $\sum_{j=1}^m \lambda_j = 1$, on a :

$$h\left(\sum_{j=1}^m \lambda_j x_{1,j}, \dots, \sum_{j=1}^m \lambda_j x_{n,j}\right) \geq \sum_{j=1}^m \lambda_j h(x_{1,j}, \dots, x_{n,j}).$$

Démonstration : D'après la discussion précédente, il suffit de démontrer le résultat pour $m = 2$. Pour cela posons $x_i = x_{i,1}$, $y_i = x_{i,2}$ et $z_i = \lambda x_i + (1 - \lambda)y_i$ pour $i \in \{1, 2\}$. Remarquons que $h(z_1, z_2) = f(z_1) + f(z_2)$ où la fonction $f : z \mapsto -z \log_2 z$ est strictement concave. Donc on a

$$h(\lambda x_1 + (1 - \lambda)y_1, \dots, \lambda x_n + (1 - \lambda)y_n) \geq \lambda f(x_1) + (1 - \lambda)f(y_1) + \lambda f(y_1) + (1 - \lambda)f(y_2).$$

Ce dernier terme est égal à $\lambda h(x_1, x_2) + (1 - \lambda)h(y_1, y_2)$, ce qui conclut la preuve. ■

Entropie conditionnelle. Intéressons-nous maintenant à la quantité d'aléa résiduelle après avoir levé d'incertitude liée à une expérience aléatoire.

Définition 2.25

Étant données deux variables aléatoires X et Y à image dans \mathcal{X} et \mathcal{Y} , et un évènement $y \in \mathcal{Y}$, on définit l'entropie conditionnelle de X sachant que l'évènement y s'est réalisé comme :

$$H(X | y) := - \sum_{x \in \mathcal{X}} p_{X|Y}(x, y) \log_2 p_{X|Y}(x, y).$$

L'incertitude de X sachant Y est alors l'espérance de $y \mapsto H(X | y)$, c'est-à-dire :

$$H(X | Y) := - \sum_{y \in \mathcal{Y}} p_Y(y) \sum_{x \in \mathcal{X}} p_{X|Y}(x, y) \log_2 p_{X|Y}(x, y) = - \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{X,Y}(x,y) \log_2 p_{X|Y}(x,y).$$

Exemple 2.26

Considérons la distribution conjointe suivante :

$p_{X,Y}(x,y)$	x_1	x_2
y_1	0	$\frac{3}{4}$
y_2	$\frac{1}{8}$	$\frac{1}{8}$

Alors, d'une part l'entropie de la variable X est

$$H(X) = -\frac{1}{8} \log_2 \frac{1}{8} - \frac{7}{8} \log_2 \frac{7}{8} = 3 - \frac{7}{8} \log_2 7 \simeq 0.544 \text{ bit}.$$

D'autre part, on observe que $H(X | Y = y_1) = 0$ bit car $p(x_1 | y_1) = 0$, et que $H(X | Y = y_2) = 1$ bit, car $p(x_1 | y_2) = p(x_2 | y_2)$. Ainsi

$$H(X | Y) = \frac{3}{4} \times 0 + \frac{1}{4} \times 1 = \frac{1}{4} \text{ bit}.$$

On note que l'incertitude de X croît lorsque $Y = y_2$, et décroît lorsque $Y = y_1$. C'est bien ce que l'on observe sur la table de $p_{X,Y}$: pour $Y = y_1$, la variable résiduelle est déterministe (toujours égale à $X = x_2$), tandis que pour $Y = y_2$, on obtient une variable résiduelle uniforme sur $\{x_1, x_2\}$.

Proposition 2.27

Soient X, Y, Z trois variables aléatoires. On a les relations suivantes :

1. (règle de chaînage) $H(X, Y) = H(Y) + H(X | Y)$,
2. $H(X | Y) \leq H(X)$, avec égalité si et seulement si X et Y sont indépendantes,
3. $H(X) \leq H(X, Y) \leq H(X) + H(Y)$, avec égalité dans la dernière inégalité si X et Y sont indépendantes,
4. $H(X, Y | Z) = H(Y | Z) + H(X | YZ)$,
5. $H(X, Y | Z) \geq H(Y | Z)$.

Avant de donner les preuves de cette proposition, nous noterons maintenant fréquemment $p_x := p_X(x)$, $p_{x,y} := p_{X,Y}(x,y)$ et $p_{x|y} := p_{X|Y}(x,y)$.

Démonstration : 1.

$$\begin{aligned}
H(X | Y) &= - \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{x,y} \log_2 p_{x|y} \\
&= - \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{x,y} (\log_2 p_{x,y} - \log_2 p_y) \\
&= - \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{x,y} \log_2 p_{x,y} + \sum_{y \in \mathcal{Y}} \log_2 p_y \underbrace{\sum_{x \in \mathcal{X}} p_{x,y}}_{=p_y} \\
&= H(X,Y) - H(Y).
\end{aligned}$$

2. La fonction $f : x \mapsto -x \log_2 x$ peut être définie sur $[0,1]$ en posant $f(0) = 0$, et elle est strictement concave sur cet intervalle. Comme $\sum_{y \in \mathcal{Y}} p_y = 1$, on obtient :

$$H(X | Y) = \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_y f(p_{x|y}) \leq \sum_{x \in \mathcal{X}} f\left(\sum_{y \in \mathcal{Y}} p_y p_{x|y}\right) = \sum_{x \in \mathcal{X}} f(p_x) = H(X). \quad (2.2)$$

Si X et Y sont indépendantes, alors on a $p_{x|y} = p_x$, puis :

$$H(X | Y) = \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_y f(p_x) = \left(\sum_{y \in \mathcal{Y}} p_y\right) \times \left(\sum_{x \in \mathcal{X}} f(p_x)\right) = 1 \times H(X) = H(X).$$

Réciproquement, le cas d'égalité dans l'équation (2.2) intervient lorsque pour tout $x \in \mathcal{X}$, on a $\sum_{y \in \mathcal{Y}} p_y f(p_{x|y}) = f(\sum_{y \in \mathcal{Y}} p_y p_{x|y})$. Fixons $x \in \mathcal{X}$. Comme $\sum_{y \in \mathcal{Y}} p_y = 1$, on peut appliquer le critère d'égalité pour les fonctions strictement concaves, qui assure alors que les $(p_{x|y})_{y \in \mathcal{Y}}$ sont tous égaux. Autrement dit, la variable aléatoire $X | Y$ ne dépend pas de Y , c'est-à-dire $H(X | Y) = H(X)$.

3. On sait que $H(Y | X) \geq 0$ d'après la Proposition 2.21. Donc, grâce au premier point de la proposition, $H(X,Y) \geq H(X)$. D'après les deux premiers points de la proposition, on a bien $H(X,Y) = H(Y) + H(X | Y) \leq H(Y) + H(X)$.
4. L'idée est d'appliquer le premier point de la proposition à différents couples de variables. Si on l'applique à la variable conjointe (X,Y,Z) et à Z , on obtient

$$H(X,Y,Z) = H(Z) + H(X,Y | Z). \quad (2.3)$$

De même, en l'appliquant respectivement aux couples $((X,Y,Z), (Y,Z))$ et $((Y,Z), Z)$, on a :

$$H(X,Y,Z) = H(Y,Z) + H(X | Y,Z) \quad \text{et} \quad H(Y,Z) = H(Z) + H(Y | Z). \quad (2.4)$$

Le résultat s'obtient alors en « retranchant » l'équation (2.3) à la somme des deux équations (2.4).

5. C'est un corollaire immédiat du quatrième point de la proposition. ■

Remarque 2.28

Il faut prendre garde au fait que l'inégalité $H(X) \geq H(X | Y)$ n'est vraie qu'en moyenne. En effet, il peut exister certaines valeurs de $y \in \mathcal{Y}$ telles que $H(X) < H(X | \{Y = y\})$, voir Exemple 2.26.

Il est naturel de penser qu'appliquer une transformation déterministe à une variable aléatoire ne peut pas faire croître sa quantité d'information. Cette intuition s'exprime dans un résultat important, appelé *principe de non-crétion d'information*.

Proposition 2.29 (Principe de non-crétion d'information)

Soit $X : \Omega \rightarrow \mathcal{X}$ une variable aléatoire, et $g : \mathcal{X} \rightarrow \mathcal{Y}$ une fonction. On définit la variable aléatoire $Y = g(X) : \Omega \rightarrow \mathcal{Y}$ par $\omega \mapsto Y(\omega) = g(X(\omega))$. Alors

$$H(Y | X) = 0.$$

Démonstration : Revenons aux définitions de p_X et $p_{X,Y}$. Soient $x \in \mathcal{X}$ et $y \in \mathcal{Y}$. Si $g(x) \neq y$, alors les événements $\{X = x\}$ et $\{Y = y\}$ sont disjoints, donc $p_{Y,X}(y, x) = 0$. Au contraire, si $g(x) = y$, alors $\{X = x\} = \{Y = y\}$ donc $p_{Y,X}(y, x) = p_X(x)$, autrement dit $p_{Y|X}(y, x) = 1$. On en déduit que

$$H(Y | X) = - \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{Y,X}(y, x) \log_2 p_{Y|X}(y, x) = 0$$

en décomposant la somme selon si $g(x)$ est égal ou non à y . ■

Corollaire 2.30

Si X est une variable aléatoire, alors

1. $H(X, g(X)) = H(X)$,
2. $H(X) \geq H(g(X))$.

Démonstration : Pour le premier point, on observe que $H(X, g(X)) = H(X) + H(g(X) | X) = H(X)$. Pour le second point, on poursuit le calcul pour obtenir $H(X) = H(X, g(X)) = H(g(X)) + H(X | g(X)) \geq H(g(X))$. ■

Proposition 2.31

Soient $X : \Omega \rightarrow \mathcal{X}$ et $Y : \Omega \rightarrow \mathcal{Y}$ deux variables aléatoires discrètes. Si $H(Y | X) = 0$, alors il existe une fonction $g : \mathcal{X} \rightarrow \mathcal{Y}$ telle que les variables aléatoires $g(X)$ et Y ont même loi.

Démonstration : Par hypothèse, $0 = H(Y | X) = - \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{x,y} \log_2 p_{y|x}$, donc tous les termes $p_{x,y} \log_2 p_{y|x}$ sont nuls. Autrement dit, pour tout $(x,y) \in \mathcal{X} \times \mathcal{Y}$, on a $p_{x,y} = 0$ ou $p_{x,y} = p_x$. Fixons $x \in \mathcal{X}$; on sait que $p_x = \sum_{y \in \mathcal{Y}} p_{x,y}$, donc il y a au plus $p_{x,y}$ non nul. On construit alors $g : \mathcal{X} \rightarrow \mathcal{Y}$ tel que $g(x)$ est l'élément $y \in \mathcal{Y}$ tel que $p_{x,y} \neq 0$. Par construction, $g(X)$ et Y ont alors la même loi. ■

Remarque 2.32

Le second point du Corollaire 2.30 exprime le fait que l'on ne peut augmenter l'entropie d'un système en lui appliquant une fonction déterministe.

Néanmoins, en cryptographie il est parfois désirable d'essayer d'augmenter artificiellement l'entropie d'une source. C'est notamment le cas lorsqu'on souhaite concevoir et utiliser un générateur pseudo-aléatoire. Comme le principe de non-crédation d'information est une barrière théorique à ce dessein, l'idée est de partir d'une petite source d'aléa $X : \Omega \rightarrow \mathcal{X}$ que l'on suppose bonne (la graine), puis d'utiliser une fonction $g : \mathcal{X} \rightarrow \mathcal{Y}$ où \mathcal{Y} est de taille bien plus grande que \mathcal{X} , et telle qu'il soit difficile de discerner un tirage de $Y : \Omega \rightarrow \mathcal{Y}$ d'un tirage de $g(X) : \Omega \rightarrow g(\mathcal{X}) \subset \mathcal{Y}$.

D'après les résultats qui précèdent, l'entropie de Y est plus forte que celle de X , donc de $g(X)$, mais comme il est supposé difficile de discerner les observations de Y de celles de $g(X)$, la fonction g semble amplifier l'incertitude de X . On parle alors d'entropie calculatoire.

Remarque 2.33

Il est possible d'interpréter l'entropie comme une borne inférieure sur le nombre moyen de questions « oui/non » à poser pour déterminer la valeur prise par une variable aléatoire. C'est une conséquence du premier théorème de Shannon que nous verrons dans un chapitre ultérieur.

2.3 Information mutuelle

Nous avons vu dans la section précédente qu'à l'aide de l'entropie conditionnelle, il est possible de quantifier l'aléa résiduel d'une variable aléatoire X après observation d'une autre variable aléatoire Y .

Il est aussi intéressant de s'interroger sur une manière de quantifier l'information que Y apporte sur X . Naturellement, on peut définir cette information comme la différence d'incertitude entre X d'une part, et d'autre part X après observation de Y . Cette grandeur est alors appelée information mutuelle entre X et Y .

Définition 2.34 (Information mutuelle)

Soient X et Y deux variables aléatoires. L'information mutuelle entre X et Y est

$$I(X; Y) := H(X) - H(X | Y).$$

En développant les calculs, on observe aussi que l'information mutuelle s'exprime comme

$$I(X; Y) = \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{X,Y}(x,y) \log_2 \frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)}. \quad (2.5)$$

Il en découle les résultats suivants.

Proposition 2.35

Soient X, Y deux variables aléatoires.

1. $I(X; Y) = H(X) + H(Y) - H(X, Y)$,
2. $I(X; Y) = I(Y; X)$ (symétrie entre les deux variables),
3. $I(X; X) = H(X)$.

Démonstration : Laissez en exercice. ■

L'information mutuelle peut également être interprétée comme un degré d'indépendance de deux variables aléatoires. Pour en venir à cette interprétation, définissons d'abord une notion de *divergence* entre distributions.

Définition 2.36

Soient $p = (p(x))_{x \in \mathcal{X}}$ et $q = (q(x))_{x \in \mathcal{X}}$ deux distributions de probabilités. La divergence de Kullback-Leibler, ou divergence KL, de p par rapport à q est :

$$D_{\text{KL}}(p \| q) = \sum_{x \in \mathcal{X}} p(x) \log_2 \left(\frac{p(x)}{q(x)} \right).$$

Par convention, on pose $0 \log_2 \frac{0}{0} = 0$, $0 \log_2 \frac{0}{1} = 0$ et $0 \log_2 \frac{1}{0} = \infty$. Ainsi, il est possible que $D_{\text{KL}}(p \| q) = \infty$.

Attention ! Si l'on peut interpréter la divergence KL comme la mesure d'un « écart » entre deux distributions, ce n'est pas une distance pour autant. En effet, elle n'est pas symétrique en p et q , et ne vérifie pas l'inégalité triangulaire.

Avant de lister des propriétés de la divergence KL, donnons un premier résultat connu sous le nom d'inégalité de Gibbs.

Proposition 2.37 (Inégalité de Gibbs)

Soit X une variable aléatoire de distribution $(p(x))_{x \in \mathcal{X}}$, et $(q(x))_{x \in \mathcal{X}}$ une distribution de support \mathcal{X} . Alors :

$$H(X) = - \sum_{x \in \mathcal{X}} p(x) \log_2 p(x) \leq - \sum_{x \in \mathcal{X}} p(x) \log_2 q(x).$$

Démonstration : Remarquons que pour tout $t \in [0, 1]$, on a $\log_2 t \leq \ln t \leq t - 1$. Notons I l'ensemble des $x \in \mathcal{X}$ tels que $p(x) \neq 0$. On obtient alors $\sum_{x \in I} q(x) \leq \sum_{x \in I} p(x) = 1$, puis

$$-\sum_{x \in I} p(x) \log_2 \left(\frac{q(x)}{p(x)} \right) \geq -\sum_{x \in I} p(x) \left(\frac{q(x)}{p(x)} - 1 \right) = -\sum_{x \in I} q(x) + \sum_{x \in I} p(x) \geq 0.$$

Comme les termes $x \in \mathcal{X} \setminus I$ ne contribuent pas dans les sommes de l'inégalité à prouver, ceci conclut la preuve. ■

Proposition 2.38

Soient p et q deux distributions de probabilités. Alors on a :

1. $D_{\text{KL}}(p \parallel q) \geq 0$,
2. $D_{\text{KL}}(p \parallel q) = 0$ si et seulement si $p = q$.

Démonstration : Le premier point correspond à l'inégalité de Gibbs. Pour le second, il est clair que si $p = q$, alors $D_{\text{KL}}(p \parallel q) = 0$. Réciproquement, par stricte concavité de la fonction logarithme, il y a égalité dans $D_{\text{KL}}(p \parallel q) \geq 0$ si pour une certaine constante c , on a $q_i = cp_i$ pour tout i . Comme les distributions somment à 1, on obtient $c = 1$ puis $p = q$. ■

Remarquons que l'équation (2.5) se réécrit comme

$$I(X; Y) = D_{\text{KL}}(p_{X,Y} \parallel p_X p_Y).$$

Autrement dit, l'information mutuelle entre deux variables aléatoires X et Y s'interprète comme une mesure d'écart entre la distribution jointe $p_{X,Y}$ et la distribution « produit », c'est-à-dire la distribution obtenue en imposant formellement que X et Y sont indépendantes.

Corollaire 2.39

Soient X, Y deux variables aléatoires. Alors

$$I(X; Y) \geq 0.$$

Chapitre 3

Codage de source

Il existe de multiples modélisations de la source d'un système de communication. Dans ce chapitre, nous nous concentrons sur les sources *discrètes* et *sans mémoire*, dans le sens où une telle source produit une séquence de *lettres* tirées dans un *alphabet* (un ensemble discret \mathcal{X}) selon une distribution $(p(x))_{x \in \mathcal{X}}$, et où les tirages de lettres sont indépendants les uns des autres.

Le *codage de source* correspond à la transformation des messages issus de la source en une suite de bits appelés *mots*. Un des enjeux majeurs du codage de source est de rendre cette transformation efficace en terme d'espace : on souhaite que la taille moyenne des mots produits par le code soit la plus petite possible, où la moyenne est prise selon la loi de probabilité de la source.

Un exemple typique de codage de source est le code Morse, utilisé en télégraphie au XIXème siècle. Ce code est constituée de deux symboles principaux : une impulsion courte « · » et une impulsion longue « – ». Le choix de l'encodage des lettres de l'alphabet dépend alors de leur fréquence d'apparition dans les textes. Par exemple, le « e » est codé par « · », tandis que le « y » est codé par « – – · – ».

3.1 Encodage d'un alphabet

Étant donné un ensemble \mathcal{X} de lettres, un *mot* x sur \mathcal{X} est une séquence de lettres de \mathcal{X} , notée $x = x_1|x_2|\dots|x_n$, ou parfois $x_1x_2\dots x_n$ lorsque le contexte le permet. On note $\ell(x)$ la longueur d'un mot x , c'est-à-dire son nombre de lettres. L'ensemble des mots de longueur finie et non-nulle est noté \mathcal{X}^+ , et on définit $\mathcal{X}^* = \mathcal{X}^+ \cup \{\epsilon\}$ où ϵ est le mot vide (de longueur 0).

Définition 3.1

Soit \mathcal{X} un alphabet. Un code (binaire) sur \mathcal{X} est une application $C : \mathcal{X} \rightarrow \{0, 1\}^+$. On dit que le code C est régulier s'il est injectif. Le codage associé à C est alors l'application :

$$\begin{aligned} C^+ : \quad \mathcal{X}^+ &\rightarrow \{0, 1\}^+ \\ x = x_1|\dots|x_n &\mapsto C^+(x) := C(x_1)|\dots|C(x_n) \end{aligned}$$

Lorsque que le code C est régulier, il est théoriquement possible de *décoder* une lettre encodée $c = C(x)$, c'est-à-dire retrouver la lettre $x \in \mathcal{X}$ à partir de c . Dès maintenant, sauf indication contraire, **on supposera donc que les codes sont réguliers.**

Néanmoins, cela ne signifie pas que l'application de codage C^+ est injective. Par exemple, sur l'alphabet $\mathcal{X} = \{a, b\}$, le code $C : a \mapsto 0, b \mapsto 00$ est bien régulier, mais il est impossible de savoir si le mot de code 000 correspond au message aaa, ab ou ba .

Définition 3.2

Une source discrète et sans mémoire est une variable aléatoire X sur un alphabet \mathcal{X} discret, que l'on étend à \mathcal{X}^* en effectuant une succession (de taille indéterminée) de tirages indépendants.

Dans ce chapitre, on utilisera plus simplement la terminologie de *source*. Par abus, on parlera également de code sur X , où X est une source sur l'alphabet \mathcal{X} .

Définition 3.3

Soit $X = (\mathcal{X}, p_X)$ une source discrète sans mémoire et C un code sur X . On note

$$\bar{\ell}(C) := \mathbb{E}_X(\ell(C)) = \sum_{x \in \mathcal{X}} p_X(x) \ell(C(x))$$

la longueur moyenne du code. L'efficacité du code est alors

$$E(C) := \frac{H(X)}{\bar{\ell}(C)}.$$

On verra par la suite que l'efficacité est un nombre réel compris entre 0 et 1, autrement dit $\bar{\ell}(C) \geq H(X)$ pour tout code sur une source X .

Un des objectifs du codage de source est de construire des codes que l'on sait décoder, et dont l'efficacité est la meilleure possible. Pour cela, nous allons distinguer deux familles des codes : l'une où la longueur de chaque lettre encodée est fixe, facilitant le décodage (voir Section 3.2); l'autre où l'on autorise des longueurs variables suivant la probabilité d'apparition de la lettre, conduisant à une meilleure efficacité (Section 3.3).

3.2 Codes à longueur fixe

Un code de longueur fixe est un code C dont tous les mots de code ont la même longueur. On peut alors parler de *la* longueur du code. Lorsque les éléments encodés sont de même longueur $n \in \mathbb{N}$, le décodage est simple : on découpe le message codé en mots de taille n , puis on applique la fonction réciproque de C sur chaque mot.

Pour des raisons d'efficacité, on souhaite construire des codes dont la longueur est la plus petite possible. Néanmoins, il existe une limite théorique à cette longueur, en fonction de la taille de l'alphabet à coder.

Proposition 3.4

Si \mathcal{X} est un alphabet de cardinal k , alors :

1. il existe un code régulier à longueur fixe de \mathcal{X} , dont la longueur est égale à $n = \lceil \log_2(k) \rceil$;
2. il n'existe pas de code régulier à longueur fixe de \mathcal{X} , de longueur strictement plus petite de $\log_2(k)$.

Démonstration : Supposons que les lettres de \mathcal{X} sont numérotées x_0, \dots, x_{k-1} . Pour l'existence, on associe à la i -ème lettre de l'alphabet, l'écriture en base 2 de l'entier i , à laquelle on ajoute des zéros pour obtenir $\lceil \log_2(k) \rceil$ bits. Par exemple, si $k = 6$, alors la lettre x_0 est codée par 000 et la lettre x_5

est codée par 101. On observe alors que le code obtenu est bien injectif (chaque entier a une unique décomposition binaire) et sa longueur est $\lceil \log_2(k) \rceil$.

Pour le résultat d'impossibilité, on note que si un code C de longueur fixe $n < \lceil \log_2(k) \rceil$ existe, alors l'image de C contient au plus $2^n < k$ éléments, ce qui contredit l'hypothèse de régularité. ■

Exemple 3.5

Si $k = 4$, alors le code représenté par le tableau suivant a longueur $\log_2(4) = 2$:

x	x_0	x_1	x_2	x_3
$C(x)$	00	01	10	11

Si $k = 5$, alors on peut construire un code de longueur fixe égale à 3, mais pas 2 (car il n'y a que 4 mots binaires de longueur 2). Un exemple de code de longueur fixe égale à 3 est :

x	x_0	x_1	x_2	x_3	x_4
$C(x)$	000	001	010	011	100

Proposition 3.6

Soit $X = (\mathcal{X}, p_X)$ une source. L'efficacité d'un code C de longueur fixe sur X vérifie $E(C) \leq 1$. De plus, si $E(C) = 1$ alors :

- $|\mathcal{X}|$ est une puissance de 2, et
- la loi p_X est la loi uniforme sur \mathcal{X} .

Démonstration : Notons n la longueur de C . D'après la Proposition 2.21, on a

$$E(C) = \frac{H(X)}{n} \leq \frac{\log_2(|\mathcal{X}|)}{n}.$$

Comme $n \geq \log_2(|\mathcal{X}|)$ d'après la Proposition 3.4, on obtient $E(C) \leq 1$.

Par ailleurs, on observe que si $E(C) = 1$, alors nécessairement $H(X) = \log_2(|\mathcal{X}|)$ et $n = \log_2(|\mathcal{X}|)$. La première condition implique que X suit une loi uniforme (voir encore la Proposition 2.21), tandis que la seconde impose que $|\mathcal{X}|$ soit une puissance de 2. ■

Exemple 3.7

Considérons le code suivant pour l'alphabet de 26 lettres $\{a, b, \dots, z\}$:

lettre	mot de code	lettre	mot de code	lettre	mot de code
a	00000	j	01001	s	10010
b	00001	k	01010	t	10011
c	00010	l	01011	u	10100
d	00011	m	01100	v	10101
e	00100	n	01101	w	10110
f	00101	o	01110	x	10111
g	00110	p	01111	y	11000
h	00111	q	10000	z	11001
i	01000	r	10001		

Le code a longueur $5 > \log_2(26) \simeq 4.70$: il n'est pas optimal car on voit que certains mots de 5 bits ne correspondent à aucune lettre (par exemple, 11111). Pour un tirage uniforme sur l'alphabet, on obtient une efficacité de 0.94.

Les codes à longueur fixes sont donc très simples à appréhender, mais ne permettent d'obtenir une efficacité maximale que dans certains cas très particuliers. Pire : pour des distributions très déséquilibrées, leur efficacité peut tendre vers 0 (voir exercices). Il est donc naturel de s'intéresser maintenant aux codes à longueur variables.

3.3 Codes à longueur variable

Contrairement aux codes à longueur fixe, les codes à longueur variable permettent d'obtenir des codes très efficaces pour une source de distribution arbitraire.

Néanmoins, le décodage d'un code à longueur variable semble moins aisé : comment découper une séquence de lettres codées pour ensuite être capable de décoder chaque lettre séparément ?

Exemple 3.8

Considérons le code

<i>lettre</i>	<i>mot de code</i>		<i>lettre</i>	<i>mot de code</i>		<i>lettre</i>	<i>mot de code</i>
<i>a</i>	0		<i>b</i>	01		<i>c</i>	10

Le message codé 0010 correspond-il à « aac » ou à « aba » ?

Définition 3.9

On dit qu'un code C est *uniquement décodable* si son codage associé C^+ est injectif.

Définition 3.10

Un code vérifie la condition du préfixe si aucun mot de code n'est le début (= préfixe) d'un autre mot de code. Autrement dit, si $c_1 = c_2|x$ où c_1 et c_2 sont deux mots de code, alors $c_1 = c_2$ et x est le mot vide. On dit alors que le code est *préfixe* ou *instantané*.

Les codes préfixes ont l'avantage de former une famille importante de codes uniquement décodables que l'on sait facilement distinguer.

Exemple 3.11

Soit C le code sur l'alphabet $\{a, b, c, d, e, f\}$ défini par

$$C(a) = 01, \quad C(b) = 11, \quad C(c) = 10, \quad C(d) = 001, \quad C(e) = 0001, \quad C(f) = 0000.$$

On vérifie aisément que le code est préfixe. On peut également se convaincre qu'il est *uniquement décodable*, et la proposition suivante nous en fournit une preuve formelle.

Proposition 3.12

Tout code préfixe est *uniquement décodable*.

Démonstration : Soit C un code préfixe et considérons $x, y \in \mathcal{X}^+$ tels que $C^+(x) = C^+(y)$. Il faut montrer que $x = y$. On peut écrire

$$C^+(x) = C(x_1)|\dots|C(x_n) \quad \text{et} \quad C^+(y) = C(y_1)|\dots|C(y_m)$$

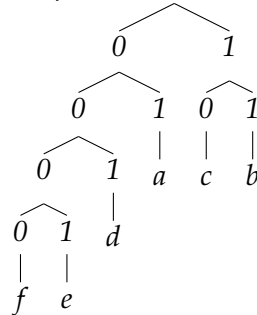
où $x = x_1|\dots|x_n$ et $y = y_1|\dots|y_m$. Supposons, par exemple, que $C(x_1)$ est plus long que $C(y_1)$. Alors, $C(y_1)$ est un préfixe de $C(x_1)$, donc $C(x_1) = C(y_1)$, ce qui implique que $x_1 = y_1$. En procédant itérativement, on montre alors que $n = m$ et $x_i = y_i$ pour tout $i \in \{1, \dots, n\}$. ■

Attention, la réciproque est fautive : tout code uniquement décodable n'est pas préfixe ! Par exemple, le code sur $\{a,b\}$ défini par $a \mapsto 1$ et $b \mapsto 10$ est uniquement décodable (pour décoder, on coupe avant chaque 1), mais il n'est pas préfixe.

Arbre binaire d'un code. À tout code binaire préfixe, on peut associer un *arbre binaire*. Dans cet arbre, toute feuille porte l'étiquette d'un message, tout nœud interne correspond à un bit, et le mot de code associé à une lettre se lit comme l'unique chemin qui mène de la racine à la feuille.

Exemple 3.13

Reprenons le code de l'Exemple 3.11. Sa représentation sous forme d'arbre binaire est la suivante :



Pour décoder un mot de code $x_1 | \dots | x_n$, on part de la racine et à chaque nœud interne, on choisit le descendant suivant la valeur de x_i . Par exemple, pour le mot 001, on part d'abord sur la gauche, de nouveau sur la gauche, puis sur la droite, pour aboutir à la lettre d . On a en effet $C(d) = 001$.

Remarque 3.14

Soit \mathcal{T} l'arbre binaire correspondant à un code binaire préfixe sur $X = (\mathcal{X}, p_X)$.

1. L'arbre \mathcal{T} a $k = |\mathcal{X}|$ feuilles.
2. La hauteur de \mathcal{T} correspond à la longueur maximale d'un mot du code.
3. La longueur moyenne du code est la longueur moyenne des chemins menant de la racine aux feuilles, pondérée par les probabilités d'occurrence des lettres.

Dans le cas des codes à longueur fixe, on a démontré une borne simple sur la longueur minimale d'un code (Proposition 3.4). Pour des codes à longueur variable, on peut également se questionner sur les séquences de longueurs possibles. L'inégalité de Kraft fournit un premier critère, dans le cas particulier des codes préfixes.

Theorème 3.15 (Inégalité de Kraft)

Soient n_1, \dots, n_k des entiers positifs. Les deux assertions suivantes sont équivalentes.

1. Il existe un code préfixe à k mots de longueurs respectives n_1, \dots, n_k .
2. Les entiers n_1, \dots, n_k vérifient

$$\sum_{i=1}^k 2^{-n_i} \leq 1.$$

Pour démontrer ce théorème, on utilise un résultat auxiliaire portant sur les longueurs des chemins dans un arbre binaire. D'abord, définissons un *arbre parfait* comme un arbre binaire dont toutes les feuilles sont à égale distance de la racine. Il est clair qu'un arbre parfait dont les feuilles sont à distance d de la racine possède 2^d feuilles.

Lemme 3.16

Soit \mathcal{T} un arbre binaire à k feuilles, dont on note d_1, \dots, d_k les longueurs des chemins menant de la racine aux feuilles. Soit $d = \max\{d_1, \dots, d_k\}$ la hauteur de l'arbre. Alors l'inégalité

$$2^d \geq \sum_{i=1}^k 2^{d-d_i}$$

est vérifiée, avec égalité si et seulement si tous les nœuds internes de l'arbre ont 2 descendants (on dit que l'arbre est localement complet).

Démonstration du lemme : Construisons un nouvel arbre à partir de \mathcal{T} . Pour chaque feuille de \mathcal{T} , on remplace la feuille par un sous-arbre parfait de hauteur $d - d_i$ (notons que si $d = d_i$, cette modification est inerte). Alors le nombre de feuilles du nouvel arbre est la somme du nombre de feuilles des k sous-arbres parfaits ajoutés, c'est-à-dire

$$\sum_{i=1}^k 2^{d-d_i}.$$

Par ailleurs, ce nouvel arbre est de hauteur d , donc contient moins de feuilles que l'arbre parfait de même hauteur. On obtient donc $2^d \geq \sum_{i=1}^k 2^{d-d_i}$. Remarquons qu'il y a égalité dans l'équation précédente si et seulement si l'arbre construit est parfait. Un arbre parfait a des nœuds internes à 2 descendants, et le procédé de construction n'élimine pas les nœuds à 1 descendant. Ainsi, le cas d'égalité se produit si et seulement si l'arbre d'origine n'a aucun nœud à 1 descendant. ■

Démonstration de l'inégalité de Kraft : ($1 \Rightarrow 2$) Posons $n = \max\{n_1, \dots, n_k\}$ et considérons l'arbre binaire associé au code. D'après le lemme précédent, si un code est préfixe, alors ses longueurs vérifient $\sum_{i=1}^k 2^{n-n_i} \leq 2^n$, et l'on obtient l'inégalité escomptée en divisant de part et d'autre par 2^n .

($2 \Rightarrow 1$) Pour la réciproque, on construit les mots itérativement, par ordre croissant de longueur. Il est facile de construire le premier ($0 \dots 0$ de longueur n_1). Supposons qu'on ait construit $i - 1$ mots c_1, \dots, c_{i-1} de longueurs $n_1 \leq \dots \leq n_{i-1}$ vérifiant la condition du préfixe, et considérons l'arbre binaire associé à ces mots. Pour construire le i -ème mot, on montre qu'il existe un nœud interne de l'arbre ayant exactement 1 descendant. Si ce n'était pas le cas, l'arbre serait localement complet, et d'après le Lemme 3.16, on aurait alors $2^{n_{i-1}} = \sum_{j=1}^{i-1} 2^{n_{i-1}-n_j}$. En multipliant de part et d'autre par $2^{-n_{i-1}}$, on observe alors une contradiction, car $1 \geq \sum_{j=1}^{i-1} 2^{-n_j} > \sum_{j=1}^{i-1} 2^{-n_j}$. ■

Le théorème de MacMillan est une extension de l'inégalité de Kraft pour les codes uniquement décodables.

Théorème 3.17 (Théorème de MacMillan)

Soient n_1, \dots, n_k des entiers positifs. Les deux assertions suivantes sont équivalentes.

1. Il existe un code uniquement décodable à k mots de longueurs respectives n_1, \dots, n_k .
2. Les entiers n_1, \dots, n_k vérifient

$$\sum_{i=1}^k 2^{-n_i} \leq 1.$$

Démonstration : Le sens ($2 \Rightarrow 1$) est un corollaire du Théorème 3.15, car un code préfixe est uniquement décodable. Il reste donc à établir le sens ($1 \Rightarrow 2$).

Considérons un code uniquement décodable C de longueurs $n_1 \leq \dots \leq n_k$, et pour un entier $m \geq 1$, notons

$$C^m := \{c = (c_1 | \dots | c_m) \text{ tel que } c_i \in C\}$$

l'ensemble des séquences binaires obtenues par concaténation d'exactly m mots de code quelconques. Alors, comme le code est uniquement décodable, à chaque élément de C^m on peut associer un unique m -uplet de mots de C , donc

$$\left(\sum_{i=1}^k 2^{-n_i} \right)^m = \sum_{c_1 \in C} 2^{-\ell(c_1)} \times \dots \times \sum_{c_m \in C} 2^{-\ell(c_m)} = \sum_{(c_1, \dots, c_m) \in C \times \dots \times C} 2^{-(\ell(c_1) + \dots + \ell(c_m))} = \sum_{c \in C^m} 2^{-\ell(c)}$$

Par ailleurs, les mots de C^m ont pour longueur maximale $m \cdot n_k$, et pour chaque $i \in \{1, \dots, m \cdot n_k\}$, on sait qu'il existe au plus 2^i mots de longueur i . Par conséquent, en regroupant la somme par longueur

on obtient

$$\sum_{c \in C^m} 2^{-\ell(c)} \leq \sum_{i=1}^{m \cdot n_k} 2^i 2^{-i} = m \cdot n_k.$$

Puis :

$$\sum_{i=1}^k 2^{-n_i} \leq (m \cdot n_k)^{1/m}, \quad \forall m \geq 1.$$

En passant à la limite pour $m \rightarrow \infty$, on trouve l'inégalité escomptée. ■

Définition 3.18

Un code est complet si la longueur de ses mots vérifie $\sum_{i=1}^k 2^{-n_i} = 1$.

Exemple 3.19

Le code défini sur $\mathcal{X} = \{a, b, c\}$ par $C(a) = 00$, $C(b) = 01$ et $C(c) = 1$ est complet. Celui de l'Exemple 3.11 l'est aussi.

Remarquons que, dans la preuve de l'inégalité de Kraft, nous avons établi le résultat suivant.

Corollaire 3.20

Un code préfixe est complet si et seulement si les nœuds de son arbre binaire ont 0 ou 2 descendants.

Après avoir déterminé quelles séquences de longueurs sont possibles pour un code uniquement décodable, on peut ensuite chercher à borner sa longueur moyenne.

Proposition 3.21

Soit X une variable aléatoire sur \mathcal{X} . Soit C un code sur \mathcal{X} uniquement décodable. Alors, la longueur moyenne de C vérifie :

$$H(X) \leq \mathbb{E}(\ell(C)).$$

Démonstration : Pour $\mathcal{X} = \{x_1, \dots, x_k\}$, notons $c_i = C(x_i)$ et $n_i = \ell(c_i)$ pour tout $i \in \{1, \dots, k\}$. Notons également $p = (p_1, \dots, p_k)$ la distribution de probabilité de X . L'idée est de construire une autre distribution de probabilité en fonction des longueurs des mots de codes. Soit $Q = \sum_{i=1}^k 2^{-n_i}$, on pose alors

$$q_i := \frac{2^{-n_i}}{Q}.$$

La divergence-KL entre les distributions p et q est positive ou nulle, donc on obtient

$$\sum_{i=1}^k p_i \log_2 \frac{p_i}{q_i} \geq 0.$$

En développant les calculs, on aboutit alors à

$$H(X) = - \sum_{i=1}^k p_i \log_2 p_i \leq - \sum_{i=1}^k p_i \log_2 q_i = \sum_{i=1}^k p_i n_i - \sum_{i=1}^k p_i \log_2 \frac{1}{Q} = \mathbb{E}(\ell(C)) + \log_2 Q \leq \mathbb{E}(\ell(C)),$$

car $Q \leq 1$ grâce au théorème de MacMillan. ■

Proposition 3.22

Soit X une variable aléatoire sur \mathcal{X} . Il existe un code sur \mathcal{X} uniquement décodable de longueur moyenne :

$$\mathbb{E}(\ell(C)) < H(X) + 1.$$

Démonstration : Soit $\mathcal{X} = \{x_1, \dots, x_k\}$ et notons $p = (p_1, \dots, p_k)$. Grâce à l'inégalité de Kraft, il suffit de construire une séquence de longueurs n_1, \dots, n_k qui vérifient

$$\sum_{i=1}^k 2^{-n_i} \leq 1 \quad \text{et} \quad \sum_{i=1}^k p_i n_i < 1 + \sum_{i=1}^k p_i \log_2 \frac{1}{p_i}.$$

Notons que les valeurs $\log_2 \frac{1}{p_i}$ pourraient satisfaire ces contraintes, mais elles ne sont pas nécessairement entières. On peut essayer de choisir pour n_i un entier suffisamment proche de $\log_2 \frac{1}{p_i}$ pour ne pas contredire la seconde contrainte, et légèrement supérieur à celui-ci afin de satisfaire la première. On pose donc

$$n_i := \left\lceil \log_2 \frac{1}{p_i} \right\rceil,$$

de telle sorte que $\log_2 \frac{1}{p_i} \leq n_i < \log_2 \frac{1}{p_i} + 1$. Il s'ensuit que

$$\sum_{i=1}^k 2^{-n_i} \leq \sum_{i=1}^k 2^{\log_2 p_i} = 1$$

et

$$\sum_{i=1}^k p_i n_i < \sum_{i=1}^k p_i \left(1 + \log_2 \frac{1}{p_i} \right) = \sum_{i=1}^k p_i + \sum_{i=1}^k p_i \log_2 \frac{1}{p_i} = 1 + H(X).$$

■

Définition 3.23

La preuve de la Proposition 3.22 fait intervenir l'inégalité de Kraft, dans laquelle on construit explicitement un code préfixe à partir de sa séquence de longueurs. Le code obtenu à partir des longueurs $n_i = \lceil -\log_2 p_i \rceil$ est appelé code de Shannon–Fano.

Exemple 3.24

Soit X une source de distribution $(p_1, p_2, p_3, p_4, p_5) = (0.6, 0.25, 0.06, 0.06, 0.03)$. L'entropie de la source X est :

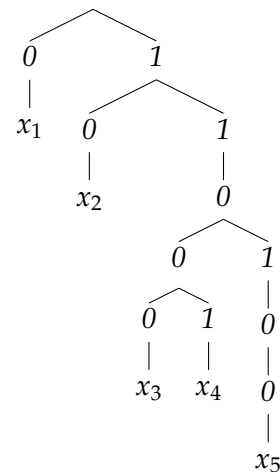
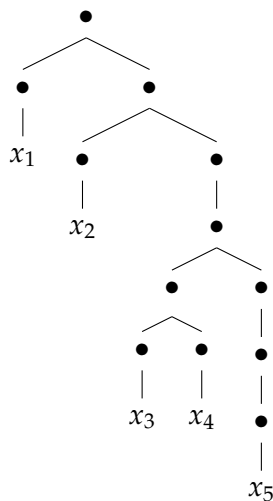
$$H(X) = - \sum_{i=1}^5 p_i \log p_i \simeq 1.58$$

Pour obtenir le code de Shannon–Fano, on calcule d'abord la séquence des longueurs $n_i = \lceil -\log p_i \rceil$ associées aux probabilités d'occurrence p_i :

p_i	0.6	0.25	0.06	0.06	0.03
n_i	1	2	5	5	6

Puis on associe à chaque lettre un mot de la longueur associée. Pour cela, on peut éventuellement construire un arbre :

Et ensuite on assigne un bit à chaque nœud interne différent de la racine :



Un code de Shannon–Fano est alors :

i	1	2	3	4	5
c_i	0	10	1100	1101	11100

Sa longueur moyenne est

$$\bar{\ell}(C) = \sum_{i=1}^5 p_i \lceil -\log p_i \rceil \simeq 1.58$$

et se retrouve bien dans l'intervalle $[H(X); H(X) + 1[$.

3.4 Code de Huffman

Dans la section précédente, l'inégalité de Kraft et le théorème de MacMillan nous a donné un critère pour évaluer l'existence de codes préfixes et/ou uniquement décodables, en fonction des longueurs présumées des mots du code.

On voudrait maintenant construire effectivement des codes dont la longueur moyenne des mots est la plus faible possible, étant donnée une certaine distribution de probabilité sur l'apparition des lettres à coder. Définissons cette notion de minimalité.

Définition 3.25

Étant donnée une source X , on dit qu'un code uniquement décodable C est optimal pour X si, pour tout code uniquement décodable C' , on a $\bar{\ell}(C) \leq \bar{\ell}(C')$.

Nous avons vu précédemment que la méthode donnée en preuve de la Proposition 3.22 construit un code (le code de Shannon–Fano) dont la longueur moyenne est distante d'au plus 1 de la longueur minimale que l'on peut espérer atteindre.

Présentons maintenant l'algorithme de Huffman qui permet de construire un code préfixe optimal. Cet algorithme prend en entrée la distribution de probabilité $p = (p_1, \dots, p_k)$ et produit l'arbre binaire d'un code préfixe. Sa description est donnée dans l'Algorithme 1.

Algorithme 1 : Algorithme de Huffman

Entrée : un entier k , et la distribution p_1, \dots, p_k de X

Sortie : l'arbre binaire d'un code préfixe sur \mathcal{X}

- 1 **Si** $k = 2$
 - 2 **Retourner** l'arbre à deux feuilles p_1, p_2 .
 - 3 Trouver les deux plus petites valeurs p_i et p_j parmi (p_1, \dots, p_k) .
 - 4 Construire une nouvelle distribution $q = (q_1, \dots, q_{k-1})$ formée des $(p_m)_{m \neq i, j}$ et de $p' = p_i + p_j$.
 - 5 Calculer l'arbre $T = \text{Huffman}(k - 1, q)$.
 - 6 Remplacer dans T la feuille p' de T par le sous-arbre $\text{Huffman}(2, (p_i, p_j))$ formé des deux feuilles p_i et p_j , puis **retourner** l'arbre obtenu.
-

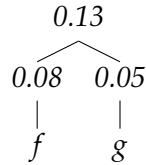
Avant de démontrer que le code produit par l'Algorithme 1 est préfixe et optimal, présentons un exemple d'exécution.

Exemple 3.26

Considérons la distribution de probabilité suivante sur $\mathcal{X} = \{a, b, c, d, e, f, g\}$:

x	a	b	c	d	e	f	g
$p(x)$	0,30	0,25	0,12	0,10	0,10	0,08	0,05

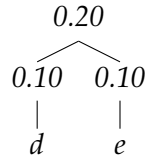
Les lettres *f* et *g* sont les moins probables. On construit donc l'arbre



et on définit une nouvelle distribution

<i>x</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f ou g</i>
<i>p(x)</i>	0,30	0,25	0,12	0,10	0,10	0,13

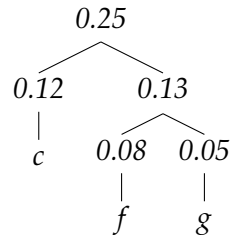
Pour cette nouvelle distribution, les événements les moins probables sont *d* et *e*. On construit donc



et on a la nouvelle distribution

<i>x</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d ou e</i>	<i>f ou g</i>
<i>p(x)</i>	0,30	0,25	0,12	0,20	0,13

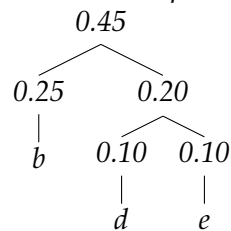
Les deux événements les moins probables sont ensuite *c* et « *f ou g* ». Cela nous mène à construire l'arbre suivant (dans lequel on a remplacé « *f ou g* » par le sous-arbre correspondant, ce qui diffère légèrement de la présentation récursive de l'algorithme) :



et la distribution devient

<i>x</i>	<i>a</i>	<i>b</i>	<i>d ou e</i>	<i>c ou f ou g</i>
<i>p(x)</i>	0,30	0,25	0,20	0,25

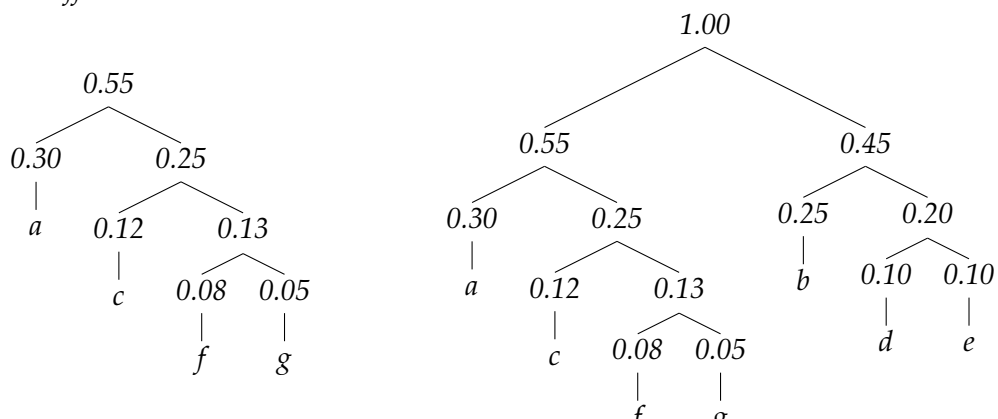
Puis, on peut choisir de manière équivalente les événements *b* et « *c ou f ou g* » à coupler avec « *d ou e* » pour construire l'arbre suivant. Choisissons la première option :



On modifie donc la distribution en

<i>x</i>	<i>a</i>	<i>b ou d ou e</i>	<i>c ou f ou g</i>
<i>p(x)</i>	0,30	0,45	0,25

Les deux dernières étapes donnent alors les arbres suivants, le dernier étant l'arbre binaire associé au code de Huffman :



Après un calcul, on observe que la distance moyenne des feuilles à la racine est 2.58, tandis que l'entropie de la variable aléatoire est d'environ 2.56. On est donc très proche de la limite théorique ! Avec le codage de Shannon–Fano, on aurait obtenu une longueur moyenne de 2.95.

On appelle code du Huffman un code produit par l'algorithme de Huffman. Démontrons maintenant le théorème attendu.

Theorème 3.27

Le code de Huffman est préfixe et optimal, c'est-à-dire que pour toute source discrète sans mémoire X , le code de Huffman C associé à X est de longueur moyenne minimale parmi tous les codes uniquement décodables sur X .

Pour établir ce théorème, on démontre une série de lemmes intermédiaires.

Lemme 3.28

Si A est un code préfixe optimal pour X , alors l'arbre binaire de A est localement complet.

Démonstration : Supposons que l'arbre de A ne soit pas localement complet. Alors il existe un nœud interne i de cet arbre qui n'a qu'un seul descendant. Considérons le sous-arbre T dont la racine est l'unique descendant de i . Si l'on remplace i par T_i , on obtient l'arbre d'un code sur X . De plus, ce code a une longueur moyenne strictement plus petite que celle de A , puisque qu'on a raccourci les longueurs de tous les mots représentés par des feuilles de T . C'est en contradiction avec l'optimalité de A . ■

Lemme 3.29

Si A est un code optimal, alors les symboles les moins fréquents codés par A sont de longueur maximale parmi les mots de A .

Démonstration : Supposons qu'il existe deux mots $A(x)$ et $A(y)$ tels que $\ell(A(x)) > \ell(A(y))$ et $p(x) > p(y)$. Alors, le code A' tel que $A'(x) = A(y)$, $A'(y) = A(x)$ et $A'(z) = A(z)$ pour tout autre symbole z , a une longueur moyenne strictement plus petite que celle de A , ce qui en contredit l'optimalité. ■

Lemme 3.30

Si T est l'arbre binaire d'un code préfixe, alors échanger des feuilles de même niveau sur l'arbre ne modifie pas la longueur moyenne de l'arbre.

Démonstration : C'est clair. ■

Lemme 3.31

Considérons deux sources X et X' , définies respectivement sur $\mathcal{X} = \{x_1, \dots, x_m\}$ et $\mathcal{X}' = \mathcal{X} \cup \{x_{m+1}\}$. On suppose que la loi p' de X' satisfait $p'(x_1) \geq p(x_2) \geq \dots \geq p'(x_{m+1})$, et que la loi p vérifie $p'(x_i) = p(x_i)$ pour tout $i \in \{1, \dots, m-1\}$ et $p'(x_m) + p'(x_{m+1}) = p(x_m)$. Soit C un code de longueur moyenne minimale pour X . Alors, le code C' pour X' obtenu à partir de C par le procédé suivant :

$$C'(x_i) = C(x_i) \text{ si } i \leq m-1, \quad C'(x_m) = C(x_m)|0, \quad \text{et} \quad C'(x_{m+1}) = C(x_m)|1,$$

est de longueur moyenne minimale pour X' .

Démonstration : Supposons qu'il existe un code A' tel que $\bar{\ell}(A') < \bar{\ell}(C')$. On peut d'abord supposer que A' est préfixe, car d'après l'inégalité de Kraft et le théorème de MacMillan, pour tout code il existe un code préfixe partageant la même séquence de longueurs. On peut également supposer que dans A' , les encodages des deux symboles les moins fréquents de X' diffèrent seulement sur leur dernier bit, d'après les Lemmes 3.28, 3.29 et 3.30. On peut donc écrire $A'(x_m) = a|0$ et $A'(x_{m+1}) = a|1$, pour un certain mot a .

Construisons alors le code A sur X à partir de A' par le procédé inverse de celui qui mène de C' à C :

$$A(x_i) = A'(x_i) \quad \text{pour } i \leq m-1 \quad \text{et} \quad A(x_m) = a.$$

Notons alors que les longueurs moyennes de C et C' diffèrent de

$$\begin{aligned} \bar{\ell}(C') - \bar{\ell}(C) &= \sum_{i=1}^{m+1} p'(x_i) \ell(C'(x_i)) - \sum_{i=1}^m p(x_i) \ell(C(x_i)) \\ &= p'(x_m) \ell(C'(x_m)) + p'(x_{m+1}) \ell(C'(x_{m+1})) - p(x_m) \ell(C(x_m)) \\ &= (p'(x_m) + p'(x_{m+1}))(1 + \ell(C(x_m))) - p(x_m) \ell(C(x_m)) \\ &= p(x_m). \end{aligned}$$

De manière similaire, on a aussi $\bar{\ell}(A') - \bar{\ell}(A) = p(x_m)$. Par conséquent, $\bar{\ell}(A) = \bar{\ell}(A') + \bar{\ell}(C) - \bar{\ell}(C') < \bar{\ell}(C)$ ce qui contredit l'optimalité de C . ■

Démonstration du Théorème 3.27 : Le résultat se montre par récurrence sur le nombre de symboles de la source. Il est clair que pour $k = 2$ éléments, le code obtenu par l'algorithme d'Huffman est optimal. Fixons maintenant $k \geq 3$, ainsi qu'une source X' à k éléments, et supposons que l'algorithme d'Huffman produit un code optimal pour toute source sur un alphabet contenant moins de $k - 1$ éléments.

On remarque que, si p est la distribution de X' , alors l'algorithme d'Huffman construit q , la distribution de probabilité de la source X donnée dans l'énoncé du Lemme 3.31. Par hypothèse de récurrence, l'appel Huffman($q, k - 1$) construit un code optimal C pour la source X . Grâce au Lemme 3.31, on déduit que la dernière étape de l'algorithme préserve l'optimalité du nouveau code ainsi construit sur X' . Ceci conclut la preuve par récurrence. ■

Chapitre 4

Codage de canal

Ce chapitre a pour but de présenter un modèle simple de canal bruité, ainsi que certaines techniques de codage permettant d'y transmettre de l'information sans perte. On supposera donc ici que les canaux sont (i) *discrets* et (ii) *sans mémoire*. Autrement dit, (i) les entrées et sorties sont choisies dans des ensembles discrets (le plus souvent finis), et (ii) les variables aléatoires qui décrivent les perturbations sur les symboles transitant dans le canal sont supposées indépendantes.

Dans ce cadre, nous donnerons un résultat majeur dû à Shannon, qui montre qu'il existe une limite théorique et atteignable sur la quantité maximale d'information qu'il est possible de transmettre à travers un tel canal.

4.1 Canaux

4.1.1 Définitions et exemples

Pour modéliser un canal, on se donne deux variables aléatoires X et Y partageant un même espace probabilisé. La variable X représente l'entrée du canal, tandis que la variable Y modélise sa sortie. On représente alors le canal par une loi de transition entre X et Y , donnée par la loi conditionnelle $p_{Y|X}$. Cette loi représente explicitement la probabilité que $y \in \mathcal{Y}$ apparaisse en sortie de canal alors que $x \in \mathcal{X}$ y a été introduit.

Formellement, on obtient alors la définition suivante.

Définition 4.1

Un canal discret sans mémoire est la donnée :

1. d'un ensemble discret \mathcal{X} , les symboles à l'entrée du canal
2. d'un ensemble discret \mathcal{Y} , les symboles en sortie de canal
3. d'une loi de transition $P : \mathcal{X} \times \mathcal{Y} \rightarrow [0,1]$ telle que :

$$\forall x \in \mathcal{X}, \sum_{y \in \mathcal{Y}} P(x, y) = 1.$$

Par commodité, on notera parfois $p(y | x) := P(x, y)$.

On suppose dans ce cours que les variables aléatoires d'entrée et de sortie prennent leurs valeurs dans des ensembles finis. Précisément, notons $\mathcal{X} = \{x_1, \dots, x_m\}$ et $\mathcal{Y} = \{y_1, \dots, y_n\}$ les domaines de définition de ces variables. On peut alors représenter le canal par une matrice

réelle de taille $(n \times m)$:

$$\Pi = \begin{pmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & p_{i,j} & \vdots \\ p_{m,1} & \cdots & p_{m,n} \end{pmatrix},$$

où

$$p_{i,j} = p_{Y|X}(y_i | x_j).$$

Cette matrice est appelée *matrice de transition* du canal. Son entrée $p_{i,j}$ représente donc la probabilité d'obtenir y_i en sortie du canal lorsque x_j a été envoyé dans le canal.

Notons que toutes éléments de la matrice sont positifs ou nuls. Par ailleurs, la somme des éléments d'un même colonne est égale à 1. Si $m = n$, une matrice vérifiant ces propriétés est appelée *matrice stochastique*.

Définition 4.2

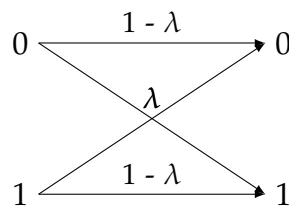
Le canal binaire symétrique (BSC pour binary symmetric channel) est un canal à entrée et sortie binaires, paramétré par $\lambda \in]0, \frac{1}{2}[$, et défini par la matrice de transition suivante :

$$\Pi = \begin{pmatrix} 1 - \lambda & \lambda \\ \lambda & 1 - \lambda \end{pmatrix}.$$

Autrement dit, si les entrée et sortie sont respectivement X et Y , on a :

$$\begin{aligned} \mathbb{P}(Y = 0 | X = 0) &= 1 - \lambda & \mathbb{P}(Y = 1 | X = 0) &= \lambda \\ \mathbb{P}(Y = 0 | X = 1) &= \lambda & \mathbb{P}(Y = 1 | X = 1) &= 1 - \lambda. \end{aligned}$$

On représente parfois les canaux par un *diagramme de transition*. C'est un graphe dont les transitions sont illustrées par des arêtes allant d'une valeur d'entrée à une valeur de sortie, et étiquetées par la probabilité d'occurrence de cette transition. Pour le cas du canal binaire symétrique, on obtient le diagramme suivant.

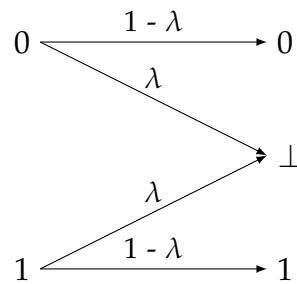


Définition 4.3

Le canal à effacement est un canal à entrée binaire et sortie dans $\{0, 1, \perp\}$, où \perp symbolise un effacement. Le canal est paramétré par $\lambda \in]0, 1[$ et défini par la matrice de transition suivante :

$$\Pi = \begin{pmatrix} 1 - \lambda & 0 \\ \lambda & \lambda \\ 0 & 1 - \lambda \end{pmatrix}.$$

Le diagramme de transition du canal est effacement est :



4.1.2 Capacité

À partir de la donnée du canal, c'est-à-dire de sa loi de transition $(x, y) \mapsto p(y | x)$, on peut s'intéresser à quantité d'information résiduelle après transmission de messages dans le canal. Comme on l'a vu au Chapitre 2, cette quantité est l'information mutuelle entre X et Y :

$$I(X; Y) = H(X) - H(X | Y).$$

On peut alors interpréter cette relation comme la différence entre $H(X)$, la quantité d'information donnée en entrée du canal, et $H(X | Y)$, la perte d'information due au bruit du canal.

Remarquons également que, dans la définition d'un canal, n'est connue que la loi de transition de X vers Y . Autrement dit, le comportement du canal dépend de la loi X en entrée. La capacité d'un canal est alors la plus grande quantité d'information que l'on **peut** éventuellement transmettre à travers le canal. Autrement dit, parmi toutes les entrées potentielles du canal, c'est la plus grande information mutuelle atteignable entre la sortie et l'entrée.

Définition 4.4

Soit C un canal d'entrée \mathcal{X} et de sortie \mathcal{Y} donné par sa loi de transition $P : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$. La capacité du canal C est la valeur maximale, pour toutes les variables aléatoires X définies sur \mathcal{X} , de l'information mutuelle $I(X; Y)$ où la variable aléatoire Y sur \mathcal{Y} est définie selon la loi de transition. Autrement dit,

$$\text{Cap}(C) := \max_X I(X; Y).$$

La capacité s'exprime en bits par unité de temps.

Un canal à petite capacité signifie donc que, pour toute source en entrée, on ne pourra pas transmettre beaucoup d'information dans le canal. En revanche, une grande capacité assure qu'il **existe** une source en entrée telle que le canal transmet beaucoup d'information sur cette source.

Remarque 4.5

On a $0 \leq \text{Cap}(C) \leq \log_2(|\mathcal{X}|)$. On ne peut donc pas transmettre plus de $\log_2(|\mathcal{X}|)$ bits d'information par unité de temps à travers un canal.

En toute généralité, le calcul de la capacité d'un canal semble difficile. Si l'entrée a taille m , alors il faut maximiser une quantité selon tous les m -uplets d'éléments de $[0, 1]$ dont la somme est égale à 1. Néanmoins, dans le cas du canal binaire symétrique, ce calcul est possible.

Proposition 4.6

La capacité du canal binaire symétrique de paramètre $\lambda > 0$ est

$$\text{Cap}(\text{BSC}_\lambda) = 1 - h(\lambda),$$

où $h(\lambda) := \lambda \log_2 \frac{1}{\lambda} + (1 - \lambda) \log_2 \frac{1}{1-\lambda}$ est l'entropie binaire de paramètre λ .

Démonstration : Soit X une variable aléatoire binaire donnée par sa loi p_X et Π la matrice du canal binaire symétrique de paramètre λ , i.e.

$$\Pi = \begin{pmatrix} 1 - \lambda & \lambda \\ \lambda & 1 - \lambda \end{pmatrix}.$$

Si Y représente la sortie du canal, alors on a $p_{XY}(x,y) = p_X(x)p_{Y|X}(y,x) = p_X(x)\Pi_{x,y}$. Puis :

$$H(Y | X) = - \sum_{x=0}^1 \sum_{y=0}^1 p_X(x)\Pi_{x,y} \log_2(\Pi_{x,y}) = \sum_{x=0}^1 p_X(x) \left(- \sum_{y=0}^1 \Pi_{x,y} \log_2 \Pi_{x,y} \right)$$

Notons alors que $-\sum_{y=0}^1 \Pi_{x,y} \log_2 \Pi_{x,y} = h(\lambda)$ ne dépend pas de x pour le canal binaire symétrique. On a donc

$$H(Y | X) = \sum_{x=0}^1 p_X(x)h(\lambda) = h(\lambda).$$

Par conséquent, pour maximiser la quantité $I(X;Y) = H(Y) - H(Y | X)$, il suffit de maximiser $H(Y)$. On sait que pour toute variable aléatoire binaire Y , l'entropie $H(Y) \leq 1$. Ainsi, la capacité du canal binaire symétrique est plus petite que $1 - h(\lambda)$.

Pour obtenir le sens opposé de l'inégalité, on analyse le cas d'égalité dans $H(Y) \leq 1$; on a vu dans un chapitre précédent que c'est le cas si et seulement si Y suit une loi uniforme. Il suffit finalement d'observer qu'on obtient une sortie uniforme en choisissant X uniforme. ■

Proposition 4.7

La capacité du canal à effacement de paramètre $\lambda > 0$ est $1 - \lambda$.

Démonstration : En exercice. ■

4.2 Codes en blocs

Le but du codage de canal est d'ajouter de la redondance à l'information en entrée du canal, afin de pouvoir corriger d'éventuelles erreurs en sortie de canal. Une façon de procéder est de considérer les symboles à transmettre par séries de taille fixe, appelées *blocs*.

Définition 4.8

Un code en bloc d'un canal d'entrée \mathcal{X} est une application injective de $\mathcal{X}^k \rightarrow \mathcal{X}^n$. L'entier n est appelé la longueur du code. La quantité $\log_2(|\mathcal{X}|) k/n$ est appelée rendement du code.

Exemple 4.9

Le code de répétition binaire, de longueur n , est l'application $C : \{0,1\} \rightarrow \{0,1\}^n$ définie par

$$C(x) = \underbrace{(x, \dots, x)}_{\leftarrow n \rightarrow}.$$

Son rendement est donc $1/n$.

Exemple 4.10

Le code de parité de longueur n sur le corps fini \mathbb{F}_q est l'application $C : \mathbb{F}_q^{n-1} \rightarrow \mathbb{F}_q^n$ définie par :

$$C(x_1, \dots, x_{n-1}) = \left(x_1, \dots, x_{n-1}, - \sum_{i=1}^{n-1} x_i \right).$$

| Son rendement est $\log(q)^{\frac{n-1}{n}}$.

Définition 4.11

Étant donné un canal C d'entrée \mathcal{X} , de sortie \mathcal{Y} et de loi de transition P , un algorithme de décodage pour le canal C est une procédure (éventuellement probabiliste) qui prend en entrée une séquence d'éléments de \mathcal{Y} et qui retourne, ou bien une erreur, ou bien une séquence d'éléments de \mathcal{X} .

Dans la suite, on considère des algorithmes de décodage dont les séquences de symboles en entrée de l'algorithme sont de longueur fixe, de même que les séquences d'éléments en sortie.

Définition 4.12

Soit Dec un algorithme de décodage sur un canal $(\mathcal{X}, \mathcal{Y}, \Pi)$. On note X et Y les variables aléatoires d'entrée et de sortie du canal. Alors, le taux d'erreur résiduel de Dec est défini comme :

$$\text{Err}_X(\text{Dec}) = \mathbb{E}(\text{Dec}(Y) \neq X).$$

Précisons que l'espérance est calculée suivant l'aléa de X , du canal et de l'algorithme de décodage.

Dans la majorité des cas, on suppose que X suit une loi uniforme. On obtient alors

$$\text{Err}_X(\text{Dec}) = \frac{1}{|\mathcal{X}|} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \Pi_{x,y} \cdot p(\text{Dec}(y) \neq x).$$

En théorie des codes, étant donné un canal on cherche à concevoir des codes avec les caractéristiques suivantes :

1. le code a un bon rendement, afin de pouvoir transmettre un maximum d'information pour une taille de message donnée,
2. le code est équipé d'un algorithme de décodage ayant un taux d'erreur résiduel faible (si possible très proche de zéro),
3. la fonction d'encodage peut être évaluée rapidement,
4. l'exécution de l'algorithme de décodage est rapide.

Dans un théorème célèbre, Shannon donne des limites théoriques aux deux premières caractéristiques. Dans un premier temps, il montre qu'il est possible de construire des codes de rendement arbitrairement proche de la capacité du canal, et pour lesquels il existe un algorithme de décodage de taux d'erreur arbitrairement proche de zéro. Dans un second temps, il démontre que toute séquence de codes équipés d'algorithmes de décodage à taux d'erreur arbitrairement faible, ont un rendement majoré de la capacité du canal.

Theorème 4.13 (Théorème de Shannon pour les canaux bruités sans mémoire)

Fixons un canal discret sans mémoire de capacité κ .

- Pour tout $R < C$, il existe une suite de codes en bloc C_n de longueur n , de rendement R_n , et admettant un algorithme de décodage de taux d'erreur résiduel E_n , telle que :

$$\lim_{n \rightarrow \infty} R_n \geq R \quad \text{et} \quad \lim_{n \rightarrow \infty} E_n = 0.$$

- S'il existe une suite de codes en bloc C_n de longueur n admettant un algorithme de décodage de taux d'erreur résiduel $E_n \rightarrow 0$, alors leur rendement R_n vérifie

$$\limsup R_n \leq \kappa.$$

La démonstration de ce théorème n'entre pas dans le cadre de ce cours. Néanmoins, il est intéressant de noter que dans le cas du canal binaire symétrique, on obtient une séquence de codes asymptotiquement optimale avec des codes et algorithmes « simples ». À savoir :

- les codes sont des codes *linéaires aléatoires* de rendement $R_n = \frac{\lceil Rn \rceil}{n}$,
- l'algorithme de décodage est un décodeur au *maximum de vraisemblance*.

Par souci d'intelligibilité, on définit succinctement ces objets dans les paragraphes suivants.

Codes linéaires. Tout d'abord, munissons l'ensemble $\{0,1\}$ de la structure de corps fini $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$. On peut ainsi voir $\mathcal{X}^n = \mathbb{F}_2^n$ comme un espace vectoriel sur \mathbb{F}_2 . Il est alors possible de s'intéresser au cas particulier des codes en blocs *linéaires*.

Définition 4.14

Un code en bloc linéaire sur un corps \mathbb{F} est une application linéaire $\phi : \mathbb{F}^k \rightarrow \mathbb{F}^n$. L'entier k est appelé la dimension du code. Une matrice génératrice du code est une matrice \mathbf{G} de taille $(k \times n)$ définie sur \mathbb{F} , telle que $\phi(\mathbb{F}_2^k) = \{\mathbf{x}\mathbf{G}, \mathbf{x} \in \mathbb{F}^k\}$.

Remarque 4.15

Par abus de langage, on confond souvent le code en bloc $\phi : \mathbb{F}^k \rightarrow \mathbb{F}^n$ et son image $\mathcal{C} = \phi(\mathbb{F}^k) \subseteq \mathbb{F}^n$.

Pour $n \geq 1$ et $k = \lceil Rn \rceil$, considérons la séquence de matrices aléatoires suivantes :

$$\mathbf{G}_n = \left(\mathbf{I}_k \mid \mathbf{M}_{k,n-k} \right)$$

où \mathbf{I}_k est la matrice identité, et $\mathbf{M}_{k,n-k}$ est une matrice de taille $(k \times n - k)$ sur \mathbb{F}_2 dont toutes les entrées sont tirées uniformément et indépendamment dans \mathbb{F}_2 . On note que le rang de \mathbf{G}_n est k . Les \mathcal{C}_n de matrice génératrice \mathbf{G}_n forment alors une séquence de codes qui entrent dans le cadre du premier point du Théorème 4.13 lorsqu'il sont équipés d'un décodeur au maximum de vraisemblance.

Décodage au maximum de vraisemblance. Pour décoder un mot bruité \mathbf{y} en sortie de canal, on cherche alors à retourner le mot de code \mathbf{x} qui est plus « vraisemblablement » à l'origine de \mathbf{y} . Ainsi, il faut chercher le mot \mathbf{x} qui maximise la probabilité conditionnelle $p(X = \mathbf{x} \mid Y = \mathbf{y})$.

Définition 4.16

Soit \mathcal{C} un code de longueur n sur un alphabet \mathcal{X} . On note X et Y l'entrée et la sortie d'un canal sur \mathcal{X} . Un algorithme décode au maximum de vraisemblance pour \mathcal{C} si, pour tout $\mathbf{y} \in \mathcal{X}^n$, la valeur \mathbf{x} de retour de l'algorithme maximise strictement la quantité

$$p(X = \mathbf{x} \mid Y = \mathbf{y}).$$

Si le mot transite à travers un canal binaire symétrique sans mémoire de paramètre $\lambda > 0$, alors on a

$$p(X = \mathbf{x} \mid Y = \mathbf{y}) = \lambda^{d_H(\mathbf{x}, \mathbf{y})} (1 - \lambda)^{n - d_H(\mathbf{x}, \mathbf{y})},$$

où la quantité $d_H(\mathbf{x}, \mathbf{y})$ est définie ci-dessous.

Définition 4.17

Soit n un entier strictement positif, et \mathcal{X} un alphabet. La distance de Hamming entre deux mots $\mathbf{x}, \mathbf{y} \in \mathcal{X}^n$ est :

$$d_H(\mathbf{x}, \mathbf{y}) = |\{i \in [1, n], x_i \neq y_i\}|.$$

Par conséquent, le décodage au maximum de vraisemblance consiste à chercher le mot de code x à distance minimale du mot reçu y . Pour chercher un tel mot, une procédure consiste à parcourir tous les mots du code et retenir celui de distance minimale ; néanmoins, la complexité de cet algorithme est importante : elle est en $O(|\mathcal{X}|^k)$.

4.3 Codes convolutifs

Les codes convolutifs, introduits par Elias en 1955, forment une classe de codes plus flexible que celle des codes en blocs. Ils permettent de gérer des séquences de symboles infinies, et atteignent de meilleures performances que les codes en blocs sur certains canaux. Du fait de leur efficacité en décodage, ce sont les codes les plus couramment utilisés en télécommunication.

4.3.1 Définition et représentations

L'idée du codage convolutif est de créer, à partir de messages $x \in \{0,1\}^+$, un ensemble de mots $c \in \{0,1\}^+$ formés de combinaisons linéaires entre les symboles consécutifs des messages (on assimile $\{0,1\}$ au corps fini \mathbb{F}_2). Quitte à rallonger les messages par des zéros, on peut alors supposer que les messages et les mots de codes sont infinis et indexés par \mathbb{N} .

Exemple 4.18

Pour un message $x = (x_n)_{n \in \mathbb{N}}$, on définit les deux séquences $c^{(1)}$ et $c^{(2)}$ par :

$$\begin{aligned} c_n^{(1)} &= x_n + x_{n-1} + x_{n-2}, \\ c_n^{(2)} &= x_n + x_{n-1}, \end{aligned}$$

où par convention on pose $x_i = 0$ pour $i \leq 0$. Les séquences $c^{(1)}$ et $c^{(2)}$ sont appelées séquences de parité.

Le code convolutif correspondant est alors l'application qui transforme le message $x = (x_n)_{n \in \mathbb{N}}$ de taille quelconque en la paire de séquences de parité $(c^{(1)}, c^{(2)})$.

On dit alors ce code a une fenêtre glissante de taille 3 (c'est la taille de l'intervalle de symboles impliqués dans la création des séquences de parité) et qu'il admet un rendement de $1/2$ (rapport entre nombre de messages et nombre de séquences de parité).

Définition 4.19

Un code convolutif de rendement k/n est la donnée de k polynômes $g_1(x), \dots, g_k(x) \in \mathbb{F}_2[x]$ de degré $\leq n-1$.

Notons $g_j[i]$ le coefficient de degré i du polynôme $g_j(x)$. Le code convolutif transforme alors un message $x = (x_0, x_1, \dots, x_m, \dots) \in \{0,1\}^+$ en k séquences $c^{(1)}, \dots, c^{(k)} \in \{0,1\}^+$ définies par :

$$c_m^{(j)} = \sum_{i=0}^{n-1} g_j[i] x_{m-i}, \quad 1 \leq j \leq k, \quad m \in \mathbb{N},$$

où par convention on pose $x_m = 0$ pour $m < 0$. Les polynômes $g_1(x), \dots, g_k(x)$ sont appelés les générateurs du code, et n est la longueur de contrainte.

Un code convolutif peut être représenté de différentes manières.

Représentation algébrique. La première représentation permet de comprendre la terminologie introduite dans la définition du code convolutif, notamment l'utilisation de polynômes. L'idée est de modéliser les opérations sur les séquences de bits, comme des multiplications de séries formelles sur \mathbb{F}_2 . Pour toute suite binaire $\mathbf{a} = (a_0, a_1, \dots, a_m, \dots)$, on définit sa série formelle associée comme

$$A(z) := a_0 + a_1z + a_2z^2 + \dots = \sum_{n \in \mathbb{N}} a_n z^n \in \mathbb{F}_2[[z]]$$

Notons que si \mathbf{a} est une suite nulle à partir d'un certain rang, alors sa série formelle $A(z)$ est simplement un polynôme en z .

Considérons maintenant $g_1(z), \dots, g_k(z)$ les polynômes générateurs d'un code convolutif. En s'appuyant sur le formalisme des séries formelles, on observe alors que les séquences de parité $\mathbf{c}^{(1)}, \dots, \mathbf{c}^{(k)}$ produisent des séries $C^{(1)}(z), \dots, C^{(k)}(z)$ définies par les relations

$$\begin{cases} C^{(1)}(z) = g_1(z) \cdot X(z), \\ C^{(2)}(z) = g_2(z) \cdot X(z), \\ \dots = \dots \\ C^{(k)}(z) = g_k(z) \cdot X(z), \end{cases}$$

où $X(z)$ est la série formelle associée au message \mathbf{x} . Encoder un message correspond donc à effectuer une multiplication d'une série formelle (le plus souvent un polynôme) par un ensemble de polynômes de petit degré.

Exemple 4.20

Reprenons le code convolutif proposé dans l'Exemple 4.18. On a alors $g_1(z) = 1 + z + z^2$ et $g_2(z) = 1 + z$. Si l'on choisit comme message $\mathbf{x} = (1, 0, 1, 1, 0, 0, 0, \dots)$, alors sa série formelle est $X(z) = 1 + z^2 + z^3$, puis on obtient les polynômes

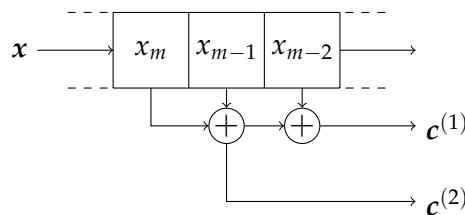
$$\begin{aligned} C^{(1)}(z) &= g_1(z) \cdot X(z) = (1 + z + z^2) \cdot (1 + z^2 + z^3) = 1 + z + z^5, \\ C^{(2)}(z) &= g_2(z) \cdot X(z) = (1 + z) \cdot (1 + z^2 + z^3) = 1 + z + z^2 + z^4, \end{aligned}$$

qui correspondent aux séquences $\mathbf{c}^{(1)} = (1, 1, 0, 0, 0, 1, 0, 0, \dots)$ et $\mathbf{c}^{(2)} = (1, 1, 1, 0, 1, 0, 0, \dots)$.

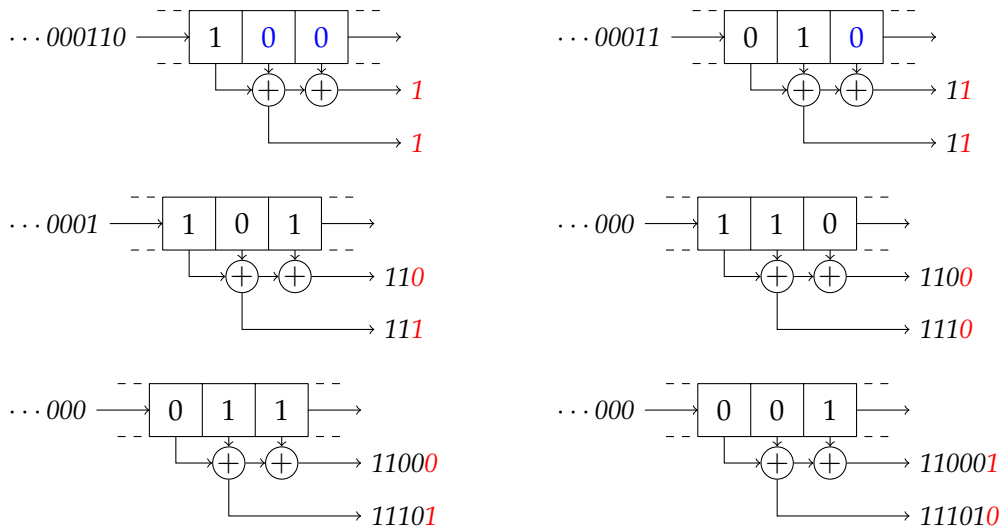
Registre à décalage. Les codes convolutifs peuvent aussi être représentés sous forme de registre à décalage. Dans cette représentation, les bits du message $\mathbf{x} \in \{0,1\}^+$ passent successivement dans des registres, dont le nombre est égal à la longueur de contrainte. À chaque entrée d'un bit x_m dans les registres, les bits de parité $c_m^{(1)}, \dots, c_m^{(k)}$ sont formés et empilés sur les séquences de parité déjà créées.

Exemple 4.21

Pour le code proposé dans l'Exemple 4.18, on obtient la représentation suivante.



Pour la séquence $\mathbf{x} = (1, 0, 1, 1, 0, 0, 0, \dots)$, voici les états successifs des registres et des mots de parité (les valeurs en bleu correspondent à l'état initial des registres) :



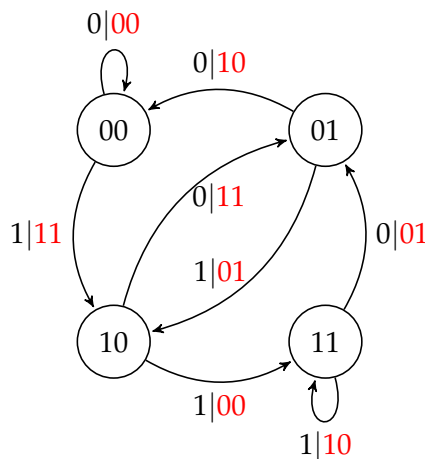
On retrouve bien les séquences de parité obtenues dans l'Exemple 4.20.

Automate. Un code convolutif a aussi une description sous forme d'automate. Rappelons qu'un automate est un graphe orienté, dont les arêtes sont étiquetées de telle sorte que les arêtes sortantes de tout nœud sont étiquetées par des valeurs distinctes.

Pour un code convolutif à k générateurs et de longueur de contrainte n , on aura 2^{n-1} nœuds, chacun correspondant à un état possible des $n - 1$ derniers bits $(x_{m-1}, \dots, x_{m-n+1})$ du registre. Chaque nœud a 2 arêtes sortantes. Ces arêtes ont pour étiquette « $x_m | c_m^{(1)} \dots c_m^{(k)}$ », et relie le nœud $(x_{m-1}, \dots, x_{m-n+1})$ au nœud (x_m, \dots, x_{m-n+2}) .

Exemple 4.22

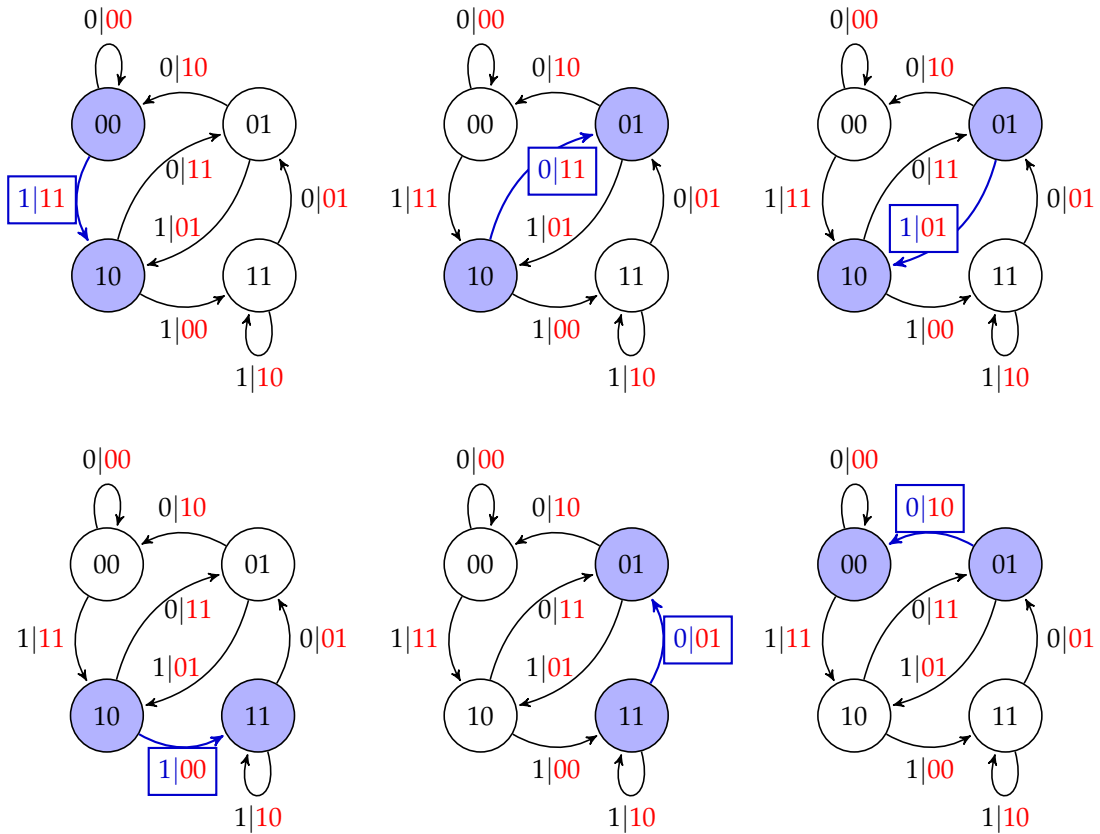
Pour le code proposé dans l'Exemple 4.18, on obtient la représentation suivante.



On peut remarquer que pour tout code convolutif de longueur de contrainte fixée (ici, 3), l'automate correspondant aura une forme similaire : seules les données écrites en rouge dépendent du choix des polynômes (leur degré, leurs coefficients).

Pour encoder un message, on part de l'état initial (00), puis on suit l'arête dont le premier bit de l'étiquette (avant le « | ») correspond au bit courant du message. Les bits suivants de l'étiquette de l'arête constituent les nouveaux bits à ajouter aux séquences de parité.

Dans le cas de $x = (1, 0, 1, 1, 0, 0, 0, \dots)$, on obtient alors les états successifs suivants.

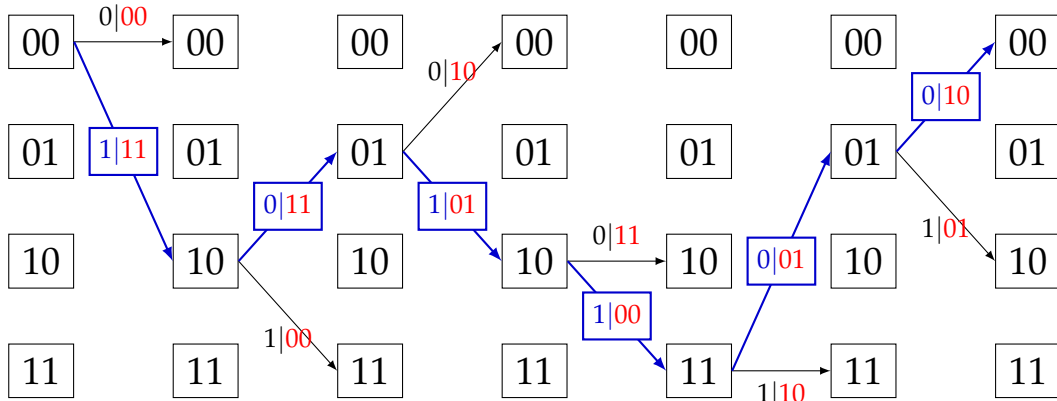


Treillis. En « développant » l'automate dans le temps, on obtient la représentation du code convolutif sous forme de treillis. Cette représentation est notamment utilisée pour décoder.

Pour chaque temps m , tous les états de l'automate sont listés en colonne. Ces états sont reliés aux mêmes ensembles d'états du temps $m + 1$, en accord avec les transitions données par l'automate. Une tranche du treillis aura donc la forme donnée en Figure 4.1. Ainsi, l'encodage d'un message correspond à une suite d'arêtes partant de l'état $00 \dots 0$ du temps 0, et allant vers la droite. L'Exemple 4.23 en propose une illustration.

Exemple 4.23

Reprenons une nouvelle fois le code de l'Exemple 4.18. L'encodage se visualise sur le treillis par le chemin bleu et les cadres associés : les bits de message sont en bleu, tandis que les bits de mots de code sont en rouge.



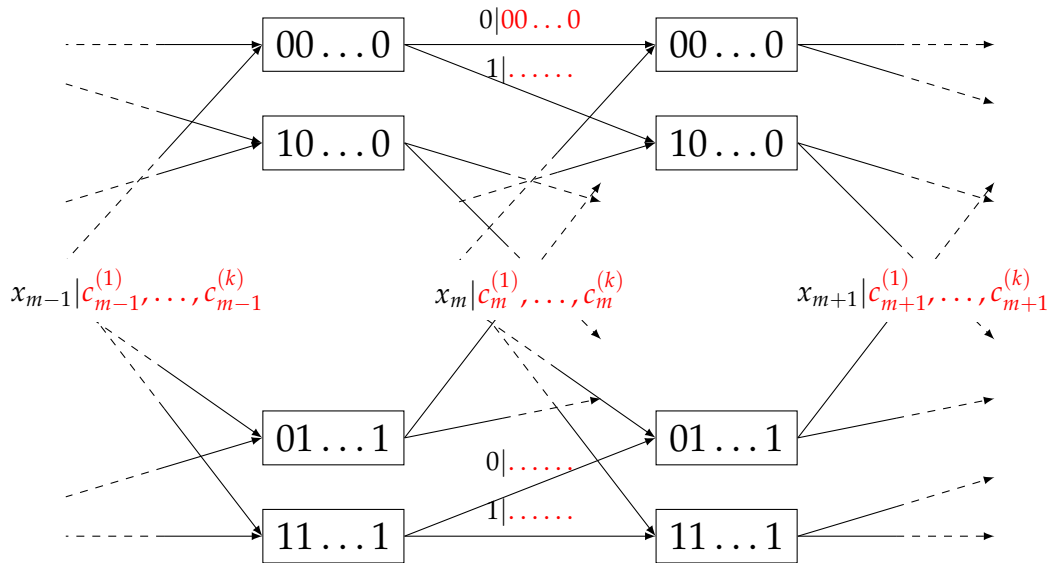


FIGURE 4.1 – Tranche de treillis d'un code convolutif.

4.3.2 Algorithme de décodage de Viterbi

L'algorithme de Viterbi est un algorithme de programmation dynamique permettant de décoder efficacement un code convolutif. L'idée est de chercher, dans le treillis du code, le chemin menant de l'état initial à l'état stationnaire final qui supporte le plus petit nombre d'erreurs. Pour cela, on procède itérativement et pour chaque état courant on tient à jour le chemin le plus court le menant à l'état initial.

Un pseudo-code de l'algorithme de Viterbi est présenté en Algorithme 2. Donnons quelques explications de cet algorithme. D'abord, les notations :

- On travaille sur la représentation sous forme de treillis. Un « état » est donc un $(n - 1)$ -uplet de bits, et on considère des instants $m = 0, \dots, M$.
- Le tableau à double entrée `weight` est rempli de telle sorte que `weight[i][m]` est le poids d'erreur minimal d'un chemin menant de l'état initial $i=0$ à l'instant initial $m=0$, à l'état i à l'instant m .
- Le tableau à double entrée `previous` contiendra l'information du chemin le plus court menant de l'état initial $i=0$ à l'instant initial $m=0$, à l'état i à l'instant m . Pour cela, `previous[i][m]` est constitué de l'indice j de l'état à l'instant $m-1$, ainsi que du bit a qui permet la transition.
- On note ∞ une valeur de poids inatteignable (prendre typiquement $\infty := 2Mk$).
- On note également `dist` la distance de Hamming entre deux vecteurs.

Dans la première partie de l'algorithme (lignes 1 à 4), on initialise les tableaux `weight` et `previous`. Dans la deuxième partie de l'algorithme (lignes 5 à 9), on met à jour ces tableaux en cherchant, pour chaque nouveau couple (état, instant) = (i, m) , l'arête $(j, m - 1) \rightarrow (i, m)$ associée à un poids d'erreur total minimal. Dans la troisième et dernière partie, on parcourt le treillis en sens inverse en partant du poids minimal à l'instant M , et en utilisant le tableau `previous`.

Algorithme 2 : Algorithme de Viterbi associé à un code à k générateurs et de fenêtre glissante n .

Entrée : des mots bruités $\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(k)} \in \{0,1\}^+$, une longueur maximale M

Sortie : un message $x \in \{0,1\}^+$ et un poids d'erreur w

1 **Pour tout état i faire**

2 **Pour tout $m = 0, \dots, M$ faire**

3 weight[i][m] $\leftarrow \infty$

4 weight[0][0] = 0

5 **Pour tout $m = 1, \dots, M$ faire**

6 **Pour tout état i faire**

7 Chercher une arête $j \rightarrow i$ d'étiquette $a|t_1 \dots t_k$ qui minimise la somme

$$s_{i,j,m} = \text{weight}[j][m-1] + \text{dist}([t_1 \dots t_k], [\mathbf{y}_m^{(1)} \dots \mathbf{y}_m^{(k)}])$$

8 previous[i][m] $\leftarrow [j, a]$

9 weight[i][m] $\leftarrow s_{i,j,m}$

10 Chercher un i tel que weight[i][M] est minimal.

11 $w \leftarrow \text{weight}[i][M]$

12 res $\leftarrow []$

13 **Pour tout $m = M, \dots, 1$ faire**

14 Ajouter previous[i][m][1] au début de res

15 $i \leftarrow \text{previous}[i][m][0]$

16 Retourner res et w

La complexité en temps et en mémoire de l'algorithme de Viterbi est en $\mathcal{O}(M2^k)$. Notons que, lors de la recherche d'éléments minimaux, plusieurs choix sont parfois possibles. Alors, par exemple on pourra choisir aléatoirement l'un des éléments atteignant la valeur minimale.

Dans le cas du canal binaire symétrique de paramètre λ , on peut montrer que la probabilité d'échec de l'algorithme de Viterbi est majorée par une expression de la forme

$$p \sim K \cdot \lambda^{\lfloor d/2 \rfloor}$$

où K est une constante positive et d est la distance libre du code convolutif. Cette distance représente le poids minimal d'un chemin associé à un mot de code valide ; elle peut se calculer en cherchant un chemin non-trivial de poids minimal entre l'état $00 \dots 0$ et lui-même dans l'automate du code. Dans le cas de l'Exemple 4.22, on obtient une distance libre égale à 4.

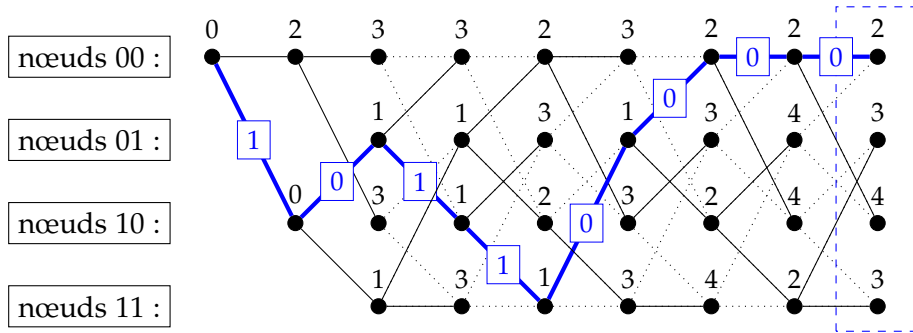
Remarque 4.24

Il faut donc retenir que dans la conception d'un code convolutif, on cherche à obtenir un rendement k/n et une distance libre d les plus grands possibles. D'un point de vue algorithmique (en mémoire comme en calcul), il est aussi intéressant d'avoir des valeurs de k et n assez petites.

Exemple 4.25

Reprenons l'Exemple 4.18 avec le message $x = (1011000 \dots)$ encodé en la paire de séquences $c^{(1)} = (1100010 \dots)$ et $c^{(2)} = (1110100 \dots)$.

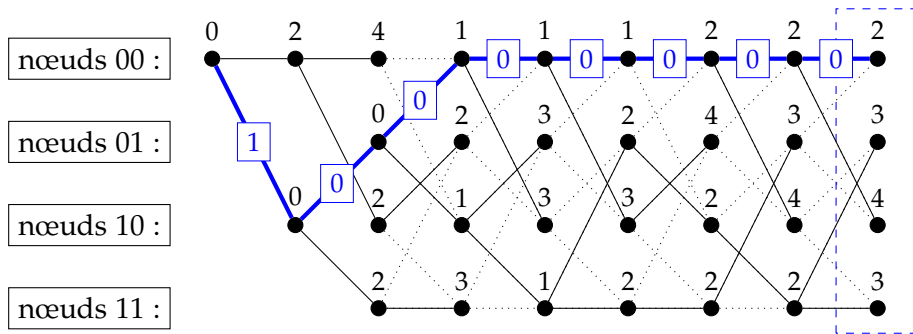
1. Dans un premier temps, on suppose qu'après passage dans le canal, on reçoit la paire de séquences bruitées $y^{(1)} = (1100010 \dots)$ et $y^{(2)} = (1010110 \dots)$. Dans la figure suivante, on présente les chemins construits successivement par l'algorithme de Viterbi. Lorsqu'un choix parmi deux arêtes doit être fait, on garde celle dont le nœud de départ est le premier dans l'ordre lexicographique (choix arbitraire).



Au-dessus de chaque nœud, en noir, on précise le poids du chemin le reliant à l'état initial. Le chemin bleu représente le chemin de poids le plus faible après $M = 8$ étapes. Pour des valeurs plus grandes de M , on obtiendrait le même chemin complété d'arêtes $00 \rightarrow 00$.

La valeur encadrée en bleu sur toute arête du chemin minimal précise le bit de message correspondant à la transition. Ainsi, on retrouve le message $x = (10110000 \dots)$. Remarquons que le poids du chemin correspond au nombre de bits de $c^{(1)}, c^{(2)}$ qui ont été corrompus par le canal.

2. Dans un second temps, on suppose avoir reçu la paire de mots bruités $y^{(1)} = (1110010 \dots)$, $y^{(2)} = (1110000 \dots)$. Notons que cette paire de mots est également à distance 2 de la paire de mots $c^{(1)}, c^{(2)}$: seuls les bits $c_2^{(1)}$ et $c_4^{(2)}$ ont été modifiés. En revanche contrairement à l'exemple précédent, l'algorithme de Viterbi va retourner un mauvais message. Comme le montre la figure ci-dessous, on obtient en effet le message $x' = (1,0,0,0, \dots)$, en ayant fait une nouvelle fois l'hypothèse que, dans l'algorithme de Viterbi, lorsqu'un choix parmi deux arêtes doit être fait, on sélectionne celle dont le nœud de départ est le premier dans l'ordre lexicographique.



Chapitre 5

Modèles de sources stochastiques

5.1 Processus stochastiques

5.1.1 Premières définitions

Dans les chapitres précédents, nous avons émis l'hypothèse selon laquelle les sources et canaux sont *sans mémoire*, c'est-à-dire que les symboles qu'ils émettent ou transmettent sont issus de variables aléatoires indépendantes. Or, les fichiers et messages manipulés et transmis en pratique sont structurés : deux symboles consécutifs sont rarement décorrélés. Par exemple, dans la langue française, le *digramme*¹ « ES » est plus fréquent que le digramme « SE », bien qu'ils soient constitués des mêmes lettres.

Ainsi, il est naturel de considérer des sources pour lesquelles la probabilité d'apparition d'un certain symbole dépend des événements passés. On donne le nom de *processus stochastique* à la suite de variables aléatoires (potentiellement dépendantes entre elles) correspondant à une telle source.

Définition 5.1

Un processus stochastique discret est une suite de variables aléatoires $X = (X_n)_{n \in \mathbb{N}}$ définies sur un même espace probabilisé (Ω, p) .

On voit la réalisation d'un processus stochastique comme une succession (discrète) de tirages aléatoires dans le temps. Étant donné un instant « présent » $n \in \mathbb{N}$, le « passé » est la réalisation des variables $\{X_i\}_{i < n}$, et le « futur » celle des variables $\{X_i\}_{i > n}$.

Différents qualificatifs sont donnés aux processus stochastiques, suivant l'influence du passé sur la réalisation de la variable aléatoire présente.

1. suite de deux lettres

Définition 5.2

Un processus stochastique $\mathbf{X} = (X_n)_{n \in \mathbb{N}}$ sur (Ω, p) est :

- sans mémoire si pour tous $0 \leq i < n$, les variables X_i et X_n sont indépendantes;
- markovienne si pour tout $n \geq 1$ et pour tout $(x_i)_{i \in \mathbb{N}}$, on a

$$p(X_{n+1} = x_{n+1} \mid (X_n, \dots, X_0) = (x_n, \dots, x_0)) = p(X_{n+1} = x_{n+1} \mid X_n = x_n);$$

- stationnaire si pour tous $n, m \geq 0$ et tout $(x_i)_{i \in \mathbb{N}}$ on a :

$$p(X_{n+m} = x_n, \dots, X_m = x_0) = p(X_n = x_n, \dots, X_0 = x_0).$$

Ainsi, informellement :

- dans un processus stochastique sans mémoire, le passé n'a aucune influence sur les réalisations actuelles;
- dans un processus stochastique markovien, seul l'instant précédent peut éventuellement influencer sur le présent;
- un processus stochastique stationnaire est invariant par translation dans le temps.

Exemple 5.3

Considérons un ensemble de vignettes $\mathcal{X} = \{x_1, \dots, x_m\}$ à collectionner avec $m \geq 2$. À chaque instant $n \geq 0$, une vignette $x \in \mathcal{X}$ est tirée, et on note X_n la variable aléatoire correspondante. La vignette est ensuite ajoutée à une collection (on suppose que la collection est vide avant le premier tirage), et on passe à l'instant $n + 1$. Si l'on note C_n la variable aléatoire correspondant à l'état de la collection après n tirages, alors on peut écrire formellement :

$$C_n = C_{n-1} \cup \{X_n\} \text{ pour } n \geq 1.$$

Si l'on suppose que les tirages $(X_n)_{n \in \mathbb{N}}$ sont deux à deux indépendants, alors par définition le processus $\mathbf{X} = (X_0, \dots, X_n, \dots)$ est sans mémoire. Par ailleurs, l'état de la collection C_n ne dépend que de l'état C_{n-1} et du tirage X_n . Ainsi le processus $\mathbf{C} = (C_1, \dots, C_n, \dots)$ est markovien. En revanche, il n'est pas stationnaire : clairement les variables C_0 et C_1 n'ont pas la même loi, puisque qu'elles ne prennent pas les mêmes valeurs.

Exemple 5.4

Soit $(X_n)_{n \geq 0}$ une suite de variables aléatoires à valeurs dans $\{0, 1\}$. On suppose que les X_n sont indépendantes et de même loi.

1. La variable aléatoire $Y_n := X_n + X_{n-1}$ définit un processus $\mathbf{Y} = (Y_1, \dots, Y_n, \dots)$ stationnaire. En effet, comme les X_i sont indépendantes et de même loi, la variable Y_1 a même loi que Y_m pour tout $m \geq 1$, et cette propriété s'étend aux variables conjointes (Y_1, \dots, Y_n) et (Y_m, \dots, Y_{n+m-1}) .
2. La variable aléatoire $\bar{X}_n := \frac{1}{n+1} \sum_{i=0}^n X_i$, pour $n \geq 0$, définit un processus markovien. En effet, on a $\bar{X}_n = \frac{n}{n+1} \bar{X}_{n-1} + \frac{1}{n+1} X_n$.
3. La variable aléatoire $I_n := \min\{X_i \mid 0 \leq i \leq n\}$ définit un processus markovien, car on peut écrire $I_n = \min\{I_{n-1}, X_n\}$.
4. Définissons la variable aléatoire M_n comme la médiane de $\{X_i \mid 0 \leq i \leq n\}$. Pour n impair, par convention la médiane est le $\frac{n+1}{2}$ -ème plus petit élément des $\{X_i \mid 0 \leq i \leq n\}$. Alors M_n n'est ni markovienne, ni stationnaire. Pour le caractère non-markovien, on peut observer que M_2 vaut M_1 si $X_2 < X_1 < X_0$ et vaut $M_0 \neq M_1$ si $X_1 < X_0 < X_2$.

Par la suite, on emploiera le terme de source sans mémoire (respectivement stationnaire, resp. markovienne) pour désigner une source dont la séquence de variables aléatoire $\mathbf{X} = (X_i)_i$ forme un processus stochastique sans mémoire (resp. stationnaire, resp. markovien).

Remarque 5.5

Un processus stochastique sans mémoire est markovien et stationnaire.

5.1.2 Taux d'entropie et codage universel

Dans les chapitres précédents, la notion d'entropie a permis d'associer à une variable aléatoire une quantité d'information émise. Essayons d'étendre cette notion à des sources plus générales $\mathbf{X} = (X_n)_{n \in \mathbb{N}}$.

Supposons un instant que la source soit constituée d'un nombre fini de variables aléatoires X_0, \dots, X_N . La quantité d'information détenue par ces variables est alors l'entropie de la variable conjointe (X_0, \dots, X_N) . Dans le cas d'une source sans mémoire, cela correspond à calculer la somme des entropies de X_0, \dots, X_N , car ces variables sont indépendantes.

Lorsque N grandit, il y a alors de bonnes chances pour que cette quantité diverge vers $+\infty$. Pour obtenir une valeur sensible, on va donc plutôt considérer la moyenne des $H(X_0), \dots, H(X_N)$ dans le cas d'une source sans mémoire, ce qui correspond dans un cas général à diviser l'entropie de la variable conjointe par $N + 1$. La limite asymptotique de cette quantité est alors appelée *taux d'entropie* de la source.

Définition 5.6

Le taux d'entropie d'une source stationnaire $\mathbf{X} = (X_n)_{n \in \mathbb{N}}$ est

$$\bar{H}(\mathbf{X}) = \lim_{n \rightarrow \infty} \frac{1}{n+1} H(X_0, X_1, \dots, X_n),$$

où $H(X_0, X_1, \dots, X_n)$ est l'entropie de la loi conjointe des variables X_0, \dots, X_n .

On peut se questionner sur l'existence de la limite donnée en Définition 5.6. La propriété suivante permet de s'en assurer, et d'en donner une autre forme. Insistons sur le fait que, dans cette définition, on suppose que la source est stationnaire.

Proposition 5.7

Pour une source stationnaire \mathbf{X} , le taux d'entropie $\bar{H}(\mathbf{X})$ est bien défini et vaut

$$\lim_{n \rightarrow \infty} H(X_n | X_0, \dots, X_{n-1}).$$

Démonstration : Posons $u_n = H(X_n | X_0, \dots, X_{n-1})$. Il est clair que $u_n \leq H(X_n | X_1, \dots, X_{n-1})$. Puis, comme \mathbf{X} est stationnaire, par translation dans le temps on a

$$H(X_n | X_1, \dots, X_{n-1}) = H(X_{n-1} | X_0, \dots, X_{n-2}).$$

Ainsi, $0 \leq u_n \leq u_{n-1}$ pour tout $n \geq 1$, donc la suite $(u_n)_{n \in \mathbb{N}}$ converge vers une limite $u \geq 0$.

Notons maintenant que, par la règle de chaînage de l'entropie, on a :

$$H(X_0, \dots, X_n) = \sum_{i=0}^n H(X_i | X_0, \dots, X_{i-1}) = \sum_{i=0}^n u_i.$$

Pour terminer la preuve, il suffit donc d'établir que la suite définie par $c_n = \frac{1}{n+1} \sum_{i=0}^n u_i$ converge également vers u . Génériquement, ce résultat est connu sous le nom de lemme de Césaro. ■

Lemme 5.8 (Lemme de Cesàro)

Soit $(u_n)_{n \in \mathbb{N}}$ une suite complexe qui converge vers une limite $u \in \mathbb{C}$. Alors, la suite $(c_n)_{n \in \mathbb{N}}$ de terme général $c_n = \frac{1}{n+1} \sum_{i=0}^n u_i$ converge, et sa limite est aussi égale à u .

Démonstration : Soit $\epsilon > 0$. Comme $(u_n)_{n \in \mathbb{N}}$ converge vers $u \in \mathbb{C}$, il existe $N \geq 0$ tel que pour tout $n \geq N$, $|u_n - u| < \epsilon/2$. Par ailleurs, par inégalité triangulaire pour tout $M \geq 0$ on a

$$\left| \left(\frac{1}{n+1} \sum_{i=0}^n u_i \right) - u \right| \leq \sum_{i=0}^n \frac{1}{n+1} |u_i - u| = \sum_{i=0}^{M-1} \frac{1}{n+1} |u_i - u| + \sum_{i=M}^n \frac{1}{n+1} |u_i - u|.$$

Posons $a_n = \frac{1}{n+1} \sum_{i=0}^{N-1} |u_i - u|$. Comme la somme $\sum_{i=0}^{N-1} |u_i - u|$ est finie, la suite $(a_n)_{n \in \mathbb{N}}$ tend vers 0 donc il existe $N' \geq 0$ tel que $a_n \leq \epsilon/2$ pour tout $n \geq N'$. Pour tout $n \geq M := \max\{N, N'\}$, on a donc :

$$\left| \frac{1}{n+1} \sum_{i=0}^n u_i - u \right| \leq \frac{\epsilon}{2} + \sum_{i=M}^n \frac{1}{n+1} |u_i - u| \leq \frac{\epsilon}{2} + \frac{n-M+1}{n+1} \cdot \frac{\epsilon}{2} \leq \epsilon. \quad \blacksquare$$

Exemple 5.9

Soit \mathbf{X} une source sans mémoire (donc stationnaire) constituée de variables de même loi. On a alors

$$H(\mathbf{X}_0) = H(\mathbf{X}_n) \quad \text{pour tout } n \in \mathbb{N}.$$

Par conséquent, la quantité

$$\frac{1}{n+1} H(\mathbf{X}_0, \dots, \mathbf{X}_n) = \frac{1}{n+1} (H(\mathbf{X}_0) + \dots + H(\mathbf{X}_n)) = H(\mathbf{X}_0)$$

est constante (la première égalité provient du fait que les X_i sont indépendantes). Le taux d'entropie de \mathbf{X} est donc $H(\mathbf{X}_0)$. Cela se vérifie avec la seconde forme du taux d'entropie :

$$\forall n \geq 1, \quad H(\mathbf{X}_n | \mathbf{X}_0, \dots, \mathbf{X}_{n-1}) = H(\mathbf{X}_n) = H(\mathbf{X}_0),$$

une nouvelle fois car les variables X_i sont indépendantes.

Comme dans le cas d'une variable aléatoire seule, on peut se questionner sur la manière optimale d'encoder une source \mathbf{X} modélisée par un processus stochastique. Pour cela, on souhaite donc minimiser l'espérance de la longueur par symbole d'un encodage de la source. Il est possible de démontrer que cette valeur minimale est le taux d'entropie de la source, et des codes permettant d'atteindre cette borne sont appelés *codes universels*. En voici une définition plus formelle.

Définition 5.10 (Codes universels)

Soit $\mathbf{X} = (X_n)_{n \geq 1}$ un processus stochastique tel que les X_n , $n \geq 1$, sont à valeurs dans un même ensemble \mathcal{X} . On considère $\mathcal{C} = (C_n)_{n \geq 1}$ une famille de codes telle que pour tout $n \geq 1$, le code C_n encode des séquences de n symboles de \mathcal{X} . Alors, la famille \mathcal{C} est dite universelle si

$$\lim_{n \rightarrow \infty} \mathbb{E}_{\mathbf{X}} \left[\frac{1}{n} \ell(C_n(\mathbf{X}_1, \dots, \mathbf{X}_n)) \right] = \bar{H}(\mathbf{X})$$

pour toutes les sources stationnaires \mathbf{X} .

La section suivante est consacrée à une construction de codes dont les propriétés s'approchent du caractère universel. Avant cela, intéressons-nous plus en détail au cas particulier des processus markoviens.

5.1.3 Le cas des processus markoviens

Nous allons nous intéresser principalement au cas de processus markoviens pour lesquels les probabilités conditionnelles $p(X_n | X_{n-1})$ sont invariantes dans le temps (attention, cela ne veut pas dire que $X = (X_n)$ est stationnaire!). Ces processus markoviens sont qualifiés d'homogènes.

Définition 5.11

Un processus markovien $X = (X_n)_{n \in \mathbb{N}}$, dont toutes les variables X_n sont à valeurs dans \mathcal{X} , est dit homogène si pour tout $n \geq 0$ et pour tous $a, b \in \mathcal{X}$, on a :

$$p(X_{n+1} = b | X_n = a) = p(X_1 = b | X_0 = a).$$

Dans cette sous-section, on fixe maintenant $X = (X_n)_{n \in \mathbb{N}}$ un processus markovien et homogène sur un ensemble $\mathcal{X} = \{x_1, \dots, x_m\}$.

Remarque 5.12

Les processus markoviens sont également souvent appelés chaînes de Markov.

Par définition, une chaîne de Markov homogène est complètement déterminée par les probabilités conditionnelles $p(X_1 = b | X_0 = a)$. Cela mène à définir, comme pour les canaux du chapitre précédent, une matrice qui enregistre ces probabilités de transitions.

Définition 5.13

La matrice de transition de X est définie comme :

$$M = \begin{pmatrix} p_{1,1} & \cdots & p_{1,m} \\ \vdots & p_{i,j} & \vdots \\ p_{m,1} & \cdots & p_{m,m} \end{pmatrix},$$

où $p_{i,j} = \mathbb{P}(X_1 = x_i | X_0 = x_j)$.

Proposition 5.14

La matrice de transition M d'un processus markovien homogène vérifie les propriétés suivantes :

1. tous les coefficients de M sont entre 0 et 1,
2. la somme des coefficients de toute colonne de M vaut 1,
3. le vecteur $\mathbf{1} := (1, \dots, 1)$ est vecteur propre à gauche de M (c'est-à-dire $\mathbf{1}M = \mathbf{1}$), associé à la valeur propre 1.

Démonstration : Le premier point est évident. Le second point découle du fait qu'à $b \in \mathcal{X}$ fixé, la loi $X_1 | X_0 = b$ est bien une loi de probabilité. Le troisième point découle directement du second point. ■

Définition 5.15

Toute matrice vérifiant les propriétés précédentes est appelée matrice stochastique.

Comme son nom l'indique, une matrice de transition permet de représenter la transition d'un instant n à l'instant $n + 1$.

Lemme 5.16

Soit $n \geq 0$. Notons $\mathbf{u}^{(n)} = (p(X_n = x_1), \dots, p(X_n = x_m)) \in [0, 1]^m$ la distribution de la variable X_n . Alors la distribution $\mathbf{u}^{(n+1)}$ de X_{n+1} est égale à $M\mathbf{u}^{(n)}$.

Démonstration : Évident. ■

Comme corollaire, si $\mathbf{u}^{(0)}$ représente la distribution de X_0 , alors la distribution de X_n est

$$\mathbf{u}^{(n)} = \mathbf{M}^n \mathbf{u}^{(0)}.$$

En particulier, si X_0 est une variable déterministe égale à x_j , alors la probabilité $p(X_n = x_i)$ est le coefficient (i, j) de la matrice \mathbf{M}^n . On a également le résultat suivant :

Lemme 5.17

Soit $(a_0, \dots, a_n) = (x_{i_0}, \dots, x_{i_n}) \in \mathcal{X}^{n+1}$ et $\mathbf{M} = (M_{i,j})$ la matrice de transition d'une chaîne de Markov homogène. Alors on a :

$$p(X_0 = a_0, \dots, X_n = a_n) = p(X_0 = a_0) \times \prod_{j=0}^{n-1} M_{i_{j+1}, i_j}.$$

Démonstration :

$$\begin{aligned} p(X_0 = a_0, \dots, X_n = a_n) &= p(X_0 = a_0) \prod_{j=1}^n p(X_j = a_j \mid X_{j-1} = a_{j-1}, \dots, X_0 = a_0) \\ &= p(X_0 = a_0) \prod_{j=1}^n p(X_j = a_j \mid X_{j-1} = a_{j-1}) \quad (\text{par la propriété de Markov}) \\ &= p(X_0 = a_0) \prod_{j=1}^n p(X_j = a_j \mid X_0 = a_{j-1}) \quad (\text{par homogénéité}) \\ &= p(X_0 = a_0) \times \prod_{j=0}^{n-1} M_{i_{j+1}, i_j} \quad (\text{par définition de } \mathbf{M}). \end{aligned}$$
■

Une chaîne de Markov homogène peut également être représentée sous forme d'automate (voir le chapitre précédent pour la définition d'un automate). Les états de l'automate sont les éléments de \mathcal{X} et les transitions sont étiquetées par les probabilités de transition. La somme des étiquettes sortantes d'un état est donc égale à 1.

Exemple 5.18

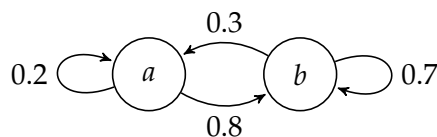
Soit X une chaîne de Markov homogène définie sur $\mathcal{X} = \{a, b\}$, par les probabilités suivantes :

$$\begin{aligned} p(X_{n+1} = a \mid X_n = a) &= 0.2, & p(X_{n+1} = b \mid X_n = a) &= 1 - 0.2 = 0.8, \\ p(X_{n+1} = b \mid X_n = a) &= 0.3, & p(X_{n+1} = b \mid X_n = b) &= 1 - 0.3 = 0.7. \end{aligned}$$

Alors, cette chaîne a pour matrice de transition :

$$\mathbf{M} = \begin{pmatrix} 0.2 & 0.3 \\ 0.8 & 0.7 \end{pmatrix}$$

et peut être représentée par l'automate



Proposition 5.19

Si \mathbf{X} est une chaîne de Markov homogène et stationnaire, alors son taux d'entropie existe bien et vaut :

$$\bar{H}(\mathbf{X}) = H(X_1 | X_0).$$

Démonstration : Par la propriété de Markov, on a $H(X_n | X_0, \dots, X_{n-1}) = H(X_n | X_{n-1})$. Puis, par homogénéité, on obtient $H(X_n | X_{n-1}) = H(X_1 | X_0)$. Par conséquent,

$$\bar{H}(\mathbf{X}) = \lim_{n \rightarrow \infty} H(X_n | X_0, \dots, X_{n-1}) = H(X_1 | X_0). \quad \blacksquare$$

Exemple 5.20

Reprenons la chaîne de Markov homogène de l'Exemple 5.18, dont la matrice de transition est $\mathbf{M} = \begin{pmatrix} 0.2 & 0.3 \\ 0.8 & 0.7 \end{pmatrix}$. Sous quelle condition ce processus est-il stationnaire ?

Pour y répondre, cherchons à résoudre $\mathbf{M}\mathbf{x} = \mathbf{x}$. On trouve que l'ensemble des solutions est un espace vectoriel engendré par le vecteur $(1, 8/3)$. Le seul élément de cet espace de solutions qui est une distribution est donc $(3/11, 8/11)$. C'est l'unique distribution stationnaire de cette chaîne de Markov.

Dans ce cas, le taux d'entropie de la chaîne est

$$\begin{aligned} H(X_1 | X_0) &= p(X_0 = a)H(X_1 | X_0 = a) + p(X_0 = b)H(X_1 | X_0 = b) \\ &= \frac{3}{11}h_2\left(\frac{1}{5}\right) + \frac{8}{11}h_2\left(\frac{3}{10}\right) \simeq 0.838. \end{aligned}$$

Proposition 5.21

Soit \mathbf{X} une chaîne de Markov homogène de matrice de transition \mathbf{M} . Notons \mathbf{x} la distribution de X_0 . Alors,

$$\mathbf{X} \text{ est stationnaire} \iff \mathbf{M}\mathbf{x} = \mathbf{x}.$$

Démonstration : Soient \mathbf{X} , \mathbf{M} et \mathbf{x} comme dans l'énoncé. Si \mathbf{X} est stationnaire, alors en particulier X_0 et X_1 ont même distribution. Par conséquent $\mathbf{x} = \mathbf{M}\mathbf{x}$. Réciproquement, supposons que $\mathbf{x} = \mathbf{M}\mathbf{x}$. Alors, pour tout $k \geq 0$, on a $\mathbf{x} = \mathbf{M}^k\mathbf{x}$; autrement dit X_0 et X_k ont même loi. Qu'en est-il des lois conjointes ? Dans le même esprit que la preuve du Lemme 5.17, on a alors :

$$\begin{aligned} p(X_k = a_0, \dots, X_{k+n} = a_n) &= p(X_k = a_0) \prod_{i=1}^n p(X_{k+i} = a_i | X_{k+i-1} = a_{i-1}, \dots, X_k = a_0) \\ &= p(X_k = a_0) \prod_{i=1}^n p(X_{k+i} = a_i | X_{k+i-1} = a_{i-1}) \quad (\text{par la propriété de Markov}) \\ &= p(X_0 = a_0) \prod_{i=1}^n p(X_i = a_i | X_{i-1} = a_{i-1}) \quad (\text{par homogénéité}) \\ &= p(X_0 = a_0, \dots, X_n = a_n), \end{aligned}$$

ce qui prouve que \mathbf{X} est stationnaire. \blacksquare

Définition 5.22

Une loi invariante d'une chaîne de Markov \mathbf{X} de matrice de transition \mathbf{M} est une loi dont la distribution \mathbf{x}' vérifie $\mathbf{x}' = \mathbf{M}\mathbf{x}'$.

Remarque 5.23

Toute chaîne de Markov homogène admet au moins une loi invariante. Elle est unique si 1 est valeur propre simple de \mathbf{M} . Par conséquent, pour toute matrice stochastique \mathbf{M} , il existe une loi X_0 pour laquelle la chaîne de Markov de loi initiale X_0 et de matrice de transition \mathbf{M} est stationnaire.

On peut aller un peu plus loin que la remarque précédente, en s'intéressant aux chemins dans l'automate représentant une chaîne de Markov.

Définition 5.24

Soient i et j deux états de l'automate d'une chaîne Markov. On dit que :

1. j est accessible à partir de i s'il existe un chemin orienté dans l'automate dont les étiquettes sont toutes strictement positives ;
2. i et j communiquent si j est accessible à partir de i et i est accessible à partir de j .

Si tous les états communiquent entre eux, on dit que la chaîne de Markov est irréductible.

Remarque 5.25

La définition précédente se retranscrit de la manière suivante sur la matrice de transition \mathbf{M} : l'état j est accessible à partir de i si et seulement si il existe $n \geq 0$ tel que $M_{i,j}^n > 0$. Par ailleurs, une chaîne de Markov est irréductible si et seulement si pour tout (i, j) , il existe $n \geq 0$ tel que $M_{i,j}^n > 0$.

Définition 5.26

Soit j un état accessible à partir de lui-même. On appelle période de l'état j le pgcd des longueurs des chemins menant de j à lui-même. Cette période est notée $d(j)$:

$$d(j) := \text{pgcd}\{n \geq 1 \mid M_{j,j}^n > 0\} ;$$

Si $d(j) = 1$, on dit que j est apériodique. Si tous les états d'une chaîne sont apériodiques, on dit que la chaîne est apériodique.

Theorème 5.27

Soit $\mathbf{X} = (X_n)_{n \in \mathbb{N}}$ une chaîne de Markov homogène, irréductible et apériodique, de matrice de transition \mathbf{M} . Alors,

1. il existe une unique distribution invariante $\mathbf{x}^{(\infty)}$ pour \mathbf{M} ;
2. pour toute loi initiale X_0 , la loi X_n converge en loi vers la loi invariante de distribution $\mathbf{x}^{(\infty)}$;
3. la chaîne de Markov est stationnaire si et seulement si X_0 a pour distribution $\mathbf{x}^{(\infty)}$.

Démonstration : Todo: à faire, ou mentionner une référence ■

5.2 Algorithmes de Lempel et Ziv

5.2.1 Présentation

Dans les années 1970, Lempel et Ziv ont développé plusieurs algorithmes de compression de données sans perte. Leurs premiers algorithmes, LZ77 et LZ78, ont été utilisés comme brique de base pour des algorithmes de compression plus évolués comme LZSS, LZMA, LZW ou DEFLATE. Ces derniers actuellement utilisés dans la plupart des logiciels de compression ou d'archivage. Par exemple, zip et gzip utilisent DEFLATE, lui-même basé sur LZ77.

5.2.2 Algorithme de Lempel–Ziv–Welch

On s'intéresse dans ce cours à la variante moderne LZW des algorithmes LZ77 et LZ78. L'algorithme LZW (pour Lempel, Ziv et Welch qui l'ont découvert) se fonde sur de la *compression par dictionnaire*. L'idée est la suivante. On recherche des similitudes entre la séquence $x \in \mathcal{X}^m$ à compresser et un ensemble de chaînes mis à jour au cours de la compression. Cet ensemble est appelé le dictionnaire. Lorsque qu'une similitude est trouvée, on encode la séquence correspondante par une référence vers la séquence similaire dans la structure, couplée à une information additionnelle.

Plus précisément, dans LZW, on cherche à diviser x en mots distincts successifs, de telle sorte que chaque mot de la liste nouvellement créé est le plus petit qui n'ait pas déjà été trouvé. Pour ce faire, on tient donc à jour une liste de mots déjà vus. Ainsi, chaque nouveau mot m est de la forme $m = c|a$, où c est un mot déjà vu, et $a \in \mathcal{X}$. On peut ainsi coder la séquence de mots obtenue comme une suite de couples (i, a) , où l'entier i est l'indice du mot c dans la suite de mots obtenue; autrement dit i est une référence vers le préfixe du nouveau mot m . La suite de couples obtenue est appelée codage LZW de x .

Algorithme 3 : Algorithme de Lempel–Ziv–Welch (LZW)

Entrée : une chaîne de caractères $x \in \mathcal{X}^m$

Sortie : une séquence de couples $((i_j, a_j))_j$ où $i_j \in \mathbb{N}$ et $a_j \in \mathcal{X}$

```

1 res ← []
2 dict ← ["" ]
3 buffer ← ""
4 i ← 0
5 Pour tout  $j \in \{1, \dots, m\}$  faire
6   Ajouter  $x_j$  à la fin de buffer
7   Si il existe une clé  $k$  tel que  $\text{buffer} = \text{dict}[k]$ 
8      $i \leftarrow k$ 
9   Sinon
10    Ajouter buffer à la fin de dict
11    Ajouter  $(i, x_j)$  à la fin de res
12    Réinitialiser buffer à ""
13    Réinitialiser  $i$  à 0
14 Retourner res
```

L'algorithme LZW remplace donc des séquences de lettres par des couples clé/lettre. Il est donc particulièrement intéressant lorsque le fichier contient de longues séquences identiques, car elles sont codées par une clé de tableau (c'est-à-dire, un entier espéré de petite taille) et une simple lettre.

Exemple 5.28

Pour rendre l'écriture plus claire, on considère l'alphabet binaire $\mathcal{X} = \{a, b\}$. Soit $x = (bbaabbabaaaaababbbbbaabbabb) \in \mathcal{X}^n$ où ici $n = 27$. On obtient la division en mots suivante :

$$b, ba, a, bb, ab, aa, aaa, bab, bbb, aaab, babb.$$

Puis, chaque mot est codé suivant l'indice de son préfixe et son dernier symbole. Par convention, si le préfixe est le mot vide, alors l'indice du préfixe est 0. Cela résulte en la suite :

$$(0, b), (1, a), (0, a), (1, b), (3, b), (2, a), (6, a), (2, b), (4, b), (7, b), (8, b).$$

Remarque 5.29

Remarquons que le dictionnaire dict construit dans l'Algorithme 3 peut s'organiser sous la forme d'un arbre, dont les noeuds sont étiquetés par la position du mot correspondant dans la liste res.

5.2.3 Analyse

Cherchons maintenant à déterminer si LZW est un bon codage, c'est-à-dire s'il permet d'obtenir des mots de codes courts. Pour simplifier l'étude de la longueur des mots du codage LZW, on se place dans le cas simple d'un alphabet binaire $\mathcal{X} = \{0, 1\}$. Dans le cadre de ce cours, on étudie d'abord le pire cas pour LZW, puis on donne un résultat théorique concernant le cas moyen.

En pire cas. Cherchons une chaîne de caractère $x \in \mathcal{X}^n$ pour laquelle le codage LZW est asymptotiquement le plus long. Pour $k \geq 1$, considérons d'abord la chaîne $\mathbf{a}^{(k)}$ définie par la concaténation de tous les i -uplets d'éléments de \mathcal{X} , pour tout i allant de 1 à k :

$$\mathbf{a}^{(k)} = 0|1|00|01|10|11|000| \cdots \cdots | \underbrace{111 \dots 11}_{k \text{ fois}}.$$

On note ensuite $\mathbf{c}^{(k)}$ le codage de $\mathbf{a}^{(k)}$ par l'algorithme LZW. Pour simplifier, on suppose que tous les indices de préfixes sont codés sur un nombre fixe de bits et on note $[i]_2$ le codage de i . Ainsi (en ajoutant des virgules et des séparateurs pour la présentation) :

$$\mathbf{c}^{(k)} = [0]_{2,0} | [0]_{2,1} | [1]_{2,0} | [1]_{2,1} | [2]_{2,0} | [2]_{2,1} | [3]_{2,0} | \cdots \cdots | [(k-2)2^k - 2]_{2,1}.$$

La longueur $\ell(\mathbf{a}^{(k)})$ est égale à $\sum_{i=1}^k i2^i$. On peut alors montrer (exercice) que

$$\ell(\mathbf{a}^{(k)}) = \sum_{i=1}^k i2^i = (k-1)2^{k+1} + 2.$$

Par ailleurs, le nombre de mots $w(\mathbf{a}^{(k)})$ dans la division de la chaîne $\mathbf{a}^{(k)}$ est

$$w(\mathbf{a}^{(k)}) = \sum_{i=1}^k 2^i = 2^{k+1} - 2.$$

Enfin, chaque indice de préfixe peut être codé sur $\lceil \log_2 w(\mathbf{a}^{(k)}) \rceil = k+1$ bits, donc la longueur du mot $\mathbf{c}^{(k)}$ associé à $\mathbf{a}^{(k)}$ est

$$\ell(\mathbf{c}^{(k)}) = w(\mathbf{a}^{(k)}) \times (k+1) = \frac{k+1}{k-1} (\ell(\mathbf{a}^{(k)}) + 2) - 2.$$

Le ratio qui nous intéresse est le rapport entre la longueur du mot encodé et celle du mot initial. Si l'on note $n = \ell(\mathbf{a}^{(k)})$, on a alors :

$$\frac{\ell(\mathbf{c}^{(k)})}{\ell(\mathbf{a}^{(k)})} = \left(1 + \frac{2}{k-1}\right) \left(1 + \frac{2}{\ell(\mathbf{a}^{(k)})}\right) - \frac{2}{\ell(\mathbf{a}^{(k)})} \stackrel{n \rightarrow \infty}{=} 1 + o\left(\frac{1}{\log_2 n}\right).$$

Remarquons enfin que si une chaîne \mathbf{b} de longueur $n \geq \ell(\mathbf{a}^{(k)})$ est telle que le codage LZW \mathbf{y} de \mathbf{b} ne contient pas tous les mots de longueurs $\leq k$, alors on peut remplacer un des mots de longueur $> k$ dans \mathbf{y} , par un mot de longueur $\leq k$. En itérant le procédé jusqu'à ne plus pouvoir, on finit par avoir tous les mots de longueur $\leq k'$ dans \mathbf{y} (pour un certain $k' \leq k$),

et on remarque qu'à chaque étape le ratio $\ell(\mathbf{y})/\ell(\mathbf{b})$ décroît. Le mot $\mathbf{a}^{(k)}$ correspond donc essentiellement au pire cas pour l'algorithme LZW.

Ajoutons qu'il est impossible de réaliser un codage qui, en pire cas, a une longueur relative $\ell(\mathbf{c}^{(k)})/\ell(\mathbf{a}^{(k)})$ plus petite que 1. En effet, il y a 2^n mots de longueur n à coder, ce qui nécessite donc au moins n bits (voir Proposition 3.4 pour une idée similaire). Le codage LZW est donc asymptotiquement optimal.

En cas moyen. Pour l'analyse du cas moyen, il faut faire des hypothèses supplémentaires sur la source. On suppose donc que $\mathbf{X}^n = (X_1, \dots, X_n)$ provient d'un processus stochastique stationnaire. Dans ce cas, le taux d'entropie $\bar{H}(\mathbf{X})$ peut bien être défini. Il faut également supposer que les statistiques de la source (ses moments) peuvent être approchées à partir d'une réalisation suffisamment longue : un tel processus est alors qualifié d'*ergodique* (nous n'en donnerons pas de définition formelle). On a alors le résultat suivant.

Theorème 5.30

Soit $C(\mathbf{x}^n)$ le codage LZW d'un message \mathbf{x}^n de longueur n , issu d'un processus stochastique \mathbf{X} stationnaire et ergodique. Alors on a

$$\lim_{n \rightarrow +\infty} \frac{1}{n} \ell(C(\mathbf{x}^n)) = \bar{H}(\mathbf{X}).$$

Étant donnée l'hypothèse supplémentaire d'ergodicité de la source, on ne peut pas alors affirmer que le codage de Lempel–Ziv–Welch est universel. On le qualifie néanmoins de ponctuellement universel (*pointwise universal*).

Chapitre 6

Autour de la compression de données

6.1 Compression sans perte

La *compression de données sans perte* a pour but de transformer une chaîne de caractères $x \in \mathcal{X}^+$ en une autre chaîne $y \in \mathcal{Y}^+$ de taille plus petite, telle qu'il soit possible de retrouver *exactement* la chaîne x à partir de y uniquement. En pratique, on souhaite que les algorithmes de compression (de x vers y) et de décompression (de y vers x) soient efficaces en temps et en mémoire.

Bien entendu, il existe une limite théorique à la taille qu'on peut espérer pour y . D'un point de vue algorithmique, on peut considérer l'ensemble de tous les programmes P qui permettent de construire x . Si l'on se donne une machine de Turing universelle, chaque programme a une taille que l'on note ℓ_P et une sortie s_P . La *complexité de Kolmogorov* de x ,

$$K(x) = \min_{\text{programmes } P} \{\ell_P \mid s_P = x\},$$

est alors une limite théorique à la taille de compression de x . Notons que la complexité de Kolmogorov d'une chaîne x n'est pas calculable génériquement.

Remarque 6.1

La compression sans perte peut s'apparenter à la construction d'un codage qui est particulièrement court pour un sous-ensemble spécifique de messages. Parmi l'ensemble \mathcal{X}^+ de toutes les chaînes de caractères, on souhaite associer à certaines chaînes x (les chaînes les plus « structurées ») un unique mot c tel que $\ell(c) < \ell(x)$.

6.1.1 Un premier algorithme de compression : RLE

L'algorithme RLE, pour *run-length encoding* ou codage par plages, est un algorithme de compression simple qui permet de réduire drastiquement la longueur de chaînes de caractères admettant des séries de caractères identiques consécutifs.

Pour simplifier le propos, on supposera que l'algorithme prend en entrée une chaîne de caractères alphabétiques (ne contenant pas de chiffres), et retourne une chaîne de caractères alphanumériques (lettres + chiffres). L'Algorithme 4 en présente une version basique.

Remarque 6.2

La sortie de l'Algorithme 4 peut être rendue plus courte en n'affichant pas la valeur 1 (on retire donc les 1 qui précèdent donc une lettre isolée dans x).

Algorithme 4 : Algorithme RLE de codage par plages (*run-length encoding*)**Entrée** : une chaîne de caractère $x = x_0 \dots x_{m-1} \in \mathcal{X}^m$ **Sortie** : une chaîne de caractères $b \in (\mathbb{N} \times \mathcal{X})^+$

```

1  $b \leftarrow ""$ 
2  $cpt \leftarrow 1$ 
3  $lettre \leftarrow x_0$ 
4 Pour tout  $i \in \{1, \dots, m-1\}$  faire
5   Si  $lettre = x_i$ 
6      $cpt \leftarrow cpt + 1$ 
7   Sinon
8      $b \leftarrow b \mid cpt \mid lettre$ 
9      $cpt \leftarrow 1$ 
10     $lettre \leftarrow x_i$ 
11  $b \leftarrow b \mid cpt \mid lettre$ 
12 Retourner  $b$ 

```

Exemple 6.3

Sur l'entrée $x = (bbaabbabaaaaababbbbaabbabb)$ de l'Exemple 5.28, l'algorithme RLE donne en sortie

$$2b2a2b1a1b5a1b1a4b3a2b1a2b.$$

L'algorithme RLE permet de réduire toute plage de n symboles identiques consécutifs en une paire alphanumérique est de taille $\lceil \log_2(n) \rceil + s$ bits, où s désigne la taille en bit du symbole à coder. Il s'avère donc particulièrement efficace lorsque le texte à compression contient des plages de caractères consécutifs identiques. Par la suite, on va donc chercher à transformer (de manière inversible) le texte à compresser en un autre texte admettant une telle structure.

6.1.2 Autour de bzip2 : la transformée de Burrows–Wheeler

L'algorithme de compression de données bzip2 a été développé à la fin des années 1990 et est actuellement très populaire, notamment sous UNIX avec l'extension .bz2. Il utilise une méthode de compression par transformée, couplée à du codage par plage et du codage à longueur variable.

Transformée de Burrows–Wheeler. L'algorithme de Burrows–Wheeler, qui est au cœur de bzip2, transforme une chaîne de caractères x en une autre chaîne y essentiellement de même taille, telle que certains caractères identiques et éloignés dans x deviennent proches dans y .

Pour définir la transformée de Burrows–Wheeler, on se donne d'abord un ordre sur l'alphabet \mathcal{X} de la source. Par exemple, pour $\mathcal{X} = \{a, b, c, \dots, y, z\}$, on peut simplement choisir l'ordre alphabétique. On définit également une fonction auxiliaire *rotate* qui prend en entrée une chaîne $x = x_0 \dots x_{m-1} \in \mathcal{X}^m$ et un entier $i \in \{0, \dots, m-1\}$, et qui retourne la chaîne x obtenue après i permutations cycliques :

$$\text{rotate}(x, i) = x_i \dots x_{m-1} x_0 \dots x_{i-1}.$$

On a aussi à notre disposition un algorithme de tri, que l'on nomme ici *sort*, qui prend en entrée une table de chaînes de caractères, et retourne la table triée selon l'ordre issu de celui de \mathcal{X} .

À l'aide de ces procédures, on peut alors définir la transformée de Burrows–Wheeler, voir l'Algorithme 5. Remarquons que l'algorithme proposé ici est peu efficace à la fois en temps et en mémoire : son coût est au moins quadratique en la taille de l'entrée. Néanmoins, il en existe des versions plus élaborées et pratiques.

Algorithme 5 : Transformée de Burrows–Wheeler

Entrée : une chaîne de caractère $x = x_0 \dots x_{m-1} \in \mathcal{X}^m$.

Sortie : un entier $n \in \{0, \dots, m-1\}$ et une chaîne de caractères $y = y_0 \dots y_{m-1} \in \mathcal{X}^m$.

- 1 $\text{Tab} \leftarrow [\text{rotate}(x, 0), \dots, \text{rotate}(x, m-1)]$
 - 2 $\text{SortedTab} \leftarrow \text{sort}(\text{Tab})$
 - 3 $y \leftarrow [\text{SortedTab}[0][m-1], \dots, \text{SortedTab}[m-1][m-1]]$
 - 4 Trouver $n \in \{0, \dots, m-1\}$ tel que $\text{SortedTab}[n] = x$
 - 5 Retourner n et y
-

Exemple 6.4

Considérons la chaîne de caractère $x = \text{ABRACADABRA}$. L'Algorithme 5 construit d'abord la table Tab puis la trie par ordre alphabétique dans SortedTab , où

Tab =	ABRACADABRA BRACADABRAA RACADABRAAB ACADABRAABR CADABRAABRA ADABRAABRAC DABRAABRACA ABRAABRACAD BRAABRACADA RAABRACADAB AABRACADABR	et SortedTab =	AABRACADABR ABRAABRACAD ABRACADABRA ACADABRAABR ADABRAABRAC BRAABRACADA BRACADABRAA CADABRAABRA DABRAABRACA RAABRACADAB RACADABRAAB
-------	---	----------------	---

L'algorithme retourne alors ($n = 2, y = \text{RDARCAAAAABB}$). On remarque notamment que 4 lettres A sont adjacentes.

La transformée de Burrows–Wheeler admet une procédure d'inversion qu'on donne dans l'Algorithme 6. Une nouvelle fois, l'algorithme présenté dans ce cours est à visée pédagogique et peu efficace en pratique.

Algorithme 6 : Inversion de la transformée de Burrows–Wheeler

Entrée : un entier $n \in \{0, \dots, m-1\}$ et une chaîne de caractères $y = y_0 \dots y_{m-1} \in \mathcal{X}^m$.

Sortie : une chaîne de caractère $x = x_0 \dots x_{m-1} \in \mathcal{X}^m$.

- 1 $\text{Tab} \leftarrow [y_0, y_1, \dots, y_{m-1}]$
 - 2 $\text{Tab} \leftarrow \text{sort}(\text{Tab})$
 - 3 **Pour tout** $i \in \{1, \dots, m-1\}$ **faire**
 - 4 **Pour tout** $j \in \{0, \dots, m-1\}$ **faire**
 - 5 Ajouter y_j au début de la chaîne $\text{Tab}[j]$
 - 6 $\text{Tab} \leftarrow \text{sort}(\text{Tab})$
 - 7 $x \leftarrow \text{Tab}[n]$
 - 8 Retourner x
-

Procédure *move-to-front*. En sortie d'une transformée de Burrows–Wheeler, on espère observer des motifs de lettres semblables et fréquents. Avant de procéder à un codage par plage, l'idée est de transformer ces motifs en « petits nombres » grâce à la procédure *move-to-front*. Pour cela, on définit d'abord une fonction auxiliaire `movefirst` de sortie

$$\text{movefirst}(T, i) = [T_i, T_0, T_1, \dots, T_{i-1}, T_{i+1}, \dots, T_{m-1}]$$

qui déplace l'élément T_i d'une table $T = [T_0, \dots, T_{m-1}]$ en première position. On peut ensuite définir la procédure *move-to-front* comme dans l'Algorithme 7.

Algorithme 7 : Algorithme *move-to-front*

Entrée : une chaîne de caractère $x = x_0 \dots x_{m-1} \in \mathcal{X}^m$.

Sortie : une suite d'entiers $a_0, \dots, a_{m-1} \in \{0, \dots, N-1\}^m$ où $N = |\mathcal{X}|$

```

1 Tab ← [a0, ..., a|\mathcal{X}|-1]; // toutes les lettres de l'alphabet \mathcal{X}
2 res ← []
3 Pour tout i ∈ {0, ..., m-1} faire
4   Trouver j ∈ {0, ..., m-1} tel que Tab[j] = xi
5   res ← res + [j]
6   Tab ← movefirst(Tab, i)
7 Retourner res
```

L'algorithme *move-to-front* ne compresse pas la taille de son entrée x . Il permet de transformer des motifs de lettres successives de x en petits nombres. Réciproquement, aux lettres peu fréquentes vont être assignés des nombres plus grands. En particulier, *move-to-front* remplace des séquences de lettres consécutives par des séquences de zéros.

Exemple 6.5

Reprenons la sortie $y = \text{RDARCAAAABB}$ de l'Exemple 6.4. Supposons que $\mathcal{X} = \{A, B, C, \dots, Z\}$ soit ordonné de manière alphabétique. Alors, *move-to-front* produit la séquence suivante : (17, 4, 2, 2, 4, 2, 0, 0, 0, 4, 0).

Voici le détail des mises à jour de `Tab` et `res` :

```

Tab = ABCDEFGHIJKLMNOPQRSTUVWXYZ; res[0] = 17 (position de R)
Tab = RABCDEFGHIJKLMNQPSTUVWXYZ; res[1] = 4 (position de D)
Tab = DRABCEFGHIJKLMNOPQRSTUVWXYZ; res[2] = 2 (position de A)
Tab = ADRBCEFGHIJKLMNOPQRSTUVWXYZ; res[3] = 2 (position de R)
Tab = RADBCEFGHIJKLMNOPQRSTUVWXYZ; res[4] = 4 (position de C)
Tab = CRADBCEFGHIJKLMNOPQRSTUVWXYZ; res[5] = 2 (position de A)
Tab = ACRDBEFGHIJKLMNOPQRSTUVWXYZ; res[6] = 0 (position de A)
Tab = ACRDBEFGHIJKLMNOPQRSTUVWXYZ; res[7] = 0 (position de A)
Tab = ACRDBEFGHIJKLMNOPQRSTUVWXYZ; res[8] = 0 (position de A)
Tab = BACRDEFGHIJKLMNOPQRSTUVWXYZ; res[9] = 2 (position de B)
Tab = BACRDEFGHIJKLMNOPQRSTUVWXYZ; res[10] = 0 (position de B)
```

L'algorithme *move-to-front* permet une compression encore meilleure lorsqu'il est couplé avec un bon code de longueur variable sur les entiers. On utilisera par exemple un code de Huffman.

Bibliographie

- [CL22] Antoine Chambert-Loir. *Théorie de l'information — trois théorèmes de Claude Shannon*. Calvage et Mounet, 2022.
- [CT01] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2001.
- [DRVET18] Jean-Guillaume Dumas, Jean-Louis Roch, Sébastien Varrette, and Éric Tannier. *Théorie des codes : compression, cryptage, correction*. Dunod, 2018.
- [HP10] William Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2010.
- [Mac03] David JC MacKay. *Information Theory, Inference, and Learning Algorithms*. 2003.
- [Rio07] Olivier Rioul. *Théorie de l'information et du codage*. 2007.
- [Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3) :379–423, 1948.