Université Paris 8 Année 2021–2022

Master Mathématiques et applications

Parcours ACC (Arithmétique, Codage et Cryptologie)
Parcours CSSD (Cyber Sécurité et Sciences des Données)

Théorie de l'information

Notes de cours

Julien Lavauzelle
julien.lavauzelle@univ-paris8.fr

15 novembre 2021

Table des matières

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

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.

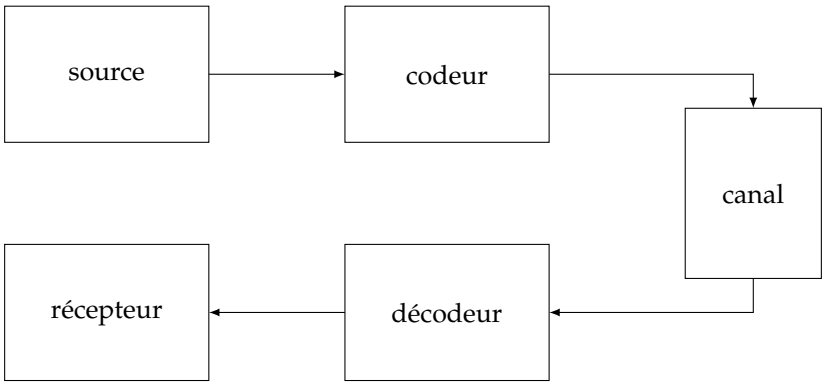


FIGURE 1.1 – Modèle générique d'un système de communication

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.

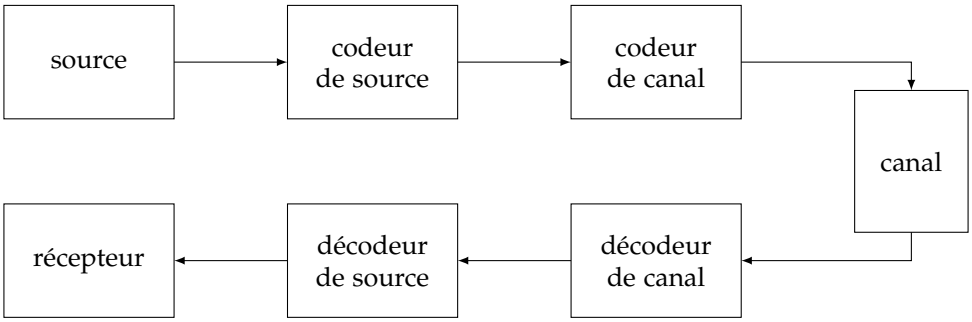


FIGURE 1.2 – Modèle d'un système de communication avec distinction des codeurs/décodeurs

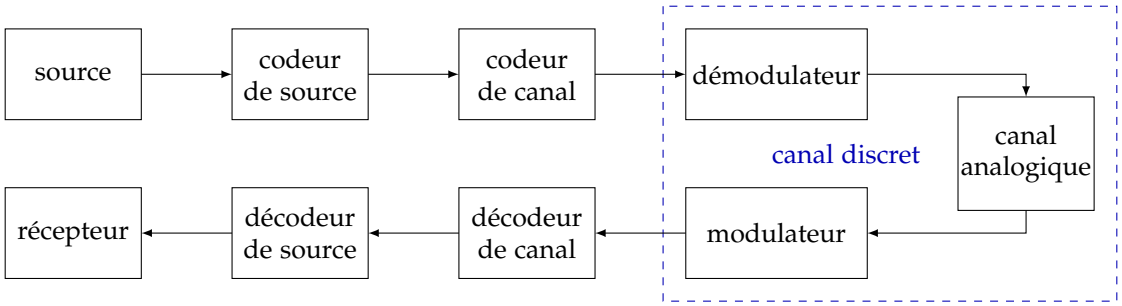


FIGURE 1.3 – Modèle raffiné d'un système de communication

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 les communications, la théorie de l'information trouve également des applications en cryptographie et en stockage.

Secret, cryptographie. Une application originelle de la théorie de l'information introduite par Shannon [Sha48] est la sécurité des communications chiffrées. Informellement, l'idée sous-jacente à beaucoup de systèmes de chiffrement est la suivante : on incorpore au message à envoyer des éléments aléatoires afin de le rendre inintelligible à toute personne n'ayant pas accès à un secret connu seulement de l'émetteur et/ou du récepteur.

Dans notre contexte, on utilisera la théorie de l'information comme un moyen d'affirmer ou de contester la sécurité d'un système de chiffrement. Par des moyens théoriques, Shannon a notamment démontré que le chiffrement de Vernam (ou *masque jetable*), dans lequel une clé aléatoire de taille aussi grande que le message lui est combinée pour former le chiffré, est inconditionnellement sûr.

On s'intéressera également au problème de l'accès confidentiel à une information stockée à distance.

Compression de données. La théorie de l'information propose aussi des outils et des limites à la compression de données. Elle permet notamment de déterminer des bornes la taille minimale de données compressées sans perte, et fournit des techniques algorithmiques pour produire une compression (presque) optimale.

Chapitre 2

Grandeurs en théorie de l'information

Dans ce chapitre, nous introduisions 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

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 $p : \Omega \to [0,1]$ 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 :*

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

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

Si p est une loi de probabilité d'un espace probabilisé (Ω, p), alors on peut prolonger p sur les évènements composés (les parties) de Ω en définissant :

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

Par définition, on obtient alors les résultats suivants.

Proposition 2.2 (*Formule d'inclusion-exclusion*)

Soit (Ω, p) *un espace probabilisé et* $A_1, A_2, \ldots, A_n$ *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 :

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

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

Corollaire 2.3

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

Définition 2.4 (*variable aléatoire*)

Une variable aléatoire *est une application $X : \Omega \to \mathcal{X}$, 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 $p_X : \mathcal{X} \to [0,1]$ 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.

Définition 2.5

La distribution *d'une loi p d'un espace probabilisé (Ω, p) est la donnée de $(p(\omega))_{\omega\in\Omega}$.*

Définition 2.6

*L'*espérance *d'une variable aléatoire réelle $X : \Omega \to \mathcal{X} \subset \mathbb{R}$ 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 de X pondérée par sa loi de probabilité, tandis que la variance mesure la dispersion de X autour de sa moyenne. Notons que également que la variance est positive ou nulle (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 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 $\lambda(1-\lambda)$.

Définition 2.8 (*Loi conditionnelle*)

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

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

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

Définissons pour $A, B \subseteq \Omega$ la quantité $p(A \mid B) \coloneqq \sum_{\alpha \in A} p(\alpha \mid B)$. On déduit alors les résultats suivants.

Proposition 2.9

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

1. $p(A \mid B) = \frac{p(A \cap B)}{p(B)}$,
2. *Formule de Bayes :* $p(B \mid A) = p(A \mid B)\frac{p(B)}{p(A)}$.

Informellement, deux évènements sont indépendants si la probabilité d'occurence de l'un ne dépend pas de l'autre. En voici une définition formelle.

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

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

Autrement dit, la connaissance de B n'apporte pas d'information sur A.

Définition 2.11 (*Loi conjointe*)

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

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

La variable aléatoire $\Omega \to \mathcal{X} \times \mathcal{Y}$ 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'occurence simultanée d'évènements associés aux variables aléatoires X et Y.

On définit également la loi conditionnelle, $p_{X|Y}(x \mid y) \coloneqq p(\{X = x\} \mid \{Y = y\})$, ainsi que sa variable associée $X \mid Y : \Omega \to \mathcal{X} \times \mathcal{Y}$ de loi $p_{X|Y}$.

Proposition 2.12

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

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

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 évèvement soit grande lorsque l'évènement est improbable,

et faible lorsqu'il est presque certain. On doit donc choisir une fonction décroissante de la probabilité d'occurence de d'évènement.

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

Définition 2.13

*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 2.14

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'occurence de x apporte beaucoup d'information. Au contraire, si l'évènement est certain (par exemple $p(x) = 1$), alors son occurence n'apporte aucune information.

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

Définition 2.15 (*Entropie de Shannon*)

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

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

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.

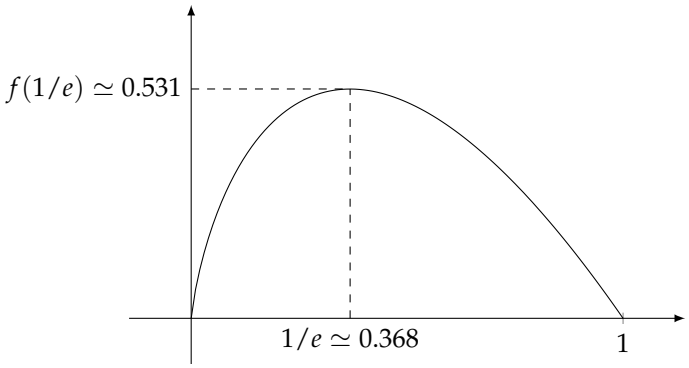


FIGURE 2.1 – La fonction $f : t \mapsto -t\log_2 t$ sur $[0,1]$.

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'occurence des $x_i \in \mathcal{X}$. Alors, la fonction entropie $h : (p_1,\ldots,p_n) \mapsto -\sum_{i=1}^n p_i\log_2 p_i$ satisfait les axiomes suivants :

A1. l'application $h : (p_1, \ldots, p_n) \mapsto -\sum_{i=1}^{n} p_i \log_2 p_i$ 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 $\big(h(1/n, \ldots, 1/n)\big)_{n\geq 1}$ est strictement croissante,
A4. si $(p_1, \ldots, p_n)$ et $(q_1, \ldots, q_m)$ sont deux distributions, alors

$$h(p_1q_1, \ldots, p_1q_m, p_2, \ldots, p_n) = h(p_1, \ldots, p_n) + p_1 h(q_1, \ldots, q_m)\,.$$

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'occurence 0) est nulle.

Exemple 2.16

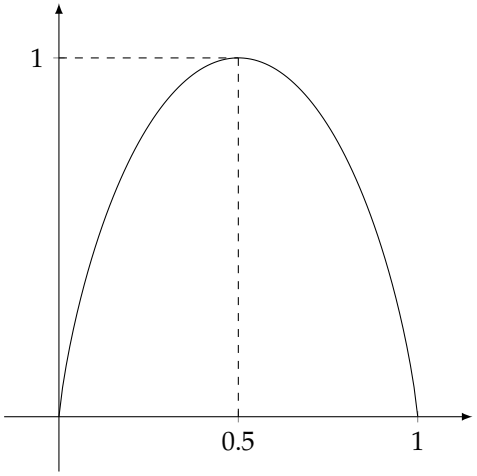
L'entropie d'un lancer de pièce idéal est 1. *Celle de N lancers consécutifs est N. L'entropie d'un lancer de dé à* 6 *faces est* $\log_2(6) \simeq 2.58$.

Exemple 2.17

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 est fondamentale en théorie de l'information. En voici un graphe :



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 2.18

Soit $X : \Omega \to \mathcal{X}$ 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 $p(x_1) = \cdots = p(x_n) = 1/n$.

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 $\log_2$:

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

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 2.18, on utilise la propriété de concavité (stricte) de la fonction $\log_2$. Par la suite, nous aurons besoin de cette notion à plusieurs occasion ; donnons-en une définition.

Définition 2.19

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 $x \in]a,b[$.

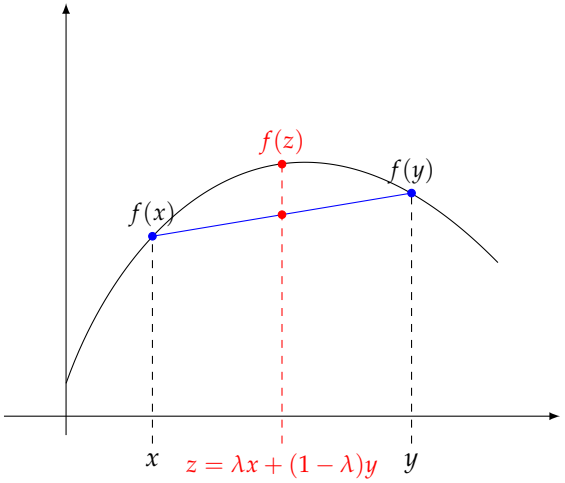


FIGURE 2.2 – Exemple de fonction concave

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 $(x_1,\ldots,x_n),(y_1,\ldots,y_n)$ distincts dans $\mathcal{A}$ et pout tout $\lambda \in]0,1[$, on a

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

Lemme 2.20

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

— *Pour tout $m \geq 1$, pour tous points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ de $\mathcal{A}$, et pour tous coefficients $\lambda_1, \ldots, \lambda_m > 0$ tels que $\sum_{i=1}^m \lambda_i = 1$, on a*

$$f\Big(\sum_{i=1}^m \lambda_i \boldsymbol{x}_i\Big) \geq \sum_{i=1}^m \lambda_i f(\boldsymbol{x}_i)\,. \tag{2.1}$$

— *Il y a égalité dans (2.1) si et seulement si les points sont égaux : $\boldsymbol{x}_1 = \cdots = \boldsymbol{x}_m$.*

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

Proposition 2.21

La fonction d'entropie de Shannon $h : (x_1, \ldots, x_n) \mapsto -\sum_{i=1}^n x_i \log_2 x_i$ 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 $\lambda_1, \ldots, \lambda_m \in]0,1[^n$ tels que $\sum_{j=1}^m \lambda_i = 1$, on a :

$$h\Big(\sum_{j=1}^m \lambda_j x_{1,j}, \ldots, \sum_{j=1}^m \lambda_j x_{n,j}\Big) \geq \sum_{j=1}^m \lambda_j h(x_{1,j}, \ldots, x_{n,j})\,.$$

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

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

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 2.22

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 :

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

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

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

Exemple 2.23

Considérons la distribution jointe 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 $H(X \mid Y = y_1) = 0$ bit car $p(x_1 \mid y_1) = 0$, et que $H(X \mid Y = y_2) = 1$ bit, car $p(x_1 \mid y_2) = p(x_2 \mid y_2)$. Ainsi

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

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 2.24

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

1. *(règle de chaînage) $H(X,Y) = H(Y) + H(X \mid Y)$,*
2. *$H(X \mid Y) \leq H(X)$, 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. *$H(X,Y \mid Z) = H(Y \mid Z) + H(X \mid YZ)$,*
5. *$H(X,Y \mid Z) \geq H(Y \mid Z)$.*

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

Démonstration : 1.

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

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 intevalle. Comme $\sum_{y\in\mathcal{Y}} p_y = 1$, on obtient :

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

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

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

   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 $X \mid Y$ ne dépend pas de Y, c'est-à-dire $H(X \mid Y) = H(X)$.
3. On sait que $H(Y \mid X) \geq 0$ d'après la Proposition 2.18. 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 $H(X,Y) = H(Y) + H(X \mid Y) \leq H(Y) + H(X)$.
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

$$H(X,Y,Z) = H(Z) + H(X,Y \mid Z). \tag{2.3}$$

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

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

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 2.25

Il faut prendre garde au fait que l'inégalité $H(X) \geq H(X \mid Y)$ n'est vraie qu'en moyenne. En effet, il peut exister certaines valeurs de $y \in \mathcal{Y}$ telles que $H(X) < H(X \mid \{Y = y\})$, voir Exemple 2.23.

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-création d'information*.

Proposition 2.26 (*Principe de non-création d'information*)

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

$$H(Y \mid X) = 0\,.$$

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 évènements $\{X = x\}$ 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

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

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

Corollaire 2.27

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 $H(X,g(X)) = H(X) + H(g(X) \mid X) = H(X)$. Pour le second point, on poursuit le calcul pour obtenir $H(X) = H(X,g(X)) = H(g(X)) + H(X \mid g(X)) \geq H(g(X))$. ■

Proposition 2.28

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

Démonstration : Par hypothèse, $0 = H(Y \mid X) = -\sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p_{x,y} \log_2 p_{y|x}$, 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 $g : \mathcal{X} \to \mathcal{Y}$ 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 2.29

Le second point du Corollaire 2.27 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-création d'information est une barrière théorique à ce dessein, l'idée est de partir d'une petite source d'aléa $X : \Omega \to \mathcal{X}$ que l'on suppose bonne (la graine), puis d'utiliser une fonction $g : \mathcal{X} \to \mathcal{Y}$ où $\mathcal{Y}$ est de taille bien plus grande que $\mathcal{X}$, et telle qu'il soit difficile de discerner un tirage de $Y : \Omega \to \mathcal{Y}$ d'un tirage de $g(X) : \Omega \to g(\mathcal{X}) \subset \mathcal{Y}$.
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 2.30

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 2.31 (*Information mutuelle*)

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

$$I(X\,;Y) \coloneqq H(X) - H(X \mid Y)\,.$$

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

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

Il en découle les résultats suivants.

Proposition 2.32

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 : Laissée 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 2.33

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 :*

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

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* $D_{\mathrm{KL}}(p \,\|\, q) = \infty$.

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 2.34 (*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

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

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 2.35

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

1. $D_{\mathrm{KL}}(p \,\|\, q) \geq 0$,
2. $D_{\mathrm{KL}}(p \,\|\, q) = 0$ *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 $D_{\mathrm{KL}}(p \,\|\, q) = 0$. Réciproquement, par stricte concavité de la fonction logarithme, il y a égalité dans $D_{\mathrm{KL}}(p \,\|\, q) \geq 0$ si pour une certaine constante c, on a $q_i = c p_i$ 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

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

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 2.36

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 lettre 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 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 $C : \mathcal{X} \to \{0,1\}^+$. On dit que C est* régulier *s'il est injectif. Le* codage *associé est alors l'application :*

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

Par abus, on parlera également de code sur X, où X est une source sur l'alphabet $\mathcal{X}$.

Lorsque que le code C est régulier, il est théoriquement possible de *décoder* une lettre encodée $C(x)$, c'est-à-dire retrouver la lettre $x \in \mathcal{X}$ correspondant. 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 ssavoir si le mot de code 000 correspond au message *aaa*, *ab* ou *ba*.

Définition 3.2

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

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

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

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

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

3.2 Codes à longueur fixe

Un code de longueur fixe est un code 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, 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 petite. Néanmoins, il existe une limite théorique à cette longueur, en fonction de la taille de l'alphabet à coder.

Proposition 3.3

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 : 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. ■

Proposition 3.4

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

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

Démonstration : Notons n la longueur de C. D'après la Proposition 2.18, 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 3.3, 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 2.18), tandis que la seconde impose que $|\mathcal{X}|$ soit une puissance de 2. ■

Exemple 3.5

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

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*.*

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 3.6

Considérons le code

lettre	*mot de code*	*lettre*	*mot de code*	*lettre*	*mot de code*
a	0	*b*	01	*c*	10

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

Définition 3.7

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

Définition 3.8

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*.*

Exemple 3.9

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.

Les codes préfixes ont l'avantage de former une famille importante et facilement distinguables de codes uniquement décodables.

Proposition 3.10

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

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

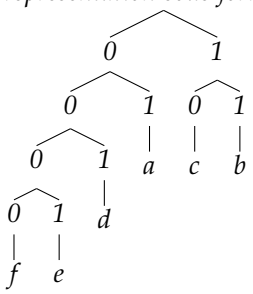
où $x = x_1|\dots|x_n$ et $y = y_1|\dots|y_m$. Supposons, par exemple, que $C(x_1)$ est plus long que $C(y_1)$. Alors, $C(y_1)$ est un préfixe de $C(x_1)$, donc $C(x_1) = C(y_1)$, ce qui implique que $x_1 = y_1$. En procédent itérativement, on montre alors que $n = m$ et $x_i = y_i$ pour tout $i \in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

Attention, la réciproque est fausse : 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, et le mot de code associé se lit sur l'unique chemin qui mène de la racine à la feuille.

Exemple 3.11

Reprenons le code de l'Exemple 3.9. 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 codée par $C(d) = 001$.

Il est clair que la profondeur de l'arbre binaire d'un code correspond à la longueur maximale d'un mot du code.

Dans le cas des codes de longueur fixe, on a démontré une borne simple sur la longueur minimale d'un code (Proposition 3.3). Pour des codes à longueur variable, on peut également se questionner sur les séquences de longueurs qui sont admissibles. L'inégalité de Kraft fournit un premier critère, dans le cas particulier des codes préfixes.

Theorème 3.12 (*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 distance égale à la racine. Il est clair qu'un arbre parfait dont les feuilles sont à distance d de la racine possède 2^d feuilles.

Lemme 3.13

Soit $\mathcal{T}$ un arbre binaire, 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

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

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'autres 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 3.13, 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 $1 \geq \sum_{j=1}^{k} 2^{-n_j} > \sum_{j=1}^{i-1} 2^{-n_j}$. ■

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

Theorème 3.14 (*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 3.12, 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

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

l'ensemble des séquences binaires obtenues par concaténation d'exactement 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 :

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

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

Définition 3.15

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

Exemple 3.16

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 3.9 l'est aussi.*

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

Corollaire 3.17

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 3.18

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 3.19

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

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

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 3.20

La preuve de la Proposition 3.19 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 3.21

à venir : exemple de code de Shannon-Fano

3.4 Code de Huffman

L'inégalité de Kraft et le théorème de MacMillan nous donnent un critère pour évaluer l'existence de codes préfixes et/ou uniquement décodables, en fonction des longueurs présupposées 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 3.22

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 $\overline{\ell}(C) \leq \overline{\ell}(C')$.*

Nous avons vu précédemment que la méthode donnée en preuve de la Proposition 3.19 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 $p = (p_1, \ldots, p_k)$ et produit l'arbre binaire d'un code préfixe. Sa description est donnée dans l'Algorithme 1.

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

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 $\mathcal{X}$

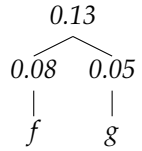
Si $k = 2$
  Retourner l'arbre à deux feuilles p_1, p_2.
Trouver les deux plus petites valeurs p_i et p_j parmi $(p_1, \dots, p_k)$.
Construire une nouvelle distribution $q = (q_1, \dots, q_{k-1})$ formée des $(p_m)_{m \neq i,j}$ et de $p' = p_i + p_j$.
Calculer l'arbre $T = \mathsf{Huffman}(k-1, q)$.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

Exemple 3.23

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

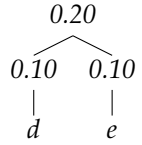
Les lettres f et g sont les moins probables. On construit donc l'arbre



et on définit une nouvelle distribution

x	a	b	c	d	e	f ou g
$p(x)$	0,30	0,25	0,12	0,10	0,10	0,13

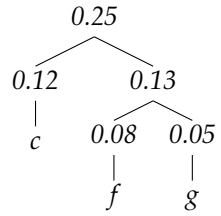
Pour cette nouvelle distribution, les évènements les moins probables sont d et e. On construit donc



et on a la nouvelle distribution

x	a	b	c	d ou e	f ou g
$p(x)$	0,30	0,25	0,12	0,20	0,13

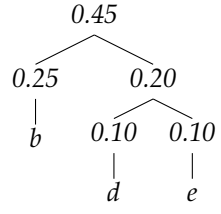
Les deux évènements les moins probables sont ensuite c et « f ou g ». Cela nous mène à construire l'arbre suivant (dans lequel on a remplacé « f ou g » par le sous-arbre correspondant, ce qui diffère légèrement de la présentation récursive de l'algorithme) :



et la distribution devient

x	a	b	d ou e	c ou f ou g
$p(x)$	0,30	0,25	0,20	0,25

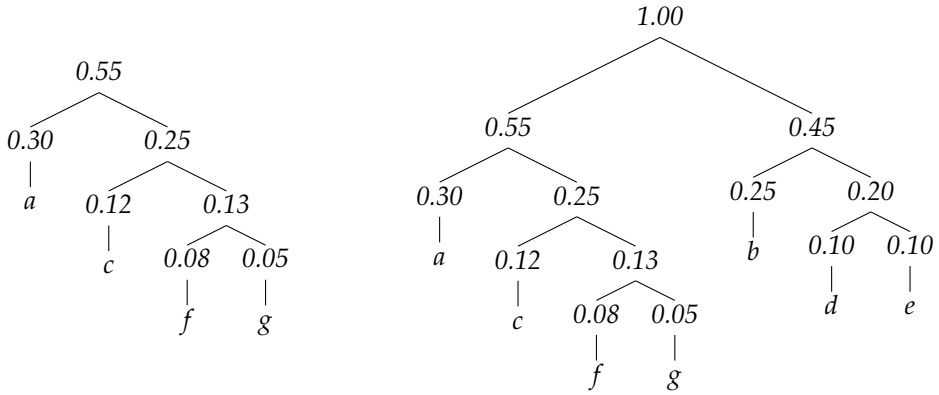
Puis, on peut choisir de manière équivalente les évènements b et « c ou f ou g » à coupler avec « d ou e » pour construire l'arbre suivant. Choisissons la première option :



On modifie donc la distribution en

x	a	b ou d ou e	c ou f ou g
$p(x)$	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 3.24

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 3.25

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 T. 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 3.26

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 3.27

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 3.28

Considérons deux sources X et X', définies respectivement sur $\mathcal{X} = \{x_1, \ldots, x_m\}$ et $\mathcal{X}' = \mathcal{X} \cup \{x_{m+1}\}$. On suppose que la loi p' de X' satisfait $p'(x_1) \geq p(x_2) \geq \cdots \geq p'(x_{m+1})$, et que la loi p vérifie $p'(x_i) = p(x_i)$ pour tout $i \in \{1, \ldots, m-1\}$ 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 :

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

est de longueur moyenne minimale pour X'.

Démonstration : Supposons qu'il existe un code A' tel que $\overline{\ell}(A') < \overline{\ell}(C')$. 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 3.25, 3.26 et 3.27. 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 :

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

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

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

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

Démonstration du Théorème 3.24 : 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 $k-1$ élements.

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 3.28. Par hypothèse de récurrence, l'appel $\mathsf{Huffman}(q, k-1)$ construit un code optimal C pour la source X. Grâce au Lemme 3.28, 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 $P : \mathcal{X} \times \mathcal{Y} \to [0,1]$ telle que :*

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

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

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

réelle de taille $(m \times n)$:

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

où

$$p_{i,j} = p_{Y|X}(y_j \mid x_i)\,.$$

Cette matrice est appelée *matrice de transition* du canal. Son entrée $p_{i,j}$ représente donc la probabilité d'obtenir y_j en sortie du canal lorsque x_i 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 ligne est égale à 1. Si $m = n$, une matrice vérifiant ces propriétés est appelée *matrice stochastique*.

Définition 4.2

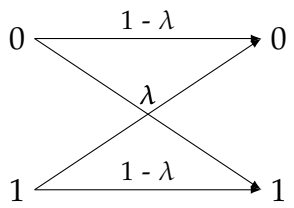
Le canal binaire symétrique (BSC pour binary symmetric channel*) est un canal à entrée et sortie binaires, paramétré par $\lambda \in]0{,}1/2[$, et défini par la matrice de transition suivante :*

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

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

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

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.

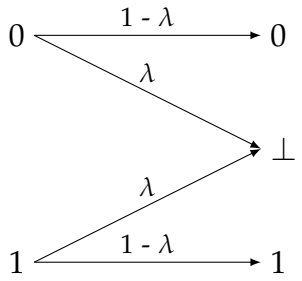


Définition 4.3

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 $(x,y) \mapsto p(y \mid x)$, 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 :

$$I(X;Y) = H(X) - H(X \mid Y)\,.$$

On peut alors interpréter cette relation comme la différence entre $H(X)$, la quantité d'information donnée en entrée du canal, et $H(X \mid Y)$, 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. Il est donc naturel d'introduire une grandeur quantifiant la qualité d'un canal indépendamment de la loi d'entrée.

Définition 4.4

La capacité *d'un canal d'entrée $\mathcal{X}$ et de sortie $\mathcal{Y}$ donné par sa loi de transition $P : \mathcal{X} \times \mathcal{Y} \to [0,1]$, 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 P.*

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

Proposition 4.5

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

$$C_{\mathrm{BSC}(\lambda)} = 1 - h(\lambda)\,,$$

où $h(\lambda) \coloneqq \lambda \log_2 \frac{1}{\lambda} + (1-\lambda)\log_2 \frac{1}{1-\lambda}$ 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 :

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

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

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

Par conséquent, pour maximiser la quantité $I(X;Y) = H(Y) - H(Y \mid X)$, 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 bnaire 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. ■

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 4.6

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

Exemple 4.7

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

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

Son rendement est donc $1/n$.

Définition 4.8

Étant donné un canal d'entrée $\mathcal{X}$, de sortie $\mathcal{Y}$ et de loi de transition P, un algorithme de décodage 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 4.9

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 :*

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

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

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

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 4.10 (*Théorème de Shannon pour les canaux bruités sans mémoire*)

Fixons un canal discret sans mémoire de capacité C.

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

$$\lim_{n\to\infty} R_n \geq R \qquad \text{et} \qquad \lim_{n\to\infty} E_n = 0\,.$$

- *S'il existe une suite de codes en bloc $\mathcal{C}_n$ de longueur n admettant un algorithme de décodage de taux d'erreur résiduel $E_n \to 0$, alors leur rendement R_n vérifie*

$$\limsup R_n \leq C\,.$$

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'intétersser au cas particulier des codes en blocs *linéaires*.

Définition 4.11

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

Remarque 4.12

Par abus de langage, on confond souvent le code en bloc $\phi : \mathbb{F}^k \to \mathbb{F}^n$ 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 :

$$\boldsymbol{G}_n = \left(\begin{array}{c|c} \boldsymbol{I}_k & \boldsymbol{M}_{k,n-k} \end{array} \right)$$

où $\boldsymbol{I}_k$ est la matrice identité, et $\boldsymbol{M}_{k,n-k}$ 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 $\boldsymbol{G}_n$ est k. Les $\mathcal{C}_n$ de matrice génératrice $\boldsymbol{G}_n$ forment alors une séquence de codes qui entrent dans le cadre du premier point du Théorème 4.10 lorsqu'il sont équipés d'un décodeur au maximum de vraisemblance.

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

Définition 4.13

Soit 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 C si, pour tout $\boldsymbol{y} \in \mathcal{X}^n$, la valeur $\boldsymbol{x}$ de retour de l'algorithme maximise strictement la quantité*

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

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

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

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

Définition 4.14

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

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

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

4.3 Codes convolutifs

Les codes convolutifs, introduits par Elias en 1955, forment une classe de codes plus flexible que celle des les 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 télé-communication.

4.3.1 Définition et représentations

L'idée du codage convolutif est de créer, à partir de messages $\boldsymbol{x} \in \{0,1\}^+$, un ensemble de mots $\boldsymbol{c} \in \{0,1\}^+$ 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 4.15

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

$$c_n^{(1)} = x_n + x_{n-1} + x_{n-2},$$
$$c_n^{(2)} = x_n + x_{n-1},$$

où par convention on pose $x_i = 0$ pour $i \leq 0$. On dit alors le code convolutif correspondant 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 4.16

Un code convolutif *de rendement k/n est la donnée de k polynômes $g_1, \dots, g_k \in \mathbb{F}_2[x]$ de degré $\leq n-1$. On note $g_j[i]$ le coefficient de degré i du polynôme g_j. Le code convolutif transforme alors un message $\boldsymbol{x} = (x_0, x_1, \dots, x_m, \dots) \in \{0,1\}^+$ en k mots $\boldsymbol{c}^{(1)}, \dots, \boldsymbol{c}^{(k)} \in \{0,1\}^+$ définis 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 $g_1, \dots, g_k$ 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 $\boldsymbol{a} = (a_0, a_1, \dots, a_m, \dots)$, on définit sa série formelle associée comme

$$A(z) := a_0 + a_1 z + a_2 z^2 + \cdots = \sum_{n \in \mathbb{N}} a_n z^n \in \mathbb{F}_2[[z]]$$

Notons que si $\boldsymbol{a}$ 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 $\boldsymbol{c}^{(1)}, \dots, \boldsymbol{c}^{(k)}$ 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 $\boldsymbol{x}$. 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 4.17

Reprenons le code convolutif proposé dans l'Exemple 4.15. On a alors $g_1(z) = 1 + z + z^2$ et $g_2(z) = 1 + z$. Si l'on choisit comme message $\boldsymbol{x} = (1,0,1,1,0,0,0,\dots)$, alors sa série formelle est $X(z) = 1 + z^2 + z^4$, puis on obtient les polynômes

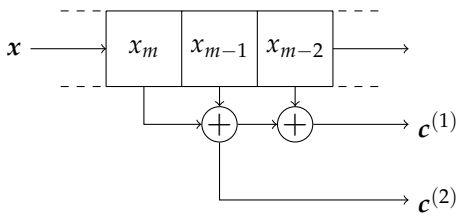
$$\begin{aligned} C^{(1)}(z) &= g_1(z) \cdot X(z) = (1 + z + z^2) \cdot (1 + z^2 + z^3) = 1 + z + z^5, \\ C^{(2)}(z) &= g_2(z) \cdot X(z) = (1 + z) \cdot (1 + z^2 + z^3) = 1 + z + z^2 + z^4, \end{aligned}$$

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

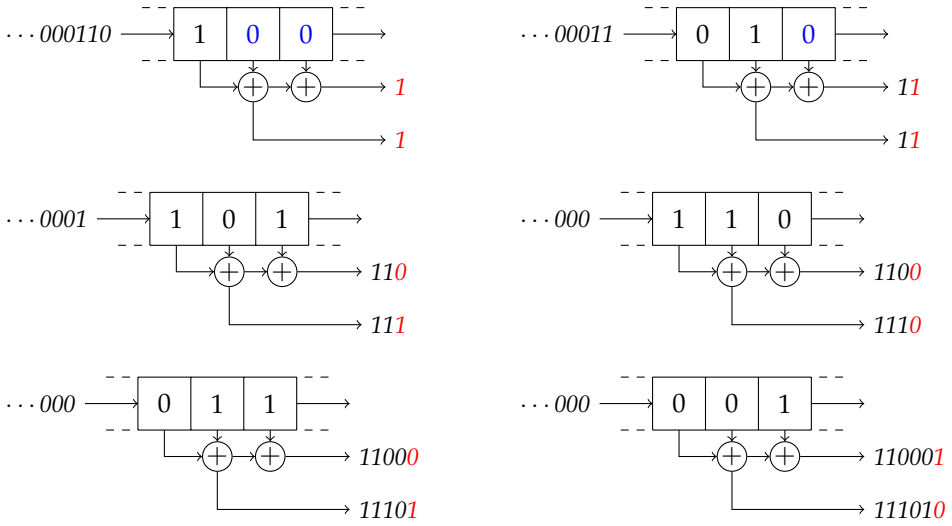
Registre à décalage. Les codes convolutifs peuvent aussi être représentés sous forme de registre à décalage. Dans cette représentation, les bits du message $\boldsymbol{x} \in \{0,1\}^+$ passent successivement dans des registres, dont le nombre est égal la longueur de contrainte. À chaque entrée d'un 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 4.18

Pour le code proposé dans l'Example 4.15, on obtient la représentation suivante.



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



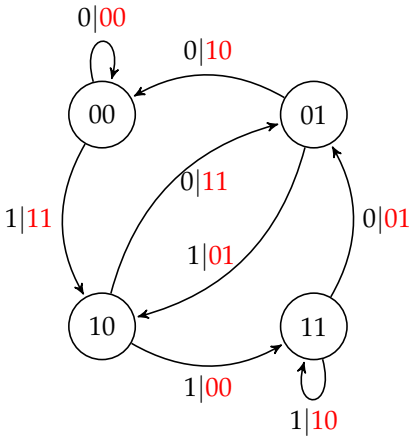
On retrouve bien les séquences de parité obtenues dans l'Exemple 4.17.

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.

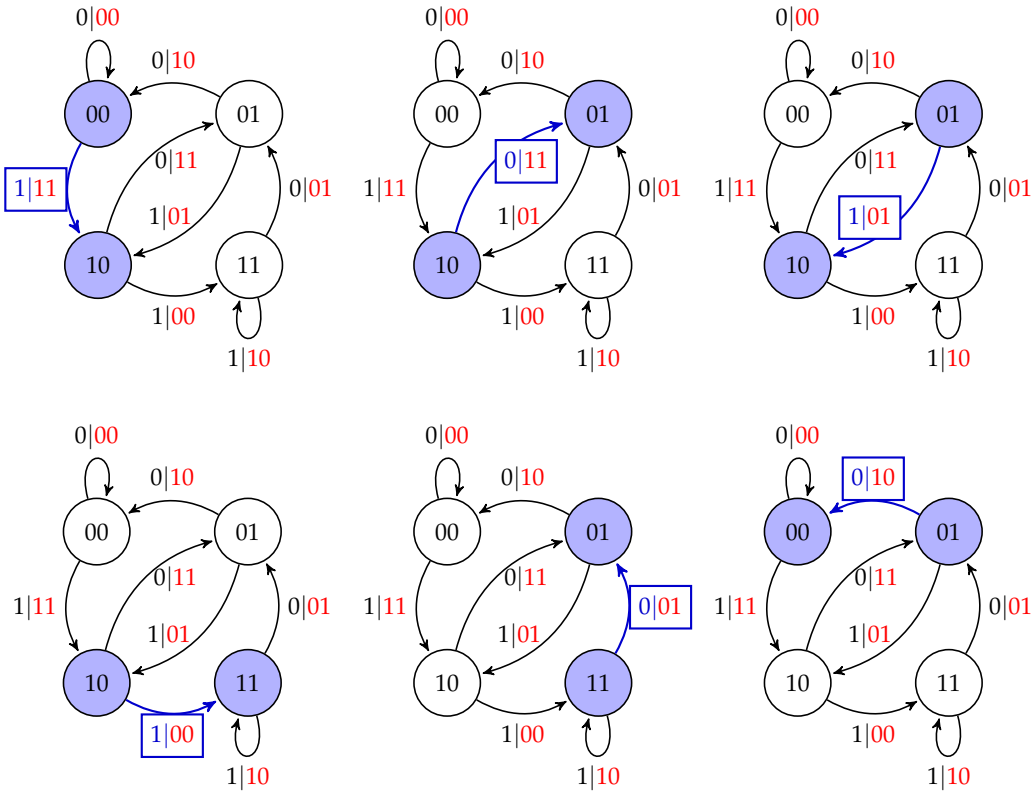
Pour un code convolutif à k générateurs et de longueur de contrainte n, on aura 2^{n-1} nœuds, chacun correspondant à un état possible des $n-1$ derniers bits $(x_{m-1},\dots,x_{m-n+1})$ du registre. Chaque nœud a 2 arêtes sortantes. Ces arêtes ont pour étiquette « $x_m|c_m^{(1)}\dots c_m^{(k)}$ », et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

Exemple 4.19

Pour le code proposé dans l'Example 4.15, 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 consituent les nouveaux bits à ajouter aux séquences de parité.*
Dans le cas de $\boldsymbol{x} = (1,0,1,1,0,0,0,\dots)$*, 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

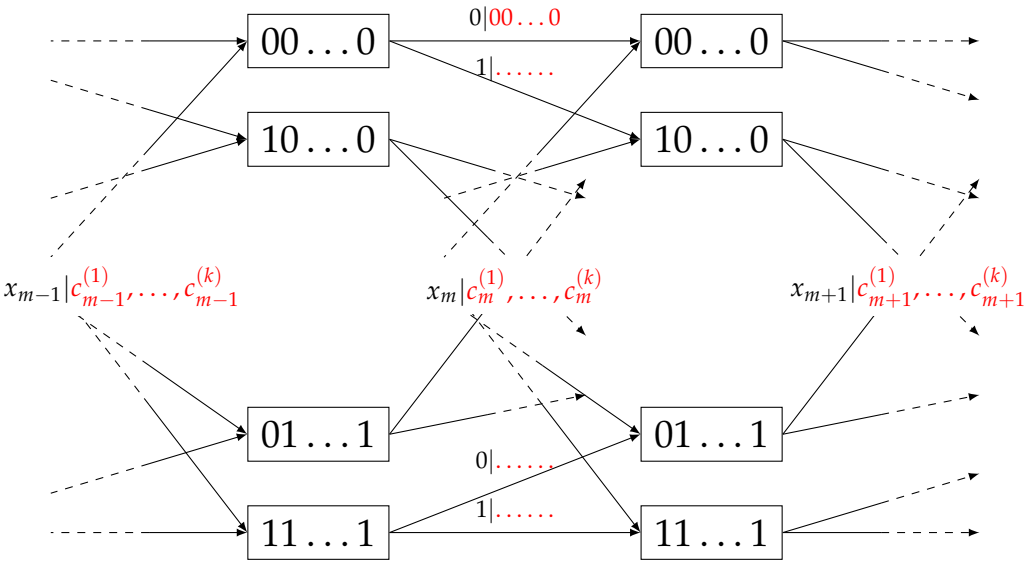
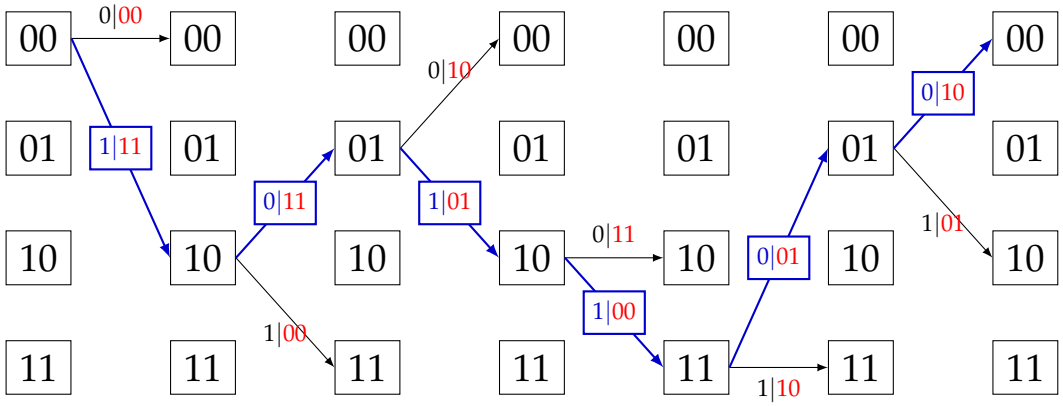


FIGURE 4.1 – Tranche de treillis d'un code convolutif.

correpond à une suite d'arêtes partant de l'état $00\ldots0$ du temps 0, et allant vers la droite. L'Exemple 4.20 en propose une illustration.

Exemple 4.20

Reprenons une nouvelle fois le code de l'Example 4.15.



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. Sa complexité en temps et en mémoire 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 atteignants la valeur minimale.

Algorithme 2 : Algorithme de Viterbi

Entrée : des mots bruités $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(k)} \in \{0,1\}^+$, une longueur maximale M
Sortie : un message $\boldsymbol{x} \in \{0,1\}^+$ et un poids d'erreur w

Pour tout *état* i **faire**
  Pour tout $m = 1, \ldots, M$ **faire**
    $\texttt{weight}[i][m] \leftarrow \infty$
$\texttt{weight}[\texttt{etat\_initial}][0] = 0$
Pour tout $m = 1, \ldots, M$ **faire**
  Pour tout *état* i **faire**
    Chercher une arête $j \to i$ d'étiquette $a|t_1 \ldots t_k$ qui minimise la somme
    $$s_{i,j,m} = \texttt{weight}[j][m-1] + d_H([t_1 \ldots t_k], [y_m^{(1)} \ldots y_m^{(k)}])$$
    $\texttt{Prec}[i] \leftarrow [j, a]$
    $\texttt{weight}[i][m] \leftarrow s_{i,j,m}$
Chercher un i tel que $\texttt{weight}[i][M]$ est minimal
$w \leftarrow \texttt{weight}[i][M]$
$\texttt{res} \leftarrow \epsilon$
Pour tout $m = 1, \ldots, M$ **faire**
  $\texttt{res} \leftarrow \texttt{Prec}[i][1] \,|\, \texttt{res}$
  $i \leftarrow \texttt{Prec}[i][0]$
Retourner res et w

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 cherche un chemin non trivial de poids minimal entre l'état $00\ldots0$ et lui-même dans l'automate du code. Dans le cas de l'Exemple 4.19, on obtient une distance libre égale à 4.

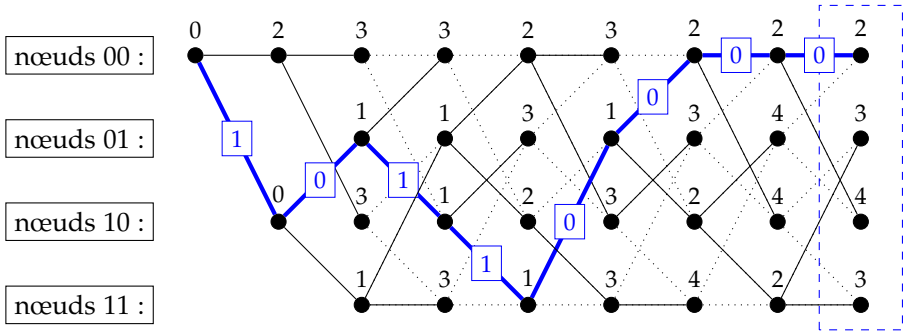
Remarque 4.21

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 4.22

Reprenons l'Exemple 4.15 avec le message $\boldsymbol{x} = (1011000\cdots)$ encodé en la paire de séquences $\boldsymbol{c}^{(1)} = (1100010\cdots)$ et $\boldsymbol{c}^{(2)} = (1110100\cdots)$.

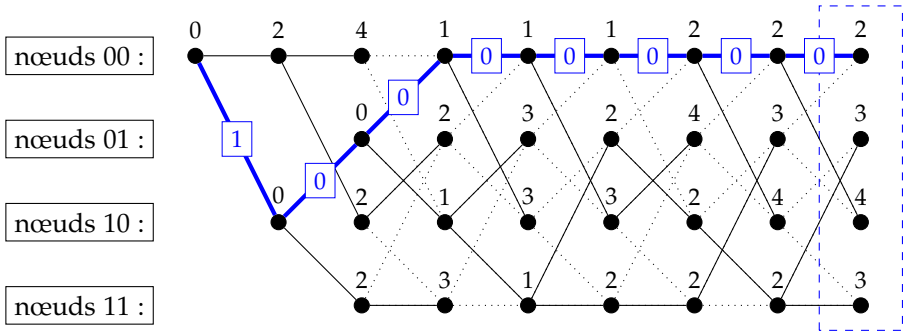
1. *Dans un premier temps, on suppose qu'après passage dans le canal, on reçoit la paire de séquences bruitées $\boldsymbol{y}^{(1)} = (1100010\cdots)$ et $\boldsymbol{y}^{(2)} = (1010110\cdots)$. Dans la figure suivante, on présente les chemins construits successivement par l'algorithme de Viterbi. Lorsqu'un choix parmi deux arêtes doit être fait, on garde celle dont le nœud de départ est le premier dans l'ordre lexigcographique (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 $00 \to 00$.

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 $\boldsymbol{x} = (10110000\cdots)$. Remarquons que le poids du chemin correspond au nombre de bits de $\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}$ qui ont été corrompus par le canal.

2. *Dans un second temps, on suppose avoir reçu la paire de mots bruités $\boldsymbol{y}^{(1)} = (1110010\cdots)$, $\boldsymbol{y}^{(2)} = (1110000\cdots)$. Notons que cette paire de mots est également à distance 2 de la paire de mots $\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}$: 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 $\boldsymbol{x}' = (1, 0, 0, 0, \ldots,)$, 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 lexigcographique.*



Chapitre 5

Autour de la compression de données

5.1 Processus stochastiques et codage universel

Dans les chapitres précédents, nous avons fréquemment émis l'hypothèse selon laquelle les sources et canaux sont *sans mémoire*, c'est-à-dire que les symboles qu'ils émettenet 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 évènements passés. On donne le nom de *processus stochastique* à la suite des variables aléatoires (potentiellement dépendantes entre elles) correspondant à une telle source.

Définition 5.1

Un processus stochastique discret *est une suite de variables aléatoires* $(X_n)_{n\in\mathbb{N}}$ *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 sur (Ω, p) est :

— sans mémoire *si pour tout $0 \leq i < n$, les variables X_i et X_n sont indépendantes ;*

— markovienne *si pour tout $n \geq 1$ et tout $(x_i)_{i\in\mathbb{N}}$, on a*

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

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

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

Ainsi, informellement dans un processus stochastique sans mémoire, le passé n'a aucun influence sur les réalisations actuelles. Dans un processus stochastique markovien, seul l'instant précédent peut éventuellement influer sur le présent, tandis qu'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 $\boldsymbol{X} = (X_0, \dots, X_n, \dots)$ 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 $\boldsymbol{C} = (C_1, \dots, C_n, \dots)$ est markovien. En revanche, il n'est pas stationnaire : clairement C_0 et C_1 n'ont pas la même loi, puisque qu'ils ne prennent pas les mêmes valeurs.

Exemple 5.4

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

1. *La variable aléatoire $Y_n := X_n + X_{n-1}$ définit un processus $\boldsymbol{Y} = (Y_1, \dots, Y_n, \dots)$ 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 $\overline{X}_n := \frac{1}{n+1}\sum_{i=0}^{n} X_i$, pour $n \geq 0$, définit un processus markovien. En effet, on a $\overline{X}_n = \frac{n}{n+1}\overline{X}_{n-1} + \frac{1}{n+1}X_n$.*
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 (resp. stationnaire, resp. markovienne) pour désigner une source dont la séquence de variables aléatoire $\boldsymbol{X} = (X_i)_i$ forme un processus stochastique sans mémoire (resp. stationnaire, resp. markovien).

Remarque 5.5

Un processus stochastique sans mémoire est markovien et stationnaire.

Dans le cas d'une source sans mémoire, par définition l'entropie de la variable conjointe $X_n, \dots, X_0$ est la somme des entropies des variables X_i, pour tout $0 \leq i \leq n$. Ce n'est pas nécessairement le cas pour une source stationnaire $\boldsymbol{X}$, mais on peut définir une notion de taux d'entropie.

Définition 5.6

Le taux d'entropie d'une source stationnaire $\boldsymbol{X}$ est

$$\overline{H}(\boldsymbol{X}) = \lim_{n\to\infty} \frac{1}{n+1} H(X_0, X_1, \dots, X_n)\,,$$

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.

Proposition 5.7

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

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

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

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

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 :

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

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 Cesàro. ∎

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 :

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

∎

Définition 5.9 (*Codes universels*)

Soit $\boldsymbol{X}$ un processus stochastique tel que les X_i, $i \geq 1$, sont à valeur dans un même ensemble $\mathcal{X}$. On considère $\mathcal{C} = (C_n)_{n\geq 1}$ une famille de codes telles 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*

$$\lim_{n\to\infty} \mathbb{E}_{\boldsymbol{X}} \left[\frac{1}{n}\ell(C_n(X_1,\ldots,X_n))\right] = \overline{H}(\boldsymbol{X})$$

pour toutes les sources stationnaires $\boldsymbol{X}$.

La section suivante est consacrée à une construction de codes dont les propriétés s'approchent du caractère universel.

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 le la séquence $\boldsymbol{x} \in \mathcal{X}^m$ 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 $\boldsymbol{x}$ 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 $\boldsymbol{m}$ est de la forme $\boldsymbol{m} = \boldsymbol{c}|a$, où $\boldsymbol{c}$ 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 $\boldsymbol{c}$ dans la suite de mots obtenue ; autrement dit i est une référence vers le préfixe du nouveau mot $\boldsymbol{m}$. La suite de couples obtenue est appelée codage LZW de $\boldsymbol{x}$.

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 5.10

Pour rendre l'écriture plus claire, on considère l'alphabet binaire $\mathcal{X} = \{a, b\}$. Soit $\boldsymbol{x} = (bbaabbabaaaaabababbbbaaabbabb) \in \mathcal{X}^n$ 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 :

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

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

Entrée : une chaîne de caractères $\boldsymbol{x} \in \mathcal{X}^m$
Sortie : une séquence de couples $((i_j, a_j))_j$ où $i_j \in \mathbb{N}$ et $a_j \in \mathcal{X}$

```
res ← []
dict ← [""]
buffer ← ""
i ← 0
Pour tout j ∈ {1,...,m} faire
    Ajouter x_j à la fin de buffer
    Si il existe une clé k tel que buffer = dict[k]
        i ← k
    Sinon
        Ajouter buffer à la fin de dict
        Ajouter (i, x_j) à la fin de res
        Réinitialiser buffer à ""
        Réinitialiser i à 0
Retourner res
```

Remarque 5.11

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

$$\boldsymbol{a}^{(k)} = 0|1|00|01|10|11|000| \cdots \cdots |\underbrace{111\ldots11}_{k \text{ fois}} .$$

On note ensuite $\boldsymbol{c}^{(k)}$ le codage de $\boldsymbol{a}^{(k)}$ 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) :

$$\boldsymbol{c}^{(k)} = [0]_2 0 \mid [0]_2 1 \mid [1]_2 0 \mid [1]_2 1 \mid [2]_2 0 \mid [2]_2 1 \mid [3]_2 0 \mid \ \cdots \cdots \mid [(k-2)2^k - 2]_2 1 .$$

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

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

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

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

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

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

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

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

Remarquons enfin que si une chaîne $\boldsymbol{b}$ de longueur $n \geq \ell(\boldsymbol{a}^{(k)})$ est telle que le codage LZW $\boldsymbol{y}$ de $\boldsymbol{b}$ ne contient pas tous les mots de longueurs $\leq k$, alors on peut remplacer un des mots de longueur $> k$ dans $\boldsymbol{y}$, 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 $\boldsymbol{y}$ (pour un certain $k' \leq k$), et on remarque qu'à chaque étape le ratio $\ell(\boldsymbol{y})/\ell(\boldsymbol{b})$ décroît. Le mot $\boldsymbol{a}^{(k)}$ 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 $\ell(\boldsymbol{c}^{(k)})/\ell(\boldsymbol{a}^{(k)})$ 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 3.3 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 $\boldsymbol{X}^n = (X_1, \ldots, X_n)$ provient d'un processus stochastique stationnaire. Dans ce cas, le taux d'entropie $\overline{H}(\boldsymbol{X})$ peut bien être défini. Il faut également supposer que les statistiques de la source (par exemple, son entropie) 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.

Proposition 5.12

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

$$\lim_{n\to+\infty} \frac{1}{n}\ell(C(\boldsymbol{x}^n)) = \overline{H}(\boldsymbol{X})\,.$$

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*).

5.3 Compression sans perte

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

Bien entendu, il existe une limite théorique à la taille qu'on peut espérer pour $\boldsymbol{y}$. D'un point de vue algorithmique, on peut considérer l'ensemble de tous les programmes P qui permettent de construire $\boldsymbol{x}$. 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 $\boldsymbol{x}$,

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

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

Remarque 5.13

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 $\boldsymbol{X}^+$ de toutes les chaînes de caractères, on souhaite associer à certaines chaînes $\boldsymbol{x}$ (les chaînes les plus « structurées ») un unique mot $\boldsymbol{c}$ tel que $\ell(\boldsymbol{c}) < \ell(\boldsymbol{x})$.

5.3.1 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.

Algorithme 4 : Algorithme RLE de codage par plages (*run-length encoding*)

Entrée : une chaîne de caractère $\boldsymbol{x} = x_0 \dots x_{m-1} \in \mathcal{X}^m$
Sortie : une chaîne de caractères $\boldsymbol{b} \in (\mathbb{N} \times \mathcal{X})^+$

```
b ← ""
cpt ← 1
lettre ← x_0
Pour tout i ∈ {1, ..., m − 1} faire
    Si lettre = x_i
        cpt ← cpt + 1
    Sinon
        b ← b | cpt | lettre
        cpt ← 1
        lettre ← x_i
b ← b | cpt | lettre
Retourner b
```

Remarque 5.14

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 $\boldsymbol{x}$).*

Exemple 5.15

Sur l'entrée $\boldsymbol{x} = (bbaabbabaaaaababbbbaaabbabb)$ de l'Exemple 5.10, 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.

5.3.2 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 $\boldsymbol{x}$ en une autre chaîne $\boldsymbol{y}$ essentiellement de même taille, telle que certains caractères identiques et éloignés dans $\boldsymbol{x}$ deviennent proches dans $\boldsymbol{y}$.

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 $\mathcal{X} = \{a, b, c, \ldots, y, z\}$, on peut simplement choisir l'ordre alphabétique. On définit également une fonction auxiliaire `rotate` qui prend en entrée une chaîne $\boldsymbol{x} = x_0 \ldots x_{m-1} \in \mathcal{X}^m$ et un entier $i \in \{0, \ldots, m-1\}$, et qui retourne la chaîne $\boldsymbol{x}$ obtenue après i permutations cycliques :

$$\texttt{rotate}(\boldsymbol{x}, i) = x_i \ldots x_{m-1} x_0 \ldots x_{i-1} .$$

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 $\boldsymbol{x} = x_0 \ldots x_{m-1} \in \mathcal{X}^m$.
Sortie : un entier $n \in \{0, \ldots, m-1\}$ et une chaîne de caractères $\boldsymbol{y} = y_0 \ldots y_{m-1} \in \mathcal{X}^m$.

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

Exemple 5.16

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

$$\texttt{Tab} = \begin{bmatrix} \texttt{ABRACADABRA} \\ \texttt{BRACADABRAA} \\ \texttt{RACADABRAAB} \\ \texttt{ACADABRAABR} \\ \texttt{CADABRAABRA} \\ \texttt{ADABRAABRAC} \\ \texttt{DABRAABRACA} \\ \texttt{ABRAABRACAD} \\ \texttt{BRAABRACADA} \\ \texttt{RAABRACADAB} \\ \texttt{AABRACADABR} \end{bmatrix} \quad \textit{et} \quad \texttt{SortedTab} = \begin{bmatrix} \texttt{AABRACADABR} \\ \texttt{ABRAABRACAD} \\ \texttt{ABRACADABRA} \\ \texttt{ACADABRAABR} \\ \texttt{ADABRAABRAC} \\ \texttt{BRAABRACADA} \\ \texttt{BRACADABRAA} \\ \texttt{CADABRAABRA} \\ \texttt{DABRAABRACA} \\ \texttt{RAABRACADAB} \\ \texttt{RACADABRAAB} \end{bmatrix} .$$

L'algorithme retourne alors ($n = 2, \boldsymbol{y} =$ `RDARCAAAABB`). *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 $\boldsymbol{y} = y_0 \dots y_{m-1} \in \mathcal{X}^m$.
Sortie : une chaîne de caractère $\boldsymbol{x} = x_0 \dots x_{m-1} \in \mathcal{X}^m$.

```
Tab ← [y_0, y_1, ..., y_{m-1}]
Tab ← sort(Tab)
Pour tout i ∈ {1, ..., m-1} faire
    Pour tout j ∈ {0, ..., m-1} faire
        Ajouter y_j au début de la chaîne Tab[j]
    Tab ← sort(Tab)
x ← Tab[n]
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

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

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 $\boldsymbol{x} = x_0 \dots x_{m-1} \in \mathcal{X}^m$.
Sortie : une suite d'entiers $a_0, \dots, a_{m-1} \in \{0, \dots, N-1\}^m$ où $N = |\mathcal{X}|$

```
Tab ← [a_0, ..., a_{|X|-1}] ;          // toutes les lettres de l'alphabet X
res ← []
Pour tout i ∈ {0, ..., m-1} faire
    Trouver j ∈ {0, ..., m-1} tel que Tab[j] = x_i
    res ← res + [j]
    Tab ← movefirst(Tab, i)
Retourner res
```

L'algorithme *move-to-front* ne compresse pas la taille de son entrée $\boldsymbol{x}$. Il permet de transformer des motifs de lettres successives de $\boldsymbol{x}$ 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 5.17

Reprenons la sortie $\boldsymbol{y} =$ `RDARCAAAABB` *de l'Exemple 5.16. Supposons que* $\mathcal{X} = \{\texttt{A}, \texttt{B}, \texttt{C}, \dots, \texttt{Z}\}$ *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 = RABCDEFGHIJKLMNOPQSTUVWXYZ;  res[1] = 4    (position de D)
Tab = DRABCEFGHIJKLMNOPQSTUVWXYZ;  res[2] = 2    (position de A)
Tab = ADRBCEFGHIJKLMNOPQSTUVWXYZ;  res[3] = 2    (position de R)
Tab = RADBCEFGHIJKLMNOPQSTUVWXYZ;  res[4] = 4    (position de C)
Tab = CRADBEFGHIJKLMNOPQSTUVWXYZ;  res[5] = 2    (position de A)
Tab = ACRDBEFGHIJKLMNOPQSTUVWXYZ;  res[6] = 0    (position de A)
Tab = ACRDBEFGHIJKLMNOPQSTUVWXYZ;  res[7] = 0    (position de A)
Tab = ACRDBEFGHIJKLMNOPQSTUVWXYZ;  res[8] = 0    (position de A)
Tab = BACRDEFGHIJKLMNOPQSTUVWXYZ;  res[9] = 2    (position de B)
Tab = BACRDEFGHIJKLMNOPQSTUVWXYZ;  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

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

[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.