Théorie de l'information et codage
(Notes de cours)

Marie-Pierre Béal* Nicolas Sendrier†

18 décembre 2006

*Institut Gaspard Monge, Université de Marne-la-Vallée, 77454 Marne-la-Vallée Cedex 2, France. beal@univ-mlv.fr.

†INRIA Rocquencourt, B.P. 105 78153 Le Chesnay Cedex, France. nicolas.sendrier@inria.fr.

Table des matières

1 Introduction aux systèmes de communication

1.1 Introduction

La théorie des communications s'intéresse aux moyens de transmettre une information depuis la source jusqu'à un utilisateur à travers un canal. La nature de la source peut être très variée. Il peut s'agir par exemple d'une voix, d'un signal électromagnétique ou d'une séquence de symboles binaires. Le canal peut être une ligne téléphonique, une liaison radio, un support magnétique ou optique. La transmission peut se faire dans l'espace ou dans le temps. Le codeur représente l'ensemble des opérations effectuées sur la sortie de la source avant la transmission. Ces opérations peuvent être par exemple la modulation, la compression, le brouillage, l'ajout de redondance pour combattre les effets du bruit, ou encore l'adaptation à des contraintes de spectre. Elles ont pour but de rendre la sortie de la source compatible avec le canal. Enfin le décodeur doit être capable, à partir de la sortie du canal, de restituer de façon acceptable l'information fournie par la source.

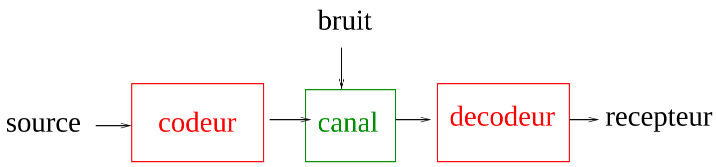

FIG. 1 – Un système de communication

La théorie de l'information a été créée par C. E. Shannon dans les années 40. Il s'agit d'une théorie mathématique qui décrit les aspects les plus fondamentaux des systèmes de communication. Elle consiste en l'élaboration et l'étude de modèles pour la source et le canal qui utilisent différents outils comme les probabilités et les automates finis.

Dans ce cours, nous étudieront certains de ces modèles qui, bien que considérablement plus simples que les sources et les canaux physiques, permettent de donner une bonne approximation de leur comportement.

Pour simplifier, on étudiera séparément les modèles de sources et les modèles de canaux ainsi que leurs codages respectifs.

- Le but du codeur de source est de représenter la sortie de la source en une séquence binaire, et cela de façon la plus économique possible.
- Le but du codeur de canal et de son décodeur est de reproduire le plus fidèlement possible cette séquence binaire malgré le passage à travers

le canal bruité.

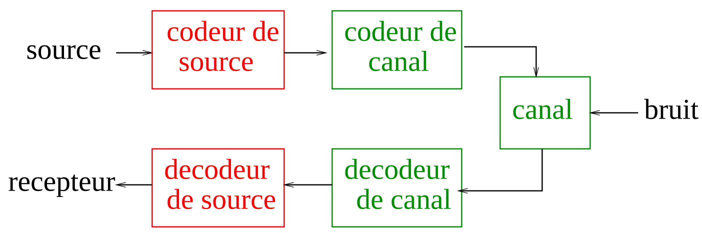

Fig. 2 – Codeur de source et codeur de canal

Cette séparation entre codage de source et codage de canal n'implique en pratique aucune limitation sur les performances du système complet.

1.2 Sources et codage de source

Parmi les classes possibles de modèles de source, nous nous intéresserons plus particulièrement aux sources discrètes sans mémoire.

La sortie d'une telle source est une séquence de lettres tirées dans un alphabet fini $A = \{a_1, \ldots, a_n\}$. Chaque lettre de la séquence est choisie aléatoirement d'après une loi de probabilité p indépendante du temps. Pour toute lettre a, $p(a)$ est la probabilité pour que cette lettre soit choisie. Il s'agit d'un réel compris entre 0 et 1. On a $\sum_{a \in A} p(a) = 1$. La donnée de $p(a_1), \ldots, p(a_n)$ définit la probabilité discrète p sur A.

Il peut sembler étonnant de modéliser une source d'information à l'aide d'une variable aléatoire. Nous allons donner un exemple qui permet de se convaincre de l'utilité de tels modèles.

Exemple : Soit une source d'information qui fournit comme information l'une des quatre lettres a_1, a_2, a_3, a_4. Supposons que le codage de source transforme cette information discrète en symboles binaires. Nous donnons deux exemples de codage différents.

Codage 1	Codage 2
$a_1 \to 00$	$a_1 \to 0$
$a_2 \to 01$	$a_2 \to 10$
$a_3 \to 10$	$a_3 \to 110$
$a_4 \to 11$	$a_4 \to 111$

Si les quatre lettres sont équiprobables, la première méthode de codage est meilleure. Elle nécessite en effet deux symboles par lettre en moyenne tandis que la deuxième méthode nécessite $\frac{1}{4}+2\times\frac{1}{4}+3\times\frac{1}{4}+3\times\frac{1}{4}=2,25$ symboles par lettres. En revanche, si l'on a une source dont la distribution de probabilités est

$$p(a_1)=\frac{1}{2}, p(a_2)=\frac{1}{4}, p(a_3)=p(a_4)=\frac{1}{8},$$

la longueur moyenne d'un symbole codé par la première méthode est toujours 2 tandis que celle d'un symbole codé par la deuxième méthode est $\frac{1}{2}+2\times\frac{1}{4}+3\times\frac{1}{4}+3\times\frac{1}{8}=1,75$. Le deuxième codage réussit donc à coder quatre symboles avec moins de deux bits. Il a réalisé une compression. Pour coder correctement une source, il est donc important de connaître son comportement statistique.

1.3 Entropie d'une source discrète

Nous allons établir un lien entre l'information fournie par une source et la distribution de probabilité de la sortie de cette source. On considère en effet que l'apparition d'un événement peu probable apporte beaucoup d'information tandis que l'occurrence d'un événement certain ne fournit au contraire aucune information.

Si une lettre a a une probabilité $p(a)$ d'être tirée, son information propre est définie par

$$I(a)=-\log_2 p(a).$$

En particulier $I(a)$ vaut zéro si $p(a)=1$. Le choix de la fonction log et de sa base seront expliqués plus loin.

La valeur moyenne de l'information propre calculée sur l'ensemble de l'alphabet revêt une grande importance. Il s'agit donc de l'espérance de la variable aléatoire I. Elle est appelée entropie de la source et est notée $H(A)$:

$$H(A)=-\sum_{a\in A} p(a)\log_2 p(a).$$

Si une source émet n lettres équiprobables (ou encore avec une loi de probabilité uniforme), son entropie est donc $\log_2 n$. Si $n=2^r$, son entropie est alors r. Or pour représenter 2^r lettres distinctes en binaires, r cases sont nécessaires.

L'entropie d'une source est quelquefois donnée en bits/seconde. Si l'entropie d'une source discrète est H et si les lettres sont émises toutes les τ_s secondes, son entropie en bits/s est H/τ_s.

1.4 Autres modèles de source

On peut également distinguer, parmi les classes de modèles de sources, les sources discrètes avec mémoire, finie ou infinie. Une entropie peut être définie pour ces sources de façon analogue.

Enfin les sources non discrètes, ou sources continues, ont une grande importance dans les applications. La sortie d'une telle source sera une fonction continue du temps, par exemple une tension qu'il faut coder par une séquence discrète binaire. La fonction continue doit être décrite le plus fidèlement possible par la séquence binaire générée par le codeur de source. Le problème dans ce cas consiste à minimiser le nombre de symboles transmis pour un niveau de distorsion donné.

1.5 Canaux et codage de canal

Pour modéliser un canal de transmission, il est nécessaire de spécifier l'ensemble des entrées et l'ensemble des sorties possibles. La cas le plus simple est celui du canal discret sans mémoire. L'entrée est une lettre prise dans un alphabet fini $A = \{a_1, \dots, a_n\}$ et la sortie est est une lettre prise dans un alphabet fini $B = \{b_1, \dots, b_m\}$. Ces lettres sont émises en séquence, et, le canal est sans mémoire si chaque lettre de la séquence reçue ne dépend statistiquement que de la lettre émise de même position.

Ainsi un canal discret sans mémoire est entièrement décrit par la donnée des probabilités conditionnelles $p(b|a)$ pour toutes les lettres a de l'alphabet d'entrée et toutes les lettres b de l'alphabet de sortie. Nous allons revenir dans la section suivante sur cette notion de probabilité conditionnelle.

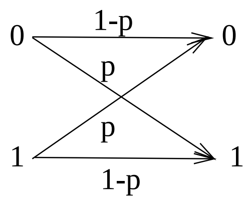

FIG. 3 – Le canal binaire symétrique

Exemple canal discret sans mémoire : Le plus connu est le canal binaire symétrique défini par $A = B = \{0, 1\}$ et dont les probabilités de transition sont représentées Figure 3. La probabilité pour qu'un symbole soit inchangé est $1 - p$, où p est un réel compris entre 0 et 1, et la probabilité pour qu'il soit changé est p.

On peut également considérer des canaux discrets à mémoire dans lesquels chaque lettre de la séquence de sortie peut dépendre de plusieurs lettres de la séquence d'entrée.

1.6 Canaux continus

Il existe une classe de modèles de canaux appelés canaux continus, beaucoup plus proches des canaux physiques. L'entrée et la sortie sont alors des fonctions continues du temps. Pour les canaux de cette classe, il est commode de séparer le codeur et le décodeur en deux parties, comme le montre la figure 4. La première partie du codeur, que nous appellerons codeur de

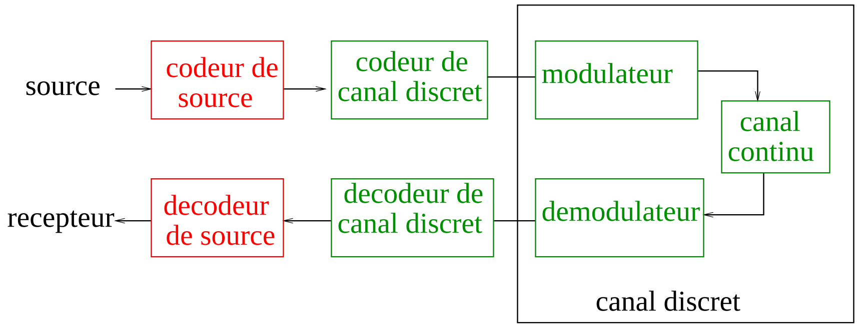

FIG. 4 – Canal continu

canal discret, transforme une séquence binaire en une séquence de lettres d'un alphabet fini $A = \{a_1, \ldots, a_n\}$. La seconde partie du codeur, le modulateur de données digitales, envoie pendant un temps τ_c sur le canal une des fonctions de temps prédéfinies $s_1(t), \ldots, s_n(t)$. La durée τ_c est l'intervalle de temps séparant l'émission de deux lettres par le codeur de canal discret. L'ensemble de ces fonctions du temps mises bout à bout est converti à la sortie du canal par le démodulateur de données digitales en une séquence de lettres d'un alphabet de sortie $B = \{b_1, \ldots, b_m\}$ au rythme, là encore, d'une lettre toutes les τ_c secondes.

1.7 Capacité du canal

Nous verrons que l'un des paramètres les plus importants pour décrire un canal est sa capacité que nous définirons plus tard. On peut montrer que l'on peut transmettre de l'information à travers un canal à n'importe quel taux de transmission inférieur à la capacité avec une probabilité d'erreur arbitrairement faible. Le taux de transmission est le nombre de symboles émis en entrée divisé par le nombre de symboles reçus en sortie.

2 Mesure de l'information

Nous allons donner une mesure de la quantité d'information qui est adaptée à la description statistique des sources et des canaux. Les énoncés qui en résultent font appel aux probabilités discrètes. Nous allons en rappeler les notions principales.

2.1 Espace probabilisé discret

Nous considérerons des ensembles finis munis d'une probabilité discrète p. L'espace probabilisé est noté (A, p). La loi de probabilité est dite uniforme si $p(a) = \frac{1}{n}$, où $n = \mathrm{card}(A)$, pour toute lettre a de A. Une variable aléatoire de (A, p) est une fonction de A dans un ensemble quelconque. Une variable aléatoire est réelle si l'espace d'arrivée est $\mathbb{R}$. L'espérance d'une variable aléatoire réelle v est le réel

$$E(v) = \sum_{a \in A} p(a)v(a),$$

encore appelé moyenne de v.

2.2 Espace probabilisé joint. Probabilités conditionnelles

Pour modéliser un canal discret, nous considérons l'espace $A \times B$ produit des deux ensembles $A = \{a_1, \ldots, a_n\}$ et $B = \{b_1, \ldots, b_m\}$. Le produit est formé des couples (a, b) avec a dans A et b dans B. On munit cet ensemble d'une loi de probabilité discrète, notée p_{AB}, appelée loi de probabilité jointe de A et B. L'espace de probabilité joint est aussi noté AB.

$$\begin{array}{cc} A & B \\ \hline \multicolumn{1}{|c|}{a_i} & \multicolumn{1}{c|}{b_j} \\ \hline \end{array}$$

La probabilité $p_{AB}(a, b)$ est la probabilité d'avoir simultanément a en entrée et b en sortie. On définit une loi de probabilité p_A sur A par

$$p_A(a) = \sum_{b \in B} p_{AB}(a, b).$$

On vérifie que c'est bien une loi de probabilité sur A. On définit une probabilité p_B sur B de façon similaire. Les deux lois p_A et p_B sont appelées lois marginales.

Nous définissons maintenant les lois conditionnelles. Soit a une lettre de A telle que $p(a) > 0$. La probabilité conditionnelle pour que l'on ait b en sortie sachant que l'on a a en entrée est définie par

$$p_{B|A}(b|a) = \frac{p_{AB}(a,b)}{p_A(a)}.$$

On dit également qu'il s'agit de la probabilité conditionnelle pour que l'on ait $\{B = b\}$ sachant que $\{A = a\}$. Notez ici l'abus de notation car A et B désignent ici des variables aléatoires. De façon symétrique, on a

$$p_{A|B}(a|b) = \frac{p_{AB}(a,b)}{p_B(b)}.$$

On dit que les événements $\{A = a\}$ et $\{B = b\}$ sont statistiquement indépendants si $p_{AB}(a,b) = p_A(a)p_B(b)$. Lorsque cette égalité est vraie pour tout couple AB, alors les espaces A et B sont dits statistiquement indépendants. On parle alors d'espace probabilisé produit.

Lorsqu'il n'y aura pas de confusion on notera p toutes les probabilités ci-dessus. Ainsi on notera $p(b|a)$ la probabilité conditionnelle pour que l'on ait $\{B = b\}$ sachant que $\{A = a\}$, et $p(a|b)$ la probabilité conditionnelle pour que l'on ait $\{A = a\}$ sachant que $\{B = b\}$. Attention à ces notations car par exemple si $A = B = \{0, 1\}$, $p_{A|B}(0|1)$ peut très bien être différent de $p_{B|A}(0|1)$.

2.3 Incertitude et information

Avant de donner la définition de la mesure de l'information proposée par Shannon, nous allons essayer de décrire le concept d'information. En suivant le modèle probabiliste, fournir une information à un utilisateur consiste à choisir un événement parmi plusieurs possibles. Qualitativement, fournir une information consiste donc à lever une incertitude sur l'issue d'une expérience aléatoire.

La notion d'information est déjà inhérente à celle de probabilité conditionnelle. Considérons les événements $\{A = a\}$ et $\{B = b\}$. La probabilité $p(a|b)$ peut être interprétée comme la modification apportée à la probabilité $p(a)$ de l'événement $\{A = a\}$ lorsque l'on reçoit l'information que l'événement $\{B = b\}$ s'est réalisé. Ainsi

- si $p(a|b) \leq p(a)$, l'incertitude sur a augmente,
- si $p(a|b) \geq p(a)$, l'incertitude sur a diminue.

Pour mesurer la variation de l'incertitude, il faut choisir une fonction décroissante de la probabilité. On choisit également une fonction continue. Le logarithme permet d'exprimer commodément les variations d'incertitude. On notera $I(a)$ l'incertitude sur a, encore appelée information propre de a :

$$I(a) = -\log_2 p(a).$$

Ainsi l'information «b est réalisé» diminue l'incertitude sur a de la quantité :

$$I(a) - I(a|b) = \log_2 \frac{p(a|b)}{p(a)}.$$

Cette dernière quantité est appelée information mutuelle de a et b. Le choix de la fonction log n'est cependant pas arbitraire. En effet il est souhaitable que la mesure de l'information choisie soit additive. La quantité d'information fournie par la réalisation de deux événements a et b statistiquement indépendants doit être égale à la somme des quantités d'information fournies par les réalisations de a et b pris séparément. On doit donc avoir $I(a, b) = I(a) + I(b)$ et puisque $p(a, b) = p(a)p(b)$ lorsque a et b sont indépendants, il est naturel d'utiliser le log. On peut même montrer que si la mesure de l'information vérifie de plus un autre axiome appelé condition de regroupement, la fonction log est la seule possible. On doit donc choisir une fonction de la forme $I(a) = \lambda \log_e p(a)$ avec $\lambda < 0$. Le choix de λ dépend de l'unité de d'information choisie. Nous utiliserons le bit.

> Un bit est égal à la quantité d'information fournie par le choix d'une alternative parmi deux équiprobables.

En clair, ceci signifie que si une lettre est choisie dans l'alphabet $A = \{0, 1\}$ muni d'une loi de probabilité uniforme, alors la quantité d'information fournie par la réalisation de l'événement $\{A = a\}$ est de un bit. On a ainsi $I(0) = \lambda \log_2 p(0) = -\lambda \log_2 2 = 1$. Donc $\lambda = -1/\log_2 2$. D'où

$$I(a) = -\log_2 p(a).$$

Le bit est ici à comprendre dans son sens originel de «binary unit» et non «binary digit». La confusion provient du fait que pour représenter une information de n bits, il faut n symboles binaires.

Exemple : Soit $A = \{a_0, \ldots, a_{15}\}$ un alphabet de 16 lettres équiprobables. L'information propre d'une lettre a quelconque est $I(a) = -\log_2(\frac{1}{16}) = 4$. Dans ce cas particulier, l'information va consister à choisir un entier i dans $\{0, 1, \ldots, 15\}$ et pour représenter cette information il faut disposer de 4 bits. Il faut prendre garde au fait que ceci n'est vrai que parce que les lettres sont équiprobables. En effet, si ce n'est pas le cas, l'information propre d'une lettre sera généralement différente de 4 et l'information propre moyenne peut même être strictement inférieure à 4.

2.4 Information mutuelle. Information propre

On considère un espace probabilisé joint AB où $A = \{a_1, \ldots, a_n\}$ et $B = \{b_1, \ldots, b_m\}$.

L'information mutuelle entre les événements $\{A = a\}$ et $\{B = b\}$ est définie par

$$I(a;b) = \log_2 \frac{p(a|b)}{p(a)}.$$

Par définition $p(a,b) = p(a|b)p(b) = p(b|a)p(a)$. Donc

$$I(a;b) = I(b;a) = \log_2 \frac{p(a,b)}{p(a)p(b)}.$$

Nous allons discuter le signe de $I(a;b)$.

- $I(a;b) > 0$ signifie que si l'un des deux événements se réalise, alors la probabilité de l'autre augmente ;
- $I(a;b) < 0$ signifie que si l'un des deux événements se réalise, alors la probabilité de l'autre diminue ;
- $I(a;b) = 0$ signifie que les deux événements sont statistiquement indépendants.

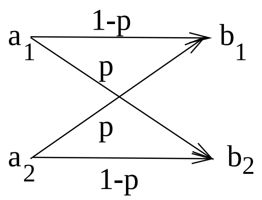

FIG. 5 – Le canal binaire symétrique

Exemple : Considérons le canal binaire symétrique de probabilité de transition p avec des entrées notées a_1, a_2 équiprobables et des sorties b_1, b_2.

La matrice de transition, notée $\Pi = (\Pi_{ij})$, est définie par $\Pi_{ij} = p(b_j|a_i)$. La matrice de transition du canal binaire symétrique est donc

$$\Pi = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}.$$

Puisque les entrées sont équiprobables, $p(a_1) = p(a_2) = \frac{1}{2}$. On en déduit la loi jointe :

$$\begin{aligned} p(a_1, b_1) &= p(a_2, b_2) = \frac{1-p}{2} \\ p(a_1, b_2) &= p(a_2, b_1) = \frac{p}{2} \end{aligned}$$

On en déduit la loi marginale sur B : $p(b_1) = p(b_2) = \frac{1}{2}$. Ceci permet de calculer l'information mutuelle de chaque couple (a_i, b_j).

$$\begin{aligned} I(a_1; b_1) &= I(a_2; b_2) = \log_2 2(1-p) = 1 + \log_2(1-p) \\ I(a_1; b_2) &= I(a_2; b_1) = \log_2 2p = 1 + \log_2 p. \end{aligned}$$

On constate que si $p < \frac{1}{2}$, $I(a_1; b_1)$ est positif et $I(a_1; b_2)$ est négatif. Ceci signifie que lorsque l'on observe la lettre b_1 à la sortie du canal, la probabilité pour que a_1 ait été émise en entrée augmente. Au contraire si b_2 est observée, la probabilité pour que la lettre a_1 ait été émise diminue. Enfin lorsque $p = \frac{1}{2}$, toutes les informations mutuelles sont nulles et donc les alphabets d'entrée et de sortie sont statistiquement indépendants, ce qui n'est pas souhaitable.

Considérons un cas particulier intéressant. Quelle est l'information mutuelle entre l'événement $\{A = a\}$ et lui-même ? Rigoureusement, ceci consiste à calculer $I(a; b)$ lorsque l'événement $\{B = b\}$ spécifie de façon unique $\{A = a\}$, c'est-à-dire lorsque $p(a|b) = 1$. On a alors $I(a; b) = \log_2 \frac{p(a|b)}{p(a)} = \log_2 \frac{1}{p(a)}$. D'où $I(a; b) = -\log_2 p(a) = I(a)$. Il s'agit en fait de la quantité maximale d'information que peut fournir $\{A = a\}$.

L'information propre de l'événement $\{A = a\}$ est $I(a) = -\log_2 p(a)$. L'information propre s'interprète comme la quantité d'information fournie par la réalisation de cet événement. Notons que l'information propre est toujours positive ou nulle et que, plus un événement est improbable, plus son information propre est grande. À l'inverse, la réalisation d'un événement certain n'apporte aucune information, ce qui semble conforme à l'intuition.

On peut également définir dans l'espace probabilisé joint AB l'information propre conditionnelle de a sachant b qui est la quantité d'information fournie par l'événement $\{A = a\}$ sachant que l'événement $\{B = b\}$ est réalisé :

$$I(a|b) = -\log_2 p(a|b).$$

L'information mutuelle entre deux événements est donc

$$I(a; b) = I(a) - I(a|b).$$

2.5 Information mutuelle moyenne. Entropie

L'information mutuelle moyenne de A et B dans l'espace probabilisé joint AB est définie par :

$$I(A;B) = \sum_{a \in A, b \in B} p(a,b) I(a;b).$$

On a donc

$$I(A;B) = \sum_{a \in A, b \in B} p(a,b) \log_2 \frac{p(a,b)}{p(a)p(b)}.$$

On peut également définir la moyenne de l'information propre d'un espace probabilisé A. Cette moyenne s'appelle entropie de l'espace A :

$$H(A) = \sum_{a \in A} p(a) I(a) = -\sum_{a \in A} p(a) \log_2 p(a).$$

Enfin l'information propre conditionnelle est une variable aléatoire réelle et nous pouvons définir sa moyenne appelée entropie conditionnelle de A sachant B. Elle est définie sur l'espace de probabilité joint AB :

$$H(A|B) = -\sum_{a \in A, b \in B} p(a,b) \log_2 p(a|b).$$

On en déduit que $I(A;B) = H(A) - H(A|B)$.

Nous allons établir quelques propriétés de l'entropie et de l'information mutuelle moyenne. On utilisera dans certaines preuves une propriété très simple de la fonction logarithme népérien. Pour tout $x > 0$, on a

$$\log_e x \leq x - 1,$$

avec égalité seulement pour $x = 1$.

Théorème 1 *Soit (A, p) un espace probabilisé discret de cardinal n. Alors $H(A) \leq \log_2 n$ avec égalité ssi la loi de probabilité p est uniforme sur A.*

Preuve. Il suffit de montrer que $H_e(A) \leq \log_e n$, où

$$H_e(A) = -\sum_{a \in A} p(a) \log_e p(a).$$

Or

$$\begin{aligned}
H_e(A) - \log_e n &= -\sum_{a\in A} p(a)\log_e p(a) - \sum_{a\in A} p(a)\log n \\
&= \sum_{a\in A} p(a)\log_e \frac{1}{np(a)} \\
&\leq \sum_{a\in A} p(a)(\frac{1}{np(a)} - 1) \\
&= (\sum_{a\in A}\frac{1}{n} - \sum_{a\in A} p(a)) \\
&= 0,
\end{aligned}$$

avec égalité ssi il y a égalité tout au long du calcul, c'est-à-dire ssi $p(a) = \frac{1}{n}$ pour tout a de A. □

Cette proposition dit que l'entropie d'un espace probabilisé de taille 2^r est toujours inférieure ou égale à r bits.

PROPOSITION 2 *Soit AB un espace probabilisé joint. L'information mutuelle moyenne $I(A;B)$ de A et B est toujours positive ou nulle. Elle est nulle ssi A et B sont statistiquement indépendants.*

Preuve. Montrons que $-I(A;B) \leq 0$.

$$\begin{aligned}
-I(A;B) &= \sum_{a\in A, b\in B} p(a,b)\log_2 \frac{p(a)p(b)}{p(a,b)} \\
&= K\sum_{a\in A, b\in B} p(a,b)\log_e \frac{p(a)p(b)}{p(a,b)} \\
&\leq K\sum_{a\in A, b\in B} p(a,b)(\frac{p(a)p(b)}{p(a,b)} - 1) \\
&= K(\sum_{a\in A, b\in B} p(a)p(b) - \sum_{a\in A, b\in B} p(a,b)) \\
&= 0,
\end{aligned}$$

où K est une constante strictement positive pour le changement de base du log et avec égalité dans les inégalités ci-dessus ssi $p(a,b) = p(a)p(b)$ pour tout a de A et tout b de B. □

Ce résultat signifie essentiellement que, en moyenne, le fait de connaître la valeur de b dans B diminue toujours l'incertitude sur A, sauf si A et B sont indépendants auquel cas aucune information n'est apportée.

3 Codage des sources discrètes

3.1 Code et codage

Nous allons nous intéresser au codage d'une source discrète sans mémoire en une séquence binaire. Le décodage devra permettre de retrouver la séquence de lettres émises par la source à partir de la séquence codée binaire. Nous verrons que le nombre minimal moyen de symboles binaires par lettre est égal à l'entropie de la source.

Un exemple célèbre et désuet de tel codage est le code Morse. En Morse la lettre e, très fréquente, est représentée par le mot «·», tandis que la lettre q, moins fréquente est représentée par un mot plus long «$\cdot \cdot - -$». De tels codes sont dits codes à longueur variable. On les appelle aussi plus simplement code.

Si A est un alphabet fini, on note A^* l'ensemble des mots sur l'alphabet A. On compte parmi eux le mot vide, encore noté ε. La concaténation de deux mots u et v est notée uv. La taille d'un mot u, notée $|u|$, est le nombre de lettres de u.

On appellera codage d'une source discrète une application qui associe à chaque séquence finie de lettres de la source une séquence binaire finie. Il s'agit donc d'une application de A^* dans $\{0,1\}^*$, où A est l'alphabet de la source. Quand on parlera de codage, on supposera que cette application est injective. Un cas particulier de ces codages consiste à associer à chaque lettre de A un mot de $\{0,1\}^*$. Si on note c cette application de A dans $\{0,1\}^*$, le codage d'un mot $u = u_1 \ldots u_n$ est alors le mot $c(u_1) \ldots c(u_n)$. On le note $c(u)$ ce qui étend c à A^*.

On note C l'ensemble des codes possibles des lettres

$$C = \{c(a) \mid a \in A\}.$$

L'application c définit bien un codage si c est injective sur A^*, c'est-à-dire si c est injective sur A et si C est un code à déchiffrage unique, au sens donné plus bas. Attention à cette distinction entre code et codage.

L'efficacité du code est définie par

$$E = \frac{H(A)}{\overline{m}},$$

où $\overline{m}$ est le nombre moyen de symboles binaires utilisés par lettre de la source :

$$\overline{m} = \sum_{a \in A} p(a)|c(a)|.$$

L'efficacité du codage est

$$\lim_{k\to\infty} \frac{H(A)}{\frac{\overline{m_k}}{k}},$$

où $\overline{m_k}$ est le nombre moyen de symboles binaires utilisés pour coder une suite de longueur k de la source. On suppose ici que cette limite existe, sinon on prend une limite supérieure. Comme la source est sans mémoire, on a

$$\begin{aligned}
\overline{m_k} &= \sum_{u\in A^*,|u|=k} p(u)|c(u)| \\
&= \sum_{a_1\in A} \dots \sum_{a_k\in A} p(a_1)\dots p(a_k)(|c(a_1)| + \dots + |c(a_k)|) \\
&= k\overline{m}.
\end{aligned}$$

L'efficacité du codage est donc égale à l'efficacité du code.

3.2 Code

Nous rappelons ci-dessous la notion de code vue dans le cours sur les automates.

Un ensemble de mots finis C sur un alphabet A est appelé code, ou encore plus précisément code à déchiffrage unique, ssi pour tous mots $u_1, \dots, u_n$, $v_1, \dots, v_m$ de C,

$$u_1 \dots u_n = v_1 \dots v_m$$

implique $n = m$ et $u_i = v_i$ pour tout $1 \leq i \leq n$.

Un ensemble préfixe de mots, c'est-à-dire un ensemble de mots tel que aucun mot n'est le «début» d'un autre, est un code. On dit parfois que c'est un code préfixe. Un code préfixe admet une représentation naturelle sous forme d'arbre k-aire si k est la taille de l'alphabet du code. Cet arbre est formé de nœuds que l'on peut identifier à un mot sur l'alphabet du code. Le père du nœud identifié à $a_1a_2\dots a_k$ est le nœud identifié à $a_1a_2\dots a_{k-1}$. La racine de cet arbre est le mot vide. Les nœuds qui n'ont pas de descendants identifiés à un mot du code sont éliminés. Le code est préfixe ssi l'ensemble des feuilles de l'arbre est le code.

Un ensemble de mots de même longueur est un code. On dit que c'est un code de longueur fixe.

3.3 Codage avec un code de longueur fixe

Si une source a pour cardinal n, il est possible de la coder avec un code de longueur fixe m tel que

$$\log_2 n \leq m < 1 + \log_2 n.$$

L'efficacité E d'un code de longueur m est égale à $\frac{H(A)}{m}$. Comme $H(A) \leq \log_2 n$, on a $E \leq 1$ et $E = 1$ ssi

– $H(A) = \log_2 n$, c'est-à-dire les lettres de la source sont équiprobables,
– $m = \log_2 n$, c'est-à-dire le cardinal de la source est une puissance de 2.

Exemple : Soit une source dont l'alphabet est $A = \{0, 1, \ldots, 9\}$ munie de la loi de probabilité uniforme. On code cette source par une code en longueur fixe de longueur 4. Par exemple, on peut prendre

lettre	0	1	2	3	4	5	6	7	8	9
mot	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

L'efficacité du code est $H(A)/4 = (\log_2 10)/4 \approx 0,83$. Ce code n'est pas optimal. Il est cependant possible d'améliorer son efficacité en considérant non plus des chiffres isolés mais des paires de chiffre. Ceci revient à changer la source A en $A^2 = \{00, 01, \ldots, 99\}$. Son cardinal est 100. Cette nouvelle source reste munie d'une loi de probabilité uniforme et son entropie vaut $H(A^2) = \log_2 100 = 2H(A)$.

La puissance de 2 immédiatement supérieure à 100 est $2^7 = 128$. Il existe donc un code de longueur 7 pour coder cette source. Son efficacité est cette fois $H(A^2)/7 = (2\log_2 10)/7 \approx 0,95$, ce qui est meilleur. En considérant la source A^3, on obtiendra une efficacité de $0,996$. On améliore ainsi l'efficacité.

D'une façon générale, il est possible de considérer la source A^l munie de la loi de probabilité produit, pour l entier positif. On a alors

$$H(A^l) = lH(A).$$

(La preuve est laissée en exercice).

Proposition 3 *Soit A une source cardinal n. Soit A^l la source des l-uplets de lettres de A. Il existe un code de longueur constante m_l pour A^l tel que*

$$\log_2 n \leq \frac{m_l}{l} < \frac{1}{l} + \log_2 n.$$

Preuve. On sait qu'il existe un code de longueur m_l pour A^l qui vérifie

$$\log_2 n^l \leq m_l < 1 + \log_2 n^l,$$

d'où le résultat. □

Pour tout entier positif l, on pose $\overline{m} = \frac{m_l}{l}$, où m_l est la longueur optimale du code donné par la proposition précédente. L'efficacité d'un tel code est

$$E_l = \frac{H(A^l)}{m_l} = \frac{lH(A)}{m_l} = \frac{H(A)}{\overline{m}}.$$

D'après la proposition précédente,

$$\lim_{l\to\infty} \overline{m} = \log_2 n.$$

Donc

$$\lim_{l\to\infty} E_l = \frac{H(A)}{\log_2 n}.$$

Ceci montre que si $H(A) < \log_2 n$, il est impossible de s'approcher d'un codage optimal avec un code de longueur fixe. Par contre, pour une source munie d'une loi de probabilité uniforme, l'efficacité du codage peut être arbitrairement proche de 1.

3.4 Codage avec un code de longueur variable

Parmi les codes à longueur variables possibles pour effectuer un codage de source, nous allons considérer les codes préfixes.

Un rappelle qu'un arbre k-aire est un arbre tel que chaque nœud a au plus k fils. Il est dit k-aire complet si chaque nœud a exactement k-fils et si tout nœud a au moins un descendant qui est une feuille.

La suite génératrice des feuilles d'un arbre est la suite $(s_n)_{n\geq 0}$, où s_n est le nombre de feuilles de hauteur n, la hauteur étant la distance à la racine. Lorsque l'arbre est fini, tous les s_n sont nuls à partir d'un certain rang. On représente parfois la suite $(s_n)_{n\geq 0}$ par la série formelle $s(z) = \sum_{n\geq 0} s_n z^n$.

Le théorème de Kraft-McMillan donne une condition nécessaire et suffisante à l'existence d'un arbre k-aire dont la série génératrice des feuilles est donnée. On dit qu'une suite $s = (s_n)_{n\geq 0}$ satisfait l'inégalité de Kraft pour l'entier k ssi

$$\sum_{n\geq 0} s_n k^{-n} \leq 1,$$

ou encore si la série $s(z)$ associée vérifie

$$s(1/k) \leq 1.$$

THÉORÈME 4 *Une suite s est la série génératrice des feuilles d'un arbre k-aire ssi elle satisfait l'inégalité de Kraft pour l'entier k.*

Preuve. Soit tout d'abord T un arbre k-aire et s la suite génératrice de ses feuilles. Il suffit de montrer que, pour tout $n \geq 0$, la suite $(s_0, \ldots, s_n)$ satisfait l'inégalité de Kraft. C'est la suite génératrice des feuilles de l'arbre fini obtenu par restriction de l'arbre T aux nœuds de hauteur au plus n. On peut donc supposer T fini. Si $s_0 = 1$, alors $s_i = 0$ pour $i > 0$ et s vérifie l'inégalité de Kraft. Sinon, on a

$$s(z) = zt_1(z) + \ldots + zt_k(z),$$

où $t_1, \ldots, t_k$ sont les suites génératrices des feuilles des sous-arbres (éventuellement vides) enracinés aux fils de la racine de T. Par récurrence sur le nombre de nœuds, on a $t_i(1/k) \leq 1$ ce qui permet de conclure.

Réciproquement, on fait une récurrence sur n pour prouver que qu'il existe un arbre k-aire dont la suite génératrice des feuilles est $(s_0, \ldots, s_n)$. Pour $n = 0$, on a $s_0 \leq 1$ et T est soit l'arbre vide, soit l'arbre réduit à un nœud. Supposons par hypothèse de récurrence, que l'on ait déjà construit un arbre T dont la suite génératrice des feuilles est $(s_0, s_1, \ldots, s_{n-1})$. On a

$$\sum_{i=0}^{n} s_i k^{-i} \leq 1,$$

donc

$$\sum_{i=0}^{n} s_i k^{n-i} \leq k^n,$$

et ainsi

$$s_n \leq k^n - \sum_{i=0}^{n-1} s_i k^{n-i}.$$

Ceci permet d'ajouter s_n feuilles à l'arbre T à la hauteur n. $\square$

Considérons le cas d'égalité dans l'inégalité de Kraft. Si $s(1/k) = 1$, alors n'importe quel arbre qui admet s pour série génératrice des feuilles est complet. La réciproque n'est pas vraie en général (voir [7, p. 231]). Mais lorsque T est un arbre complet régulier (c'est-à-dire qui n'a qu'un nombre fini de sous-arbres non isomorphes), sa série génératrice des feuilles satisfait $s(1/k) = 1$.

La preuve donnée dans le théorème de Kraft-McMillan est constructive pour les arbres finis.

Le résultat ci-dessus montre que si C est un code préfixe sur un alphabet à k lettres dont la suite génératrice est $s = (s_i)_{i \geq 0}$, où s_i est le nombre de mots de C de longueur i, alors $s(1/k) \leq 1$. Si C est un code qui n'est pas préfixe, ce résultat est encore vrai comme le montre le résultat suivant :

THÉORÈME 5 *Si C est un code sur un alphabet à k lettres de série génératrice s, $s(1/k) \leq 1$.*

Preuve. On note p la probabilité uniforme sur l'alphabet à k lettres. Le résultat serait encore valable avec une probabilité non uniforme. On définit une mesure π sur les mots sur l'alphabet A à partir de p de la façon suivante. Si a est un lettre de A, si u, v sont des mots de A^*, on a

$$\begin{aligned}\pi(\varepsilon) &= 1\\ \pi(a) &= p(a)\\ \pi(uv) &= \pi(u)\pi(v)\\ \pi(L) &= \sum_{u \in L} \pi(u).\end{aligned}$$

On en déduit que

$$\pi(u) = \sum_{a \in A} \pi(ua).$$

On cherche à montrer que $\pi(C) \leq 1$.

Montrons tout d'abord que cette propriété est vraie pour un code fini. Si C est un code fini, on a

$$C \subset A \cup A^2 \cup \ldots \cup A^r.$$

D'où, pour tout entier positif n,

$$C^n \subset A \cup A^2 \cup \ldots \cup A^{nr}.$$

Donc $\pi(C^n) \leq nr$. Comme C est un code, $\pi(C^n) = \pi(C)^n$, pour tout entier positif n. En effet ,

$$\pi(C^n) = \sum_{u \in C^n} \pi(u) = \sum_{c_1, \ldots, c_n \in C} \pi(c_1) \ldots \pi(c_n) = \pi(C)^n.$$

Supposons $\pi(C) > 1$, on peut poser $\pi(C) = 1 + \varepsilon$, avec $\varepsilon > 0$. On a donc $\pi(C)^n = (1 + \varepsilon)^n \leq nr$, ce qui est absurde. Donc $\pi(C) \leq 1$. Si C est un code infini, on note C_n l'ensemble des mots de C de longueur inférieure ou égale à n. Cet ensemble est encore un code. On a donc $\pi(C_n) \leq 1$. De plus $\pi(C) = \sup_{n \geq 1} \pi(C_n)$ car π est une mesure. Donc $\pi(C) \leq 1$. $\square$

La construction donnée dans la preuve précédente est encore valable avec une mesure π définie à partir d'une probabilité p sur les lettres qui n'est pas uniforme. La mesure π est appelée mesure de Bernoulli.

Exemple : Soit $A = \{a, b\}$ et $C = \{a, ba, bb\}$. Cet ensemble est un code. Soit p définie par $p(a) = p$ et $p(b) = q = 1-p$. On a $\pi(C) = p + pq + q^2 = 1$.

PROPOSITION 6 *Soit C un code sur un alphabet A. S'il existe une mesure de Bernoulli π sur A^* telle $\pi(a) > 0$ pour toute lettre a de A et telle que $\pi(C) = 1$ alors C est un code maximal.*

Preuve. Si C est strictement inclus dans un autre code C', soit $u \in C' - C$. Alors $\pi(C') \geq \pi(C) = 1 = 1 + \pi(u)$. Ceci implique $\pi(u) = 0$ ce qui est impossible car π est positive. □

Exemple : Soit $A = \{a, b\}$ et $C = \{a, ab, ba\}$. Soit p définie par $p(a) = p = \frac{2}{3}$ et $p(b) = q = 1 - p = \frac{1}{3}$. On a $\pi(C) = q + 2pq = \frac{10}{9} > 1$. Donc C n'est pas un code.

3.5 Le premier théorème de Shannon

On considère ici des sources discrètes sans mémoire.

PROPOSITION 7 *Pour toute source A d'entropie $H(A)$ codée en binaire au moyen d'un code de longueur moyenne $\overline{m}$ dans , on a*

$$H(A) \leq \overline{m}.$$

Pour toute source A d'entropie $H(A)$, il existe un code permettant de coder la source et de longueur moyenne $\overline{m}$ telle que

$$H(A) \leq \overline{m} < H(A) + 1.$$

Preuve. Pour la première partie, on montre que $H(A) - \overline{m} \leq 0$, où A est la source discrète sans mémoire. On note C le code et $c(a)$ le mot codant la lettre a de A. On a

$$\begin{aligned} H(A) - \overline{m} &= -\sum_{a\in A} p(a)\log_2 p(a) - \sum_{a\in A} p(a)|c(a)| \\ &= \sum_{a\in A} p(a)\log_2 \frac{2^{-|c(a)|}}{p(a)} \\ &\leq (\log_2 e)^{-1}\sum_{a\in A} p(a)(\frac{2^{-|c(a)|}}{p(a)} - 1) \\ &= (\log_2 e)^{-1}(\sum_{a\in A} 2^{-|c(a)|} - \sum_{a\in A} p(a)) \\ &\leq 0, \end{aligned}$$

d'après le théorème 5.

Pour le deuxième point, on note m_a l'unique entier vérifiant

$$2^{-m_a} \leq p(a) < 2^{-m_a+1}.$$

Donc

$$\sum_{a\in A} 2^{-m_a} \leq \sum_{a\in A} p(a) = 1.$$

On utilise le théorème 4 pour construire un code préfixe que l'on peut mettre en bijection avec A et tel que le mot en bijection avec a a pour longueur m_a. Comme

$$-m_a \leq \log_2 p(a) < -m_a + 1$$

La longueur moyenne de ce code $\overline{m}$ vérifie alors

$$-\overline{m} \leq -H(A) < -\overline{m} + 1.$$

ou encore

$$H(A) \leq \overline{m} < H(A) + 1.$$

□

L'efficacité $E = \frac{H(A)}{\overline{m}}$ d'un code qui vérifie ces inégalités ne peut donc pas excéder 1. Pour que l'efficacité s'approche de 1, on va considérer la source A^l formée des l-uplets de lettres de A.

Notons que l'on a $H(A) = \overline{m}$ dans le théorème précédent ssi $p(a) = 2^{-m_a}$ pour toute lettre a de la source c'est-à-dire si la distribution de probabilité de la source n'est formée que d'inverses de puissances de 2.

Proposition 8 *Pour toute source A d'entropie $H(A)$, il existe un code permettant de coder la source A^l et de longueur moyenne $\overline{M_l}$ telle que*

$$H(A) \leq \frac{\overline{M_l}}{l} < H(A) + \frac{1}{l}.$$

Preuve. D'après la proposition précédente,

$$H(A^l) \leq \overline{M_l} < H(A^l) + 1.$$

Or $H(A^l) = lH(A)$, donc

$$H(A) \leq \frac{\overline{M_l}}{l} < H(A) + \frac{1}{l}.$$

□

Nous pouvons maintenant prouver le premier théorème de Shannon.

Théorème 9 *Pour toute source discrète sans mémoire, il existe un codage (injectif) permettant de coder la source et dont l'efficacité est arbitrairement proche de* 1.

Ceci s'écrit encore de la façon suivante : pour tout réel $\varepsilon > 0$, il existe un codage permettant de coder la source et dont l'efficacité est strictement supérieure à $1 - \varepsilon$.

Preuve. Nous allons décrire un codage injectif pour un entier l fixé. Pour coder $N = kl + r$ symboles de la source, où $0 \leq r < l$, on code les k premiers l-uplets de lettres à l'aide du code construit pour A^l. Le bloc restant contient r lettres. On le complète avec $l - r$ symboles aléatoires tirés avec la probabilité p sur A par exemple, et on code ce bloc avec le même code. On transmet également, pour garantir l'injectivité du codage, l'écriture de r en binaire sur $\log_2 l$ bits. Si $\overline{M_l}$ est la longueur moyenne du code permettant de coder A^l, on a

$$H(A) \leq \frac{\overline{M_l}}{l} < H(A) + \frac{1}{l}.$$

Le codage décrit nécessite donc en moyenne $(k+1)\overline{M_l} + \log_2 l$ symboles binaires pour N lettres de la source. La moyenne pour une lettre est donc

$$\frac{(k+1)\overline{M_l} + \log_2 l}{kl + r}.$$

Lorsque N tend vers l'infini, cette moyenne converge vers $\overline{m} = \frac{\overline{M_l}}{l}$. L'efficacité du codage est $E = \frac{H(A)}{\overline{m}}$. D'où

$$\frac{1}{E} < 1 + \frac{1}{lH(A)}.$$

On choisit l tel que $l > \frac{1}{\varepsilon H(A)}$, on a alors $\frac{1}{E} < 1 + \varepsilon$. D'où $E > \frac{1}{1+\varepsilon} \geq 1 - \varepsilon$.
$\square$

4 Canaux discrets sans mémoire

Dans le codage de source, aucun élément extérieur ne vient modifier l'information. Au contraire lorsque l'information transite dans un canal, elle est perturbée par un bruit.

Le résultat principal de cette section est qu'il est possible de coder de l'information de façon à ce que la détérioration soit négligeable. Ceci se fait au prix d'une redondance de l'information ou encore d'une vitesse de transmission ou de capacités de stockage plus faibles.

Pour tout canal de transmission on définit une grandeur caractéristique appelée capacité du canal et que l'on peut interpréter comme la quantité maximale d'information pouvant transiter à travers le canal.

4.1 Définitions

Pour définir un canal de transmission, on décrit l'ensemble des entrées et des sorties possibles du canal ainsi que la bruit qui peut perturber la transmission.

Le canal discret est le plus simple des canaux de transmission. L'ensemble des entrées et l'ensemble des sorties sont deux alphabets finis A et B. Le canal est discret sans mémoire si le bruit se modélise comme une probabilité conditionnelle de B sachant A indépendante du temps. Un canal discret sans mémoire est donc défini par

- l'alphabet d'entrée A,
- l'alphabet de sortie B,
- la matrice de transition $\Pi = (\Pi_{ij})$ où $\Pi_{ij} = p(b_j|a_i)$.

Par exemple, le canal binaire symétrique de probabilité d'erreur p avec des entrées notées a_1 et a_2 et des sorties notées b_1 et b_2, est défini par les probabilités conditionnelles données par la matrice de transition Π :

$$\Pi = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix},$$

Le canal à effacement est un canal dont les entrées sont des bits dans $\{0,1\}$ et les sorties des symboles de l'alphabet $\{0,\infty,1\}$. La matrice stochastique Π pour ce canal est

$$\Pi = \begin{bmatrix} 1-p-p_\infty & p_\infty & p \\ p & p_\infty & 1-p-p_\infty \end{bmatrix}.$$

Certaines classes de canaux ont des propriétés remarquables. Soient $A = \{a_1, \dots a_n\}$ et $B = \{b_1, \dots b_m\}$.

- Un canal est dit sans perte si $H(A|B) = 0$ pour toutes les distributions en entrée. On encore, on peut partitionner les sorties B en $B_1, B_2, \dots, B_n$ de telle sorte que pour tout i, $\sum_{b \in B_i} p(b|a_i) = 1$. L'entrée de ce canal est déterminée par la sortie.
- Un canal est dit déterministe si $p(b_j|a_i) = 0$ ou 1 pour tout i, j. Ou encore si $H(B|A) = 0$. La sortie de canal est déterminée par l'entrée.
- Un canal est dit sans bruit s'il est déterministe et sans perte.
- Un canal est inutile si $I(A; B) = 0$ pour toutes les toutes les distributions en entrée. Ceci équivaut à avoir $H(A|B) = H(A)$ pour toutes probabilités p_A. Ceci est équivaut encore à $p(b_j|a_i) = p(b_j)$ pour tout i, j. Les lignes de la matrice Π pour ce canal sont donc toutes identiques. Ce canal mélange donc les entrées et la connaissance des sorties n'apporte aucune information sur les entrées.
- Un canal est symétrique si chaque ligne de Π contient (à permutation près) les mêmes valeurs $p_1, \dots p_m$ et chaque colonne de Π contient (à permutation près) les mêmes valeurs $q_1, \dots q_n$. Par exemple le canal donné par
$$\Pi = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$
est symétrique.

4.2 Capacité d'un canal

On considère un canal discret sans mémoire donné par A, B, Π. On rappelle que l'entropie mutuelle jointe $I(A; B)$ dépend aussi de la distribution de probabilité p_A sur A. Par définition

$$I(A; B) = \sum_{a \in A, b \in B} p(a, b) \log_2 \frac{p(a|b)}{p(a)} = \sum_{a \in A, b \in B} p(a, b) \log_2 \frac{p(b|a)}{p(b)}.$$

On rappelle que l'on a donc aussi

$$I(A; B) = \sum_{a \in A, b \in B} p(a, b) \log_2 \frac{p(a, b)}{p(a)p(b)}.$$

Elle vérifie (voir feuille d'exercice 1),

$$I(A; B) = H(B) - H(B|A) = H(A) - H(A|B) = H(A) + H(B) - H(A, B).$$

Ici $H(A, B)$ désigne l'entropie de l'espace joint considéré comme une source :

$$H(A, B) = -\sum_{a \in A} \sum_{b \in B} p(a, b) \log_2 p(a, b).$$

Lorsqu'on fait varier la distribution d'entrée p_A, $I(A;B)$ varie. La valeur maximale que l'on peut atteindre s'appelle la capacité de Shannon du canal :

$$C = \max_{p_A} I(A;B).$$

La capacité est donc un réel positif ou nul, inférieur ou égal au log à base 2 du cardinal de A et de B.

Nous allons calculer la capacité d'un canal symétrique. Considérons donc un canal symétrique dont les valeurs communes des lignes de Π sont $p_1, \ldots p_m$ et les valeurs communes des colonnes sont $q_1, \ldots q_n$.

Proposition 10 *Pour un canal symétrique, $H(B|A)$ ne dépend pas de la distribution p_A.*

Preuve. Par définition,

$$\begin{aligned} H(B|A) &= -\sum_{a\in A}\sum_{b\in B} p(a,b)\log_2 p(b|a) \\ &= -\sum_{a\in A}\sum_{b\in B} p(a)\log_2 p(b|a)p(b|a) \\ &= -\sum_{a\in A} p(a)(\sum_{b\in B} p(b|a)\log_2 p(b|a)) \\ &= -\sum_{a\in A} p(a)(\sum_{j=1}^{m} p_j\log_2 p_j) \\ &= -\sum_{j=1}^{m} p_j\log_2 p_j. \end{aligned}$$

La dernière égalité ne dépend plus de p_A d'où la conclusion. □

Comme $I(A;B) = H(B) - H(B|A)$ et puisque $H(B|A)$ ne dépend pas p_A, il suffit de maximiser $H(B)$ pour maximiser $I(A;B)$. D'après le théorème 1, $H(B) \leq \log_2 m$ avec égalité si et seulement si p_B est la loi uniforme. Ainsi, si on trouve une distribution p_A en entrée telle que la loi en sortie soit la loi uniforme, on aura réussi à calculer la capacité du canal. Nous allons voir que la loi uniforme en entrée va faire l'affaire.

Supposons donc que p_A est la loi uniforme en entrée. La loi p_B se calcule par

$$p(b) = \sum_{a\in A} p(a,b) = \sum_{a\in A} p(b|a)p(a) = \frac{1}{n}\sum_{a\in A} p(b|a).$$

Or $\sum_{a\in A} p(b|a)$ est la somme des éléments de la colonne d'indice b de Π. Cette quantité est la même pour toutes les colonnes et donc p_B est la loi uniforme sur B. La capacité d'un canal symétrique est donc

$$C_{\text{sym}} = \log_2 m + \sum_{j=1}^{m} p_j \log_2 p_j.$$

Par exemple la capacité du canal binaire symétrique est

$$C_{\text{BSC}} = 1 + p \log_2 p + (1-p)\log_2(1-p) = 1 - H(p, 1-p).$$

5 Codage de canal

Nous considérons des suites binaires sources qui sont des blocs de longueur fixe m de lettres de l'alphabet $A = \{0, 1\}$. Chaque bloc est d'abord codé à l'aide d'un code binaire de longueur constante égale à n, et de cardinal $M \leq 2^m$. Ce codage est appelé codage de canal par blocs. Les blocs codés passent ensuite dans un canal discret sans mémoire (B, C, Π), où B est l'alphabet d'entrée du canal et C est son alphabet de sortie. On suppose ici que $B = C = \{0, 1\}$. Après le passage dans le canal, on souhaite retrouver la suite binaire initiale. C'est le rôle joué par le décodeur. Le décodage transforme donc une suite de n lettres de C en une suite source de m lettres de A.

On définit le taux ou rendement d'un code binaire de cardinal M constitué par des mots de longueur n par

$$R = \frac{\log_2 M}{n}.$$

Un code aura un taux 1 lorsque $M = 2^n$, c'est-à-dire lorsqu'aucune redondance n'aura été ajoutée. Un code de taux 1/2 signifie que $M = 2^{n/2}$, c'est-à-dire que le code double la longueur des séquences binaires de la source.

5.1 Le deuxième théorème de Shannon

Nous allons expliquer sans le prouver le deuxième théorème de Shannon pour le canal binaire symétrique. On rappelle que la capacité de ce canal est $C_{\mathrm{BSC}} = 1 + p \log_2 p + (1-p) \log_2(1-p)$. Pour ce canal $B = C = \{0, 1\}$.

On note $\mathbf{a}$ un bloc source de longueur m, $\mathbf{b}$ un mot de longueur n du code $X = \{c(\mathbf{a}) \mid \mathbf{a} \in A^m\} \subset B^n$, et enfin $\mathbf{c}$ désigne un mot de C^n. Si ϕ est un algorithme de décodage, on note $\phi(\mathbf{c})$ le décodage de $\mathbf{c}$ par ϕ.

Définition Le taux d'erreur résiduel d'un algorithme de décodage est égal à la probabilité d'un décodage erroné lorsque la loi d'émission de la source est uniforme.

Si p_e est le taux d'erreur résiduel, on a

$$\begin{aligned} p_e &= \sum_{\mathbf{a} \in A^m, \mathbf{c} \in C^n \mid \phi(\mathbf{c}) \neq \mathbf{a}} p(\mathbf{a}, \mathbf{c}) \\ &= \sum_{\mathbf{a} \in A^m, \mathbf{c} \in C^n \mid \phi(\mathbf{c}) \neq \mathbf{a}} p(\mathbf{c}|\mathbf{a}) p(\mathbf{a}) = \sum_{\mathbf{b} \in X, \mathbf{c} \in C^n \mid c(\phi(\mathbf{c})) \neq \mathbf{b}} \frac{p(\mathbf{c}|\mathbf{b})}{2^m}. \end{aligned}$$

Exemple Considérons le code binaire à répétition de longueur n. Il est constitué de deux mots de longueur n : $0 \ldots 0$ et $1 \ldots 1$. On a donc $M = 2$.

On suppose que n est impair. On peut donc poser $n = 2r+1$. Un algorithme de décodage possible pour ce code est le décodage dit majoritaire. Une suite $\mathbf{c}$ de longueur n est décodée en 0 si elle a une majorité de 0 et en 1 sinon.

La probabilité pour que le mot $0\ldots0$ soit décodée en $1\ldots1$ est égale à la probabilité pour que le nombre de symboles erronés après passage dans le canal soit supérieur ou égal à $r+1$, soit

$$p_e = \sum_{i=r+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}.$$

Il est possible de montrer que pour une probabilité d'erreur $p < 1/2$, le taux d'erreur résiduel peut être rendu arbitrairement petit.

Théorème 11 *Soit un canal binaire symétrique de capacité C. Pour tout réel positif $R < C$, il existe une suite C_n de codes binaires de longueur n, de rendement R_n, avec des algorithmes de décodage de taux d'erreur résiduel $p_{e,n}$ telle que*
- $\lim_{n\to\infty} R_n = R$,
- $\lim_{n\to\infty} p_{e,n} = 0$.

Ce théorème signifie donc qu'il existe des codes en bloc permettant de réaliser un code dont le taux de transmission est aussi proche qu'on le souhaite de la capacité du canal avec un taux d'erreur arbitrairement petit.

Le théorème suivant constitue une réciproque au théorème précédent.

Théorème 12 *Soit un canal binaire symétrique de capacité C. Pour tout code de taux $R > C$, il existe une constante strictement positive $K_{R,C}$ ne dépendant que de R et C telle que tout algorithme de décodage de ce code est tel que le taux d'erreur résiduel vérifie $p_e > K_{R,C}$.*

Ce résultat indique qu'il est inutile de chercher des codes de taux supérieur à la capacité du canal. La capacité est donc bien le taux de codage maximal que l'on puisse atteindre pour faire transiter une information dans un canal.

6 Introduction au codage correcteur

6.1 Codage par registres à décalage

Nous donnons un exemple de codage correcteur d'erreurs qui s'implémente à l'aide de circuits électroniques très simples appelés registres à décalage.

Il s'agit d'un codage par blocs dit code de Hamming [7,4,3]. Les deux premiers paramètres indiquent le taux de transmission ou rendement 4 : 7. Le schéma de codage est le suivant. On note $m = a_0a_1a_2a_3$ le bloc de bits

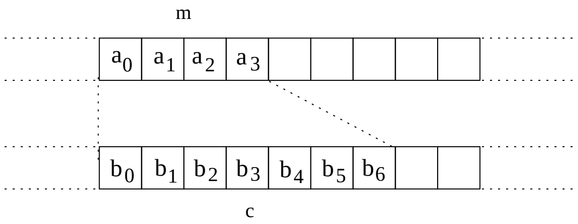

FIG. 6 – Un codage de taux 4 : 7

à coder de longueur 4. On note $c = b_0b_1b_2b_3b_4b_5b_6$ le bloc de bits codé de longueur 7. À m on associe le polynôme $m(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ et à c on associe le polynôme $c(x) = b_0 + b_1x + b_2x^2 + \ldots b_6x^6$. On considère que ces polynômes sont à coefficients dans $\mathbb{F}_2$, où $\mathbb{F}_2$ est le corps à deux éléments $\mathbb{Z}/2\mathbb{Z}$. Dans $\mathbb{F}_2[x]$, on peut effectuer

– des additions, par exemple

$$(1 + x + x^2) + (x + x^3) = 1 + x + x^2 + x^3.$$

– des multiplications, par exemple

$$(1 + x)^2 = 1 + x^2.$$

– des divisions, par exemple

$$(x^3 + x^2 + x) = (x^2 + 1)(x + 1) + 1.$$

Plus généralement, on peut diviser un polynôme $p(x)$ par un polynôme $d(x)$. Il existe une décomposition unique en

$$p(x) = q(x)d(x) + r(x),$$

où $d(x)$ et $r(x)$ sont des polynômes tels que le degré de $r(x)$ est strictement plus petit que le degré de $d(x)$. Le polynôme $r(x)$ est appelé reste de la division de $p(x)$ par $d(x)$ et est noté $p(x) \bmod d(x)$. On dit que $p(x) = q(x) \bmod d(x)$ si et seulement si $d(x)$ divise $p(x) - q(x)$.

Nous allons utiliser le polynôme

$$g(x) = 1 + x + x^3.$$

On commence par calculer $1, x, x^2, x^3, \ldots \bmod g(x)$.

	1	x	x^2	x^3	x^4	x^5	x^6
1	1	0	0	1	0	1	1
x	0	1	0	1	1	1	0
x^2	0	0	1	0	1	1	1

Le tableau ci-dessus signifie par exemple que $x^4 = x + x^2 \bmod g(x)$. On note $n = 3$ le degré de g. On remarque que les $2^n - 1 = 7$ colonnes du tableau sont toutes distinctes et non nulles. On en déduit que $1, x, x^2, x^3, \ldots, x^{2^n-2}$ sont distincts et non nuls. Un tel polynôme g est dit primitif.

Définition du codage

Le codage associe $m(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ à $c(x)$ défini par

$$c(x) = m(x)x^3 + (m(x)x^3 \bmod g(x)).$$

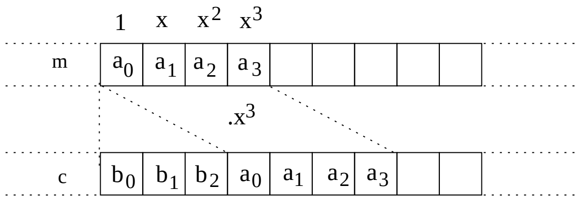

FIG. 7 – Le codage

Sur la figure 7, $b_0 + b_1 x + b_2 x^2 = m(x)x^3 \bmod g(x)$. Les bits $b_0 b_1 b_2$ sont appelés bits de correction et les bits $a_0 a_1 a_2 a_3$ sont appelés bits d'information. Le calcul de $c(x) = c_0 + c_1 x + \cdots + c_n x^n \bmod g(x)$ s'effectue au moyen du registre à décalage de la figure 8. Chaque décalage à droite correspond à une multiplication par x. Le polynôme $c(x)$ vérifie la propriété suivante :

$$c(x) = 0 \bmod g(x).$$

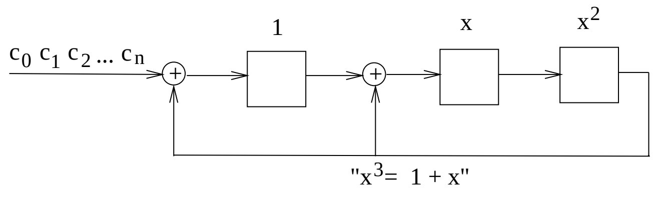

FIG. 8 – Un registre à décalage

Définition du décodage

On va tout d'abord supposer qu'une erreur au plus peut survenir sur un bloc codé c de 7 bits. Le polynôme à décoder est donc $\hat{c}(x) = c(x) + e(x)$, où $e(x)$ représente l'erreur. Avec l'hypothèse, on a $e(x) = x^i$ où i est un entier compris entre 0 et 6 qui est inconnu.

La correction d'une erreur éventuelle se fait de la façon suivante. On calcule $\hat{c}(x) \bmod g(x)$ avec le même registre à décalage que celui utilisé lors du codage. Deux cas se présentent alors

- Si $\hat{c}(x) \bmod g(x) = 0$, on conclut qu'il n'y a pas eu d'erreurs.
- Si $\hat{c}(x) \bmod g(x) = x^i$, on conclut qu'il y a eu une erreur sur le bit numéro i. En effet,

$$\hat{c}(x) \bmod g(x) = (c(x) + e(x)) \bmod g(x) = e(x) \bmod g(x) = e(x).$$

  Le calcul de i s'effectue avec un circuit permettant de retrouver l'indice de la colonne du tableau des puissances de x modulo $g(x)$ à partir des valeurs de cette colonne.

Après correction, on a $c(x)$, donc c. On renvoie les quatre derniers bits de c qui sont m.

Jusqu'à présent, nous avons fait l'hypothèse que le canal ne produit qu'une seule erreur au plus par bloc. En pratique, si l'on prend le cas du canal binaire symétrique, une erreur se produit de façon aléatoire avec une probabilité p. Si par exemple $p = 10^{-3}$, un bit sur mille en moyenne est faux. Cela signifie, pour la transmission d'un texte, environ une erreur toutes les deux lignes. Avec le codage décrit ci-dessus par le Hamming [7,4,3], la probabilité de transmission correcte d'un bloc de quatre bits est $1 - p_e$, où p_e est le taux d'erreur résiduel. Cette probabilité est donc, si $q = 1 - p$,

$$q^7 + 7pq^6 = (1+6p)(1-p)^6 = (1+6p)(1-6p+15p^2+o(p^2)) \sim 1 - 21.10^{-6}.$$

Tout se passe donc comme si la probabilité d'erreur sur un bit n'était plus que de l'ordre de 10^{-5}. Sur un texte, cette probabilité donne moins d'une erreur sur cent lignes. La capacité du canal binaire symétrique avec $p = 10^{-3}$ est $1 - H(p, 1-p) = 0,98859224$.

Exemple
Le codage correcteur du Minitel a un taux 120 : 136. Le dernier octet (sur 17) d'un mot de code est un octet de validation. Il ne comporte que des zéros. Il détecte des erreurs importantes comme celles provoquées par la foudre par exemple. Lorsqu'une partie du message a été perdue, cet octet contient le début du message suivant et sert donc à détecter ce type d'erreurs. Les 16 premiers octets sont des mots d'un code de Hamming obtenu avec le polynôme primitif $g(x) = x^7 + x^3 + 1$. Il contient 15 octets d'information et 7 bits de correction. Les 7 bits de correction sont complétés d'un bit de parité qui en font un octet de correction. Un tel codage s'appelle un codage de Hamming étendu. Il permet de corriger une erreur et d'en détecter deux. Expliquer pourquoi.

6.2 Codage correcteur convolutif

Nous présentons le principe du codage correcteur convolutif. Il s'agit d'un codage dit à fenêtre glissante. Il existe deux entiers m (pour mémoire) et a (pour anticipation) tels que le symbole d'indice n de la suite codée ne dépend que du bloc source fini compris entre les indices $n - m$ et $n + a$. La suite codée se calcule ainsi en promenant une fenêtre de taille bornée le long de la suite à coder.

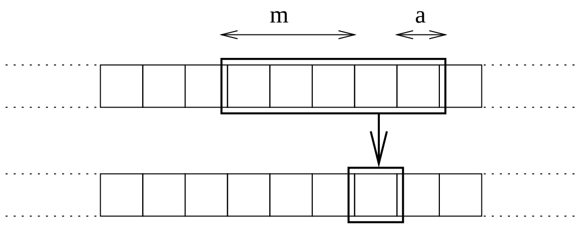

FIG. 9 – Une fenêtre glissante

Nous présentons le principe du codage convolutif sur un exemple. On considère une source qui émet une suite $I_0 = (i_0, i_1, \ldots, i_n, \ldots)$ de bits et va traverser un canal bruité. Chaque bit i_n est codé par un bloc de deux bits $(t_n^{(0)}, t_n^{(1)})$. Le taux de transmission de ce codage sera donc constant et égal à 1 : 2. Ces bits sont calculés modulo 2 par les formules suivantes

$$\begin{aligned} t_n^{(0)} &= i_{n-2} + i_n \\ t_n^{(1)} &= i_{n-2} + i_{n-1} + i_n \end{aligned}$$

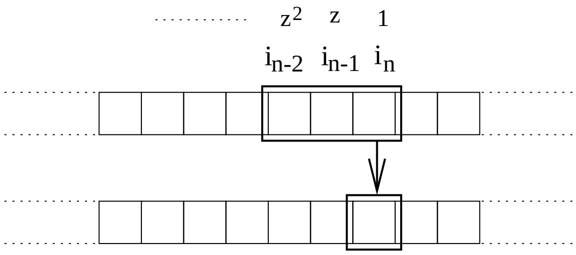

FIG. 10 – Une fenêtre de codage

6.2.1 La méthode algébrique

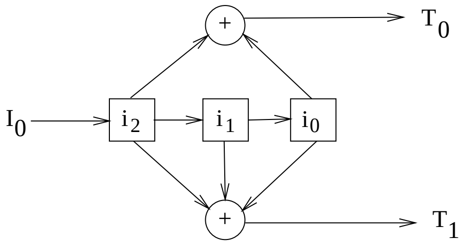

FIG. 11 – Un codage de convolution

Nous allons tout d'abord tenter de décrire de façon algébrique le codage et le décodage. Pour ceci, il est commode de décrire les suites binaires par des séries formelles où les coefficients sont dans $\mathbb{Z}/2\mathbb{Z}$. On note ainsi

$$I_0(z) = i_0 + i_1 z + i_2 z^2 + \dots$$

On définit de même T_0 et T_1 les séries correspondant à $(t_n^{(0)})_{n\geq 0}$ et $(t_n^{(1)})_{n\geq 0}$. On déduit des définitions que

$$\begin{aligned} T_0(z) &= (1+z^2)I_0(z), \\ T_1(z) &= (1+z+z^2)I_0(z). \end{aligned}$$

On pose $T(z) = T_0(z^2) + zT_1(z^2)$ et $I(z) = I_0(z^2)$. La suite $T(z)$ est une série formelle dont la suite des coefficients est la suite codée de I_0. On obtient

$$T(z) = (1 + z^4)I(z) + (1 + z^2 + z^4)zI(z).$$

D'où

$$T(z) = (1 + z + z^3 + z^4 + z^5)I(z).$$

Donc $T(z) = G(z)I(z)$. De cette formule provient le terme de codage de convolution.

Nous allons maintenant décrire le décodage sans se préoccuper pour l'instant des éventuelles erreurs survenues dans le canal. Il s'agit de retrouver $I_0(z)$ à partir de $T_0(z)$ et $T_1(z)$. Soit $G_0(z) = 1 + z^2$ et $G_1(z) = 1 + z + z^2$. Le pgcd de ces deux polynômes est 1. Donc d'après le théorème de Bézout, il existe deux polynômes P_0 et P_1 tels que

$$P_0G_0 + P_1G_1 = 1.$$

D'où

$$P_0T_0 + P_1T_1 = I_0.$$

On trouve ici $P_0(z) = 1 + z$ et $P_1(z) = z$. Donc le décodage est lui aussi à fenêtre glissante avec une fenêtre de longueur 2. On obtient

$$i_n = t_{n-1}^{(0)} + t_{n-1}^{(1)} + t_n^{(0)}.$$

On peut même obtenir une fenêtre de longueur 1 avec décalage de une unité de temps sur les indices. En effet, sur cet exemple puisque

$$\begin{array}{rcl} T_0(z) & = & (1 + z^2)I_0(z), \\ T_1(z) & = & (1 + z + z^2)I_0(z), \end{array}$$

on a, toujours modulo 2, $T_0(z) + T_1(z) = zI_0(z)$ et donc $i_n = t_{n+1}^{(0)} + t_{n+1}^{(1)}$. On déduit de ces calculs que si des erreurs surviennent sur la suite codée, ces erreurs ne se propagent pas au décodage.

Nous allons maintenant généraliser cet exemple. Un codage de convolution de taux $1 : k$ peut être obtenu en codant chaque bit source de la suite I_0 par k bits de $T_0, T_1, \ldots, T_{k-1}$ respectivement, où

$$T_i(z) = G_i(z)I_0(z) \quad 0 \le i \le k - 1.$$

Deux cas sont alors possibles.

– Si les polynômes G_i sont premiers entre eux, ou si leur pgcd est une puissance de z, il existe des polynômes P_i tels que

$$\sum_{i=0}^{k-1} P_i(z)G_i(z) = z^r.$$

Donc

$$\sum_{i=0}^{k-1} P_i(z)T_i(z) = I_0(z)z^r.$$

Donc le décodage se fait dans ce cas à l'aide d'une fenêtre glissante le long des suites T_i. Ce décodage est sans propagation d'erreur.

– Si le pgcd des polynômes G_i n'est pas une puissance de z mais vaut $z^r P(z)$ avec $P(z) = 1 + Q(z)$, $Q(0) = 0$ et $Q(z) \neq 0$. Il existe des polynômes P_i, premiers dans leur ensemble, tels que

$$\sum_{i=0}^{k-1} P_i(z)G_i(z) = P(z)z^r,$$

et

$$\sum_{i=0}^{k-1} P_i(z)T_i(z) = I_0(z)P(z)z^r,$$

Nous allons montrer qu'une erreur survenue sur la suite codée peut parfois se propager à l'infini. Un tel codage est dit catastrophique. Supposons qu'une erreur survienne sur un bit de T_i, $T_i(z)$ est changé en $\hat{T}_i(z) = T_i(z)+z^n$ pour un certain entier n. La suite T est décodée en $\hat{I}_0$ tel que $(\hat{I}_0 - I_0)(z)P(z)z^r = P_i(z)z^n$. Supposons que $(\hat{I}_0 - I_0)(z)$ est un polynôme, ce qui signifie que l'erreur ne se propage pas à l'infini lors du décodage. Alors $P(z) = 1 + Q(z)$, avec $Q(0) = 0$, divise $P_i(z)z^n$. Donc $P(z)$ divise $P_i(z)$. Pour que le décodage soit sans propagation d'erreur quel que soit le bit sur lequel survient l'erreur, il faut donc que $P(z)$ divise $P_i(z)$ pour tout i, ce qui contredit le fait que les P_i sont premiers dans leur ensemble. Dans ce cas, aucun décodage ne peut être à fenêtre glissante.

6.2.2 La méthode automatique

Pour pouvoir corriger des erreurs, c'est une autre méthode de décodage qui est utilisée en pratique. Il s'agit d'un décodage dit en treillis.

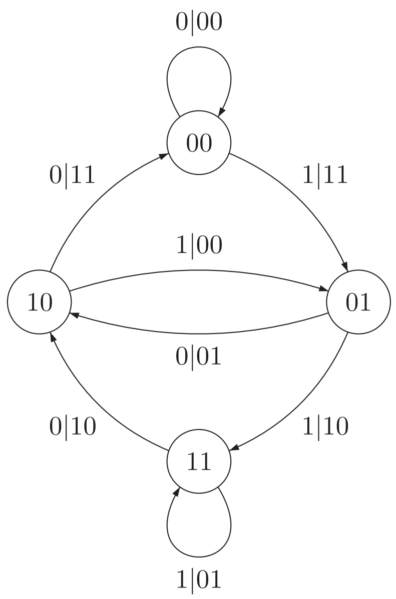

FIG. 12 – Transducteur de codage.

On représente tout d'abord le codeur à l'aide d'un transducteur. Il s'agit d'un automate fini dont les flèches ont des entrées et des sorties. L'automate d'entrée du transducteur est obtenu en masquant les sorties et l'automate de sortie est obtenu en masquant les entrées. En général il y a une lettre en entrée et une lettre en sortie, chaque lettre pouvant elle-même représenter un bloc de plusieurs bits. Le codage décrit ci-dessus est représenté par le transducteur de la figure 12 dont les états sont des blocs de deux bits source représentant la mémoire de l'entrée, et dont les flèches sont définies par

$$(i_{n-2}i_{n-1}) \xrightarrow{i_n|t_n^{(0)}t_n^{(1)}} (i_{n-1}i_n).$$

En recodant les sorties par

$$\begin{aligned} a &= 00 \\ b &= 01 \\ c &= 10 \\ d &= 11 \end{aligned}$$

on obtient le transducteur de la figure 13.

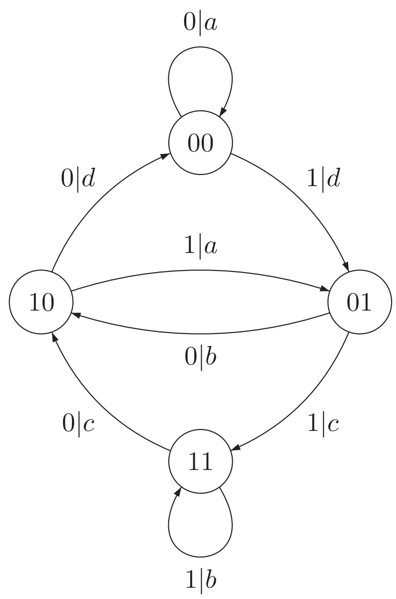

FIG. 13 – Transducteur de codage lettre à lettre.

Nous donnons maintenant quelques définitions plus techniques sur les automates. La plus importante est la notion de localité qui traduit des propriétés de synchronisation très fortes d'un automate fini.

Un chemin bi-infini dans l'automate est parfois noté $((p_i, a_i, p_{i+1}))_{i\in\mathbb{Z}}$. Un chemin fini de longueur n sera noté $((p_i, a_i, p_{i+1}))_{0\leq i<n}$. Dans les deux cas, p_i est appelé état de passage à l'instant i.

Un automate est dit non-ambigu, ou sans diamant, si deux chemins qui partent d'un même état, arrivent sur un même état et ont même étiquette, sont égaux.

Un automate déterministe admet au plus une flèche partant d'un état donné avec une étiquette donnée. Il est dit déterministe complet pour un certain alphabet si de chaque état part exactement une flèche d'étiquette donnée dans l'alphabet.

Soient m et a deux entiers positifs ou nuls. Un automate est dit (m, a)-local, ou (m, a)-défini, si chaque fois que l'on a deux chemins de longueur $m+a$, $((p_i, a_i, p_{i+1}))_{0\leq i<(m+a)}$ et $((p'_i, a_i, p'_{i+1}))_{0\leq i<(m+a)}$, de même étiquette, alors $p_m = p'_m$. Il s'agit d'une propriété de confluence des chemins à un instant donné. On remarquera que cette notion est non orientée. Un automate est local, ou défini, s'il est (m, a)-local pour deux entiers positifs ou nuls m et a (m est po ur mémoire et a pour anticipation). Lorsque l'automate est déterministe local, la propriété est satisfaite avec une anticipation nulle. La plus petite valeur de $m + a$ telle qu'un automate local est (m, a)-local est parfois appelée délai de synchronisation de l'automate.

Le transducteur de la figure 13 est déterministe en entrée et local en sortie. Sur cet exemple, le transducteur est même déterministe local en sortie.

L'algorithme utilisé pour le codage est le suivant. On choisit un état de départ, on lit la suite à coder sur l'entrée de l'automate. On produit le mot lu en sortie. La suite à coder peut être supposée infiniment longue en théorie. Il ne s'agit donc pas d'un codage par blocs. En pratique, les suites à coder sont finies mais elles sont très longues.

L'algorithme utilisé pour le décodage s'appelle l'algorithme de Viterbi. Il s'agit d'un algorithme de programmation dynamique. On part de l'état de départ fixé lors du codage. Le principe est le suivant. On veut trouver le chemin de l'automate issu de l'état de départ dont l'étiquette de sortie a la même longueur que la suite à décoder et est la plus proche possible de la suite à décoder. La notion de proximité peut varier. On considérera tout d'abord la distance dite de Hamming pour laquelle deux suites sont d'autant plus proche que leur nombre de bits différents est petit.

On appelle distance de Hamming entre deux suites de même longueur, finies ou infinies, le nombre de bits différents entre les deux suites. Parfois, d'autres distances sont prises en compte. On note $d(\mathbf{u}, \mathbf{v})$ la distance entre deux suites $\mathbf{u}$ et $\mathbf{v}$. Si les sorties ont été recodées, on tient compte du nombre de bits distincts qu'elles représentent. Ainsi $d(aaa, bbb) = 3$ et $d(aaa, ddd) = 6$.

On note Q l'ensemble des états du transducteur. L'état correspondant à un bloc de 0 est noté $\mathbf{0}$. On choisit cet état comme état de départ au moment du codage.

Description de l'algorithme de Viterbi

Soit $t = t_1 \dots t_N$ la suite à décoder, on effectue N étapes de la façon suivante.

- Le coût d'un chemin de longueur i est la distance de Hamming entre son étiquette de sortie et le préfixe de la suite à décoder $t_1 t_2 \dots t_i$.
- À l'étape i $(1 \leq i \leq N)$, et pour chaque état q de l'automate, on mémorise un unique chemin, noté $c(q, i)$ de longueur i, d'origine l'état de départ $\mathbf{0}$ et d'arrivée l'état q, et de coût minimal parmi tous ces chemins. On note ce coût $w(q, i)$. À chaque étape, le nombre de chemins mémorisés est donc borné par le nombre d'états du transducteur.
- L'algorithme de calcul des coûts et chemins est le suivant. On note π le prédécesseur d'un sommet sur un chemin.

  Viterbi(automate de sortie du codeur, suite t)

pour i allant de 0 à N et tout état q **faire**
$w(q, i) \leftarrow \infty$
$w(\mathbf{0}, 0) \leftarrow 0$
pour i allant de 1 à N **faire**
pour toute flèche (p, a, q) de l'automate **faire**
si $(w(p, i-1) + d_H(a, t_i) < w(q, i))$ **alors**
$w(q, i) \leftarrow w(p, i-1) + d_H(a, t_i)$
$\pi(q, i) = (p, i-1)$
return (p, N) tel que $w(p, N) = \min_{q \in Q} w(q, N)$.

Lorsque plusieurs couples (p, N) minimisent les $w(q, N)$, on en choisit un aléatoirement. La complexité en temps et espace est donc $O(|A| \times |Q| \times N)$, où A est l'alphabet recodé en sortie.

Ce décodage s'appelle aussi décodage en treillis. La figure 14 illustre le calcul lorsque l'état de départ du codeur est l'état 00 et lorsque la suite à décoder est la suite

$$acd = 001011.$$

La suite est donc corrigée ici en $aad = 000011$. Un bit a été corrigé.

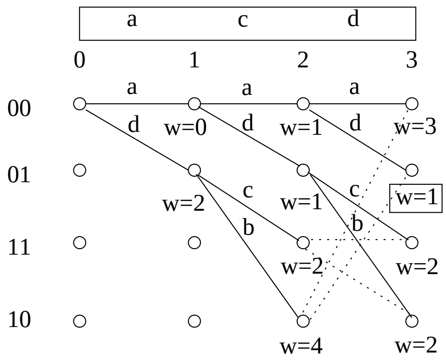

FIG. 14 – L'algorithme de Viterbi

6.2.3 Calcul de la distance libre

Nous allons évaluer la performance de ce décodage. Le principe de l'algorithme de Viterbi repose sur le fait que la recherche d'un chemin d'étiquette de sortie la plus proche de la suite codée permet de retrouver le chemin utilisé lors du codage, et ainsi la suite qui a été codée. Lorsque la suite s'est altérée dans le canal, l'algorithme peut conduire à un mauvais chemin. On cherche donc à calculer le taux d'erreur résiduel pour cet algorithme de codage et décodage. Plus précisément, si p est le paramètre d'erreur du canal binaire symétrique, on cherche à évaluer la probabilité pour qu'un symbole 0 ou 1 de la suite décodée soit faux. Le paramètre adapté pour l'appréciation de la qualité du décodage de Viterbi s'appelle la distance libre.

La distance libre d'un automate est la distance (de Hamming) minimale entre deux mots bi-infinis qui sont étiquettes de deux chemins bi-infinis distincts. On note la distance libre d_f.

PROPOSITION 13 *La distance libre d'un automate est non nulle si et seulement si cet automate est local. Dans ce cas, elle est aussi la plus petite distance de Hamming entre les étiquettes de deux chemins finis distincts de même origine et de même arrivée.*

Preuve. Exercice. □

PROPOSITION 14 *La sortie du transducteur est un automate local si et seulement si le pgcd des polynômes G_i utilisés dans le codage de convolution est une puissance de z.*

Preuve. Exercice. □

Dans la suite nous considérerons uniquement des codages de convolution tels que la sortie du transducteur est un automate local, l'autre cas correspondant aux codes catastrophiques.

Dans le cas d'un code de convolution, on a un codage dit linéaire : si T et T' sont deux suites infinies codées alors $T + T'$ aussi. En effet, on a $T(z) = G(z)I(z)$, $T'(z) = G(z)I'(z)$ et donc $T(z) + T'(z) = G(z)(I(z) + I'(z))$. La distance minimale entre deux suites codées infinies est donc aussi dans ce cas la distance minimale entre une suite codée infinie et la suite nulle infinie. C'est encore le poids minimal d'une suite codée infinie non nulle. Ce poids est le poids minimal de l'étiquette non nulle d'un chemin de la sortie du transducteur allant de $\mathbf{0}$ à $\mathbf{0}$. Il s'agit donc de la distance libre de l'automate.

La distance libre de l'automate de sortie du transducteur de la figure 13 est $d_f = 5$.

La distance libre de l'automate de sortie du transducteur permet d'évaluer le taux d'erreur résiduel par bit du décodage de Viterbi. On peut en effet montrer que la probabilité pour qu'un bit quelconque de la suite décodée soit erroné est majorée par

$$P_{d_f,p} = \sum_{i=m}^{d_f} \binom{d_f}{i} p^i (1-p)^{d_f - i},$$

avec $m = \frac{d_f - 1}{2}$ si d_f est impair et $m = \frac{d_f}{2}$ si m est pair. On a $P_{d_f,p} \sim Kp^{d_f/2}$, où K est un constante positive (sur l'exemple, on a $P_{d_f,p} \sim 15p^2$). Cette probabilité est d'autant meilleure que d_f est grand.

Calcul de la distance libre

- si la sortie du transducteur n'est pas locale, $d_f = 0$.
- sinon, on modifie l'automate de sortie du codeur de la façon suivante. On élimine pour cela la boucle d'étiquette 0 sur l'état $\mathbf{0}$. On duplique cet état en $\mathbf{0}_1$ et $\mathbf{0}_2$ de telle sorte que les flèches qui sortaient de $\mathbf{0}$ sortent de $\mathbf{0}_1$ et les flèches qui arrivaient sur $\mathbf{0}$ arrivent sur $\mathbf{0}_2$. On calcule par l'algorithme de Dijkstra (voir le cours d'algorithmique) un plus court chemin allant de l'état $\mathbf{0}_1$ à l'état $\mathbf{0}_2$. Le coût de ce chemin est la distance libre du codage.

Le temps de calcul est donc $O(|Q| \log |Q|)$.

Sur l'exemple, on obtient l'automate modifié sur la figure 15 et un plus court chemin de coût 5 représenté sur la figure 16.

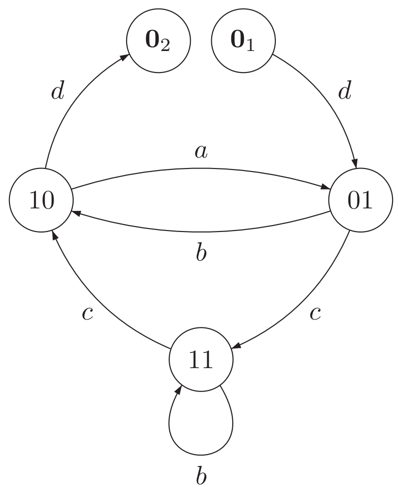

FIG. 15 – Automate modifié.

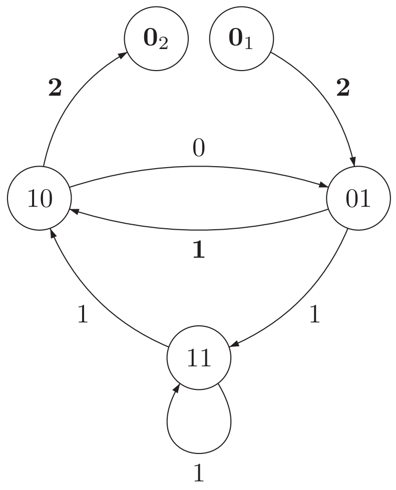

FIG. 16 – Calcul de la distance libre : $d_f = 5$.

7 Compression de texte sans perte

La compression d'un texte sans perte est un codage de source. Tous les codages tendent à s'approcher de la borne donné par le premier théorème de Shannon. Il est a noter que tout codage (injectif) ne peut comprimer tous les textes. En effet supposons qu'un codage c transforme tout texte de longueur n sur un alphabet A en un texte de longueur inférieure ou égale à n sur ce même alphabet, et qu'il transforme au moins un texte de longueur n en un texte strictement plus court. Il réalise alors une fonction de $A^{\leq n}$ dans $A^{\leq n}$ privé d'un mot. Un tel codage ne peut être injectif.

Nous présentons trois méthodes de compression de texte sans perte. Ces trois méthodes admettent diverses variantes.

7.1 Le codage de Huffman statique

Le codage de Huffman a été décrit dans le cours d'algorithmique. Nous en rappelons brièvement le principe et nous renvoyons au cours d'algorithmique pour une étude de la complexité et de l'implantation.

7.1.1 Description du codage de Huffman statique

Un code de Huffman est un code préfixe binaire associé à une source discrète sans mémoire (A, p) de cardinal n. On le construit par récurrence sur n.

- Pour $n = 1$, le code est formé du mot vide.
- Pour $n = 2$, le code est l'ensemble $\{0, 1\}$.
- Pour $n > 2$, soient a_1 et a_2 deux lettres de A de plus petits poids, où le poids est donné par la distribution p. Soit $B = A - \{a_1, a_2\} + \{b\}$, où b n'appartient pas à A. Le nouvel alphabet B est muni de la distribution p en définissant $p(b) = p(a_1) + p(a_2)$. Soit C un code de Huffman pour l'alphabet B, de cardinal $n - 1$, associé à B par c. On définit le code C' associé à A par c' de la façon suivante. On a

$$\begin{aligned} c'(a) &= c(a) \text{ pour } a \neq a_1, a_2, \\ c'(a_1) &= c(b)0 \\ c'(a_2) &= c(b)1. \end{aligned}$$

Un code est dit optimal pour une source discrète sans mémoire s'il minimise la longueur moyenne du code d'une lettre parmi tous les codes possibles pour cette source. Comme on ne considère pas des blocs sources de taille l avec $l > 1$, on dit qu'il s'agit d'un codage d'ordre 0.

Proposition 15 *Un code de Huffman pour une source discrète sans mémoire est optimal parmi tous les codes possibles pour cette source.*

7.1.2 Efficacité du codage de Huffman

On considère que le codage est appliqué sur une source sans mémoire A de cardinal k de distribution de probabilité p.

L'efficacité du codage est

$$E = \frac{H(A)}{\bar{m}},$$

où $\bar{m}$ est la longueur moyenne d'un mot du code. On a

$$\begin{aligned} H(A) &= -\sum_{a \in A} p(a) \log p(a) \\ \bar{m} &= \sum_{a \in A} p(a) |c(a)|, \end{aligned}$$

où $|c(a)|$ est la longueur du mot du code associé à la lettre a.

On sait (voir le calcul de $H(A)-\bar{m}$ pour le 1er théorème de Shannon) que $E \leq 1$. On sait également (voir toujours le 1er théorème de Shannon) que cette efficacité n'est égale à 1 que si $p(a) = 2^{-|c(a)|}$ ce qui n'est pas toujours le cas puisque les probabilités ne sont pas toujours des inverses de puissance de 2. Le codage de Huffman n'est donc pas optimal.

7.2 Codage de Huffman adaptatif

Une variante de la méthode que nous présentons a donné lieu à la commande Unix `compact`.

Le codage de Huffman dynamique, ou adaptatif, fonctionne en transformant successivement un arbre correspondant à la partie du texte déjà traité. Les arbres construits sont d'un type un peu spécial et dits *arbre de Huffman évolutifs*. Ce sont, par définition, des arbres binaires complets pondérés par une fonction p dont les valeurs sont des entiers strictement positifs (sauf éventuellement une qui est nulle) et qui vérifie la condition suivante : les noeuds peuvent être ordonnés en une suite $(x_1, x_2, \ldots, x_{2n-1})$, où n est le nombre de feuilles de telle sorte que :

1. la suite des poids $(p(x_1), p(x_2), \ldots, p(x_{2n-1}))$ est croissante.
2. pour tout $i, 1 \leq i \leq n$, les noeuds x_{2i-1} et x_{2i} sont frères.

7.2.1 La compression

Supposons déjà construit l'arbre de Huffman évolutif $H(T)$ correspondant au texte T et examinons le calcul de l'arbre $H(Ta)$ correspondant à l'ajout au texte T de la lettre suivante a. Deux cas se présentent :

1. La lettre a est déjà apparue dans le texte T.

   La traduction de a est dans ce cas le mot binaire représentant le chemin de la racine à la feuille représentant a dans $H(T)$. On augmente d'une unité le poids de la feuille x_i correspondant à la lettre a. Si la propriété 1 de la définition des arbres de Huffman adaptatifs n'est plus vérifiée, on échange x_i avec le noeud x_j où j est le plus grand indice tel que $p(x_i) > p(x_j)$, sauf si x_j est le père de x_i. On répète cette opération avec le père de x_i jusqu'à atteindre la racine. Lorsqu'on échange les noeuds, les sous-arbres enracinés en ces noeuds sont également échangés.

2. La lettre a n'a pas encore été rencontrée dans le texte T.

   On maintient dans l'arbre une feuille de poids nul destinée à accueillir les nouvelles lettres. On code a par le chemin correspondant à cette

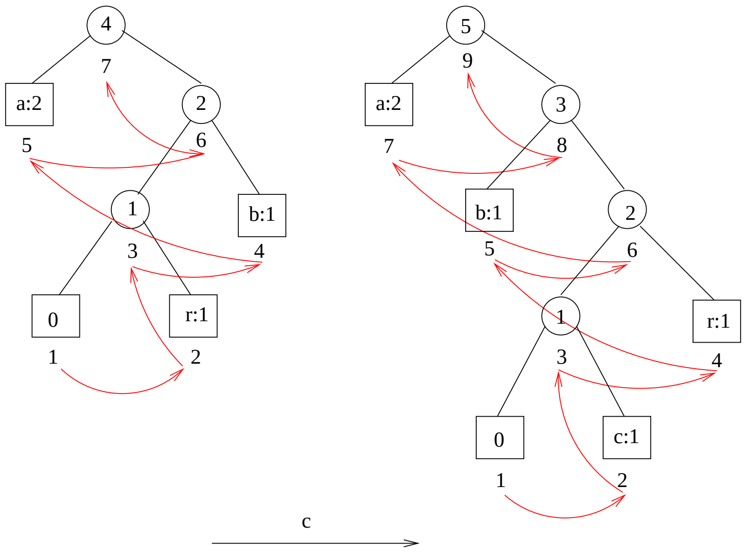

FIG. 17 – Le passage de $H(abra)$ à $H(abrac)$

feuille suivi du code de départ de la lettre (son code ASCII). Elle est ensuite remplacée par une arbre pondéré à deux feuilles. La première est la nouvelle feuille de poids nul. La seconde, de poids 1, est associée à la lettre a. On répète les opérations de mise à jour de l'arbre à partir du père de ces feuilles comme ci-dessus en 1.

7.2.2 La décompression

Il n'est pas nécessaire de transmettre l'arbre, comme dans le cas du code de Huffman non évolutif, pour pouvoir décoder. Si une suite binaire w correspondant à un texte T a déjà été décodée, l'algorithme dispose d'un arbre $H(T)$. On regarde à quelle feuille de $H(T)$ correspondent les bits suivants à décoder. On déduit de cette feuille et de ces bits de quelle lettre a il s'agit. On a ainsi décodé la suite correspondant au texte Ta et on construit $H(Ta)$ pour continuer le décodage.

Par exemple, si $T = abra$, le passage de l'arbre $H(abra)$ à l'arbre $H(abrac)$ est représenté sur la figure 17. On a indiqué en dessous de chaque sommet son indice dans la suite x_i, ainsi que le chaînage des suivants dans cette suite (en rouge).

7.3 Codage de Ziv-Lempel

Nous présentons maintenant la méthode de Lempel et Ziv (LZ78) dont une variante est utilisée dans la commande unix `compress`.

Elle consiste à coder une suite de caractères de la manière suivante : cette suite est découpée en phrases. Chaque phrase est constituée d'une phrase rencontrée précédemment (la plus longue possible) plus un caractère. Les phrases sont numérotées (dans l'ordre où elles sont rencontrées) et chacune d'elle est codée par le numéro de la phrase précédente sur laquelle elle est construite plus le caractère à ajouter. La phrase vide a pour numéro 0.

Voici un exemple :

texte initial :	$aaabbabaabaaabab$						
phrases :	a	aa	b	ba	baa	$baaa$	bab
numéro de phrase :	1	2	3	4	5	6	7
texte codé :	$(0, a)$	$(1, a)$	$(0, b)$	$(3, a)$	$(4, a)$	$(5, a)$	$(4, b)$

Les caractères peuvent être codés par leur code ASCII mais on peut aussi considérer toutes les variantes où les caractères sont codés sur 1 bit (alphabet à deux lettres), 2 bits (alphabet à 4 lettres), 3 bits (alphabet à 8 lettres), etc. , 8 bits (alphabet à 256 lettres).

D'autre part, le nombre k de phrases déjà rencontrées est variable. Aussi pour économiser sur la taille du codage des numéros de phrases, la phrase courante est codée par un caractère et le numéro d'une des k phrases déjà rencontrées qui est codé sur $\lceil \log k \rceil$ bits.

Ainsi sur l'exemple ci-dessus avec l'alphabet $A = \{a, b\}$, on prend **0** comme code de a et **1** comme code de b. Dans ce cas, le texte codé de

$$aaabbabaabaaabab$$

devient (en marquant le code de la lettre en gras et en ajoutant des espaces blancs pour la lisibilité)

$$0\mathbf{0}\ 1\mathbf{0}\ 00\mathbf{1}\ 11\mathbf{0}\ 100\mathbf{0}\ 101\mathbf{0}\ \ 100\mathbf{1}.$$

Remarquez que sur cet exemple, le codage est une expansion !

7.3.1 La compression

Pour que la compression soit rapide, il faut utiliser une structure de donnée arborescente appelée « trie ». Le problème consiste en effet à trouver rapidement le numéro de la plus longue phrase rencontrée précédemment qui se retrouve à la position courante de lecture dans le fichier. Un trie répond à cette attente, il permet de stocker toutes les phrases rencontrées précédemment sous forme d'arbre.

Chaque phrase correspond à un nœud de l'arbre. La phrase 0 qui représente la suite vide est la racine de l'arbre. Les fils de chaque nœud sont numérotés par une lettre (on dit que cette lettre est l'étiquette de l'arc entre le père et le fils). Un nœud de l'arbre correspond à la phrase obtenue en lisant les étiquettes sur les arcs du chemin de la racine au nœud. Le numéro de la phrase est stocké dans le nœud correspondant de l'arbre. (Un nœud peut avoir autant de fils qu'il y a de lettres dans l'alphabet.)

Le moyen le plus simple de comprendre cela est d'examiner le trie obtenu après la lecture du texte de l'exemple vu plus haut (voir la figure 1).

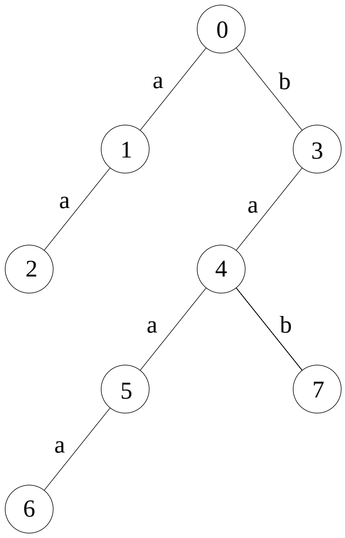

FIG. 18 – Le trie obtenu après lecture du texte « aaabbabaabaaabab ».

Le traitement de la phrase courante est le suivant. On lit les lettres du texte à partir de la position courante une à une et on parcourt le trie en partant de la racine en même temps. Chaque fois que le fils du nœud où l'on se trouve correspondant à la lettre lue existe, on se positionne à ce nœud. Si le fils n'existe pas, on le créé et on s'arrête là, c'est le nœud qui correspond à la phrase courante. On a donc lu une phrase de plus et on peut passer à la suivante.

Par exemple, si le texte de la figure 1 se poursuit par « babbbaab », on ajoutera la phrase 8 « babb » comme fils d'étiquette b du nœud 7 et la phrase 9 « baab » comme fils d'étiquette b du nœud 5.

Pour les gros fichiers, il ne faut pas saturer la mémoire avec un trop gros trie. Une stratégie consiste à recommencer à zéro quand on a lu un certain nombre de caractères. Une autre consiste à recommencer à zéro quand le trie atteint une certaine taille. Cette éventuelle optimisation doit être prise en compte lors de la décompression. L'idéal étant de maintenir l'espace mémoire utilisé par le programme en dessous d'un certain seuil, 1 Mo par exemple.

7.3.2 La décompression

Lors de la décompression, on stocke dans un tableau toutes les phrases déjà rencontrées pour pouvoir décoder immédiatement la phrase courante. La décompression est très rapide et ceci est un des points forts de la méthode de Ziv-Lempel.

7.4 Compression de Burrows-Wheeler

La compression de Burrows-Wheeler utilise une transformation, dite transformation de Burrows-Wheeler (BWT) [5]. Des compresseurs à base de cette méthode sont implémentés dans les commandes `bzip2` (see [11]),`szip`. Ce sont les meilleurs compresseurs de texte sans perte actuels.

L'idée est de changer la suite de lettres w à compresser par une permutation $\mathrm{BWT}(w)$ de w de telle sorte que cette permutation soit plus facile à compresser.

7.4.1 La transformation de Burrows-Wheeler (BWT)

7.4.2 Définition de la transformée

Soit $w = w_0 w_1 \dots w_{n-1}$ le mot à comprimer. Sa transformée $\mathrm{BWT}(w)$ est obtenue de la façon suivante :

- On construit tous les décalages circulaires du mot w.
- On trie ces mots suivant l'ordre lexicographique et on les écrit dans l'ordre comme les lignes dans une matrice.
- $\mathrm{BWT}(w)$ est la dernière colonne de cette matrice.
- La sortie de la transformation de Burrows-Wheeler est le mot $\mathrm{BWT}(w)$ et l'indice I de la ligne de la matrice qui contient w.

$$\begin{array}{ccccccccc} & & & F & & & & & L \\ & & & \downarrow & & & & & \downarrow \\ & & 1 & a & a & b & r & a & c \\ I & \rightarrow & 2 & a & b & r & a & c & a \\ & & 3 & a & c & a & a & b & r \\ & & 4 & b & r & a & c & a & a \\ & & 5 & c & a & a & b & r & a \\ & & 6 & r & a & c & a & a & b \end{array}$$

Sur l'exemple ci-dessus, BWT(w) est *caraab* et $I = 2$ puisque le mot original apparaît en deuxième ligne. On remarque que la première colonne, notée F, contient la suite des lettres de w dans l'ordre lexicographique.

7.4.3 Réversibilité de la transformation

La transformée de Burrows-Wheeler est réversible. Connaissant BWT(w) et I, il est possible de retrouver w. Nous montrons comment retrouver w sur l'exemple ci-dessus. On note L la dernière colonne de la matrice et F la première. La propriété fondamentale qui est utilisée est la suivante : si F_i (resp. L_i) designe la ième lettre de F (resp. de L), F_i est le symbole qui suit L_i dans w pour tout $i \neq I$.

$$\begin{array}{ccccccccc} & & & F & & & & & L \\ & & & \downarrow & & & & & \downarrow \\ & & 1 & a_3 & a_1 & b & r & a_2 & c \\ I & \rightarrow & 2 & a_1 & b & r & a_2 & c & a_3 \\ & & 3 & a_2 & c & a_3 & a_1 & b & r \\ & & 4 & b & r & a_2 & c & a_3 & a_1 \\ & & 5 & c & a_3 & a_1 & b & r & a_2 \\ & & 6 & r & a_2 & c & a_3 & a_1 & b \end{array}$$

Cette propriété permet de retrouver w à partir de L et I.

- Le dernier symbole de w est L_I. Sur l'exemple, il s'agit d'un a.
- Ce a se retrouve en colonne F. De quel a s'agit-il ? En fait, si ce a est le premier a à apparaître dans L, ce sera aussi le premier a à apparaître dans F. En effet l'ordre dans lequel les a apparaissent dans F et L est le même. L'ordre des a dans F est déterminé par l'ordre lexicographique du contexte droit de a de taille $|w|-1$ dans ww, ici $abrac, braca, caabr$. Or l'ordre des a dans L est déterminé par l'ordre lexicographique du contexte droit du a de taille $|w| - 1$ dans ww également. Ce contexte apparaît à gauche du a sur la même ligne. Il s'agit donc sur l'exemple

du premier a de F (le troisième a de w). La lettre qui précède est donc la dernière lettre de ligne 1, c'est un c.

- Le c apparaît comme étant F_5. Donc la lettre qui précède ce c est L_5, soit un a (noté a_2). C'est le troisième a de F, soit F_3. Le lettre de w qui précède ce a est $L_3 = r$. Lettre r est en F_6. Donc $L_6 = b$ précède r. Comme $F_4 = b$, $L_4 = a$ précède ce b dans w. Enfin ce a apparaît en position 2 dans L et le processus s'arrête car $I = 2$.
- Le mot obtenu est bien *abraca*.

7.4.4 Interêt de la transformation

Le mot obtenu $\mathrm{BWT}(w)$ a la propriété remarquable suivante : pour chaque facteur x de w, les caractères qui précèdent x dans w sont regroupés dans $\mathrm{BWT}(w)$. Pour un facteur x assez grand, ces caractères sont presque toujours la même lettre et ainsi le mot $\mathrm{BWT}(w)$ est localement homogène, c'est-à-dire, il consiste en la concaténation de plusieurs facteurs ayant peu de lettres distinctes. C'est pour cette raison que le mot $\mathrm{BWT}(w)$ est ensuite plus facile à compresser.

7.4.5 Le codage complet

Burrows et Wheelers ont suggéré de coder ensuite $\mathrm{BWT}(w)$ à l'aide du codage dit "Move-To-Front" (MTF).

Dans le codage MTF, un symbole a et codé par un entier égal au nombre de symboles distincts rencontrés depuis la dernière occurrence de a. Le codeur maintient une liste de symboles ordonnés en fonction de la date de leur dernière apparition. Quand le symbole suivant arrive, le codeur renvoie sa position dans la liste et le met en tête de liste. Cette opération est également réversible.

Une suite sur l'alphabet $A = \{a_1, a_2, \ldots, a_k\}$ est transformée en une suite sur l'alphabet $\{0, \ldots, k-1\}$.

Par exemple, si $\mathrm{BWT}(w) = caraab$ et si l'alphabet a trois lettres a, b, c, r, on part d'une liste des positions arbitraire $l = a, b, c, r$. Le codage de chaque lettre de $\mathrm{BWT}(w)$ est donné ci-dessous avec la nouvelle liste des positions.

c	2	$l = c, a, b, r$
a	1	$l = a, c, b, r$
r	3	$l = r, a, c, b$
a	1	$l = a, r, c, b$
a	0	$l = a, r, c, b$
b	3	$l = a, b, r, c$

La suite MTW(BWT(w) est donc 213103.

Ansi la suite

$$\hat{w} = \mathrm{MTF}(\mathrm{BWT}(w))$$

a la même taille que la suite w, mais cette suite est très fortement compressible car elle va contenir beaucoup de petit entiers, principalement des 0 et des 1. Par exemple, sur un texte en anglais, elle contient 50% de 0.

Pour compresser cette dernière suite, on peut utiliser un compresseur simple comme Huffman statique, ou encore un codage arithmétique[1].

Le codeur se résume donc à

$$w \rightarrow \mathrm{Huffman}(\mathrm{MTF}(\mathrm{BWT}(w))).$$

Cette transformation est reversible car chacune des trois étapes l'est.

7.4.6 Mise en œuvre

La compression se fait en découpant le texte source en blocs de taille fixe. La taille d'un bloc se situe en pratique entre 100Kb et 900Kb.

Pour un calcul plus rapide de la transformée de Burrows-Wheeler, on calcule en fait la transformée du mot $w\$$, où \$ est un symbole qui n'appartient pas à l'alphabet et que l'on suppose inférieur à tous les autres pour l'ordre alphabétique.

La transformée de $abraca\$$ est $\mathrm{BWT}(abraca\$) = ac\$raab$. On renvoie $acraab$ et l'indice $I = 3$. Le symbole \$ est en position I dans $ac\$raab$.

$$\begin{array}{ccccccccc}
 & & & F & & & & & L \\
 & & & \downarrow & & & & & \downarrow \\
 & & 1 & \$ & a & b & r & a & c & a \\
 & & 2 & a & \$ & a & b & r & a & c \\
I & \rightarrow & 3 & a & b & r & a & c & a & \$ \\
 & & 4 & a & c & a & \$ & a & b & r \\
 & & 5 & b & r & a & c & a & \$ & a \\
 & & 6 & c & a & \$ & a & b & r & a \\
 & & 7 & r & a & c & a & \$ & a & b
\end{array}$$

L'intérêt de l'ajout du symbole \$ réside dans le fait que les permutations circulaires de $w\$$ sont données par l'ordre lexicographiques des suffixes de w. Or il existe plusieurs structures, toutes calculables en temps linéaire, pour représenter les suffixes d'un mot dans l'ordre lexicographique, l'automate

[1] non décrit dans ce cours

déterministe des suffixes, l'automate compact des suffixes, le trie compact des suffixes. On peut aussi calculer une table des suffixes en temps $O(n \log n)$ si $n = |w|$ ou en temps linéaire. Cette dernière structure est moins coûteuse en place.

Les suffixes de $w = abraca$ dans l'ordre lexicographique sont

ε
a
$abraca$
aca
$braca$
ca
$raca$

Si le mot est indicé à partir de 0 :

0	1	2	3	4	5
a	b	r	a	c	a

,

la table des suffixes est un tableau A tel que $A[i]$ est l'indice dans w du début du $i$ème suffixe pour l'ordre lexicographique.

$A =$

6	5	0	3	1	4	2

.

On peut calculer immédiatement BWT($w\$$) à partir de w et de la table A des suffixes de w. En effet, la $i$ème lettre du mot codé est la lettre qui précède le $i$ème suffixe dans w, c'est-à-dire la lettre en position $A[i] - 1$, où \$ si $A[i] = 0$.

Pour calculer les structures ci-dessus, nous renvoyons à [6].

8 Codage de canal contraint

8.1 Canal contraint

Un canal contraint est un ensemble de suites de symboles caractérisé par des contraintes sur ces suites. Ces contraintes sont en général l'autorisation ou l'interdiction de blocs particuliers (contraintes de spectre).

On appelle canal sofique ou canal régulier un canal défini comme l'ensemble des suites étiquettes d'un chemin d'un automate fini. On notera par S l'ensemble de suites bi-infinies étiquettes de chemins bi-infinis de l'automate. Tous les états sont alors initiaux et terminaux dans cet automate.

Exemple : Le canal de la partie droite de la figure 19 est appelé canal du nombre d'or. Il représente les suites qui ont un nombre pair de 0 entre deux 1. Le canal de la partie gauche est dit canal sans contrainte. Il représente toutes les suites sur l'alphabet $\{0, 1\}$.

Fig. 19 – Canaux sofiques

On appelle canal de type fini l'ensemble des suites qui ne comportent pas de blocs appartenant à une liste finie de blocs interdits. Considérons par exemple la liste de blocs interdits $I = \{11\}$, les suites bi-infinies autorisées sont celles qui sont reconnues par l'automate de la figure 20.

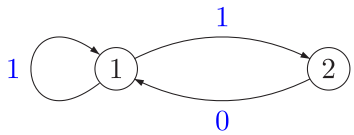

Fig. 20 – Canal de type fini

Un exemple très courant de contraintes rencontrées est celui des contraintes dites $[d, k]$. Les suites qui satisfont cette contrainte doivent comporter un minimum de d symboles 0 entre deux symboles 1 et un maximum de k symboles 0 consécutifs. Ces contraintes sont de type fini. En effet un ensemble fini possible de blocs interdits pour la contrainte $[d, k]$ est

$$\begin{array}{c} 11 \\ 101 \\ 1001 \\ \vdots \\ 1\overbrace{0\cdots0}^{d-1}1 \\ \overbrace{00\ldots00}^{k+1} \end{array}$$

Le canal $[2, 7]$ est représenté sur la figure 21. Cette contrainte modélise le canal du disque IBM 3380. Les CD audio utilise un canal $[2, 10]$.

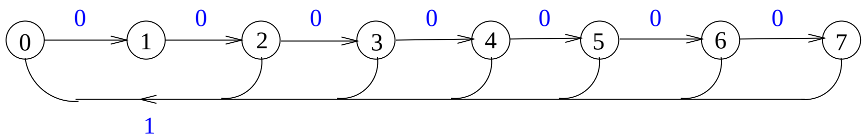

FIG. 21 – La contrainte $[2, 7]$.

La proposition suivante indique que les systèmes de type fini sont ceux qui peuvent être reconnu par un automate local.

PROPOSITION 16 *Un système de type fini peut être reconnu par un automate local. Inversement un système qui est reconnu par un automate local est de type fini.*

Preuve. Exercice. □

Le canal du nombre d'or par exemple n'est pas de type fini. On ne peut pas le caractériser par une liste finie de blocs interdits.

8.2 Capacité d'un canal

La notion de capacité que nous allons utiliser dans les codages pour canaux contraints est la suivante. Étant donné un canal S, sa capacité est définie par

$$\operatorname{cap}(S) = \limsup_{n \to \infty} \frac{1}{n} \log_2 \operatorname{card} B_n,$$

où B_n est l'ensemble des blocs finis (ou facteurs) pouvant apparaître dans une suite (bi-infinie) du canal.

Nous allons considérer des canaux sofiques irréductibles, c'est-à-dire qui peuvent être reconnu par un automate fini déterministe de graphe fortement connexe.

Proposition 17 *Pour tous les entiers positifs n, m,*

$$\operatorname{card} B_{n+m} \leq \operatorname{card} B_n \times \operatorname{card} B_m.$$

Preuve. Laissé en exercice. □

Pour ces canaux, on peut déduire de la proposition 17 que la limite supérieure ci-dessus est une limite simple et on peut calculer facilement la capacité du canal avec le théorème suivant dont on admettra la preuve.

Si G est un graphe, sa matrice d'adjacence est une matrice carrée telle que le coefficient d'indice p, q est le nombre de flèches de p vers q.

Proposition 18 *Soit S un canal sofique reconnu par un automate déterministe. La capacité de S est égale au logarithme en base deux de la plus grande valeur propre de la matrice d'adjacence du graphe de l'automate.*

Exemple : Considérons le canal S sans contrainte sur l'alphabet $\{0, 1\}$. On a $\operatorname{card} B_n = 2^n$, donc $\operatorname{cap}(S) = \log_2 2 = 1$.
Exemple : Considérons le canal du nombre d'or S de la figure 19. On peut démontrer que les matrices à coefficients positifs ont une valeur propre réelle supérieure ou égale au module de chacune des autres valeurs propres (théorème de Perron-Frobénius). On la nomme rayon spectral. La capacité de S est alors $\log \lambda$, où λ est le rayon spectral de la matrice

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

Le polynôme caractéristique de M est $X^2 - X - 1$. Donc $\lambda = \frac{1+\sqrt{5}}{2}$.

Étant donnés un système sofique S reconnu par un automate $\mathcal{A} = (Q, F)$ sur l'alphabet A, où Q est l'ensemble des états et F l'ensemble des flèches, et un entier positif r, on note S^r le système sofique reconnu par l'automate $\mathcal{A}^r = (Q, F')$ sur l'alphabet A^r, tel qu'il existe une flèche de p à q d'étiquette u dans $\mathcal{A}^r$ si et seulement s'il existe un chemin de p à q d'étiquette u dans $\mathcal{A}$.

Proposition 19 *Si S est un système sofique et r un entier positif, on a*

$$\operatorname{cap}(S^r) = r \operatorname{cap}(S).$$

Le résultat est également vrai si S n'est pas sofique.

Preuve. Si M est la matrice d'adjacence de $\mathcal{A}$, M^r est la matrice d'adjacence $\mathcal{A}^r$. Si λ est la plus grande valeur propre de M, alors λ^r est la plus grande valeur propre de M^r, d'où le résultat. □

8.3 Codage

Le codage pour canaux contraints consiste à transformer une suite sans contraintes en une suite qui satisfait une contrainte.

On a le résultat de codage pour canaux contraints suivant.

Théorème 20 *Soit S un canal de type fini irréductible de capacité strictement supérieure à $\log_2 k$, où k est un entier positif. Il existe un transducteur dont l'entrée est un automate déterministe complet reconnaissant une source sans contrainte sur un alphabet à k lettres et dont la sortie est un automate local reconnaissant un système de type fini inclus dans S.*

Pour coder une source sur k symboles dans ce canal, on utilise le transducteur comme dans un codage de Viterbi. Pour décoder, on utilise un décodage à fenêtre glissante. Le décodage ne permet pas de corriger des erreurs éventuelles mais évite la propagation des erreurs à l'infini.

Prenons l'exemple le canal S de la contrainte $[2, 7]$, sa capacité est strictement plus petite que 1 mais est plus grande que 1/2. On ne peut donc pas coder une source binaire avec un taux de transmission 1 : 1 dans le canal S. Comme la capacité de S^2 est strictement supérieure à 1, on va pouvoir coder une source binaire sans contrainte dans le canal S^2 avec un taux de transmission 1 : 1, ce qui revient à coder une source sans contrainte dans le canal S avec un taux de transmission constant égal à 1 : 2. Lorsqu'un tel système de codage est utilisé pour stocker des données sur un support, la capacité de stockage est donc réduite de moitié.

La canal S^2 est représenté sur la figure 22 avec $a = 00, b = 01, c = 10$.

Le codeur utilisé dans le disque IBM pour la contrainte $[2, 7]$ est le suivant. La figure 24 représente le décodeur que l'on peut implémenter avec une fenêtre glissante de longueur 4 seulement.

Nous allons illustrer la construction sur un exemple plus simple. Soit S le canal reconnu par l'automate $\mathcal{A}$ de la figure 25.

Sa capacité est supérieure à 1. Pour construire le codeur, on choisit un état, par exemple l'état 1 et on sélectionne des chemins de premier retour sur l'état 1 tels que l'ensemble de leurs étiquettes est un code fini C dont la distribution de longueur satisfait l'égalité de Kraft pour l'entier $k = 2$. On construit un code préfixe P sur un alphabet à deux lettres $\{0, 1\}$ de même

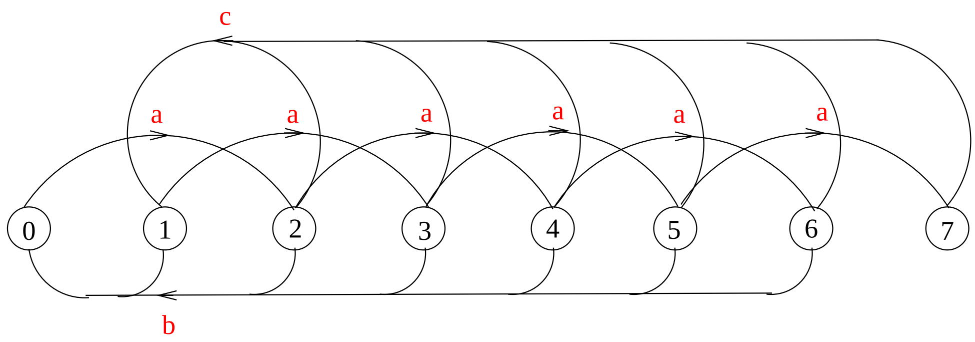

FIG. 22 – La contrainte [2, 7] au carré.

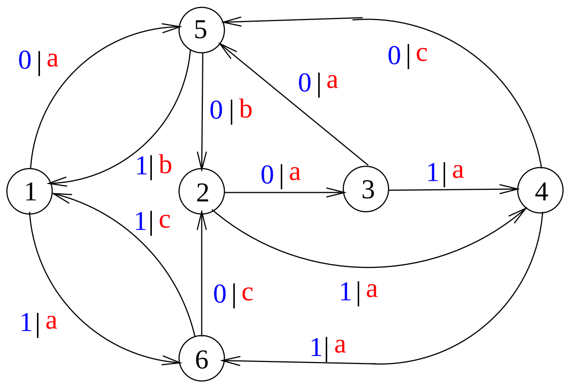

FIG. 23 – Le codage de Franaszek.

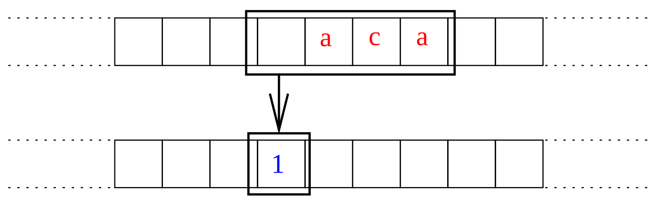

FIG. 24 – Le décodage à fenêtre glissante de longueur 4.

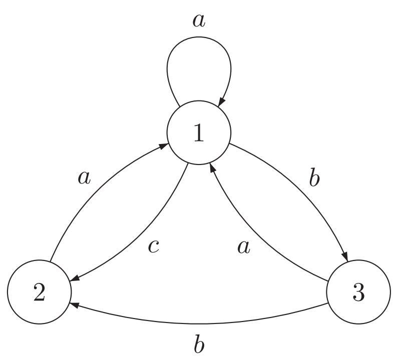

FIG. 25 – Canal de type fini S

distribution de longueur. On associe bijectivement un mot de C à un mot de P de même longueur. On forme ainsi des couples (p, c), où $p \in P$ et $c \in C$.

Le code C est un code circulaire fini (voir la feuille de travaux dirigés numéro 2). Son automate en pétales est dans ce cas un automate local.

On construit alors l'automate en pétales de ces couples de mots. On obtient un transducteur qui n'est pas déterministe mais qui est déterministe avec un certain délai fini. La sortie du transducteur est un automate local.

Sur l'exemple, on peut prendre

C	P
a	0
ba	10
ca	11

Le transducteur obtenu est donné sur la figure 26.

On peut ensuite transformer ce transducteur pour avoir un transducteur qui a les mêmes propriétés et est déterministe en entrée. On obtient le transducteur de la figure 27.

Dans le codage de Franaszek pour la contrainte $[2, 7]$, les codes utilisés sont les codes ci-dessous. Le code C représente des mots pouvant être lus en partant de l'état 2 ou de l'état 3 dans l'automate de la figure 22, et en

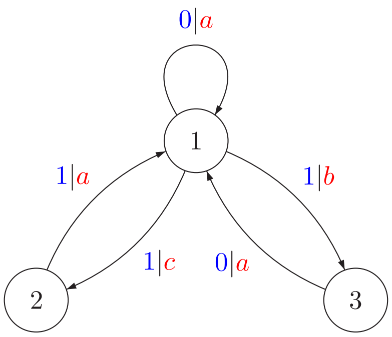

FIG. 26 – Codeur pour le canal S.

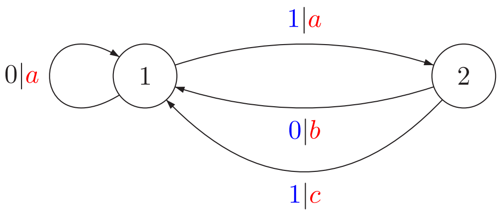

FIG. 27 – Codeur déterministe pour le canal S.

arrivant sur l'un de ces deux états. Ce code est encore un code circulaire.

C	P
ba	10
ca	11
aba	000
cba	010
aca	011
$acba$	0010
$aaca$	0011

Codes de Franaszek.

Bibliographie

[1] R. B. Ash, *Information Theory*, Dover Publications, Inc, New-York, 1990.

[2] G. Battail, *Théorie de l'Information. Application aux techniques de comunication*, Masson, 1997.

[3] M.-P. Béal, *Codage Symbolique*, Masson, 1993.

[4] R. E. Blahut, *Digital Transmission of Information*, Addison-Wesley, 1990.

[5] M. Burrows and D. J. Wheeler, *A block sorting lossless data compression algorithm*, Tech. Rep. 124, Digital Equipment Corporation, Paolo Alto, California, 1994.

[6] M. Crochemore, C. Hancart, and Th. Lecroq, *Algorithmique du Texte*, Vuibert, Paris, 2001.

[7] S. Eilenberg, *Automata, Languages and Machines*, vol. A, Academic Press, 1974.

[8] D. A. Lind and B. H. Marcus, *An Introduction to Symbolic Dynamics and Coding*, Cambridge, 1995.

[9] F. M. Reza, *An Introduction to Information Theory*, Dover Publications, Inc, New-York, 1994.

[10] N. Sendrier, *Théorie de l'Information.* communication personnelle.

[11] J. Seward, *The BZIP2 home page.* http ://www.bzip.org, 1997.