Chapitre 2

Codage de source

1. Introduction

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.

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 «· ·——». De tels codes sont dits codes à longueur variable. On les appelle aussi plus simplement code.

2. Débit moyen d'une information

Si **k** est la taille de l'alphabet (chaine de caractères ou un message) d'une source discrète sans mémoires **X**.

On appelle :

$\frac{1}{\tau_S}$: Vitesse d'émission/ vitesse d'émission de la source, exprimée en symboles/seconde.

τ_S: Le temps entre deux caractères transmis.

Débit d'information *:* Le débit moyen d'information d'une source sera défini comme étant le produit de l'entropie de la source (valeur moyenne de l'information propre par symbole) par le nombre moyen de symboles par seconde. En notant la durée moyenne d'un symbole par , le débit d'information de la source sera :

$$D = \frac{H(X)}{\bar{\tau}} = H(X) \times \bar{f} \qquad \text{Exprimé en bits/seconde ou Shannon/s.}$$

$\bar{f} = \frac{1}{\bar{\tau}}$: est la fréquence moyenne d'émission des symboles.

3. Codage avec mots de longueur fixe, efficacité

Nous prendrons par la suite le cas d'un codage binaire.

4.1. Mots de codes de longueur fixe :

Une manière simple de coder en binaire l'alphabet d'une source est d'attribuer à chaque symbole R bits.

Il y a donc 2^R codes possibles et bien sûr nous avons la condition $2^R \geq K$ l'égalité étant possible

- lorsque le nombre K de symboles de la source est une puissance de 2.
- Dans le cas contraire nous aurons : $2^{R-1} < K < 2^R$

Cette dernière relation permet de déterminer le nombre R de bits nécessaires au codage de l'alphabet d'une source de K symboles.

$$R = Int[Log(k)] + 1$$

Int : Integer (entier)

Nous savons que :

$$H(X) \leq Log_2(k)$$

$$R \geq Log_2(k)$$

L'égalité a lieu lorsque tous les symboles de la source sont équiprobables et lorsque K est une puissance de 2.

$$\text{Alors : } R \geq H(X)$$

Exemple :

Si $k = 15$ alors $R = Int[Log_2(15)] + 1 = Int[3{,}9] + 1 = 4$

Le nombre de codes possible est 2^4

4.2. L'efficacité :

Un codage est dit d'autant plus efficace que le nombre de codes possibles inutilisés est faible.

L'efficacité dépend aussi de la quantité d'information moyenne de la source. L'efficacité η d'un codage sera ainsi définie par :

$$\eta = \frac{H(X)}{R}$$

Elle est exprimée en %.

Exemple :

Si $K = 24 \rightarrow log_2(K) = 4{,}585 \rightarrow R = 5 \rightarrow 2^R = 32 \rightarrow$8 codes non utilisés.

Si tous les symboles sont équiprobables :

$$\eta = \frac{H(X)}{R}$$

On calcule $H(X)$:

$$H(X) = Log_2(24) = 4{,}58$$

Et R :

$$R = Int[Log_2(24)] + 1 = Int[4{,}58] + 1 = 5$$

On peut calculer l'efficacité :

$$\eta = \frac{4{,}58}{5} = 0{,}916$$

Alors

$$\eta = 91{,}69\%$$

4.3. Codage par blocs, extension de la source

Pour améliorer l'efficacité du codage, on peut transmettre et donc coder les symboles non pas individuellement mais par bloc de ***j*** symboles, cette technique est appelée l'extension de la source.

Exemple

Avec $\{A, B\}$ (Source primaire) On peut faire avec $j = 2$ les blocs $\{AA, BB, AB, BA\}$

À partir d'une source dite primaire, nous fabriquons une source secondaire.

Source primaire de K symbole, avec une extension on aura une source secondaire $\boldsymbol{K^j}$ symboles.

Si nous utilisons ***N bits*** de codage par blocs :

$$R \geq log_2(k)$$

$$N \geq Log_2\left(k^j\right)$$

$$N \geq jLog_2(k)$$

$$\text{Et } N = Int[jlog_2(k)] + 1$$

Le nombre de bits par symbole de la source primaire est

$$R = \frac{N}{j} = \frac{Int[jLog(k)]}{j} + \frac{1}{j} = \frac{1}{j} Int[jLog(k)] + \frac{1}{j}$$ (Remarque : R n'est plus un entier).

L'efficacité de codage de la source primaire est égale à :

$$\eta_2 = \frac{H(X)}{R}$$

$$\eta_2 = \frac{H(X)}{1/j\ Int[jLog(k)] + 1/j}$$

Résultat à comparer à celui obtenu sans extension de source :

$$\eta_1 = \frac{H(X)}{Int[Log_2(k)] + 1}$$

Et, à cause du terme en 1/J en dénominateur de η_2, nous avons

$$\eta_2 > \eta_1$$

La technique d'extension de source peut ainsi améliorer l'efficacité du codage pour des mots de codes de longueur fixe.

Exemple

$$k = 24; \quad j = 3$$

$$k^j = 24^3 = 13824$$

$$N = Int\left[log_2\left(k^j\right)\right] + 1 = Int[13.75] + 1 = 14; \qquad R = \frac{N}{j} = \frac{14}{3} = 4{,}66$$

Si tous les symboles sont équiprobables :

$$H(X) = log_2(k) = 4.585$$

$$\eta_2 = \frac{4{,}585}{4{,}666} = 98.25\%$$

4.4. Théorie du codage de source « théorie de Shannon »

Il découle de l'extension de source:

Pour une source étendue $N \geq j \times H(X) + 1$

Pour la source primaire :$\frac{N}{j} = R \geq H(X) + \frac{1}{j}$

En posant $\frac{1}{j} = \varepsilon$, ε peut être en théorie rendu aussi petit que l'on veut. D'où le premier théorème de Shannon:

1er théorème de Shannon : Pour avoir un codage sans erreurs, une source **X** doit être codée en moyenne avec au moins $H(X)$ bits.

$$R \geq H(X)$$

4. Codage, mot de longueur variable

Lorsque tous les symboles de l'alphabet ne sont pas équiprobables, l'extension de la source ne permet pas d'augmenter jusqu'à **100%** l'efficacité, le code de Morse résous ce problème ; l'idée étant utilisée un codage (court), pour les symboles plus utilisés (de probabilité élevée) et en réservant un codage plus « long » aux symboles peu utilisés (de probabilité faible).

C'est cette idée qui est reprise et formalisée dans le codage avec mot de longueur variable.

a. Codes préfixes

Pour introduire les notions essentielles nous allons utiliser en exemple trois codages possibles pour une source de 4 symboles. Ces exemples sont dans le tableau ci-dessous:

Caractères	Probabilité	Code 1	Code 2	Code 3
I	$\frac{1}{2}$	1	0	0
B	$\frac{1}{4}$	00	10	01
F	$\frac{1}{8}$	01	110	011
O	$\frac{1}{8}$	10	111	111

Supposons que nous cherchions à transmettre le message BOF

Exemple :

Code 1

BOF 001001

Avec le code I, le message envoyé est : 001001. C'est ce que voit le récepteur. Comment peut–il l'interpréter? De manière correcte bien sûr mais aussi 00 1 00 1 c'est à dire BIBI. Problème, le message n'est pas décodable de manière unique. Ceci est dû au fait que le 1, code attribué à I est le début d'un autre code 10 attribué au O. Pour éviter cette situation, il ne faut pas qu'un code soit le "préfixe" d'un autre code. Les codes qui remplissent cette condition sont des codes préfixes.

Code 3

Avec le code III, le message envoyé est 01111011. Au décodage nous pouvons voir 0 111….c'est à dire IO…Mais ici nous nous rendons compte du fait que ce qui suit c'est à dire soit 1, soit 10, soit 101 ne sont pas des codes et donc, nous pouvons revenir en arrière pour modifier l'interprétation soit 01 111 011 et retrouver le bon message. Le code n'est pas décodable de manière instantanée.

Ceci est aussi dû au fait que le code utilisé n'est pas un code préfixe.

BOF 01111011

Code 2 : Code préfixe

Le code II est lui un code préfixe et nous avons les deux propriétés souhaitées: décodable de manière unique et de manière instantanée.

BOF 10111110 déchiffra d'une manière instantanée et unique

Définitions

Un code préfixe est, par définition, un code dont aucun code n'est le préfixe d'un autre.

Un code préfixe est décodable de manière unique et instantanée.

b. **Longueur moyenne de code**

Pour une source **X** d'alphabet $\{X_k\}$ de probabilité associée $\{P_k\}$ la longueur moyenne du code est définie par :

$$\bar{R} = \sum_{k=1}^{N} n_k P_k$$

où n_k est la longueur du code associé au symbole x_k.

Exemple :

$$I = 1,\ P(I) = \frac{1}{2} \qquad \bar{R} = 1 \times \frac{1}{2}$$

5.1. Codage de Huffman (longueur variable)

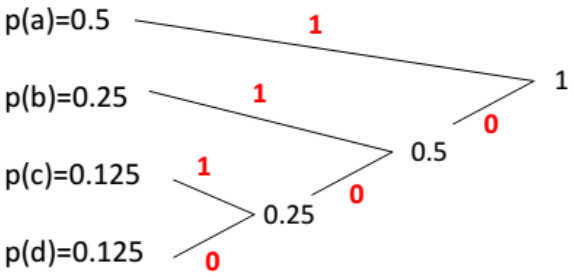
Le codeur de Huffman crée un arbre ordonné à partir de tous les symboles et de leur fréquence d'apparition. Les branches sont construites récursivement en partant des symboles les moins fréquents.

La construction de l'arbre se fait :

1- En ordonnant dans un premier temps les symboles par fréquence d'apparition successivement.

2- Les deux symboles de plus faible fréquence d'apparition sont retirés de la liste et rattachés à un nœud dont le poids vaut la somme des fréquences des deux symboles.

3- Le symbole de plus faible poids (à gauche) est affecté à la branche 0, l'autre à la branche 1 et ainsi de suite en considérant chaque nœud formé comme un nouveau symbole, jusqu'à obtenir un seul nœud parent appelé *racine*.



a	1
b	01
c	001
d	000

H(X)=1.75 bits/symboles, L=1.75 bits/symboles, E=100%, R=0%

5.2. Code de Shannon-Fano

Le code de Shannon-Fano est le premier code à avoir exploité la redondance d'une source. On en expose à présent le principe.

1. Ranger les états du système par probabilités décroissantes.

2. Subdiviser les états du système en 2 groupes G_0 et G_1 de probabilités voisines, sans modifier l'ordre dans lequel ils ont été rangés en 1.

3. Chaque groupe G_i est subdivisé en 2 sous-groupes G_{i0} et G_{i1} de probabilités aussi voisines que possibles, une fois encore sans modifier l'ordre des états.

4. La procédure s'arrête lorsque chaque sous-groupe est constitué d'un unique élément. L'indice du groupe donne le mot de code.

Pour élaborer un code de Shannon-Fano, on procède ainsi :

état	p_i	étape 1	étape 2	étape 3	code
s_1	0.35	0	0		00
s_2	0.22	0	1		01
s_3	0.18	1	0		10
s_4	0.15	1	1	0	110
s_5	0.10	1	1	1	111

5.3. Codage arithmétique

Cette partie va présenter une compression sans perte appelé codage arithmétique. Elle va d'abord définir l'intérêt de l'algorithme avant de présenter la compression et la décompression effectuées par ce codage

5.3.1. Description

Le codage arithmétique est un codage statistique, c'est-à-dire que plus un caractère est représenté, moins il faudra de bits pour le coder.
Il s'agit d'un cousin du codage de Huffman qui cependant reste toujours plus efficace que ce dernier (sauf dans le cas particulier où tous les poids des feuilles/nœuds/racines de l'arbre de Huffman sont des puissances de 2). Il est aussi plus simple à implémenter.
L'avantage que possède le codage arithmétique sur le codage de Huffman est que ce dernier va coder un caractère sur un nombre entier de bits (il ne peut coder sur 1.5 bits) là où le codage arithmétique le peut. Par exemple, si un caractère est représenté à 90%, la taille optimale du code du caractère serait de 0.15 bit, alors que Huffman coderait sûrement ce symbole sur 1 bit, soit 6 fois trop.
Ce codage n'est que très peu utilisé en pratique mais elle reste présente, notamment dans le format JPEG2000.

5.3.2. Compression

Pour présenter la compression, nous allons utiliser un exemple et nous décrirons chaque étape de compression. Codons le mot "ESIPE" à l'aide du codage arithmétique.
La première étape consiste à décompter chaque lettre du mot. Nous avons donc 2 'E', 1 'S', 1 'I' et 1 'P'. Nous en générons alors une probabilité de présence dans le mot soit 40% de chance de trouver un E et 20% de chance pour les autres lettres. Dernière actions à effectuer pour cette première partie, nous affectons à chaque lettre un intervalle entre 0 et 1 de la manière suivante :

- La lettre 'E' à une probabilité de 40% (soit 0.4). Son intervalle est donc [0,0.4[
- La lettre 'P' a une probabilité de 20% (soit 0.2). Son intervalle est donc [0.4,0.6[
- Etc...

On obtient dès lors le tableau suivant :

Lettre	Probabilité	Intervalle

E	4/10	[0,0.4[
S	2/10	[0.4,0.6[
I	2/10	[0.6,0.8[
P	2/10	[0.8,1.0[

Le codage va maintenant consister à remplacer le mot ESIPE par un nombre flottant lui correspondant. Pour cela, le mot va se voir affecter un intervalle compris entre 0 et 1 où chaque nombre compris entre les deux intervalles permettra de retrouver le mot ESIPE.

L'algorithme appliqué est le suivant : le mot commence avec un intervalle de [0,1[. Puis pour chaque lettre croisée, nous appliquons la formule suivante :

- La borne inférieure (BI) du mot est modifiée avec le résultat du calcul "BI + (BS – BI) * Borne_Inférieure_Lettre"
- La borne supérieure (BS) du mot est modifiée avec le résultat du calcul "BI + (BS – BI) * Borne_Supérieure_Lettre"

Le tableau suivant montre les étapes du calcul:

Lettre	Borne Inférieure	Borne Supérieure
	0.0	1.0
E	0.0	0.4
S	0.16	0.24
I	0.208	0.224
P	0.2208	0.224
E	0.2208	0.22208

Dès lors, tous nombre flottant entre 0.2208 et 0.22208 est le format compressé du mot "ESIPE"

5.3.3. Décompression

De la même manière, nous allons présenter la décompression par l'exemple en décompressant notre format compressé.

Prenons le nombre 0.2208 qui code le mot "ESIPE". Le principe de la décompression est très simple. Celle-ci suit les deux étapes suivantes qui se répète jusqu'à l'obtention du mot :

- La prochaine lettre du mot est celle dont l'intervalle contient le nombre du mot actuel (Ex : 0.2208 est dans l'intervalle de E donc la première lettre est E).
- On modifie le nombre représentant le mot à l'aide du calcul « (nombre du mot – borne inférieure de la lettre) / probabilité de la lettre (Ex : nombre du mot = (0.2208 – 0.0) / 0.4 = 0.552)

Le tableau suivant montre les différentes étapes de la décompression :

Mot	Lettre	Nouveau code
	E	0.552
E	S	0,76
ES	I	0,8
ESI	P	0,0
ESIP	E	

Ainsi, nous avons récupéré notre premier mot: ESIPE

5.4. Codage de Lemplel-Ziv

Dans bien des cas pratiques, on ne connaît pas exactement la distribution de la source X. Dès lors, il est impossible de construire un code de Huffman. Sans connaître la distribution P_X, on peut néanmoins connaître son entropie H(X). On s'est alors posé la question de construire des codes de source "universels", susceptibles de coder toutes les sources dont l'entropie ne dépasse pas une borne fixée R en utilisant R bits par valeur à transmettre.

Une autre technique de codage universel a été développée pour les sources sur lesquelles on ne dispose d'aucune information. Cette technique, due à Lempel et Ziv (76), cumule de nombreux avantages :

• les algorithmes de codage et de décodage sont très simples,

• elle ne demande aucune connaissance sur la source,

• elle est asymptotiquement optimale,

• elle fonctionne même avec des sources à mémoire.

Par souci de simplicité, on suppose que X est une source binaire, éventuellement avec mémoire, à coder sur un alphabet binaire. L'algorithme procède en deux passes, sur une suite finie de bits x1,..., x_n. La première passe décompose le train de bits en segments de sorte que chaque nouveau segment n'ait pas été rencontré précédemment.

Sur la suite : 1011010100010...cela donne la segmentation: 1|0|11|01|010|00|10|...

Il vient que chaque segment est composé d'un préfixe, correspondant à un segment déjà rencontré, plus un bit. À la fin de la première passe, on compte alors le nombre de segments, noté c(n). La deuxième passe procède au codage proprement dit. Elle numérote les segments par ordre d'apparition, ce qui demande $\lceil \log_2(c(n)) \rceil$ bits par segment, et représente chaque segment s à l'aide d'une paire (a, b) où a est le numéro du préfixe de s, et b le bit qui a été rajouté (le suffixe). Sur l'exemple précédent, on a 7 segments différents, que l'on va numéroter et coder sur 3 bits. La segmentation devient, sous forme codée:

(000, 1) (000, 0) (001, 1) (010, 1) (100, 0) (010,0) (001, 0)...

La longueur de la séquence codée est donc c(n) $[1+\lceil \log_2 c(n) \rceil]$. Le décodage se fait par le procédé inverse (il faut néanmoins connaître c(n)). Sur l'exemple ci-dessus, la séquence codée est plus longue que la séquence d'entrée du codeur... Mais il faut voir que pour des suites d'entrée longues, les segments que l'on identifie sont eux-mêmes de plus en plus longs, et l'on y gagne beaucoup à les décrire sous la forme compacte (a, b).