2 – Codes de source – Codes de Huffman

Codes de source

On suppose par défaut que l'on code sur l'alphabet binaire $\Sigma = \{0,1\}$

- un codeur de source est le plus souvent une application de l'alphabet de source Ω vers l'ensemble Σ^*
- les mots du code sont les images des symboles de Ω
- le code C est alors l'ensemble des mots du code

- à l'action de coder doit correspondre l'action de décoder
- le fait que l'application soit une injection ne suffit pas à assurer un décodage sans ambiguïté
- il faut pour cela que l'extension de Ω^* à Σ^* soit elle aussi injective
- ce qui se traduit par une bijection entre Ω^* et C^*
- un décodage ambigu correspond donc à un codage avec perte d'informations

Exemple Le codage : $a \rightsquigarrow 1$, $b \rightsquigarrow 01$ et $c \rightsquigarrow 10$ est ambigu contrairement au codage : $a \rightsquigarrow 1$, $b \rightsquigarrow 00$ et $c \rightsquigarrow 10$

Codes à longueur variable

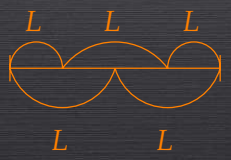
- un code à longueur variable C est un langage caractérisé par le fait que tout mot de C^+ a une factorisation unique en mots de C
- en *Théorie des langages*, on les appelle tout simplement des codes
- de façon équivalente, C est un code ssi :

$$C^{-1}C \cap C^*C^{*^{-1}} = \{\varepsilon\}$$

- en *Théorie des codes*, on les distingue des autres par l'appellation de codes non ambigus
- en *Cryptographie*, on parlera de codes uniquement déchiffrables

Exemple $C = \{0, 01, 110\}$ est un code (non-ambigu)
mais pas $L = \{0, 010, 101\}$: en effet, le mot 0101010 a deux factorisations sur L

Un langage L n'est pas un code si un mot de L^+ admet 2 factorisations distinctes :



Codes préfixes

- Un langage qui ne contient pas 2 mots distincts dont l'un est préfixe de l'autre est clairement un code non ambigu
- en *Théorie des langages*, on les appelle des codes préfixes (plus logiquement, en anglais, prefix-free codes)
- ainsi un code préfixe P vérifie :

$$P^{-1}P = \{\varepsilon\}$$

- en *Théorie des codes*, on parle de codes ayant la propriété du préfixe
- ils sont aussi appelés codes instantanés car le décodage a lieu dès qu'on parvient à lire un mot du code en entier
- ils sont encore appelés codes instantanément déchiffrables ou irréductibles
- à noter : tout code à longueur fixe possède la propriété du préfixe !

Exemple $C = \{0, 01\}$ est un code mais n'est pas un code préfixe

Par contre, le langage $P = \{01, 001, 10\}$ est un code préfixe

Algorithme de Sardinas–Patterson

- Cet algorithme de 1953 permet de décider si un langage (rationnel) donné L est un code non-ambigu
- il consiste en la construction d'une suite inductive d'ensembles :

Initialisation	$X_0 = L^{-1}L \setminus \{\varepsilon\}$	
Etape inductive	$X_{n+1} = ((X_n)^{-1}L) \cup (L^{-1}X_n)$	
Deux cas d'arrêt	• $\varepsilon \in X_n$	$\Rightarrow$ L n'est pas un code
	• $X_{n-1} = X_n$	$\Rightarrow$ L est un code

Exemple

On peut vérifier avec cet algorithme si les langages rationnels suivants sont des codes (non-ambigus) ou pas :

$$K : 0 + 01 + 101$$

$$L : 0 + (01)^*10$$

$$M : 0 + 101 + 100 + 111 + 1101 + 1100$$

Inégalités de Kraft - Mac Millan

On suppose que le codage a lieu sur l'alphabet binaire

Théorème (Inégalité de Kraft)

Il existe un code *instantanné* dont les n mots sont de longueur $l_1, ..., l_n$ ssi :

$$\sum_{i=1}^{n} \frac{1}{2^{l_i}} \leq 1$$

- la même condition nécessaire et suffisante a été établie antérieurement par Mac Millan pour les codes non-ambigus
- on en déduit que tout code non-ambigu possède un code préfixe *équivalent*
- ce résultat n'est pas *constructif* : il ne dit rien sur la manière de trouver un tel code

Exemple Soit le langage $L = \{10, 11, 000, 101, 111, 1100, 1101\}$

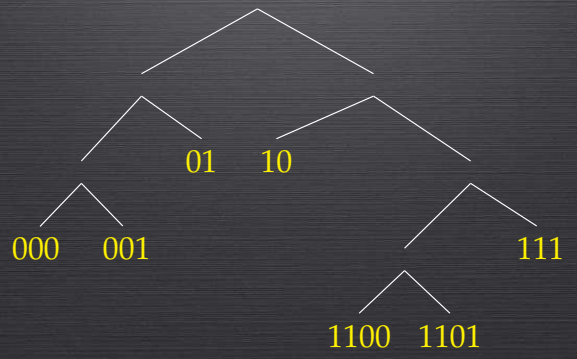
$$\sum_{i=1}^{7} \frac{1}{2^{l_i}} = 2.2^{-2} + 3.2^{-3} + 2.2^{-4} = 1/2 + 3/8 + 1/8 = 1$$

Il existe un code préfixe avec autant de mots et de même longueur que dans L, par exemple :

$$P = \{01, 10, 000, 001, 111, 1100, 1101\}$$

Arbre de codage

- Un arbre de codage aussi dit arbre de Huffman est un arbre binaire complet *i.e.* chaque nœud a 0 ou 2 fils
- les arêtes menant aux fils gauches sont étiquetées par 0 (resp. 1 à droite)
- à chaque feuille correspond un mot du code obtenu en concaténant les étiquettes de la racine à la-dite feuille
- un code est l'ensemble des mots correspondant aux chemins dans l'arbre de la racine à une de ses feuilles



Longueur moyenne

Soit $S = (\Omega, p)$ une source avec $| \Omega | = n$

- chaque symbole de Ω est codé par un mot de C de longueur l_i
- si pour tout i, la longueur l_i est constante, on parle de codes à longueur fixe
- dans le cas contraire, on parle de codes à longueur variable
- la longueur moyenne (pondérée) d'un code est définie par :

$$L = \sum_{i=1}^{n} p_i \, l_i$$

elle est donc sans intérêt dans le cas des codes à longueur fixe ...

- Théorème (conséquence des inégalités de Kraft-Mac Millan)
  La longueur moyenne L des mots d'un code non-ambigu vérifie :

$$H(S) \leq L$$

- lorsque chaque mot du code est d'une longueur égale à celle de son entropie, la longueur moyenne du code est la plus faible possible et égale à l'entropie du message :

$$L = H(S)$$

Théorème de Shannon (th. du codage sans bruit)

On suppose que le codage a lieu sur l'alphabet binaire

- Premier théorème de Shannon
  Soit S une source sans mémoire, d'entropie $H(S)$, il existe un code (non-ambigu) pour S dont la longueur moyenne L des mots de code est aussi voisine que l'on veut de l'entropie.

- toute source discrète sans mémoire admet donc un code non-ambigu vérifiant :

$$H(S) \leq L < H(S) + 1$$

- en théorie, il existe des codes s'approchant indéfiniment de l'entropie mais rien n'est dit sur la façon de les trouver

- l'idée des codes entropiques (qui suivent) est de coder chaque symbole au plus près de son entropie pour approcher la borne inférieure

Méthode directe de codage binaire

Exemple

- On considère une source $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5\}$ à 5 symboles suivant la distribution de probabilités :

$$\begin{array}{llll} p_1 & 0.40 & -log_2\, p_1 = 1,32 & \rightarrow \quad l_1 = 2 \\ p_2 & 0.20 & -log_2\, p_2 = 2,32 & \rightarrow \quad l_2 = 3 \\ p_3 & 0.15 & -log_2\, p_3 = 2,73 & \rightarrow \quad l_3 = 3 \\ p_4 & 0.15 & -log_2\, p_4 = 2,73 & \rightarrow \quad l_4 = 3 \\ p_5 & 0.10 & -log_2\, p_5 = 3,32 & \rightarrow \quad l_5 = 4 \end{array}$$

- à l'aide d'un arbre, il est possible de trouver un code préfixe vérifiant les conditions sur les longueurs
- on obtient par exemple :

$$\omega_1 \rightsquigarrow 00 \quad \omega_2 \rightsquigarrow 010 \quad \omega_3 \rightsquigarrow 011 \quad \omega_4 \rightsquigarrow 100 \quad \omega_5 \rightsquigarrow 1010$$

Codage de Shannon-Fano

- On connaît les symboles de l'alphabet et leurs probabilités d'apparition
  1. on les classe par ordre décroissant de fréquence
  2. à chaque symbole on associe le mot de code temporaire ε
  3. on partitionne l'ensemble ordonné en 2 moitiés à peu près équiprobables
  4. on ajoute 1 à la suite du code des symboles de la $1^{\text{ère}}$ moitié, 0 pour la 2^{de}
  5. on reprend en 2. pour chacun des sous-ensembles jusqu'à épuisement
- s'il est impossible de scinder en 2 ensembles de probabilités comparables, l'optimum n'est pas atteint
- c'est pour cela que ce codage qui date de 1949 a été détrôné par celui des codes de Huffman

Exemple

symbole	prob.						code
a	0.25	1	1				11
b	0.20	1	0				10
c	0.15	0	1	1			011
d	0.15	0	1	0			010
e	0.10	0	0	1			001
f	0.10	0	0	0	1		0001
g	0.05	0	0	0	0		0000

Codes de Huffman

Une source probabilisée $S = (\Omega, p)$ avec $\Omega = \{\omega_1, ..., \omega_n\}$ et $p = \{p_1, ..., p_n\}$

- L'algorithme de Huffman date de 1952 et consiste à contruire inductivement un arbre de codage associant systématiquement les mots de codes les plus courts aux symboles les plus fréquents (*cf.* TP 2) :

  1. on construit à partir d'Ω un ensemble d'arbres-feuilles étiquetés par (ω_i, p_i)
  2. on pré-trie ces arbres par probabilité croissante
  3. on assemble en un seul arbre les 2 arbres de moindres probabilités
  4. on étiquette l'arbre obtenu de la concaténation des symboles et de la somme des probabilités
  5. si il reste plus d'un arbre, on reprend en 2.
  6. on associe à chaque élément d'Ω son mot de code

Exemple

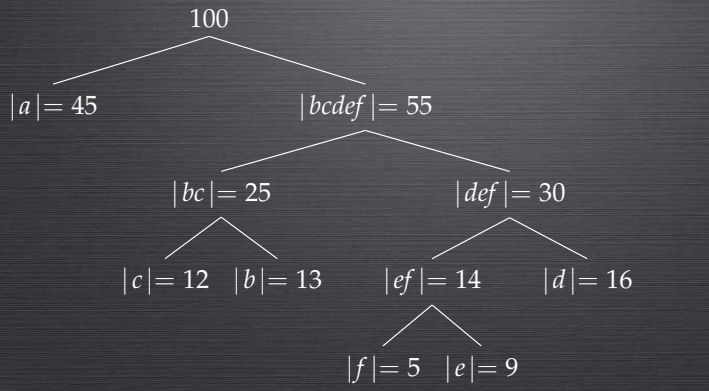
Fréquences d'apparition en % :

a	b	c	d	e	f
45	13	12	16	9	5

Exemple

Fréquences d'apparition en % :

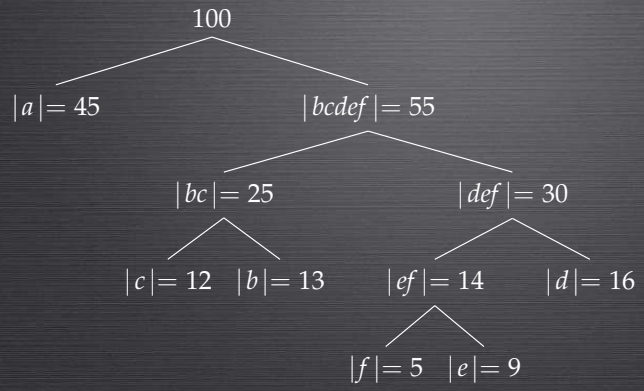
a	b	c	d	e	f
45	13	12	16	9	5



Exemple

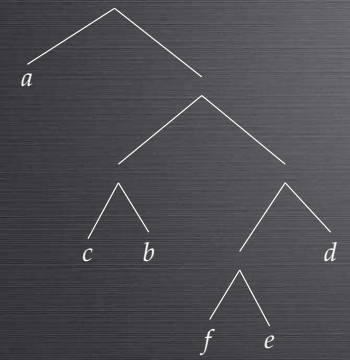
Fréquences d'apparition en % :

a	b	c	d	e	f
45	13	12	16	9	5



a	b	c	d	e	f
0	101	100	111	1101	1100

Exemple : décodage



a	b	c	d	e	f
0	101	100	111	1101	1100

A décoder :

- 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 1 1 0 1

Un 2e exemple

Voici une distribution de probabilités d'une source Ω qu'on veut coder en binaire :

Symbole	Proba. d'apparition
A	0,30
B	0,10
C	0,28
D	0,20
E	0,12

Exercice :

- Quelle est l'entropie de cette source Ω ?
- Quel arbre de Huffman pour cette source Ω ?
- Quelle est la longueur moyenne pondérée des mots du code ?
- Quelle conclusion ?

Un codage optimal ?

- les codes de Huffman ont par construction la propriété du préfixe
- tout code qui possède la propriété du préfixe est même contenu dans un code de Huffman
- ces codes nécessitent une connaissance statistique préalable de la distribution de symboles
- un tel code est optimal car la longueur moyenne L de ses mots est minimale :

$$H(S) \leq L < H(S) + 1$$

- cependant, ce codage s'effectue en bit entier et on peut lui préférer le codage arithmétique (1990)
- les codes de Huffman restent une technique de compression courante couplée à d'autres codages spécifiques à la nature de la source : image, vidéo ou son
- JPEG, MPEG, MP3 et même LZH utilisent les codes de Huffman ou leurs variantes (semi-adaptatives ou adaptatives).

Codage arithmétique

- il s'agit d'un codage entropique
- cette méthode statistique utilise un tableau des fréquences d'apparition des symboles
- elle s'avère meilleure que les codes de Huffman dans la mesure où l'encodage n'a pas lieu en bits entiers
- on encode les caractères par des intervalles
- la sortie de l'encodage est un réel dans $[0, 1]$
- pour éviter les problèmes de portabilité, il y a moyen de travailler sur des entiers
- d'autres optimisations sont possibles pour manier des entiers les plus petits possibles

Ce codage sera présenté en détails au Cours 3 qui traite de compression.

Codage non injectif

Certains codes sont utiles même s'ils ne permettent pas un décodage non-ambigu :

- compression avec perte : son, images
- détection des erreurs : calcul d'empreinte pour vérifier l'intégrité d'un message
- fonction de hachage :

$$h : \{0,1\}^* \to \{0,1\}^n$$

- les transformées de Fourier (DFT)

Exemples

Le bit de parité est un modeste exemple du calcul d'une empreinte :

- si $m = \sigma_1 ... \sigma_n$ alors il vaut $\sum_{i=1}^{n} \sigma_i \mod 2$

Utilisation des fonctions de hachage :

- MDC pour l'intégrité des messages
- MAC pour l'intégrité et l'authentification

A suivre ...