Université Paris Cité

Projet Mathématiques - Informatique

Simulation de variables aléatoires

Florent Bordas
Ryan Timeus

Mai 2023

Dirigé par
Max Fathi

Table des matières

Introduction

Le but de ce projet

L'objectif de ce projet a pour but de simuler une variable aléatoire X de loi P à partir de lancers de pièces équilibrées, naturellement on se pose le problème suivant, de combien de lancers avons-nous besoin en moyenne pour cette simulation ? Cette problématique s'apparente à un problème de compression de données, l'un des deux principaux axes d'étude de la théorie de l'information.

Les propos tenus s'appuient essentiellement sur l'ouvrage [1] proposé par M. Fathi. Par ailleurs, les algorithmes implémentés sont codé en R, un langage spécialisé pour l'étude statistique, il nous servira à comparer nos algorithmes au générateur natif de R.

Qu'est ce que la théorie de l'information ?

La théorie de l'information est née de la préoccupation de transmettre des messages de la façon à la fois la plus économique et la plus fiable. Elle a pour but ultime de répondre à deux questions fondamentales : quelle est la compression maximale et quel est le débit d'information maximal que l'on peut atteindre. L'information possède un caractère essentiellement aléatoire, et cette incertitude sera donc prise comme méthode de mesure de cette dernière. Ainsi, une information est uniquement définie par sa probabilité : $I = -log(P)$. Cependant, l'information possède les caractères fondamentaux de toute réalité physique organisée : abandonnée à elle-même, elle ne peut évoluer que dans le sens de sa désorganisation, c'est-à-dire l'accroissement d'entropie. L'entropie va donc nous permettre de mesurer la quantité d'information moyenne d'un ensemble d'événements. En somme, notre objectif est d'une part de répondre à la problématique en encodant une source dont la distribution de probabilités est connue, de la manière la plus économique et optimale possible et d'autre part d'expliquer les outils utilisés pour encoder une source dont la distribution est, au départ, inconnue et possédant une entropie inférieure à un certain taux de transfert.

Notations

Probabilités

Sauf mention du contraire, X désigne une variable aléatoire (v.a.) *discrète* sur l'espace probabilisé $(\Omega, \mathcal{A}, P)$.

- L'ensemble des valeurs prises par X sera noté par $\mathcal{X} = X(\Omega)$.
- Pour $x \in \mathcal{X}$, on écrira $P(x) = P_X(x) = P(\{X = x\})$.
- On dit que X suit la loi $P = (p_1, \ldots, p_n)$, si $\mathcal{X} = \{x_1, \ldots, x_n\}$ et $\forall i \in [\![1, n]\!]$, $P(x_i) = p_i$.

Théorie de l'information

- Pour tout ensemble fini $\mathcal{D}$, on note $\mathcal{D}^*$ l'ensemble des mots de longueurs finis sur $\mathcal{D}$. De plus le mot vide ϵ est dans $\mathcal{D}^*$.
- Soient $X_1, \ldots, X_n$ une séquence (i.i.d.) de n symboles sur l'alphabet fini $\mathcal{X}$, on notera alors $\mathbf{x} = x_1, \ldots, x_n$, la séquence donnée par $X_1, \ldots, X_n$.

Chapitre 1

Simulation de variables aléatoires

Dans ce chapitre, nous allons nous intéresser à la simulation de variables aléatoires, qui est un élément clé de nombreux domaines tels que la finance, la physique, la statistique, etc.

Après avoir introduit les notions de bases de la compression de données (entropie et codage), nous étudierons le code de *Huffman*, un algorithme de compression efficace permettant de trouver les codes optimaux pour les données. Nous conclurons le chapitre par la résolution de notre problème, plus formellement, étant donné une suite de lancers de pièces indépendants (i.i.d.), extraire (décoder) un échantillon $(X_1, \ldots, X_n)$ où chacun des X_i sont de même loi. Le problème nous oriente donc sur la recherche d'un codage pour une variable aléatoire le plus optimal et le plus rapide possible.

1.1 Introduction à la compression des données

La compression de données est un processus qui a pour but de réduire la quantité de données utilisées pour représenter l'information. L'objectif est de stocker ou de transmettre les données en utilisant le moins d'espace possible, ce qui peut réduire les coûts de stockage et les temps de transfert de données. Nous verrons par la suite que l'entropie d'une source de données est considérée comme une mesure de la quantité d'information contenue dans cette source. Ainsi, la compression de données est étroitement liée à l'entropie, ce qui nous permettra de donner l'existence d'une compression optimale.

1.1.1 Entropie

Définition 1.1 (Entropie)**.** *L'entropie de X noté H(X) est définie par*

$$H(X) = -\sum_{x \in \mathcal{X}} P(x) \log(P(x))$$

Remarque 1.2. La base du logarithme dépend de l'unité voulu, sauf mention du contraire, on utilisera le log en base 2, ainsi $H(X)$ est une mesure en bits. De plus, on prendra comme convention que $0 \log 0 = 0$, qui se justifie par continuité.

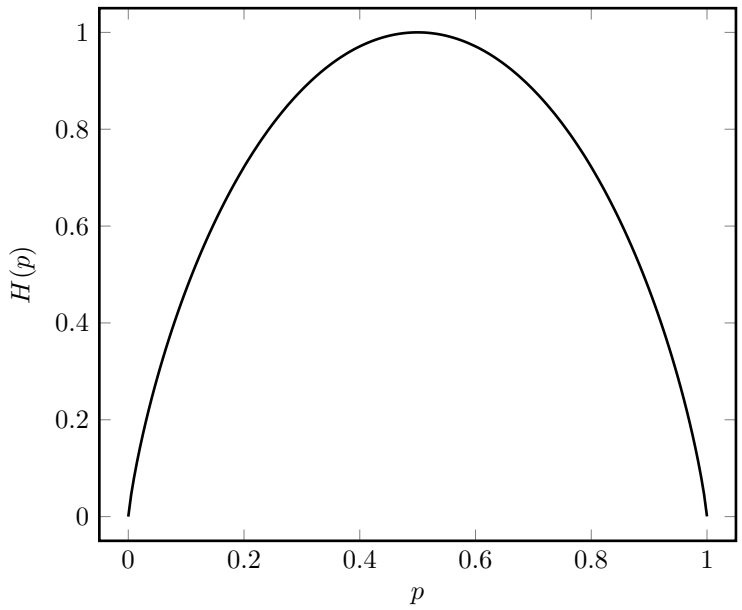
Remarque 1.3. Pour $x \in \mathcal{X}$, on a $P(x) \in [0, 1]$ donc le log est négatif d'où $H(X) \geq 0$. D'autre part, plus l'entropie est élevée, plus la variable est prévisible et contient d'information. En effet, l'entropie est maximale lorsque toutes les possibilités sont équiprobables et à l'inverse lorsque toutes les probabilités sont concentrées en un faible nombre de points, l'entropie est faible.

Exemple 1.4. Un exemple simple pour illustrer ces propos est de regarder l'entropie dans le cas où X est une variable de Bernoulli de paramètres p. Ainsi, l'entropie dépend uniquement de p et vaut

$$H(X) = \begin{cases} -p\log(p) - (1-p)\log(1-p) & \text{si } 0 < p < 1 \\ 0 & \text{si } p = 0 \text{ ou } p = 1 \end{cases}$$

En observant le graphe de l'entropie de X en fonction de p on remarque bien que l'entropie est maximal lorsque p vaut 0.5 et vaut 1 bit dans ce cas.

Entropie de X en fonction de p



Nous aborderons dans le **chapitre 2** des notions connexes à l'entropie comme la divergence de *Kullback-Leibler*, aussi appelé entropie relative, elle nous permettra de mesurer une *« distance »* entre deux distributions de probabilités, en d'autres termes elle mesure la différence entre l'observation et le modèle théorique.

Le concept d'entropie ainsi définie est utilisé en compression de données pour déterminer la quantité minimale de bits nécessaires pour représenter une certaine quantité d'information. En effet, nous cherchons à réduire la taille des données d'une source, qui est une variable aléatoire discrète, en utilisant des méthodes qui exploitent les régularités ou les motifs de celle-ci, afin de réduire le nombre de bits nécessaires pour les stocker ou les transmettre.

1.1.2 Encodage d'une source

Pour compresser une variable aléatoire discrète, nous devons encoder chacune des valeurs qu'elle prend, en une suite de bits finie, dont le but est de contenir les mêmes informations. Ce processus de codage est *sans perte*, il doit nous permettre après de restituer à l'identique les informations originales sur la variable.

Définition 1.5 (Code source)**.** *Soit $\mathcal{D}$ un alphabet fini, un code source C d'une variable aléatoire est une fonction de $\mathcal{X}$ dans $\mathcal{D}^*$. On notera $C(x)$ le codage de $x \in \mathcal{X}$ et $l(x)$ la longueur du mot $C(x)$.*

Définition 1.6. *Soit C un code source pour la variable aléatoire X, la longueur moyenne de C noté $L(C)$ est définit par :*

$$L(C) = \sum_{x \in \mathcal{X}} P(x) l(x)$$

Exemple 1.7. Un codage sans perte connu est le *code Morse*, qui permet de transmettre du texte, où chaque caractère est représenté par une séquence unique de traits et de points. On remarquera que le *code Morse* code les symboles les plus fréquents comme la lettre « e » ou la lettre « i » avec 1 et 2 points respectivement et inversement pour la lettre « j » codé sur un mot de longueur 4.

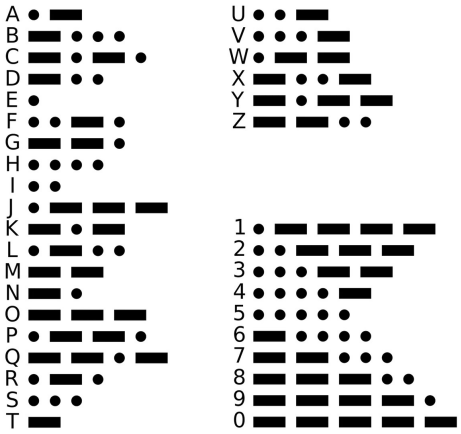


FIGURE 1.1 – Code Morse

Cette idée de codage est voisine de celle du code de *Huffman* qui utilise des codes de longueurs variables en fonction de la fréquence d'apparition des symboles.

Exemple 1.8. Supposons que la variable X soit définie sur $\mathcal{X} = \{1,\ 2,\ 3,\ 4\}$ et $\mathcal{D} = \{0,\ 1\}$, alors un codage pour X serai $C(1) = 0,\ C(2) = 1,\ C(3) = 01$ et $C(4) = 11$.

Cette exemple illustre parfaitement les restrictions que doit posséder un codage. En effet, prenons la suite binaire suivante 01001001100, ce codage peut être décoder par 31313211 ou par 12112112211. Nous avons donc un problème, le message coder peut être décoder de plusieurs manière, le code ainsi défini dans l'exemple n'est pas un code dit *préfixe*, c'est un code qui présente des ambiguïtés.

Définition 1.9. *On dit qu'un code C est préfixe, si $\forall x, y \in \mathcal{X}$ tels que $x \neq y$, aucun des deux mots $C(x)$ et $C(y)$ n'est le préfixe de l'autre.*

Dans la suite du chapitre on désignera par $\mathcal{D}$ l'alphabet binaire (i.e $\mathcal{D} = \{0,\ 1\}$), de plus tout codage pour X sera supposé préfixe sauf mention du contraire.

Exemple 1.10. Reprenons *l'exemple 1.8* et rendons le code choisit en un code préfixe, on peut choisir : $C(1) = 00,\ C(2) = 01,\ C(3) = 10$ et $C(4) = 11$. En pratique, l'avantage des codes préfixe est que pour tout mot $b \in \mathcal{D}^*$, nous avons pas besoin de le parcourir en entier pour en décoder le premier élément de $\mathcal{X}$. En effet, pour décoder le mot b, il suffit de reconnaître un préfixe de celui-ci et de le décoder afin obtenir $b = C(x)b'$ avec $x \in \mathcal{X}$ et $b' \in \mathcal{D}^*$, puis itérer le processus sur le mot b'.

Remarque 1.11. On observe dans *l'exemple 1.8* que si X suit une loi uniforme on a $L(C) = 2$ et $H(X) = 2$, nous avons donc trouvé un code qui a la même longueur moyenne que l'entropie de X. Dans le cas où X suit la loi $P = (0.3,\ 0.1,\ 0.4,\ 0.2)$ on aurait $L(C) = 2 \geq 1.85 = H(X)$. On montrera plus tard que l'entropie est une borne inférieure de la longueur moyenne d'un code source.

Remarque 1.12. Les codes préfixe peuvent se représenter visuellement à l'aide d'arbres. En particulier, pour $x \in \mathcal{X}$, $C(x)$ est une feuille de l'arbre et les branches sont les symboles qui compose le mot $C(x)$.

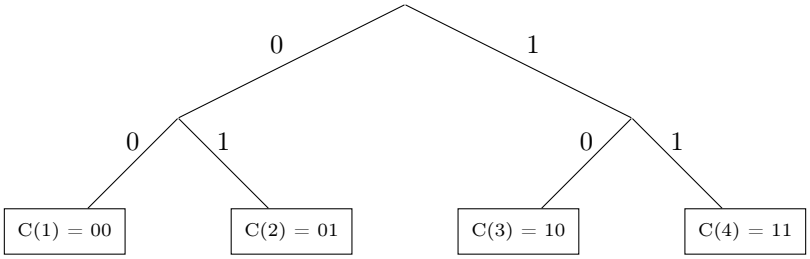


FIGURE 1.2 – Arbre représentant le code de X pour *l'exemple 1.8*

1.1.3 Inégalité de Kraft et code optimal

Proposition 1.13 (Inégalité de Kraft)**.** *Soient X une variable aléatoire définie sur $\mathcal{X}$ et C un code sur $\mathcal{X}$ dans l'alphabet $\mathcal{D}$. Alors si C est un code préfixe on a l'inégalité*

$$\sum_{x \in \mathcal{X}} |\mathcal{D}|^{-l(x)} \leq 1$$

Réciproquement, étant donné une suite $(l_n)_{n\in\mathbb{N}}$ vérifiant cette inégalité, alors il existe un code préfixe C dont les mots sont de longueurs l_i.

Démonstration. On commence par démontrer le premier point, comme C est un code préfixe on utilise sa représentation sous forme d'arbre. Soit $l_{max} = \max_{x\in\mathcal{X}} l(x)$, alors pour $x \in \mathcal{X}$, $C(x)$ est à profondeur $l(x)$ de l'arbre et possède $|\mathcal{D}|^{l_{max}-l(x)}$ descendants à profondeur l_{max}. Notons que ces descendants sont disjoints pour tout x car C a la propriété d'être préfixe, de cette façon en sommant sur les $x \in \mathcal{X}$ et en tenant compte qu'il y ai au plus $|\mathcal{D}|^{l_{max}}$ feuilles on obtient l'inégalité

$$\sum_{x \in \mathcal{X}} |\mathcal{D}|^{l_{max}-l(x)} \leq |\mathcal{D}|^{l_{max}}$$

Puis en divisant par $|\mathcal{D}|^{l_{max}}$ on obtient l'inégalité voulu.

Démontrons l'autre sens, soit donc $(l_n)_{n\in\mathbb{N}^*}$ une suite vérifiant l'inégalité de *Kraft*. Sans perte de généralités on suppose la suite $(l_n)_{n\in\mathbb{N}^*}$ trié par ordre croissant. Reprenons $l_{max} = \max_{i\in[1,n]} l_i = l_n$ ainsi que l'arbre de taille l_{max}, d'après l'inégalité de *Kraft* on peut placer n paquets disjoints de $|\mathcal{D}|^{l_{max}-l(x)}$ feuilles au dernier niveau de l'arbre. Ainsi en identifiant le père de chaque paquet qui est à profondeur l_i, on en extrait le code préfixe associé. □

Proposition 1.14. *Soit C un code préfixe pour X dans l'alphabet $\mathcal{D}$, alors la longueur moyenne de C est plus grande ou égale à l'entropie de X :*

$$L(C) \geq H(X)$$

L'égalité a lieu si, et seulement si, pour $x \in \mathcal{X}$, $|\mathcal{D}|^{-l(x)} = P(x)$.

Démonstration. Supposons que X suit la loi $P = (p_1, \ldots, p_n)$ et $D = \text{Card}(\mathcal{D})$, on s'intéresse au problème de minimisation sous la contrainte de l'inégalité de *Kraft* de la fonction f défini par

$$f(l_1, \ldots, l_n) \ = \ \sum_{i=1}^{n} p_i l_i$$

Pour cela, on utilise la méthode des multiplicateurs de *Lagrange* qui permet de trouver les points critiques d'une fonction dérivable sous contrainte. Soit donc $\lambda \in \mathbb{R}$, on étudie les dérivées partielles du *lagrangien* :

$$L(l_1, \ldots, l_n, \lambda) = f(l_1, \ldots, l_n) + \lambda \left(\sum_{i=1}^{n} D^{-l_i} - 1 \right) \quad \text{et} \quad \frac{\partial L}{\partial l_i}(l_1, \ldots, l_n, \lambda) = p_i - \lambda D^{-l_i} \ln D$$

On cherche alors les points l_i^* et λ tels que $\frac{\partial L}{\partial l_i}(l_1^*, \ldots, l_n^*, \lambda) = 0$ et $\frac{\partial L}{\partial \lambda}(l_1^*, \ldots, l_n^*, \lambda) = 0$. On a :

$$\frac{\partial L}{\partial l_i} = 0 \iff D^{-l_i} = \frac{p_i}{\lambda \ln D} \quad \text{d'où} \quad \frac{\partial L}{\partial \lambda} = 0 \iff \sum_{i=1}^{n} \frac{p_i}{\lambda \ln D} = 1 \iff \lambda = \frac{1}{\ln D}$$

Avec $\lambda = \frac{1}{\ln D}$ on a alors

$$p_i = D^{-l_i} \implies l_i^* = -\log p_i$$

Ainsi, sous la contrainte de l'inégalité de *Kraft* la fonction f atteint son minimum au point l^* de coordonnées $l_i^* = -\log p_i$. Par conséquent, on a

$$f(l_1^*, \ldots, l_n^*) = \sum_{i=1}^{n} p_i l_i^* = -\sum_{i=1}^{n} p_i \log p_i = H(X)$$

□

1.2 Code de Huffman

Dans les sections précédentes, nous avons vu comment représenter une variable aléatoire par des suites de bits. Par ailleurs, nous voulons que cette représentation soit minimale et on sait dorénavant que cette compression est limitée par l'entropie de la variable aléatoire.

Ainsi, pour générer une variable aléatoire X à partir de lancer de pièces, on veut un code C qui vérifie ces conditions :

1. Le code doit être préfixe et optimal dans le sens où sa longueur moyenne doit être au plus proche de l'entropie de X.
2. Pour tout $x \in \mathcal{X}$, $P(x) = 2^{-l(x)}$. En d'autre termes comme c'est un code supposé préfixe, sa représentation sous forme d'arbre doit posséder les propriétés suivantes :
   — Chaque éléments de $\mathcal{X}$ encodé sur une feuille de profondeur d à une probabilité égale à 2^{-d}.
   — L'arbre est complet, c'est-à-dire que chaque noeud est soit une feuille, soit qu'il possède deux descendants.

Pour obtenir un tel code, nous allons utiliser le code de *Huffman* dont l'algorithme de construction du code est récursive sur la taille de $\mathcal{X}$. Cet algorithme nous donnera un arbre qui sera la représentation de notre code préfixe, en particulier sa taille moyenne $L(C)$ est le nombre de moyens de lancers nécessaires pour simuler X.

1.2.1 Algorithme

Le code de *Huffman* est un algorithme de compression de données qui permet de réduire la taille d'un message en utilisant une méthode de codage préfixe. Il a été inventé par David A. Huffman en 1952 alors qu'il était étudiant en doctorat au MIT. L'idée est la suivante, les symboles ayant une haute probabilité d'arriver sont associés à des codes courts et inversement ceux à basse probabilité avec des codes plus longs.

Supposons X prenant ses valeurs dans $\mathcal{X} = \{x_1, \ldots, x_n\}$ de loi $P = (p_1, \ldots, p_n)$. Pour construire un code binaire afin de représenter X on suit les étapes suivantes :

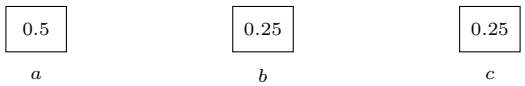
— Sans perte de généralité, on suppose que $p_1 \leq \ldots \leq p_n$, on fusionne les deux plus petites probabilités p_1 et p_2 pour former une nouvelle probabilité $p_{(1,2)} = p_1 + p_2$.

— Les deux noeuds correspondant à p_1 et p_2 forment les deux fils du nouveau noeud qui correspond à $p_{(1,2)}$. On répète l'étape précédente sur le nouvel ensemble de probabilités qui comprend la somme des deux probabilités précédentes : $(p_{(1,2)}, p_3, \ldots, p_n)$, jusqu'à ce qu'il ne reste qu'un seul noeud, l'ensemble de probabilité est donc réduit à $(p_{(1,2,\ldots,n)}) = p_1 + p_2 + \ldots + p_n = 1$.

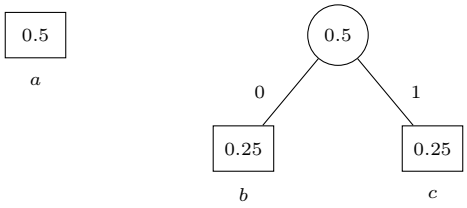
Ainsi, le dernier nœud obtenu $(p_{(1,2,\ldots,n)})$ est la racine de l'arbre voulu. Pour décoder une suite de lancers de pièces, il suffit de parcourir l'arbre et de descendre à gauche ou à droite selon le résultat du lancer, jusqu'à atteindre une feuille qui correspond à un $x_i \in \mathcal{X}$.

Exemple 1.15. Pour illustrer cette algorithme on considère X à valeurs dans $\mathcal{X} = \{a,\ b,\ c\}$ suivant la loi de probabilité suivante : $P(X = a) = \frac{1}{2},\ P(X = b) = \frac{1}{4},\ P(X = c) = \frac{1}{4}$. On détaille les étapes de la construction de l'arbre :

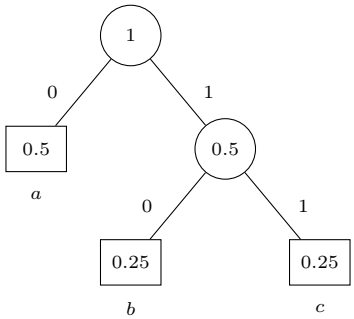
Étape 0 :



Étape 1 :



Étape 2 :



En R, nous allons procéder itérativement sur notre ensemble de nœuds, qui au départ, lorsque $\mathcal{X} = \{x_1, \ldots, x_n\}$ et $P = (p_1, \ldots, p_n)$, représentent tous un couple $(x_i,\ p_i)$. Nous enlevons en boucle les deux nœuds les plus petits par rapport à leur probabilité et on en créer un nouveau comme décrit ci-dessus, la boucle se finit lorsque la taille de l'ensemble des nœuds vaut 1. La correction de l'algorithme se justifie par la finitude de $\mathcal{X}$. Voici notre implémentation en R :

```
.get_huffman_tree <- function(nodes_pq) {
  while (nodes_pq$size() > 1) {
    right <- nodes_pq$pop()
    left <- nodes_pq$pop()

    nodes_pq$push(
      Node$new(NULL, left$freq + right$freq, left, right),
      priority = -log2(left$freq + right$freq)
    )
  }

  return(nodes_pq$pop())
}
```

La variable `nodes_pq` est une *priority queue*, elle nous permet d'ajouter et de retirer des éléments avec une meilleure complexité qu'avec des algorithmes classiques, on utilise la fréquence d'un nœud pour la priorité. L'algorithme de décodage est donc implémenté comme décrit auparavant, en R cela donne :

```
decode <- function(h_tree, encoded_msg) {
  decoded_msg <- NULL

  current <- h_tree
  for (b in encoded_msg) {
    if (b == 0) {
      current <- current$left
    } else {
      current <- current$right
    }

    if (is_leaf(current)) {
      decoded_msg <- c(decoded_msg, current$sym)
      current <- h_tree
    }
  }

  return(decoded_msg)
}
```

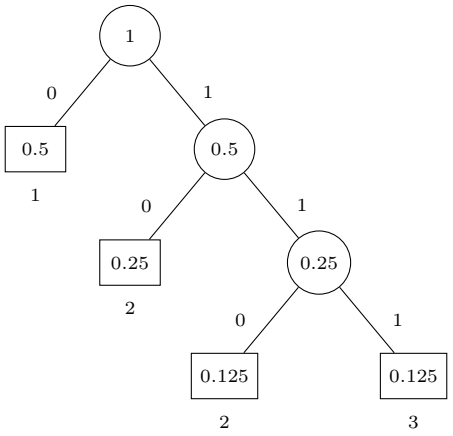
Comme précisé au début de la section, tous les éléments de $\mathcal{X}$ encodés sur une feuille de profondeur d à une probabilité égale à 2^{-d}. Cela nous restreint au codage de variables aléatoires dites dyadique uniquement.

Définition 1.16. *On dit que X est une variable aléatoire dyadique si $\forall x \in \mathcal{X},\ \exists k \in \mathbb{N}$ tel que $x = 2^{-k}$.*

Naturellement, notre but est de résoudre ce problème afin de simuler une variable aléatoire quelconque. Supposons que X ne soit pas dyadique, l'objectif est d'approcher la loi de X en décomposant chacune des probabilités en une somme de fraction dyadique. Soit donc $x \in \mathcal{X}$, alors $\exists N \subseteq \mathbb{N}$ non-vide et de cardinal minimal vérifiant $P(x) = \sum_{k \in N} 2^{-k}$. Avec ces décompositions, on construit l'arbre de la même façon que pour une variable aléatoire dyadique, par exemple si $P(x) = \sum_{k \in N} 2^{-k}$ il y aura $\mathrm{Card}(N)$ feuilles de profondeur k toutes associés au symbole x.

Remarque 1.17. La somme $P(x) = \sum_{k \in N} 2^{-k}$ peut être infinie et donc engendré des arbres de taille infini.

Exemple 1.18. Supposons que X soit à valeur dans $\mathcal{X} = \{1,\ 2,\ 3\}$, de loi $P = (\frac{1}{2},\ \frac{3}{8},\ \frac{1}{8})$. Comme $\frac{3}{8} = \frac{1}{4} + \frac{1}{8}$ on obtient l'arbre suivant :



Pour la simulation en R on va se donner une précision maximale pour nos approximations dyadique, définie par la variable `.max_dyadic_precision`, la variable `.dyadic_atoms` représente donc l'ensemble des fractions dyadique. En R on rajoute une telle décomposition avec la fonction `.add_dyadic_decomposition` :

```
.max_dyadic_precision <- 16
.dyadic_atoms <- 1 / (2 ^ -(1:.max_dyadic_precision))

.add_dyadic_decomposition <- function(pq, freq, symbol) {
  for (atom in .dyadic_atoms) {
    if (freq == atom) {
      pq$push(Node$new(symbol, atom), priority = -log2(atom))
      return()
    }
    if (freq > atom) {
      pq$push(Node$new(symbol, atom), priority = -log2(atom))
      freq <- freq - atom
    }
  }
}
```

1.2.2 Optimalités

Le code de *Huffman* nous assure un code optimal, en effet, dans le cas où X est dyadique, on a $L(C) = H(X)$ qui d'après la proposition 1.14, est le minimum possible en termes de compression. Lorsque X n'est pas dyadique l'optimalité est conservé car on choisit une décomposition optimale de chaque valeurs prises par X.

1.3 Simulation d'une variable aléatoire discrète en R

Maintenant, que nous avons tous les outils pour simuler une variable aléatoire, nous allons comparer en R notre algorithme de simulation avec celui que propose nativement le langage R (via la fonction `sample`) afin d'observer les différents résultats, comparer la précision et la complexité de l'algorithme.

1.3.1 Implémentation

Sans perte de généralité et pour simplifier les affichages, toutes les variables aléatoires simuler seront à valeurs dans $\mathcal{X} = \{1, \ldots, n\}$. Pour générer une variable aléatoire discrète X on utilise les fonctions suivantes :

```
get_nodes <- function(frequencies) {
  nodes_pq <- priority_queue()

  for (i in seq_along(frequencies)) {
    .add_dyadic_decomposition(nodes_pq, frequencies[i], i)
  }

  return(nodes_pq)
}

generate_va <- function(freqs) {
  nodes_pq <- get_nodes(freqs)
  tree <- .get_huffman_tree(nodes_pq)

  return(tree)
}

tests_random_va <- function(freqs) {
  symbols <- seq_along(freqs)
  tree <- generate_va(freqs)

  afb <- tree$average_fair_bits()

  msg_encoded <- rbinom(afb * 10000, 1, 0.5)
  msg_decoded <- decode(huffman_tree, msg_encoded)

  sample_size <- length(msg_decoded)
  sample <- sample(x = symbols, sample_size, replace = TRUE, prob = freqs)
}
```

Premièrement, nous devons créer les objets nécessaires à la création de l'arbre de *Huffman*, c'est la fonction `generate_va` qui à partir d'un tableau de fréquences créer d'abord avec la fonction `get_distribution` l'ensemble des couples (i, p_i) quand `freqs` $= \{p_1, \ldots, p_n\}$ et construit finalement l'arbre de *Huffman*.

La fonction `tests_random_va` utilise donc les objets créer afin de simuler une variation aléatoire. La variable `afb` contient la taille moyenne de l'arbre et donc le nombre de lancers de pièces pour simuler X, on génère ensuite `afb * 50000` lancers de pièces pour enfin en décoder un échantillon de taille ≈ 10000. Enfin la variable `msg_decoded` est l'échantillon de loi donné par `freqs` généré depuis les lancers de pièces.

1.3.2 Résultats

Pour illustrer ces propos et avoir des éléments de comparaison, on utilise la fonction `sample` de R, qui nous génère également un échantillon depuis un ensemble de fréquences. Voici deux exemples qui permettent de visualiser ces simulations :

1. X suit la loi $P = (0.15, 0.4, 0.2, 0.25)$.
2. X suit une loi binomiale de paramètres 10000 et 0.2.

Avec ces configurations, on obtient respectivement les affichages suivants :

Exemple 1

```
freqs <- c(0.15, 0.4, 0.2, 0.25)
tests_random_va_from_freqs(freqs)
```

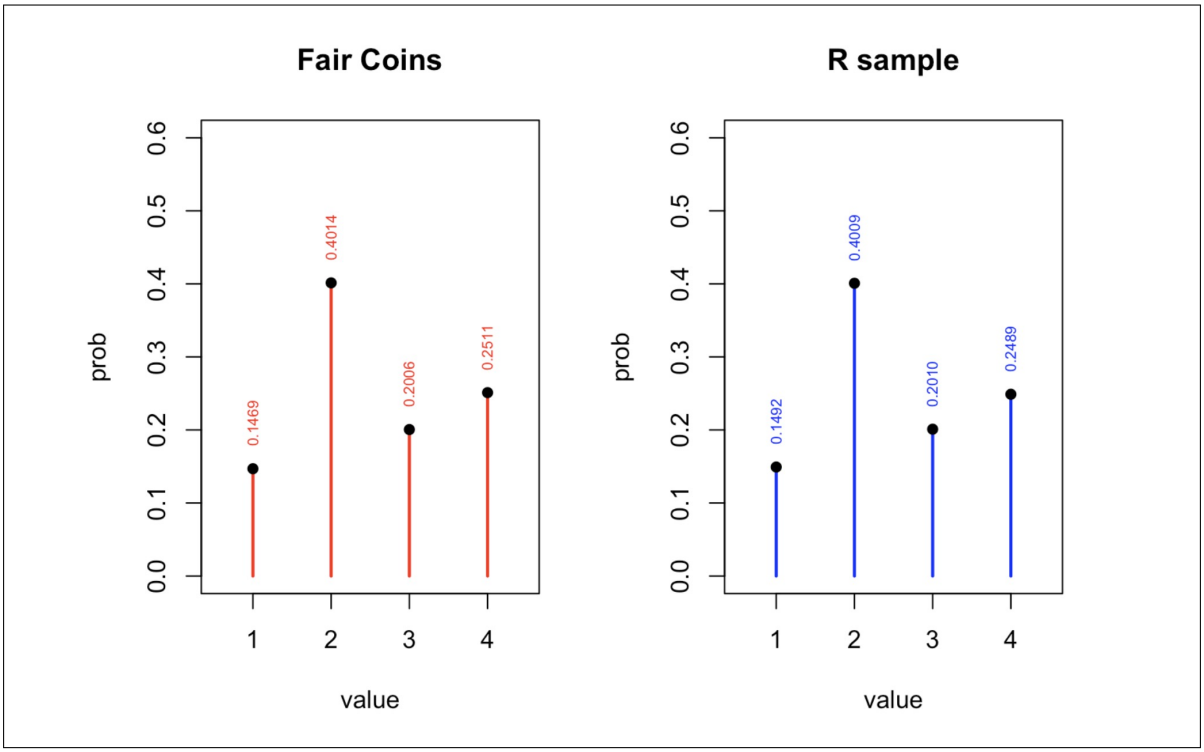


FIGURE 1.3 – Exemple 1

```
##         Entropie =  1.90370
## Longueur moyenne =  2.79971
```

```
## Encodage: 0.003 sec elapsed
## Decodage: 1.945 sec elapsed
##    Total: 1.949 sec elapsed

## R sample: 0.001 sec elapsed
```

Exemple 2

```
n <- 10000
p <- 0.2
freqs <- dbinom(0:n, n, p)
tests_random_va_from_freqs(freqs)
```

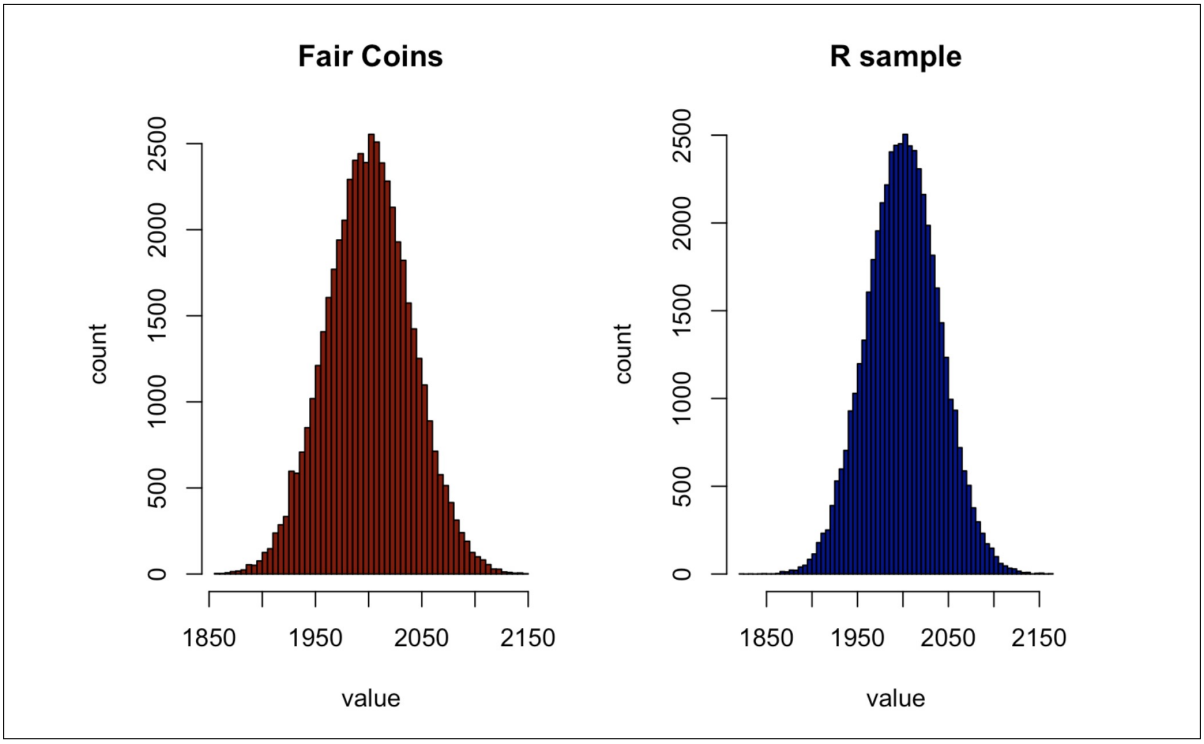


FIGURE 1.4 – Exemple 2

```
##         Entropie =  7.36899
## Longueur moyenne =  8.31427
```

```
## Encodage: 0.057 sec elapsed
## Decodage: 2.200 sec elapsed
##    Total: 2.268 sec elapsed

## R sample: 0.002 sec elapsed
```

Chapitre 2

Théorie de l'information et statistique

Ce chapitre contient les recherches additionnelles à la problématique du projet, elles nous ont été proposées à la suite de la partie principale et dont les résultats ont suscité notre intérêt. Notre but a été donc de s'intéresser à certaines sections du chapitre 11 de l'ouvrage [1] donné. En particulier, beaucoup de résultats théoriques y sont présentés, afin, d'approcher et de comprendre au mieux le sujet choisit.

Dans cette seconde partie, nous nous penchons sur la relation entre les statistiques et la théorie de l'information. La première section définit la divergence de *Kullback-Leibler* autrement appelé entropie relative que nous allons utiliser intensivement. La seconde section aura pour but de présenter la méthode des types qui va nous permettre d'obtenir des résultats utiles pour enfin dans la troisième section prouver l'existence d'un encodage universel pour toute source indépendantes et identiquement distribuées.

2.1 Entropie relative

Définition 2.1. *Soient P et Q à valeurs dans $\mathcal{X}$ deux distributions de probabilités discrètes, l'entropie relative (ou divergence de* Kullback-Leibler*) de P par rapport à Q est définie par :*

$$D(P\,||\,Q) = -\sum_{x\in\mathcal{X}} P(x)\log\left(\frac{Q(x)}{P(x)}\right) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$

Lemme 2.2 (Log sum inequality)**.** *Soient $(a_n)_{n\in\mathbb{N}^*}$ et $(b_n)_{n\in\mathbb{N}^*}$ deux familles de réels positifs.*
Notons $a = \sum_{i=1}^n a_i$ et $b = \sum_{i=1}^n b_i$, alors on a

$$\sum_{i=1}^n a_i \log\left(\frac{a_i}{b_i}\right) \geq a\log\left(\frac{a}{b}\right)$$

Démonstration. On pose $f(x) = x\log(x)$, une fonction convexe, on a :

$$\begin{aligned}\sum_{i=1}^n a_i \log\left(\frac{a_i}{b_i}\right) &= \sum_{i=1}^n b_i f\left(\frac{a_i}{b_i}\right) = b\sum_{i=1}^n \frac{b_i}{b} f\left(\frac{a_i}{b_i}\right)\\ &\text{(Inégalité de Jensen)} \geq bf\left(\sum_{i=1}^n \frac{b_i}{b}\frac{a_i}{b_i}\right) = bf\left(\frac{1}{b}\sum_{i=1}^n a_i\right) = a\log\left(\frac{a}{b}\right)\end{aligned}$$

□

Proposition 2.3 (Exercice)**.** *La fonction à deux variables $D(P\,||\,Q)$ est convexe en P et Q.*

Démonstration. On démontre que $\forall \alpha \in [0,1]$ et pour tous couples de distributions de probabilités (P_1, Q_1) et (P_2, Q_2) à valeurs dans $\mathcal{X}$, on a :

$$D(\alpha P_1 + (1-\alpha)P_2 \,||\, \alpha Q_1 + (1-\alpha)Q_2) \leq \alpha D(P_1 \,||\, Q_1) + (1-\alpha)D(P_2 \,||\, Q_2)$$

En posant $a_1 = \alpha P_1$, $a_2 = (1-\alpha)P_2$, $b_1 = \alpha Q_1$ et $b_2 = (1-\alpha)Q_2$ on a :

$$\begin{aligned} D(\alpha P_1 + (1-\alpha)P_2 \,||\, \alpha Q_1 + (1-\alpha)Q_2) &= D(a_1 + a_2 \,||\, b_1 + b_2) \\ &= \sum_{x \in \mathcal{X}} (a_1 + a_2)(x) \log\left(\frac{(a_1 + a_2)(x)}{(b_1 + b_2)(x)}\right) \\ &= \sum_{x \in \mathcal{X}} (a_1(x) + a_2(x)) \log\left(\frac{a_1(x) + a_2(x)}{b_1(x) + b_2(x)}\right) \end{aligned}$$

On applique le Lemme 2.2 pour $i = 2$ et on a :

$$\sum_{x \in \mathcal{X}} (a_1(x) + a_2(x)) \log\left(\frac{a_1(x) + a_2(x)}{b_1(x) + b_2(x)}\right) \leq \sum_{x \in \mathcal{X}} a_1(x) \log\left(\frac{a_1(x)}{b_1(x)}\right) + \sum_{x \in \mathcal{X}} a_2(x) \log\left(\frac{a_2(x)}{b_2(x)}\right)$$

$$= \sum_{x \in \mathcal{X}} \alpha P_1(x) \log\left(\frac{\alpha P_1(x)}{\alpha Q_1(x)}\right) + \sum_{x \in \mathcal{X}} (1-\alpha) P_2(x) \log\left(\frac{(1-\alpha)P_2(x)}{(1-\alpha)Q_2(x)}\right) = \alpha D(P_1 \,||\, Q_1) + (1-\alpha) D(P_2 \,||\, Q_2)$$

□

2.2 Méthode des Types

La méthode des types consiste à regarder des séquences ayant la même distribution de probabilités. Avec cette restriction on peut facilement calculer et borner le nombre de séquences avec une distribution spécifique et la probabilité de chaque séquence dans ce même ensemble.

Définition 2.4. *Le type $P_{\mathbf{x}}$ d'une séquence $\mathbf{x}$ de caractères issus d'un alphabet $\mathcal{X}$, est le nombre d'occurrences de chaque caractère dans cette séquence. Autrement dit, $\forall a \in \mathcal{X}, P_{\mathbf{x}}(a) = N(a|\mathbf{x})/n$ où n est la taille de la séquence et $N(a|\mathbf{x})$ est le nombre d'occurrences du caractère a dans la séquence.*

Définition 2.5. *L'ensemble des types de taille n que l'on note $\mathcal{P}_n$, dénombre toutes les distributions de caractères possibles pour un alphabet donné.*

Définition 2.6. *La classe d'un type $P \in \mathcal{P}_n$ que l'on note $T(P)$, est l'ensemble des séquences $\mathbf{x}$ de taille n et de type P, soit ayant même distribution de probabilités P. Formulé autrement on regarde toutes les combinaisons de caractères possibles sous la distribution de probabilités P.*

Exemple 2.7. Prenons $\mathcal{X} = \{a, b, c, d\}$ et $\mathbf{x}$ la séquence *acdabca*. Alors le type $P_{\mathbf{x}}$ est :

$$P_{\mathbf{x}}(a) = \frac{3}{7},\ P_{\mathbf{x}}(b) = \frac{1}{7},\ P_{\mathbf{x}}(c) = \frac{2}{7},\ P_{\mathbf{x}}(d) = \frac{1}{7}$$

Dans ce cas la classe du type $P_{\mathbf{x}}$, possède $\frac{7!}{3!1!2!1!} = 420$ éléments :

$$T(P_{\mathbf{x}}) = \{aaabccd, aaaccbd, \ldots, dbccaaa\}$$

Proposition 2.8. *On peut borner le nombre de types de la manière suivante :*

$$|\mathcal{P}_n| \leq (n+1)^{|\mathcal{X}|}$$

On constate donc il n'y a qu'un nombre polynomiale de types. Donc si le nombre de séquences grandit de manière exponentielle en fonction de n, on en déduit que au moins un type a un nombre exponentielle de séquences dans sa classe. On suppose maintenant que les séquences de caractères obtenues sont tirées i.i.d. selon une distribution de probabilités $Q(x)$. De plus pour $\mathbf{x} \in \mathcal{X}^n$, on définit $Q^n(\mathbf{x}) = \prod_{i=1}^n Q(x_i)$.

Proposition 2.9. *On peut maintenant donner un résultat très important, la probabilité d'une séquence* $\mathbf{x}$ *dépend uniquement de son type* $P_{\mathbf{x}}$ *et est noté :*

$$Q^n(\mathbf{x}) = 2^{-n(H(P_{\mathbf{x}})+D(P_{\mathbf{x}} \,||\, Q))}$$

Corollaire 2.10. *Si* $\mathbf{x} \in T(Q)$ *alors :*

$$Q^n(\mathbf{x}) = 2^{-nH(Q)}$$

Ce corollaire nous permet de donner un encadrement pour le cardinal de la classe du type $P \in \mathcal{P}_n$. En fait, on pourrait donner la valeur exacte de $|T(P)|$, c'est juste un problème de combinatoire mais son expression est difficile à manipuler d'où le théorème suivant :

Proposition 2.11. *Soit* $P \in \mathcal{P}_n$*, on a :*

$$\frac{1}{(n+1)^{|\mathcal{X}|}} 2^{nH(P)} \leq |T(P)| \leq 2^{nH(P)}$$

Avec ces résultats, on peut enfin encadrer (pour les mêmes raisons données auparavant) la probabilité d'une classe du type $P \in \mathcal{P}_n$.

Définition 2.12. *Soit* $P \in \mathcal{P}_n$*, pour toute distribution* Q *la probabilité de la classe* $T(P)$ *sous* Q^n *est définit par :*

$$Q^n(T(P)) = \sum_{\mathbf{x} \in T(P)} Q^n(\mathbf{x})$$

Proposition 2.13. *La probabilité de la classe* $T(P)$ *sous* Q^n *peut être encadré de la sorte :*

$$\frac{1}{(n+1)^{|\mathcal{X}|}} 2^{-nD(P \,||\, Q)} \leq Q^n(T(P)) \leq 2^{-nD(P \,||\, Q)}$$

Ainsi, tous les résultats obtenus (en particulier le dernier) vont nous être utile lorsque l'on étudie le comportement de longues séquences. Par exemple, pour les longues séquences tirées i.i.d. selon une certaine distribution, le type de la séquence est proche de la distribution générant la séquence, et nous pouvons utiliser les propriétés de cette distribution pour estimer les propriétés de la séquence.

En effet, on peut établir un autre énoncé pour la loi des grands nombres. On rappelle que par définition du type $P_{\mathbf{x}}$, on a lorsque $\mathbf{x}$ est tirée selon la loi Q :

$$P_{\mathbf{x}}(a) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{x_i = a\}}$$

La loi (faible) des grands nombres nous affirme que $P_{\mathbf{x}}(a)$ tend en probabilité vers $\mathbb{E}[\mathbb{1}_{\{X_1 = a\}}] = Q(a)$. Alors, une autre façon d'obtenir ce résultat est de considérer pour $\epsilon > 0$ l'ensemble :

$$T_Q^\epsilon = \{\mathbf{x} \,|\, D(P_\mathbf{x} \,||\, Q) < \epsilon\}$$

On peut alors regarder la probabilité du complémentaire de T_Q^ϵ sous Q^n :

$$\begin{aligned}
1 - Q^n(T_P^\epsilon) &= \sum_{\{P \,|\, D(P\,||\,Q) > \epsilon\}} Q^n(T(P)) \\
&\leq \sum_{\{P \,|\, D(P\,||\,Q) > \epsilon\}} 2^{-nD(P\,||\,Q)} \qquad \text{Proposition 2.13} \\
&\leq \sum_{\{P \,|\, D(P\,||\,Q) > \epsilon\}} 2^{-n\epsilon} \\
&\leq (n+1)^{|\mathcal{X}|} 2^{-n\epsilon} \qquad \text{Proposition 2.8} \\
&= 2^{-n\left(\epsilon - |\mathcal{X}| \frac{\log(n+1)}{n}\right)},
\end{aligned}$$

qui quand $n \longrightarrow \infty$, tend vers 0. En découle le résultat suivant :

$$Pr(\{D(P_\mathbf{x} \,||\, Q) > \epsilon\}) \leq 2^{-n\left(\epsilon - |\mathcal{X}| \frac{\log(n+1)}{n}\right)}$$

Et par conséquent, $D(P_\mathbf{x} \,||\, Q)$ converge en probabilité vers 0, on retrouve bien le même résultat. Un autre domaine d'application de la méthode des types est le *Codage Universel d'une Source.*

2.3 Codage Universel d'une Source

Cette section a pour but de répondre à la problématique que l'on a rencontré avec le code de Huffman, il est sensible à la distribution de départ. En effet, si le code est utilisée pour une distribution incorrect on insère une pénalité de $D(P \,||\, Q)$. Grâce aux résultats obtenus sur la méthode des types, nous allons pouvoir décrire un schéma permettant d'affirmer qu'il existe un codage universel de taux R et qui peut décrire toutes les sources i.i.d. avec une entropie tel que $H(X) < R$.

Cependant, les affirmations ci-dessus lèvent une problématique tout à fait naturelle, qu'il est impossible à partir d'un seul code de respecter la condition que l'entropie soit inférieur au taux de transmission. On rappelle qu'il y a un nombre polynomiale de types et un nombre exponentielle de séquences de chaque type. La probabilité d'une classe d'un type dépend uniquement de l'entropie relative entre une distribution P et notre distribution réelle Q. Ainsi, les classes dont la distance à notre distribution réelle est trop grande ne sont pas cohérentes à prendre en compte car leur probabilité d'apparition croît exponentiellement vers 0 avec une probabilité certaine.

Définition 2.14. *On définit donc notre ensemble typique fort*

$$A_\epsilon^{*(n)} = \left\{ \mathbf{x} \in \mathcal{X}^n \,:\, \begin{array}{ll} \left|\frac{1}{n} N(a|\mathbf{x}) - P(a)\right| < \frac{\epsilon}{|\mathcal{X}|} & si\ P(a) > 0. \\ N(a|\mathbf{x}) = 0 & si\ P(a) = 0. \end{array} \right\}$$

On peut voir que les séquences ont une probabilité qui diffère très peu de notre distribution réelle et notre ensemble a une typicalité forte.

Définition 2.15. *Un code à bloc au taux fixe R pour une source i.i.d. selon une distribution inconnue Q consiste en deux fonctions l'encodeur et le décodeur qu'on nomme f_n et ϕ_n. La probabilité d'erreur de notre code par rapport à la distribution Q inconnue se définit de la manière suivante :*

$$P_e^{(n)} = Q^n(X^n : \phi^n(f_n(X^n)) \neq X^n)$$

Pour que notre code soit universel, il faudra d'une par que notre décodeur et notre encodeur ne dépendent pas de la distribution Q et d'autre part que l'erreur $P_e^{(n)} \underset{n\to\infty}{\longrightarrow} 0$ pour n'importe quelle source où $R > H(Q)$

Théorème 2.16. *Il existe une séquence de $(2^nR, n)$ codes universels tel que $P_e^{(n)} \underset{n\to\infty}{\longrightarrow} 0$ pour chaque source Q où $H(Q) < R$*

On pose d'abord,

$$R_n = R - |\mathcal{X}|\frac{log(n+1)}{n}$$

et on considère un ensemble $A = [\mathbf{x} \in \mathcal{X}^n : H(P_\mathbf{x}) \leq R_n]$ qui s'apparente à un dictionnaire. Pour donner un peu de contexte pour la démonstration qui va suivre, supposons que l'on veuille encoder notre dictionnaire. Dans un cas optimal, on voudrait que la fonction d'encodage donne en sortie, l'indice de la séquence dans A si notre séquence appartient à A et 0 sinon. La fonction de décodage renverrait directement chaque indice sur le bon élément dans A et les séquences restantes seraient des erreurs. Ainsi, tout les éléments seraient correctement récupérés. On sait que $|A| = 2^{nR}$, il nous faut maintenant démontrer que ce schéma existe.

Démonstration. Supposons que la distribution de notre échantillon $(X_1, X_2, \ldots, X_n)$ est $H(Q) < R$, alors on a :

$$\begin{aligned}
P_e^{(n)} &= 1 - Q^n(A) \\
&= \sum_{\{P|H(P)>R_n\}} Q^n(T(P)) \\
&= (n+1)^{|\mathcal{X}|} \max_{\{P|H(P)>R_n\}} Q^n(T(P)) \\
&= (n+1)^{|\mathcal{X}|} 2^{-n \min_{\{P|H(P)>R_n\}} D(P\,||\,Q)}
\end{aligned}$$

Comme $R_n \underset{n\to\infty}{\longrightarrow} R$ et $H(Q) < R$, il existe un rang n_0 à partir duquel $\forall n \geq n_0, R_n > H(Q)$: $min_{\{P|H(P)>R_n\}} D(P\,||\,Q) \geq 0$ et la probabilité d'erreur converge vers 0 de manière exponentielle quand n est très grand. □

Comme annoncé dans l'introduction de notre rapport, nous travaillons uniquement sur la problématique de transfert de sources i.i.d. Le schéma d'encodage universel que nous venons de décrire est donc universel seulement dans le cas des sources i.i.d. Par ailleurs, il existe d'autres schémas pour résoudre la problématique des sources ergodiques comme l'algorithme de *Lempel-Ziv*. Cependant, il reste une question en suspens : Pourquoi utiliser l'algorithme de Huffman quand on possède un schéma universel ? L'encodage universel nécessite en réalité une plus grande longueur de bloc pour fonctionner de manière optimal, ce qui se répercute en une plus grande complexité des fonctions d'encodage et de décodage.

Dans la section précédente on a montré que des types incohérents ont une probabilité d'apparition très faible et que ce résultat permet l'existence d'un schéma universel. Nous allons maintenant estimer la

probabilité d'un ensemble de types qui ne seraient à priori pas typiques par rapport à notre distribution de départ. Pour cela nous allons utiliser le concept de grande déviation.

Supposons E, le sous-ensemble de fonctions de probabilités avec espérance μ. On pose :

$$Q^n(E) = Q^n(E \cap \mathcal{P}_n) = \sum_{\{\mathbf{x} | P_{\mathbf{x}} \in E \cap \mathcal{P}_n\}} Q^n(\mathbf{x})$$

Comme pour la section précédente, si E possède une entropie relative proche de celle de Q, alors par la loi faible des grands nombres, E à une probabilité d'apparition certaine, et une probabilité nulle si l'entropie relative est trop grande.

On peut illustrer la théorie de grande déviation en prenant l'exemple d'une suite de variables aléatoires i.i.d. de Ber($\frac{1}{3}$). La probabilité que la moyenne empirique soit proche de $\frac{1}{3}$ est une petite déviation du résultat attendu, et la probabilité que notre moyenne empirique soit supérieur ou égale à $\frac{3}{4}$ C'est ce qu'on appelle une grande déviation et la probabilité de cette dernière est exponentiellement faible.

On suppose par la suite que par observation, la moyenne g(X) de notre échantillon est supérieur ou égal à une constante α : $\frac{1}{n}\sum_i g(x_i) \geq \alpha$ c'est équivalent à l'évènement $P_{\mathbf{x}} \in E \cap \mathcal{P}_n$ où $E = \{P \mid \sum_{a \in \mathcal{X}} g(a)P(a) \geq \alpha\}$

Théorème 2.17. *On peut maintenant borner la probabilité de notre sous-ensemble E grâce au théorème de* Sanov. *Soit $(X_1, X_2, ..., X_n)$ une source i.i.d. selon une distribution $Q(x)$. On pose $E \subseteq \mathcal{P}$ notre sous-ensemble de types, alors :*

$$Q^n(E) = Q^n(E \cap \mathcal{P}_n) \leq (n+1)^{|\mathcal{X}|} 2^{-nD(P^* \,||\, Q)}$$

où

$$P^* = \arg\min_{P \in E} D(P \,||\, Q)$$

soit le type P qui a l'entropie relative la plus proche de Q. Si de plus l'ensemble E est la fermeture de son intérieur, alors :

$$\frac{1}{n} log Q^n(E) \underset{n \to \infty}{\longrightarrow} -D(P^* \,||\, Q)$$

Démonstration. D'abord on démontre la borne supérieure :

$$\begin{aligned}
Q^n(E) &= \sum_{P \in E \cap \mathcal{P}_n} Q^n(T(P)) \\
&\leq \sum_{P \in E \cap \mathcal{P}_n} 2^{-nD(P \,||\, Q)} \\
&\leq \sum_{P \in E \cap \mathcal{P}_n} \max_{P \in E \cap \mathcal{P}_n} 2^{-nD(P \,||\, Q)} = \sum_{P \in E \cap \mathcal{P}_n} 2^{-n \min_{P \in E \cap \mathcal{P}_n} D(P \,||\, Q)} \\
&\leq \sum_{P \in E \cap \mathcal{P}_n} 2^{-n \min_{P \in E} D(P \,||\, Q)} = \sum_{P \in E \cap \mathcal{P}_n} 2^{-nD(P^* \,||\, Q)} \\
&\leq (n+1)^{|\mathcal{X}|} 2^{-nD(P^* \,||\, Q)}
\end{aligned}$$

La borne inférieure nécessite un ensemble E qui nous convient, c'est-à-dire que pour n très grand, on peut trouver une distribution dans $E \cap \mathcal{P}_n$ qui est proche de P^*. Si on suppose que E est la fermeture de son intérieur, en conséquence, l'intérieur est non vide puis $\cup_n \mathcal{P}_n$ est dense dans l'ensemble des distributions de probabilités. On obtient que $E \cap \mathcal{P}_n$ est non vide à partir d'un certain rang n_0. On peut donc prendre une distribution de probabilités P_n tel que $P_n \in E \cap \mathcal{P}_n$ et $D(P_n \,||\, Q)) \underset{n \to \infty}{\longrightarrow} D(P^* \,||\, Q)$. On obtient

donc :

$$\forall n \geq n_0 : Q^n(E) = \sum_{P \in E \cap \mathcal{P}_n} Q^n(T(P)) \geq Q^n(T(P)) \geq \frac{1}{(n+1)^{|\mathcal{X}|}} 2^{-nD(P\,||\,Q)}$$

□

Le théorème de *Sanov* est donc utilisé pour donner une borne sur la probabilité d'observer une séquence atypique dans notre échantillon. Il permet d'identifier la fonction de taux pour les grandes déviations de la moyenne empirique de notre source i.i.d. En montrant que la probabilité d'un ensemble de types par rapport à une distribution Q est essentiellement déterminé par l'élément dans l'ensemble qui est le plus proche de Q, on peut renforcer notre argument. En effet, la probabilité de notre ensemble est similaire à la probabilité du type P^* et la somme des probabilités des types qui sont éloignés de P^* sont négligeables. Ce qui implique que le type observé dans l'expérience sera P^* avec une grande probabilité, c'est ce qu'on appelle un théorème de limite conditionnelle.

Bibliographie

[1] T. Cover and J. Thomas. *Elements of Information Theory.* John Wiley and Sons, second edition, 2006.