Semaine 10 : Série d'exercices sur la compression de données [Solutions]

1 Algorithme de compression

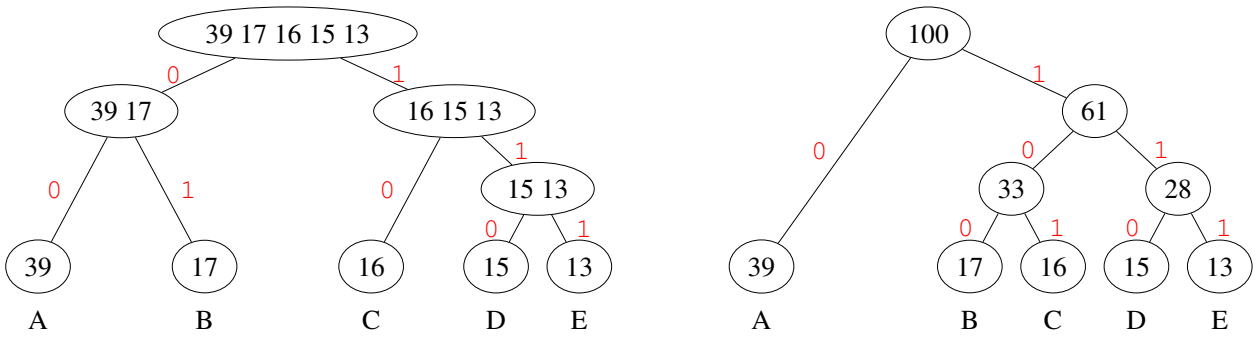


FIGURE 1 – à gauche : code de Shannon-Fano ; à droite : code de Huffman

L'arbre binaire pour les deux différents codes apparaît sur la figure ci-dessus. Les mots de codes obtenus pour un code de Shannon-Fano sont par exemple :

$$A \to 00, B \to 01, C \to 10, D \to 110, E \to 111$$

Donc le nombre moyen de bits utilisé par le code de Shannon-Fano est :

$$\frac{(39+17+16)\times 2 + (15+13)\times 3}{100} = \frac{228}{100} = 2.28 \text{ bits/lettre}$$

Les mots de codes obtenus pour un code de Hufman sont par exemple :

$$A \to 0, B \to 100, C \to 101, D \to 110, E \to 111$$

Donc le nombre moyen de bits utilisé par le code de Huffman est :

$$\frac{39\times 1 + (17+16+15+13)\times 3}{100} = \frac{222}{100} = 2.22 \text{ bits/lettre}$$

La limite théorique donnée par le théorème de Shannon est l'entropie :

$$\begin{aligned} \text{Entropie} &= -0.39\log_2(0.39) - 0.17\log_2(0.17) - 0.16\log_2(0.16) - 0.15\log_2(0.15) - 0.13\log_2(0.13) \\ &\approx 2.18 \text{ bits/lettre} \end{aligned}$$

A noter que dans ce cas, la borne inférieure ne peut pas être atteinte. Du fait que les codes de Huffman sont optimaux (pour la compression sans pertes), on ne peut donc pas trouver dans ce cas un schéma de compression qui fasse mieux que 2.22 bits/lettre. Le théorème de Shannon est vrai pour *tous* les codes et *toutes* les sources. Dans certains cas, la borne inférieure que constitue l'entropie ne peut être atteinte (de fait, le théorème ne dit pas qu'elle doit l'être).

2 Un peu de magie noire

a) Calcul de l'entropie de la séquence (que l'on va appeler X comme dans le cours) : le A a une probabilité d'apparition de $\frac{5}{12}$, le V et le D de $\frac{2}{12}$ et les 3 lettres restantes de $\frac{1}{12}$. Donc

$$\begin{aligned} H(X) &= \frac{5}{12}\log_2\left(\frac{12}{5}\right) + 2\,\frac{2}{12}\log_2\left(\frac{12}{2}\right) + 3\,\frac{1}{12}\log_2(12) \\ &= \log_2(12) - \frac{5}{12}\log_2(5) - \frac{1}{3}\log_2(2) = \frac{5}{3} + \log_2(3) - \frac{5}{12}\log_2(5) \end{aligned}$$

ce qui donne numériquement : $H(X) \simeq 2.28$ bit.

b) Avec l'algorithme de Shannon-Fano, on peut trouver les mots de codes suivants, p.ex. :

		version 1		version 2		version 3	
lettre	nb app.	nb quest.	mot de code	nb quest.	mot de code	nb quest.	mot de code
A	5	2	11	2	11	1	1
V	2	2	10	2	10	3	011
D	2	3	011	2	01	3	010
K	1	3	010	3	001	3	001
E	1	3	001	4	0001	4	0001
R	1	3	000	4	0000	4	0000

Avec les codes 1) et 2), la séquence codée contient 29 bits ; avec le code 3), la séquence codée contient 28 bits.

c) Avec l'algorithme d'Huffman, on trouve par exemple le code suivant :

lettre	nb apparitions	mot de code
A	5	1
V	2	011
D	2	010
K	1	001
E	1	0001
R	1	0000

et la séquence codée contient 28 bits. On peut aussi trouver des codes différents suivant l'arbre que l'on construit, mais la longueur de la séquence codée reste invariablement de 28 bits dans ce cas.

d) Avec l'algorithme de Shannon-Fano, la longueur moyenne du code $L(C_{SF}) \simeq 2.42$ ou 2.33. Avec l'algorithme de Huffman, la longueur moyenne du code $L(C_H) \simeq 2.33$, et l'entropie de la séquence vaut $H \simeq 2.28$. On vérifie donc bien les inégalités du cours :

$$L(C_{SF}) \geq L(C_H) \geq H(X)$$

3 Encore plus de codes

a) En utilisant l'algorithme de Shannon-Fano, on obtient par exemple :

lettre	nb apparitions	nb questions	mot de code
I	4	3	111
N	4	3	110
O	4	3	101
C	4	3	100
M	3	3	011
A	3	4	0101
T	3	4	0100
L	2	4	0011
U	2	4	0010
F	1	4	0001
R	1	5	00001
E	1	5	00000

b) Au total, on a donc besoin de $19 \times 3 + 11 \times 4 + 2 \times 5 = 111$ bits, c.-à-d. $L(C) = \frac{111}{32} = 3.468975$ bits par lettre.

c) En utilisant l'algorithme de Huffman, on obtient par exemple :

lettre	nb apparitions	mot de code
I	4	111
N	4	001
O	4	000
C	4	101
M	3	011
A	3	010
T	3	1101
L	2	1001
U	2	1000
F	1	11001
R	1	110001
E	1	110000

Sa longueur moyenne est de

$$\frac{1}{32} \times ((4+4+4+4+3+3) \times 3 + (3+2+2) \times 4 + 1 \times 5 + (1+1) \times 6) = \frac{111}{32}$$

On retrouve ici un cas *particulier* où un code de Shannon-Fano obtient les mêmes performances que les codes de Huffman.

d) L'entropie de la séquence est égale à $H(X) = 3.42923$ bits. On vérifie donc bien que $H(X) \leq L(C) \leq H(X)+1$, on on voit en fait que $L(C)$ est beaucoup plus proche de $H(X)$ que de $H(X)+1$.

e) La séquence contient 12 lettres différentes : si on veut utiliser le même nombre de bits par lettre, on a donc besoin de 4 bits par lettre au minimum (car $2^4 = 16$, qui est la puissance de 2 la plus

proche au-dessus de 12), et donc de $32 \times 4 = 128$ bits au total. On utilise donc 17 bits de plus qu'avec un code de Shannon-Fano ou un code de Huffman.

f) La séquence « DIDON etc. » a un très grand nombre de D (11 pour être plus précis) : on s'attend donc à une plus faible entropie.

g) En utilisant l'algorithme de Shannon-Fano, on obtient par exemple :

lettre	nb apparitions	nb questions	mot de code
D	11	2	11
N	6	2	10
O	5	3	011
I	4	3	010
U	3	3	001
A	1	4	0001
T	1	5	00001
S	1	5	00000

Le nombre total de bits utilisés est cette fois-ci de $17 \times 2 + 12 \times 3 + 1 \times 4 + 2 \times 5 = 84$ bits, c.-à-d. $L(C) = \frac{84}{32} = 2.625$ bits par lettre. Quand à l'entropie de la séquence, elle est égale à $H(X) = 2.56475$. A nouveau, on vérifie bien que $H(X) \leq L(C) \leq H(X) + 1$, et que $L(C)$ est plus proche de $H(X)$ que de $H(X) + 1$, même si les probabilités des lettres sont assez irrégulières dans le cas présent.

En utilisant l'algorithme de Huffman, on obtient par exemple :

lettre	nb apparitions	mot de code
D	11	11
N	6	01
O	5	101
I	4	100
U	3	001
A	1	0001
T	1	00001
S	1	00000

On retrouve encore ici un cas *particulier* où un code de Shannon-Fano obtient les mêmes performances que les codes de Huffman. [1]

La séquence contient 8 lettres différentes : si on veut utiliser le même nombre de bits par lettre, on a donc besoin de 3 bits par lettre au minimum (car $2^3 = 8$) et donc de $32 \times 3 = 96$ bits au total. On utilise donc 12 bits de plus qu'avec un code de Shannon-Fano ou de Huffman.

1. Ce n'est pas toujours le cas, comme le montre l'exercice 1, mais pas rare non plus, les codes de Shannon-Fano n'étant pas si mauvais que ça (sans être optimaux).

4 Enquête policière et littérature « scientifique »

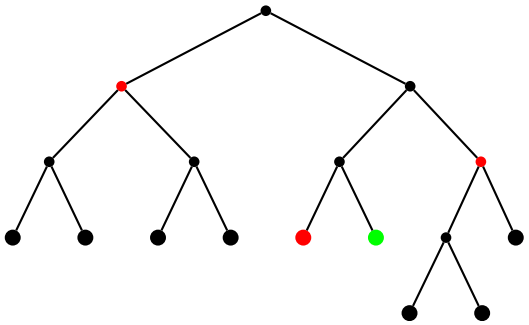
1. Ce que l'auteur du problème suggère, c'est d'effectuer des tests en mélangeant plusieurs échantillons prélevés dans les verres. En d'autres termes, de construire un arbre binaire de décision (empoisonné/pas empoisonné) dont chaque niveau représente un test [2].

   Et l'auteur pense que la solution optimale consiste à procéder par dichotomie. Pour cela l'idéal serait d'avoir un nombre de verres égal à une puissance de deux. Le fait que la solution optimale du mathématicien soit de commencer en testant d'abord (une fois) n'importe quel verre suggère que le nombre de verres est une puissance de deux plus 1. Le seul nombre compris entre 100 et 200 qui remplisse cette condition est $129 = 2^7 + 1$.

   Il y avait donc 129 verres.
2. Cependant nous savons que l'arbre de décision optimal est l'arbre correspondant à un codage de Huffman de la situation.

   Dans le cas où le nombre de feuilles (c'est-à-dire le nombre de cas à tester – des verres, en l'occurence) est une puissance de deux, cet arbre optimal est effectivement l'arbre équilibré auquel pense l'auteur. Cependant, dans le cas où le nombre de feuilles n'est justement pas une puissance de deux mais, par exemple, une puissance de deux plus un, l'arbre n'est **pas** l'arbre équilibré avec une branche supplémentaire à son sommet. C'est précisément l'idée clé de Huffman et du codage (optimal) que de procéder à la fin et non pas au début à ce test, peut-être inutile !

   Représentons l'arbre de décision optimal (code de Huffman binaire de « lettres » avec la même probabilité) dans le cas où l'on a $2^3 + 1 = 9$ verres [3] :



   avec en rouge une séquence de tests possible si le poison est par exemple dans le verre représenté par la feuille verte.

   Quel est le nombre moyen de tests ? Soit $N = 2^n + 1$ le nombre de verres. La plupart du temps, $n = \lfloor \log_2 N \rfloor$ tests détermineront la situation, sauf quand le verre empoisonné est dans le dernier groupe de deux verres (pas de chance !). Dans ce cas, il faut un test de plus. Mais ceci n'arrive en moyenne que dans 2 cas sur N. Le nombre moyen de tests est donc :

$$\overline{T} = \frac{N-2}{N}\lfloor \log_2 N \rfloor + \frac{2}{N}(\lfloor \log_2 N \rfloor + 1) = \lfloor \log_2 N \rfloor + \frac{2}{N}$$

   Avec la procédure suggérée par l'auteur du problème, c'est-à-dire la dichotomie, le nombre de tests est 1 si, par chance, le verre empoisonné est choisi en premier (ce qui se produit avec

2. En effet, un seul test de mélange de la moitié restante suffit à déterminer laquelle des moitiés est empoisonnée.
3. Vous généraliserez facilement à 129 verres.

une probabilité de $1/N$). Il faut $1 + \lfloor \log_2 N \rfloor$ tests dans les autres cas. Le nombre moyen de tests de la procédure du mathématicien est donc de :

$$\begin{aligned} \overline{T} &= \frac{1}{N} + \frac{N-1}{N}\left(1 + \lfloor \log_2 N \rfloor\right) \\ &= \lfloor \log_2 N \rfloor + \frac{N - \lfloor \log_2 N \rfloor}{N} \end{aligned}$$

En moyenne on gagne donc $\frac{N - \lfloor \log_2 N \rfloor - 2}{N}$ tests avec la procédure vraiment optimale !

Pour 129 verres, en moyenne la procédure du mathématicien conduit à 7.95 tests et la procédure optimale à 7.02 tests.

A 10'000 francs le test, cela fait déjà une petite différence !

5 Codage par plages (run-length encoding)

a) Vu que chaque ligne de la première image est unicolore, son code RLE est donné par

1111|1111|0111|0111|1111|1111|0111|0111 (longueur totale de 32 bits)

Sur chaque ligne de la deuxième image, on voit le même motif de 4 pixels noirs (encodés par 1011) suivis de 4 pixels blancs (encodés par 0011), donc le code RLE de l'image est

1011|0011|1011|0011|... (longueur totale de 64 bits)

Sur la troisième image, les deux premières lignes sont des séries alternées de noir et blanc, donc avec le codage RLE, chacun des pixels est représenté par 4 bits ! (0000 si le pixel est blanc, ou 1000 si le pixel est noir). Donc rien que l'encodage des deux premières lignes requiert $16 \times 4 = 64$ bits [4] ! Les six prochaines lignes ne requièrent chacune que 4 bits, comme sur la première image. Au total, on a donc besoin de $64 + 24 = 88$ bits ; clairement pas une compression !

b) Dans cet exercice, on a choisi d'encoder l'image ligne par ligne, mais rien ne nous empêche de faire la même chose colonne par colonne. Pour préciser notre choix dans le code RLE, il suffit d'ajouter un bit au début : 0 pour «ligne par ligne» et 1 pour «colonne par colonne». On peut aini représenter la deuxième image avec 33 bits au total :

1|1111|1111|1111|1111|0111|0111|0111|0111

c) On voit bien que le codage RLE des deux premières lignes est loin d'être optimal. Une façon de remédier à ça est de changer la structure des paquets, en ajoutant un bit au début de chaque paquet (dont la longueur passe donc à 5 bits) : si ce bit vaut 0, alors les quatre prochains bits sont à interpréter au sens du code RLE ci-dessus ; par contre, si le premier bit du paquet vaut 1, alors les quatre prochains bits sont simplement les couleurs des 4 pixels de l'image. Ainsi, la troisième image sera encodée par la séquence de bits suivante :

11010|11010|10101|10101|00111|00111|01111|01111|00111|00111

4. On néglige ici les deux pixels blancs qui se suivent de la fin de la première ligne au début de la seconde.

pour une longueur totale de 50 bits (51 si on ajoute le premier bit 0 de la partie b) pour dire qu'un procède ligne par ligne).

Pour aller plus loin encore on pourrait aussi considérer un codage différent où cette fois on n'utilise plus 1 mais 2 bits au début pour définir le pattern à reproduire et ensuite le nombre de fois qu'il se répète de manière consécutive (en retranchant 1 selon le même principe que le codage RLE). Avec des blocs de 4 bits on aurait donc 2 bits pour le schéma et 2 bits pour la longueur de la répétition ce qui nous donnerait pour la 3e image, la séquance suivante sur 32 bits :

1011|0111|0011|0011|1111|1111|0011|0011

On pourrait aussi utiliser des blocs de 5 bits avec 3 bits pour le nombre de répétitions et on trouverait :

10011|01011|00111|11111|00111

qui ne nécéssite plus que 25 bits pour encoder l'image 3.

Ce codage permettrait de réduire la taille nécessaire pour l'image 1 à 20 bits :

11111|00111|11111|00111

Ce genre d'idées lié à la recherche (dynamique [5]) de la longueur optimale de patterns de bits consécutifs à coder a conduit aux algorithmes de Lempel-Ziv utilisé dans zip.

5. car on voit bien que cela va dépendre de l'image à coder