Remise à niveau – Traitement du signal

Énoncé de la séance N°4

L'objectif de cette séance est de vous familiariser avec quelques techniques de compression. Comme les séances précédentes, la mise en oeuvre s'effectue au travers du logiciel Matlab. Ce document (librement inspiré de [Inc03, Rio03]) ainsi que le matériel nécessaire pour cette séance sont disponibles à l'adresse suivante : `http://nicolaslerme.fr/`.

1 Éléments de compréhension

Tout processus générant de l'information peut être considéré comme source ou signal émettant une séquence $\mathcal{S}$, finie ou non, d'*évènements*. Chaque évènement prend sa valeur sur un *alphabet* $\mathcal{A}$ à N symboles $(\alpha_i)_{0\leq i<N}$. En fonction du niveau de corrélation entre ces évènements, on parle de source *aléatoire* (de longueur infinie) ou *markovienne* (de longueur finie m). Dans ce qui suit, on considèrera uniquement des sources markoviennes.

Il est important de remarquer que l'information portée par un évènement est d'autant plus importante que cet évènement est rare (et inversement). Le poids des lettres au jeu du scrabble en est un bon exemple : la lettre `Z`, rare dans la langue française, rapporte davantage de points que la lettre `A`, beaucoup plus fréquente.

L'étude d'un signal consiste à mesurer la quantité d'information portée par chaque évènement ainsi que l'information portée par le signal dans son ensemble. Si la somme des informations des évènements pris indépendamment est supérieure à l'information globale, cela signifie qu'une certaine quantité d'information est redondante, c'est-à-dire présente plusieurs fois dans le signal. Le processus visant à supprimer cette redondance, c'est-à-dire réduire le nombre d'évènements, est appelé compression. Ce processus a un double avantage : d'une part, il permet d'augmenter la quantité de données stockable sur un même support et d'autre part, il permet de transmettre, pour un même débit, davantage de données à travers un réseau. Cette compression peut être réalisée sans ou avec perte d'information. Dans le premier cas, la source d'origine est restituée exactement. Dans le second cas, il s'agit d'un processus non conservatif visant à obtenir une source compressée la plus proche possible de la source d'origine. Cela n'a évidemment du sens que lorsque la source est perceptuelle (audio, vidéo, image). L'opération inverse, consistant à reconstruire une source proche voire identique à la source d'origine, s'appelle la décompression.

Un résumé des techniques de compression est donné sur la figure 1. Il est important de noter que ces techniques ne s'excluent pas forcément les unes par rapport aux autres mais peuvent être complémentaires. Certaines de ces techniques sont utilisées dans des formats de compression sans perte (TGA, GIF, PNG, TIFF, JPEG, etc.) et avec perte (JPEG, TIFF, JPEG-2000, etc.). Certains de ces formats proposent donc les deux modes de compression.

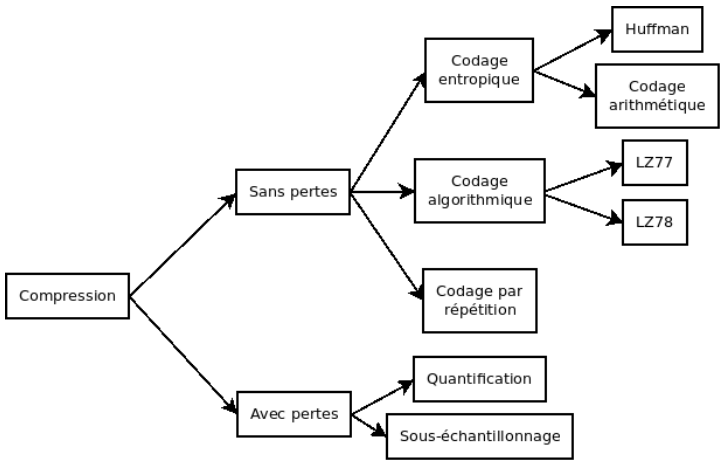

FIGURE 1 – Classification des techniques de compression.

1.1 Codage RLE

Le codage RLE (Run-Length Encoding) est un codage par répétition sans perte. Son principe est le suivant : diminuer la redondance d'information en résumant k évènements successifs ayant le même symbole α par un couple $\{k, \alpha\}$. Considérons par exemple la portion de ligne d'une image dont les intensités sont codées dans l'intervalle $\{0, \ldots, 255\}$:

50	50	50	50	52	52	52	50	50	50	48	48	50	49	49	49

En appliquant la méthode RLE, on obtient alors la nouvelle séquence suivante :

$$(50, 4) \quad (52, 3) \quad (50, 3) \quad (48, 2) \quad (50, 1) \quad (49, 3)$$

Cette technique est utilisée dans plusieurs formats de compression tels que BMP, TGA, TIFF et bien d'autres.

1.2 Codage de Huffman

Le codage de Huffman [Huf52] appartient à la famille des codages entropiques (ou codes à longueur variable) sans perte. Avant de présenter l'approche, nous allons commencer par donner quelques définitions. Pour une source $\mathcal{S}$ de longueur m avec un alphabet $(\alpha_i)_{0 \leq i < N}$, on définit la probabilité d'un symbole α_i par

$$p(\alpha_i) = \frac{k}{m},$$

où k est le nombre d'occurences du symbole α_i. Et bien sûr, on a donc :

$$\sum_{i=0}^{N-1} p(\alpha_i) = 1.$$

Le nombre d'occurrences k de chaque symbole α_i étant strictement positif, il est important de remarquer que l'on a toujours $p(\alpha_i) > 0$, $\forall i \ \in \{0, \ldots, N-1\}$.

Supposons maintenant que l'on veuille coder la source $\mathcal{S}$ sur un alphabet $(\beta_j)_{0\leq j<N'}$. Ce recodage consiste à associer à chaque symbole α_i un mot code $\mathcal{M}(\alpha_i)$ constitué de symboles de l'alphabet cible $(\beta_j)_{0\leq j<N'}$. Soit $L(\alpha_i)$ la longueur (en bits N'-aires) du mot code $\mathcal{M}(\alpha_i)$. On définit la longueur moyenne (en nombre de bits / symbole) du code permettant de passer de l'alphabet $(\alpha_i)_{0\leq i<N}$ à l'alphabet $(\beta_j)_{0\leq j<N'}$ par

$$\overline{L} = \sum_{i=0}^{N-1} p(\alpha_i) L(\alpha_i).$$

La compression de la source $\mathcal{S}$ est donc d'autant plus forte que la longueur moyenne des mots code est faible. La distribution de probabilité des symboles étant fixe, la seule possibilité est par conséquent de choisir des mots code les plus courts possibles afin de rendre la longueur moyenne minimale. Une première possibilité est d'utiliser un code à longueur fixe (FLC) où chaque mot code a la même longueur $L(\alpha_i) = \lceil \log_{N'}(N) \rceil$. Lorsque la longueur d'un mot code et les correspondances entre mots code et symboles sont connues, ce code permet de décoder rapidement un signal codé. Ce type de code n'est en revanche généralement pas optimal.

Dans son deuxième théorème, Shannon a démontré qu'un code est optimal lorsque la longueur moyenne $\overline{L}$ est égale à l'entropie [1] de $\mathcal{S}$, donnée par

$$H(\mathcal{S}) = \sum_{i=0}^{N-1} p(\alpha_i) \underbrace{\log_{N'}\left(\frac{1}{p(\alpha_i)}\right)}_{I(\alpha_i)} = -\sum_{i=0}^{N-1} p(\alpha_i) \log_{N'}(p(\alpha_i)).$$

La longueur de chaque mot code $\mathcal{M}(\alpha_i)$ est donc variable, non entière et sa valeur optimale est

$$L^*(\alpha_i) = -\log_{N'}(p(\alpha_i)).$$

Un tel code est appelé code à longueur variable (VLC). L'entropie donne une borne inférieure que l'on ne peut pas dépasser pour le nombre de bits de chaque mot code. Dans ce cas, on peut facilement voir que la longueur d'un symbole (plus précisément du mot code qui lui est associé) est d'autant plus importante que sa probabilité est faible. Ce résultat est conforme à notre objectif cité plus haut. En revanche, ce théorème ne nous dit pas comment construire un tel code.

La construction d'un VLC est plus délicate qu'un FLC car un mot code peut être décodé de plusieurs manières. Par exemple, si l'on associe le mot code `110` à un symbole α, le mot code `1101` à un symbole β et le mot code `10` à un symbole γ, le code `110110` pourrait être décodé de deux manières : $\alpha\alpha$ ou $\beta\gamma$. L'ambiguïté vient du fait que les mots code `110` et `1101` ont un début commun. Pour lever les ambiguités, il faut donc veiller à ce qu'un code ne contienne aucun mot code qui ne soit préfixe d'un autre. Une première solution (naïve) est d'utiliser un séparateur entre chaque mot émis pour rendre préfixe un code non préfixe. Cette solution n'est pas très élégante et clairement pas optimale. Une autre solution, plus élégante, est de construire directement un code préfixe.

1. L'entropie permet de mesurer l'incertitude sur la nature d'un message donné par rapport au message qui le précède. Elle est nulle pour un signal infini constant et maximale pour un signal aléatoire.

Symbole/occurrence	FLC optimal	$p(\alpha_i)$	$\lceil I(\alpha_i) \rceil$	VLC optimal
E_4	...	...	...	...
N_3	...	...	...	...
O_3	...	...	...	...
S_3	...	...	...	...
D_2	...	...	...	...
C_1	...	...	...	...
I_1	...	...	...	...
M_1	...	...	...	...
P_1	...	...	...	...
R_1	...	...	...	...
	$\overline{L} = \ldots b/s$			$\overline{L} = \ldots b/s$

TABLE 1 – Comparaison des performances entre FLC et VLC optimaux.

L'algorithme d'Huffman produit un code préfixe optimal pour un codage *par symbole*. Autrement dit, pour une source et un alphabet donnés, on ne peut pas faire mieux. On peut montrer que la longueur moyenne d'un code obtenu par cet algorithme vérifie

$$H(\mathcal{S}) \leq \overline{L} < H(\mathcal{S}) + 1.$$

L'algorithme de codage construit un arbre binaire où chaque feuille correspond à un symbole et un mot code. La construction de l'arbre et des mots code peut se faire en une ou deux passes. On présente ci-dessous la version de cet algorithme en deux passes :

1. **Construction de l'arbre :**
   (a) Ordonner dans un tableau les symboles par poids croissants, chaque symbole étant considéré comme un arbre dégénéré (c'est-à-dire, réduit à sa racine),
   (b) Fusionner les arbres de poids minimums (les deux symboles les moins fréquents) qui deviennent les deux fils d'un arbre binaire dont la racine prend pour poids la somme des poids de ses fils,
   (c) Supprimer du tableau les arbres de poids minimums et y insérer le nouvel arbre,
   (d) Continuer jusqu'à ce qu'il ne reste plus qu'un seul arbre dont toutes les feuilles sont des symboles de l'alphabet (`GOTO 1`).
2. **Construction du code :** pour chaque symbole, le mot code associé est obtenu en parcourant l'arbre de la racine à sa feuille en attribuant 0 à gauche et 1 à droite.

Ce codage s'appuyant uniquement sur les statistiques de la source $\mathcal{S}$, des séquences présentant les mêmes probabilités de symboles auront le même arbre. L'algorithme peut être étendu pour coder des séquences sur des alphabets à un nombre de symboles $N' > 2$. L'étape de déco-

dage est plus simple car il suffit de remplacer chaque mot code par son symbole. Cela implique donc de connaître les correspondances entres symboles et mots codes.

1.3 Codage arithmétique

Bien que le codage de Huffman produise un code optimal *par symbole* (c'est-à-dire une source de symboles n'ayant pas de rapports entre eux), rien ne garantit que ce soit le cas lorsque cette contrainte est supprimée. Ce codage possède quelques limitations. Par exemple, un nombre entier de bits est nécessaire pour représenter chaque symbole alors que les longueurs optimales sont des valeurs réelles. Si on cherche à coder un symbole α en binaire avec une probabilité $p(\alpha) = 0.9$, la longueur optimale du mot code associé à ce symbole est $-\log_2(0.9) \simeq 0.15$. Avec le codage de Huffman, sa longueur sera supérieure à 1 bit, soit 6 fois trop.

Plutôt que d'associer un mot code à chaque symbole d'une source, le codage arithmétique code la source avec un seul nombre réel. Comme le codage d'Huffman, le codage arithmétique appartient à la famille des codages entropiques (ou codes à longueur variable) sans perte. Pour une source $\mathcal{S}$ codée sur un alphabet $(\alpha_i)_{0\leq i<N}$, le réel associé est obtenu comme suit :

1. Pour chaque symbole α_i, on commence par calculer sa probabilité $p(\alpha_i)$,
2. À chaque symbole, on associe un sous-intervalle I_{α_i} de longueur $p(\alpha_i)$:
$$I_{\alpha_i} = [\mathrm{BINF}(\alpha_i), \mathrm{BSUP}(\alpha_i)[= \left[\sum_{k=0}^{i-1} p(\alpha_k), \sum_{k=0}^{i} p(\alpha_k)\right[.$$
3. La séquence $\mathcal{S}$ est ensuite codée en générant une suite d'intervalle emboîtés convergeant vers un réel de $]0,1[$ en prenant pour chaque symbole émis le sous-intervalle I_{α_i} déterminé par sa probabilité dans l'intervalle courant correspondant aux évènements déjà traités. Pour chaque symbole α émis, on maintient une borne inférieure (BINF) et supérieure (BSUP) de l'intervalle courant que l'on met à jour ainsi (l'ordre des opérations est important) :
$$\begin{cases} \mathrm{LONG} & \leftarrow & (\mathrm{BSUP} - \mathrm{BINF}) \\ \mathrm{BSUP} & \leftarrow & \mathrm{BINF} + \mathrm{LONG} \times \mathrm{BSUP}(\alpha) \\ \mathrm{BINF} & \leftarrow & \mathrm{BINF} + \mathrm{LONG} \times \mathrm{BINF}(\alpha). \end{cases}$$
4. La valeur du code représentant la source $\mathcal{S}$ est donnée par le réel BINF.

Le décodage de la séquence $\mathcal{S}$ s'effectue de manière symétrique au codage à partir du réel CODE, en supposant connu les sous-intervalles $I_{\alpha_i} = [\mathrm{BINF}(\alpha_i), \mathrm{BSUP}(\alpha_i)[$ (ou de manière équivalente, les symboles et leurs probabilités respectives) :

1. On trouve le premier symbole du message en recherchant lequel des sous-intervalles contient l'entrée CODE (c'est celui tel que $\text{BINF}(\alpha_i) \leq \text{CODE} < \text{BSUP}(\alpha_i)$).
2. On édite le symbole trouvé et on le supprime de l'entrée : on modifie la valeur de CODE en lui soustrayant $\text{BINF}(\alpha_i)$ puis en divisant par la longueur de l'intervalle I_{α_i}, c'est-à-dire $p(\alpha_i)$:
$$\text{CODE} \leftarrow \frac{\text{CODE} - \text{BINF}(\alpha_i)}{p(\alpha_i)}.$$
3. On recommence avec cette nouvelle valeur de CODE (`GOTO 1`).

D'un point de vue théorique, le codage arithmétique et le codage de Huffman ne font tous les deux aucune hypothèse sur la nature du signal et sont les seuls à réellement garantir que le signal codé occupera moins (au pire autant) de place que le signal initial.

2 Exercices

2.1 Codage de Huffman

Question 1 : On considère la séquence $\mathcal{S}$ =`"COMPRESSIONDEDONNEES"` à $m = 20$ caractères, codée sur un alphabet $(\alpha_i)_{0\leq i<N}$ = {`C,D,E,I,M,N,O,P,R,S`} à $N = 10$ symboles. On souhaite recoder $\mathcal{S}$ sur l'alphabet $(\beta_j)_{0\leq j<N'} = \{0, 1\}$ à $N' = 2$ symboles. Trouvez un FLC optimal pour cette séquence et appliquez l'algorithme d'Huffman en donnant l'arbre et le code associés. Complétez ensuite le tableau 1. Comparez les longueurs moyennes $\overline{L}$ obtenues. Qu'observez-vous ? Quel est le gain par rapport à un codage ASCII standard utilisant 8 bits par symbole ? Calculez l'entropie de $\mathcal{S}$. Concluez.

Question 2 : On considère la séquence $\mathcal{S}$ =`"ABDAACBADABAAACB"` à $m = 16$ caractères, codée sur un alphabet $(\alpha_i)_{0\leq i<N}$ = {`A,B,C,D`} à $N = 4$ symboles. On souhaite recoder $\mathcal{S}$ sur l'alphabet $(\beta_j)_{0\leq j<N'} = \{0, 1\}$ à $N' = 2$ symboles. Appliquez l'algorithme d'Huffman sur la séquence $\mathcal{S}$ en donnant l'arbre et le code correspondants. Calculez l'entropie de $\mathcal{S}$ et comparez-la avec la longueur moyenne du code obtenu. Qu'observez-vous ? Peut-on généraliser à des séquences codées sur un nombre de bits $N' \neq 2$?

Question 3 : Soit un $\mathcal{T}$ l'arbre de Huffman obtenu après application de l'algorithme de codage avec un ensemble de feuilles $\mathcal{F}$. À partir de cet arbre, comment peut-on retrouver la longueur moyenne d'un code $\overline{L}$? Donnez son expression.

Question 4 : Soit $\mathcal{S}$ =`"GATGATGATAATGATCACGATGATAATGATGAAGATGATGATCACGATGATGCCGATGAT"` à $m = 60$ caractères, codée sur un alphabet $(\alpha_i)_{0\leq i<N}$ = {`A,T,G,C`} à $N = 4$ symboles. On souhaite recoder $\mathcal{S}$ sur l'alphabet $(\beta_j)_{0\leq j<N'} = \{0, 1\}$ à $N' = 2$ symboles. Donnez l'arbre de Huffman et le code associés. En uilisant Huffman, est-il possible de faire mieux ?

2.2 Codage RLE

Question 1 : Complétez le code fourni pour compresser/décompresser une image.

Question 2 : Testez en utilisant la fonction `checkerboard` avec un nombre croissant de carrés et calculez le taux de compression associé. Tracez la courbe du taux de compression en fonction du nombre de carrés présent sur l'échiquier. Pouvez-vous dire jusqu'à quel point cette méthode présente un intérêt ?

Question 3 : Testez cette approche sur plusieurs images en niveaux de gris. L'approche est-elle toujours intéressante ?

2.3 Codage arithmétique

Question 1 : $\mathcal{S}$ =`"ABACBAC"` à $m = 8$ caractères, codée sur un alphabet $(\alpha_i)_{0 \leq i < N} =$ {`A,B,C`} à $N = 3$ symboles. Appliquez l'algorithme de codage arithmétique sur la séquence $\mathcal{S}$ et donnez le réel correspondant.

Question 2 : Recommencez avec la séquence $\mathcal{S}'$ =`"AAACCBB"`. Le réel obtenu est-il le même que pour la séquence $\mathcal{S}$?

Question 3 : Implémentez les algorithmes de codage/décodage en complétant les fichiers fournis. Quel inconvénient voyez-vous à ces implémentations ?

Références

[Huf52] D.A. Huffman. "a method for the construction of minimum-redundancy codes". In *Proceedings of the Institute of Radio Engineers*, pages 1098–1102, September 1952.

[Inc03] E. Incerti. *Compression d'image : algorithmes et standards*. 2003.

[Rio03] O. Rioul. Codage entropique à longueur variable, 2003.