Chapitre 3

Codage sans perte de l'information

Sommaire du chapitre

Nous continuons de suivre le cheminement du signal. Après avoir été échantillonné puis quantifié, il est maintenant représenté par une suite de 0 et de 1. Nous allons voir comment on peut préparer sa transmission dans de bonnes conditions. Plus précisément, deux raffinements très utiles vont être présentés dans ce chapitre : tout d'abord ce qui est parfois appelé le *codage source*, i.e. une méthode de compression du signal, comme le fait par exemple le programme zip, ensuite, ce qui est parfois appelé le *codage canal*, *i.e.* l'ajout judicieux de *bits de correction* qui vont permettre de corriger les erreurs éventuelles qui vont affecter

le signal au cours de sa transmission. Nous regarderons un codage particulier appelé le code de HAMMING qui permet simplement de détecter deux erreurs par mot transmis et d'en corriger une.

3.1 Codage source et compression sans perte

On appelle donc codage source l'ensemble des techniques permettant de compresser avec ou sans perte d'information un signal numérique. Ceci revient en fait à coder de manière astucieuse les symboles émis par une source de manière à réduire le nombre total de bits utilisés.

Dans cette section on démontre le théorème fondamental du codage source, qui indique une limite théorique de compression sans perte et on présente un algorithme, dû à HUFFMAN [1], permettant d'approcher arbitrairement cette limite.

3.1.1 Définitions

On commence par poser deux définitions.

Définition 3.1 — Source de symboles m-aire

On appelle source de symboles m-aire tout dispositif émettant des *mots* construits par concaténations de symboles pris dans un *alphabet* de taille m.

Par exemple en informatique, l'alphabet est constitué de deux symboles 0 et 1. L'anglais utilise quant à lui un peu plus de 26 symboles (car on peut ajouter aux 26 lettres de l'alphabet latin quels autres symboles comme les signes de ponctuation).

On appelle *source sans mémoire* une source pour laquelle la probabilité d'émission d'un symbole ne dépend pas des symboles précédemment émis. Dans ce cours, on ne va considérer que des sources sans mémoire, mais la plupart des résultats présentés dans ce chapitre, et en particulier le théorème de SHANNON, peuvent être étendu à des cas beaucoup plus généraux.

Définition 3.2 — Extension d'ordre k

On appelle extension d'ordre k d'une source S, la source S_k dont l'alphabet est obtenu par concaténation de k symboles consécutif de la source S.

1. David Albert HUFFMAN (9 août 1925 - 7 octobre 1999, Ohio),chercheur américain pionnier dans le domaine de l'informatique.

Évidemment, le débit de la source S_k est k fois moins élevé que celui de la source initiale S. De plus, si S est une source de symboles m-aire, l'alphabet de la source S_k sera de taille au plus m^k, cette valeur étant atteinte si S est sans mémoire.

3.1.2 Entropie et mesure de la quantité d'information

Pour pouvoir compresser efficacement les symboles émis par une source, il est utile de savoir mesurer la quantité d'information apportée par les symboles qu'elle produit. Intuitivement, un symbole qui apparaît rarement, par exemple le *w* dans les textes en français, apporte plus d'information qu'un symbole très fréquent, par exemple le *e* toujours dans les textes en français [2].

Quantité d'information par symbole

Étant donné une source S émettant des symboles d'un alphabet A, on commence par calculer la probabilité d'apparition des symboles de A. Ce calcul peut-être fait *offline*, c'est-à-dire *a posteriori*, en calculant la fréquence d'apparition de chaque symbole si l'on dispose de l'ensemble du signal émis, ou bien *online*, c'est-à-dire *a priori*, si l'on connaît certaines propriétés de la source, par exemple si l'on sait que la source émet des mots dans une certaine langue écrite.

Notons h la quantité d'information — que nous n'avons pas encore définie — apportée par un symbole $x \in A$. Faisons une liste des propriétés de h attendues.

1. Tout d'abord, on souhaite que la quantité d'information apportée par l'émission d'un symbole x augmente lorsque la probabilité d'émission de x, notée $p(x)$ dans la suite, diminue, ce qui peut se modéliser par
$$h(x) = f\left(\frac{1}{p(x)}\right),$$
où f est un fonction *croissante*.
2. Si la probabilité d'émission d'un symbole est 1, alors celui-ci n'apporte aucune information, on peut alors imposer
$$f(1) = 0.$$
3. On souhaite enfin que la quantité d'information apportée par deux messages *indépendants* soit la somme des quantités d'information apportées par chacun des messages. On peut alors demander à f de vérifier
$$f\left(\frac{1}{p(x \text{ et } y)}\right) = f\left(\frac{1}{p(x) \cdot p(y)}\right) = f\left(\frac{1}{p(x)}\right) + f\left(\frac{1}{p(y)}\right).$$

2. à l'exception de *La disparition* de G. Perec, un livre ne contenant pas -dans un but bien précis- la lettre *e*.

Ces différentes propriétés impliquent que la fonction h doit être choisie de la forme :

$$h(x) = -\log_q(p(x)).$$

La base q du logarithme va dépendre de la dimension de la «dimension» de codage que l'on va utiliser.

Entropie d'une source

On souhaite maintenant définir la quantité moyenne d'information apportée par la source S. Cette quantité est appelée *entropie* et est naturellement définie par la formule :

$$H(S) := \sum_{x \in A} p(x)h(x) = -\sum_{i=1}^{m} p_i \log_q(p_i),$$

où l'on a noté p_i la probabilité d'émission du i-ème symbole de l'alphabet A.

Maximisation de l'entropie

Notre but est maintenant de recoder les symboles de S de telle sorte que son entropie soit maximisée. Pour ce faire, on introduit le résultat préliminaire suivant.

Lemme 3.1 — Inégalité de Gibbs Soit n nombres p_i et q_i tels que :

$$0 < p_i < 1, \qquad 0 < q_i < 1,$$

et

$$\sum_{i=1}^{n} p_i = 1, \qquad \sum_{i=1}^{n} q_i \leq 1.$$

Alors :

$$\sum_{i=1}^{n} p_i \log_q \left(\frac{q_i}{p_i}\right) \leq 0,$$

avec égalité si et seulement si

$$\forall i,\ 0 \leq i \leq n, \qquad p_i = q_i.$$

Exercice 3.1 :

Montrer ce lemme pour le logarithme népérien.

Cette inégalité est également appelée *inégalité de Jensen* [3] dans d'autres contextes.

Proposition 6 : *L'entropie vérifie l'inégalité suivante*

$$H(S) \leq \log_q(m),$$

avec égalité dans le cas où tous les symboles de S sont équiprobables, c'est-à-dire que pour tout $i \in \{1, \dots, m\}$, $p_i = 1/m$.

Preuve : À partir de l'inégalité de Gibbs, on pose $q_i = 1/m$ pour tout $i \in \{1, \dots, m\}$.

On remarque que l'incertitude sur le résultat d'une expérience est d'autant plus grande que tous les résultats possibles sont équiprobables.

3.1.3 Propriétés d'un codage source

Comme précédemment annoncé, on va chercher à construire un nouvel alphabet de codage maximisant l'entropie, ou, du point de vue de la compression, minimisant la taille moyenne des nouveaux symboles obtenus :

$$\bar{\ell} = \sum_{i=1}^{m} n_i p_i,$$

où n_i est la taille du nouveau code du i-ème symbole de A.

Supposons que la source (après quantification) soit constituée de Q états, c'est-à-dire $S = \{s_1, \dots, s_Q\}$. À chacun de ces états s_i, on va associer une suite de n_i symboles d'un nouvel alphabet q-aire. Ceux-ci constituent un code source que l'on note $C = \{c_1, \dots, c_q\}$.

Définition 3.3 — Déchiffrabilité

Un code est appelé un code *déchiffrable*, ou encore à décodage unique, si toute suite de mots de code ne peut être interprétée que de manière unique.

Il y a plusieurs manières d'avoir un code déchiffrable :

- définir un code à longueur fixe : c'est-à-dire avec des mots de longueur fixe, qu'on peut décoder sans ambiguïté (par exemple : le code ASCII) ;
- consacrer un symbole de l'alphabet de destination comme séparateur de mot ;

3. Johan Ludwig William Valdemar Jensen, surtout connu comme Johan Jensen, (8 mai 1859, Nakskov, Danemark - 5 mars 1925, Copenhague, Danemark) était un mathématicien et ingénieur danois. Il est surtout connu pour sa fameuse inégalité de Jensen. En 1915, il démontra également la formule de Jensen en analyse complexe.

— on évite qu'un mot du code soit identique au début d'un autre mot.

On va donc ajouter une autre contrainte sur le nouvel alphabet. On souhaite que celui-ci soit *auto-ponctué*[4], c'est-à-dire que les nouveaux symboles soient émis par simple concaténation. Ceci est formalisé par la définition suivante.

Définition 3.4 — Code irréductible

On appelle code irréductible, ou code préfixe, ou code auto-ponctué, un ensemble de mots tel qu'aucun mot ne soit le début d'un autre.

De la sorte, le récepteur d'un message sait toujours comment découper la suite de symboles qu'il reçoit, car il sait identifier les moments où l'on passe d'un symbole à un autre.

Tous les codes utilisés dans la vie courante ne sont pas irréductibles. Les langues écrites ne le sont pas en général, le code morse non plus par exemple.

Problématique

Soit S une source caractérisée par un débit D_s (symbole Q-aire/seconde). Soit un canal non-bruité de débit maximal D_c (symbole q-aire/seconde). On définit :

— le taux d'émission de la source $T = D_s H(S)$;

— la capacité du canal $C = D_c \log(q)$.

Si $T > C$, le canal ne peut écouler l'information. Il faut donc être dans le cas contraire. Si on dispose d'un code q-aire dont la longueur moyenne $\bar{\ell}$ des mots est telle que $\bar{\ell} D_s \leq D_c$, alors celui-ci peut être utilisé pour la transmission.

Le codage source vise à éliminer la redondance d'information *sans perte*.

Arbre q−aire

Les codes auto-ponctués sont de manière naturelle associés à la notion d'arbre q-aire, que l'on introduit maintenant.

Définition 3.5 — Arbre q-aire

On appelle arbre q-aire un arbre dont chaque noeud est soit une feuille, soit possède q descendants.

En algorithmique, on distingue plusieurs types d'arbres :

— les listes, où à chaque élément de la liste n'est associé qu'au plus un autre élément ;

4. On parle aussi de code *préfixe*.

— les arbres binaires, où à chaque élément de la liste n'est associé qu'au plus deux éléments ;
— les arbres q-aire, où à chaque élément de la liste n'est associé qu'au plus q éléments.

Par exemple un arbre binaire est obtenu en considérant l'arbre généalogique des ascendants d'une personne (et en se fixant une limite dans le temps). La notion d'arbre q-aire est le bon concept pour représenter les codes auto-ponctués. Le nombre q représente le nombre de caractères utilisés pour le nouveau codage des symboles de la source A et les mots sont donnés par l'ensemble des feuilles. Le découpage du message reçu se fait donc par parcours successifs de l'arbre, avec retour à la racine à chaque fois que l'on obtient une feuille.

Si l'alphabet est un alphabet binaire (composé de `0` et de `1`) alors, l'arbre correspondant est un arbre binaire très utile en algorithme et pour lesquels il existe beaucoup de résultats et de librairies.

Les arbres q-aires vérifient l'inégalité de Kraft [5] qui nous servira dans la suite.

Lemme 3.2 — Inégalité de Kraft et réciproque Les longueurs $(n_i)_{1\leq i\leq m}$ des m chemins allant vers les m feuilles d'un arbre q-aire vérifient :

$$\sum_{i=1}^{m} q^{-n_i} \leq 1. \qquad (\text{ inégalité de Kraft})$$

Réciproquement, si l'on se donne un entier q et m entiers $(n_i)_{1\leq i\leq m}$ vérifiant l'inégalité de Kraft, alors on peut construire un arbre q-aire ayant m feuilles situées à des profondeurs $(n_i)_{1\leq i\leq m}$.

Exercice 3.2 :

Montrer cette inégalité pour les code irréductibles.

Théorème fondamental du codage source

On dispose maintenant de tous les outils pour énoncer et démontrer le théorème fondamental du codage source, qui est également un théorème de Shannon.

5. Cette inégalité fut publiée pas Leon Kraft en 1949. Dans l'article correspondant Kraft ne considère que les codes auto-ponctués et attribue l'analyse qu'il présente pour obtenir son résultat à Raymond M. Redheffer. Cette inégalité est aussi parfois appelée "Théorème de Kraft-McMillan" après que Brockway McMillan l'ait découverte indépendamment en 1956. McMillan prouve le résultat pour une classe plus large de codes et attribue quant à lui la version correspondant aux codes auto-ponctués de Kraft à des observations de Joseph Leo Doob en 1955.

Théorème 3.1 — fondamental du codage source — Dans le codage d'une source S sans mémoire à l'aide d'un alphabet de taille q, la longueur moyenne $\bar{\ell}$ des mots du code utilisé pour coder un symbole de S vérifie :

$$H(S) \leq \bar{\ell},$$

et l'on peut approcher cette limite aussi près que l'on veut, quitte à coder les extensions de S au lieu de S elle-même.

Preuve : On considère m symboles codés par des mots de longueurs $(n_i)_{1\leq i\leq m}$. Si ces mots sont obtenus comme les feuilles d'un arbre q-aire, alors leurs longueurs vérifient :

$$Q := \sum_{i=1}^{m} q^{-n_i} \leq 1.$$

D'après l'inégalité de Gibbs, on a également :

$$\sum_{i=1}^{m} p_i \log_q \left(\frac{q^{-n_i}}{p_i}\right) \leq 0,$$

où p_i est la probabilité d'émission du i-ème symbole de S avec n_i la longueur du nouveau code du i-ème symbole. On en déduit :

$$-\sum_{i=1}^{m} p_i \log_q p_i \leq \log_q q \times \sum_{i=1}^{m} p_i n_i = 1 \times \sum_{i=1}^{m} p_i n_i.$$

Or :

- $H(S) = -\sum_{i=1}^{m} p_i \log_q p_i$,
- $\sum_{i=1}^{m} p_i = 1$,
- $\sum_{i=1}^{m} p_i n_i = \bar{\ell}$, la longueur moyenne d'un mot du nouveau code.

Finalement, on trouve :

$$H(S) \leq \bar{\ell}.$$

Nous avons donc obtenu la borne théorique de compression annoncée dans l'introduction de cette section.

Montrons maintenant qu'on peut s'approcher aussi près que l'on veut de cette borne. On cherche à minimiser $\bar{\ell}$. Pour ce faire, on peut essayer d'approcher le cas d'égalité dans les inégalités de Kraft et de Gibbs utilisées, c'est-à-dire à avoir :

$$Q = 1,\ q^{-n_i} = p_i,$$

qui se traduit pour n_i par :

$$n_i = -\log_q p_i.$$

Mais cette dernière fraction n'est pas forcément égale à un entier, on choisit donc de définir n_i par :

$$-\log_q p_i \leq n_i \leq -\log_q p_i + 1.$$

En conséquence, on a :

$$\sum_{i=1}^{m} q^{-n_i} \leq \sum_{i=1}^{m} p_i = 1,$$

et l'inégalité de Kraft est vérifiée. Il existe donc un arbre q-aire correspondant aux longueurs n_i, dont on peut déduire les mots (de longueurs n_i) du nouveau codage. De plus on a :

$$-p_i \log_q p_i \leq p_i n_i \leq -p_i \log_q p_i + p_i,$$

d'où l'on déduit par sommation :

$$H(S) \leq \bar{\ell} \leq H(S) + 1.$$

Pour approcher arbitrairement la limite théorique, une stratégie judicieuse est de considérer non plus la source S comme source initiale, mais son extension d'ordre k, S_k. En répétant le raisonnement précédent, on obtient :

$$H(S_k) \leq \bar{\ell}_k \leq H(S_k) + 1,$$

où $\bar{\ell_k}$ représente la longueur moyenne des mots du nouveau codage de S_k. Si on considère les symboles comme des variables aléatoires prenant des valeurs dans $A = \{s_1, \dots, s_m\}$, alors

$$\bar{\ell}_k = \sum_{x_1,\dots,x_k} p(x_1, \dots, x_k) n(x_1, \dots, x_k)$$

où la somme est sur toutes les combinaisons possibles, et p est la probabilité d'avoir la combinaison et n la longueur de codage de cette suite de symboles. Pour cette source là, on applique le théorème que l'on vient d'obtenir, et on obtient :

$$H(S_k) \leq \bar{\ell}_k \leq H_k + 1,$$

inégalités que l'on divise par k :

$$\frac{H(S_k)}{k} \leq \frac{\bar{\ell}_k}{k} \leq \frac{H(S_k)}{k} + \frac{1}{k}.$$

La quantité $\frac{\bar{\ell}_k}{k}$ est alors notre longueur moyenne pour la source S et notons $\mathscr{H} = \frac{H(S_k)}{k}$. On a alors

$$\begin{aligned}\mathscr{H} &= \frac{1}{k} \sum_{x_1,\dots,x_k} p(x_1, \dots, x_k) \log_q \left(\frac{1}{p(x_1, \dots, x_k)} \right) \\ &= \frac{1}{k} \mathbf{E} \left[\log_q \left(\frac{1}{p(X_1, \dots, X_k)} \right) \right]\end{aligned}$$

où on a noté X_k les variables aléatoires du processus. Dans ce cours, on ne considère que des sources sans mémoire, et donc les symboles sont indépendants, ainsi $p(x_1, \dots, x_k) =$

$p(x_1)p(x_2)\cdots p(x_k)$ et donc

$$\begin{aligned}\mathscr{H} &= \frac{1}{k}\mathbf{E}\left[\log_q\left(\prod_{i=1}^{k}\frac{1}{p(X_i)}\right)\right]\\ &= \frac{1}{k}\sum_{i=1}^{k}\mathbf{E}\left[\log_q\left(\frac{1}{p(X_i)}\right)\right]\\ &= \frac{1}{k}\sum_{i=1}^{k}H\\ &= H.\end{aligned}$$

Remarque 3.1 :

Nous avons démontrer ce résultat dans le cas d'une source sans mémoire, mais on peut aussi montrer que $\mathscr{H} \leq H$ dans le cas d'une source avec mémoire.

Remarque 3.2 :

Évidemment, approcher cette limite a un certain coût ! En effet, la technique de codage considérée implique à un moment ou à un autre la transmission d'un dictionnaire permettant le décodage pour retrouver les symboles émis initialement. Si l'on code les extensions de S au lieu de S elle-même, la taille du dictionnaire va augmenter et ce exponentiellement par rapport à l'extension considérée.

3.1.4 Algorithme de Huffman

Code optimal

Un code optimal est un code préfixé de longueur moyenne minimale. La compression est d'autant plus forte que la longueur moyenne des mots de code est faible. Trouver un code optimal revient donc à choisir les longueurs des mots de codes, par rapport à la distribution de probabilité des symboles de source, afin de rendre la longueur moyenne minimale. Pour trouver un tel code, il faut minimiser la longueur moyenne du code $\bar{\ell}$ sous les conditions de l'inégalité de Kraft (qui donne une condition nécessaire et suffisante pour qu'un code possède un code préfixé équivalent). Mathématiquement cela revient à résoudre le problème de minimisation de la longueur moyenne

$$\bar{\ell} = \sum_{i=1}^{m} p_i n_i$$

sur l'ensemble des longueurs $\{n_i\}_{i\in\{1,\dots,m\}}$ et sous la contrainte donnée par l'inégalité de Kraft :

$$\sum_{i=1}^{m} q^{-n_i} \leq 1.$$

Par la méthode des *multiplicateurs de Lagrange*, on définit le lagrangien J :

$$J = \sum_{i=1}^{m} p_i n_i + \lambda \left(\sum_{i=1}^{m} q^{-n_i} - 1 \right)$$

que l'on différentie par rapport aux n_i. Un rapide calcul donne les longueurs optimales $n_i^* = -\log_2(p_i)$, c'est-à-dire une longueur moyenne égale à l'entropie. Bien évidemment, ces longueurs optimales ne sont pas des longueurs entières. Le but de cette section est de présenter un algorithme qui permet d'approcher cette limite.

L'algorithme

Il nous reste à donner une méthode permettant la mise en pratique du résultat théorème 3.1 (qui n'est pas constructif). C'est l'algorithme de HUFFMAN qui fournit ce résultat à partir d'une source de m symboles, dont on connaît les probabilités $(p_i)_{1\leq i\leq m}$ d'émission. En pratique cet algorithme explique comment construire un *arbre de codage*.

Algorithm 2 Algorithme de HUFFMAN d'un code q-aire de N symboles

Require : $(C_i)_{i\in\{1,\dots,N\}}$ symboles de distribution de probabilité $(p_i)_{i\in\{1,\dots,N\}}$

1 : Définir N nœuds actifs correspondant aux N symbole
2 : Calculer r comme le reste de la division de $1-N$ par $q-1$ *(si on a un code binaire, alors $r=0$)*
3 : **while** il reste plus d'un nœud actif **do**
4 : Grouper, un tant que fils d'un nœud nouvellement créé, les $q-r$ nœuds actifs les moins probables et les r nœuds inutilisés
5 : Marquer les $q-r$ nœuds actifs comme *non-actifs* et le nœud nouvellement crée comme *actif*
6 : Assigner au nœud nouvellement créé une probabilité égale à la somme des probabilités des $q-r$ nœuds qui viennent d'être désactivés.
7 : Poser $r=0$
8 : **end while**

Exercice 3.3 :

On considère l'alphabet $S = s_1, s_2, \dots, s_8$ avec la distribution de fréquence suivante :

s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
0.4	0.18	0.1	0.1	0.07	0.06	0.05	0.04

1. Calculer l'entropie de la source d'information S.
2. Calculer un codage de HUFFMAN de cette source puis la longueur moyenne de ce codage. Comparer avec la longueur moyenne à l'entropie.

Remarque 3.3 :

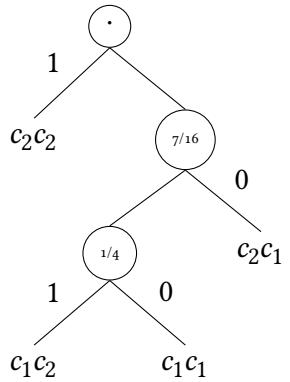
En ayant calculé l'entropie de la source S, on peut établir une prescription d'un objectif pour la taille moyen du code $\bar{\ell}$. Si l'objectif de longueur moyenne n'est pas atteint, alors il suffit de reprendre l'algorithme avec une extension d'ordre plus élevée du code.

Exemple : Réduction de la longueur moyenne par extension

Considérons un code simpliste : soit $S = (C, P)$ avec $C = (c_1, c_2)$ et $P = (p_1, p_2) = (1/4, 3/4)$. Le codage de HUFFMAN est alors juste un arbre à un nœud et deux feuilles qui donne $c_1 \to 0$ et $c_2 \to 1$ de taille respective $n_1 = n_2 = 1$, de longueur moyenne

$$\bar{\ell} = 1 \times 1/4 + 1 \times 3/4 = 1.$$

Considérons désormais l'extension d'ordre 2, c'est-à-dire $S^2 = (C^2, P^2)$ où $C^2 = (c_1c_1, c_1c_2, c_2c_1, c_2c_2)$ et $P^2 = (1/16, 3/16, 3/16, 9/16)$. Cette extension d'ordre 2 donne donc l'arbre suivant.



$c_1c_1 \to 010$, $c_1c_2 \to 011$, $c_2c_1 \to 00$ et $c_2c_2 \to 1$.

La longueur moyenne est alors de

$$\bar{\ell}_2 = 3 \times 1/16 + 3 \times 1/16 + 2 \times 3/16 + 9/16 = 1.6875,$$

ce qui donne

$$\bar{\ell}^* = 0.5 \times \bar{\ell}_2 < \bar{\ell}.$$

3.2 Codage canal et correction d'erreur

On considère maintenant uniquement des signaux binaires même de nombreux définitions et résultats présentés ici peuvent être généralisés. Une fois le signal – c'est-à-dire une suite de 0 et de 1 – compressé, il va maintenant s'agir de le transmettre. Le medium utilisé pour la transmission est appelé *canal*. Évidemment, les canaux peuvent être de nature très variées. Il peut s'agir d'une onde électromagnétique envoyée dans une fibre optique, dans l'atmosphère ou dans dans l'espace dans le cas de la communication par satellite, d'impulsions électriques envoyées dans un réseau, ou encore d'une clef USB, d'un disque dur ou d'un disque optique codé sous le format `.wav` pour les CD ou sous des formats plus évolués pour les DVD.

Chacun de ces canaux rend intrinsèquement possible l'introduction d'erreurs. Il y a concrètement toujours un risque que le médium transforme un 0 en 1 et réciproquement. Cela peut ne pas avoir d'effet grave si le code supporte une image ou un son, mais peut être beaucoup plus lourd de conséquence s'il s'agît du texte d'un programme exécutable par exemple. Dans ce cas, la transmission échouera si le moindre symbole du programme est modifié.

Il apparaît donc nécessaire de mettre en place une stratégie permettant au

moins de détecter, lors de la réception, les erreurs de transmission, voir de les corriger. De nombreuses méthodes atteignant cet objectif ont été conçues depuis une quarantaine d'années. Elles reposent toutes sur l'ajout de *bits de correction*, ou *bits de contrôle*, c'est-à-dire qu'elles ont toujours pour effet négatif de grossir la taille du signal à envoyer. Un compromis doit donc être trouvé entre qualité de correction et alourdissement du signal.

Dans cette section, on présente des outils permettant de quantifier le facteurs intervenant dans ce compromis et une stratégie générale de codage canal ainsi qu'une de ses réalisations concrètes.

Dans la suite on désignera indifféremment de *code correcteur*, *codage canal* ou encore *code* la stratégie de correction considérée.

3.2.1 Une approche naïve

Une idée simple que l'on peut avoir est de répéter un certain nombre de fois chaque symbole à transmettre. Si on choisit par exemple de doubler le symbole, c'est-à-dire d'appliquer la fonction :

$$0 \to 00,$$

$$1 \to 11,$$

on pourra facilement détecter une erreur (si les symboles reçus ne coïncident pas deux à deux). Par contre, on ne pourra corriger l'erreur correctement qu'une fois sur deux en moyenne - en choisissant arbitrairement de remplacer la séquence erronée détectée par un 0, par exemple.

Dans cette démarche, on a ajouté un bit de correction par bit transmis. Si on choisit maintenant d'en ajouter deux, en adoptant une stratégie de triplement, on constate que la correction sera plus efficace : on remplacera une séquence erronée, par exemple 001, par le signe apparaissant majoritairement [6] dans le triplet, 0 dans notre exemple. Cette méthode permet de détecter deux erreurs par bloc de trois bits et d'en corriger une : en effet, si deux erreurs affectent, lors de la transmission la séquence de trois bits, alors la stratégie choisie donnera lieu, lors du décodage, à une erreur. Pour aller plus loin, on peut regarder l'exemple d'une canal introduisant 5% d'erreurs, avec lequel on applique la stratégie du triplement. Puisqu'une erreur est corrigée, la probabilité qu'une séquence de trois bits soit transmise correctement ou avec une seule erreur est de :

$$p = 0,95^3 + 3 \times 0,95^2 \times 0,005 = 0,99275.$$

Ainsi le taux de succès est passé de 95% à un taux supérieur 99%. Évidemment, ceci a un coût ! Il a en effet fallu tripler la taille du signal à transmettre.

On va voir dans la suite comment optimiser l'usage des bits de correction.

6. On parle parfois de *stratégie du vote majoritaire*.

3.2.2 Codes linéaires par blocs

Découpage en bloc et structure algébrique des blocs

Un préalable souvent nécessaire à la correction d'erreur est le découpage du signal à transmettre en blocs de taille fixe, disons k. L'ensemble sur lequel on va travailler est donc B^k où $B = \{0, 1\}$. En munissant celui-ci de l'addition composante par composante dans le groupe $(\mathbf{Z}/2\mathbf{Z}, +)$, ainsi que du corps des scalaires (fini) $(\mathbf{Z}/2\mathbf{Z}, +, \times)$, on voit que l'on peut doter B^k d'une structure d'espace vectoriel (fini) que l'on notera $\mathbf{F}_2^k$.

Remarque 3.4 :

— Le plus petit corps fini est noté $\mathbf{F}_2$. Il est composé de deux éléments distincts, 0 qui est l'élément neutre pour l'addition, et 1 qui est élément neutre pour la multiplication. Ceci détermine les tables de ces deux opérations en dehors de $1 + 1$ qui ne peut alors être que 0, car 1 doit avoir un opposé. On vérifie alors qu'elles définissent bien un corps commutatif.

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Le corps $\mathbf{F}_2$ peut s'interpréter diversement. C'est l'anneau $\mathbf{Z}/2\mathbf{Z}$, les entiers pris modulo 2, c'est-à-dire que 0 représente les entiers pairs, 1 les entiers impairs (c'est le reste de leur division par 2), et les opérations se déduisent de celles sur $\mathbf{Z}$.
C'est aussi l'ensemble des valeurs de vérité classiques, 0 pour le faux, et 1 pour le vrai. L'addition est le « ou exclusif » (xor), la multiplication le « et ».

— Une généralisation naturelle de $\mathbf{F}_2 = \mathbf{Z}/2\mathbf{Z}$ est, pour p premier, le corps $\mathbf{Z}/p\mathbf{Z}$ à p éléments, noté aussi $\mathbf{F}_p$. Pour que l'anneau $\mathbf{Z}/p\mathbf{Z}$ soit un corps, il faut et il suffit que p soit premier.

Théorie des codes correcteurs

Introduisons quelques définitions pour pouvoir travailler mathématiquement avec ces concepts.

Définition 3.6 — Poids d'un mot

On définit le poids d'un mot m de $\mathbf{F}_2^k$ comme le nombre de composantes non nulles du mot. On le notera $w(m)$.

On définit alors une distance sur cet espace vectoriel.

Définition 3.7 — Distance de Hamming

La distance de Hamming entre deux mots m_1 et m_2 de $\mathbf{F}_2^k$ est le nombres de composantes distinctes de m_1 et m_2. On la note $d(m_1, m_2)$.

On peut montrer que cette distance de Hamming [7] est bien une distance.

Proposition 7 : *La distance de* Hamming *est une distance sur* $\mathbf{F}_2^k$.

On a par exemple :

$$d(111, 101) = 1.$$

Une propriété intéressante de cette distance est qu'elle vérifie :

$$d(m_1, m_2) = d(m_1 + m_2, 0_{\mathbf{F}_2^n}), \tag{3.1}$$

où :

$$0_{\mathbf{F}_2^n} = (\underbrace{0, \dots, 0}_{n \text{ fois}}).$$

Autrement dit, pour connaître la distance entre deux mots, il suffit de les additionner et de compter le nombre de composantes égales à 1 dans le résultat. Derrière cette notion de distance, se cache une stratégie de correction : étant donné que les mots du code ne forment qu'une sous-partie de $\mathbf{F}_2^n$, lorsqu'on reçoit en sortie de canal un mot ne figurant pas dans $C(k, n)$, on le remplace par le mot du code le plus proche.

Code Dans le cadre que l'on vient d'introduire, l'ajout de r bits de correction peut être vu comme l'application d'une certaine fonction injective :

$$g : \mathbf{F}_2^k \to \mathbf{F}_2^n,$$

7. Richard Wesley Hamming (11 février 1915, Chicago - 7 janvier 1998, Monterey, Californie) est un mathématicien américain surtout connu pour son algorithme de correction d'erreur, le Code de Hamming. Il travailla avec Claude Shannon entre 1946 et 1976 aux laboratoires Bell.

où $n = k+r$. La stratégie de correction est dite *linéaire*, lorsque g est une application linéaire (entre les espaces $\mathbf{F}_2^k$ et $\mathbf{F}_2^n$). Si on note m un mot de taille k et m' son image par le codage, on a donc :

$$m' = mG,$$

où G est la transposée de la matrice de g de dimensions (n, k) et à coefficients dans $\mathbf{Z}/2\mathbf{Z}$.

Puisque $n > k$, l'image de g est un sous-espace vectoriel de $\mathbf{F}_2^n$.

Définition 3.8 — Code linéaire

On appelle codage linéaire le sous-espace vectoriel $\mathrm{Im}(g)$ que l'on note $C(k, n)$. En tant que sous-espace de dimension k, il est caractérisé par $n-k = r$ équations linéaires traduisant les dépendances entre les bits des blocs codés.

Définition 3.9 — Matrice génératrice

La matrice G est appelée matrice génératrice du code $C(k, n)$.

Exemple

Un exemple particulièrement simple est le bit de parité : on ajoute à la fin du message un bit correspondant à la parité de la somme des éléments du message. Si la somme est paire, alors le bit vaut 0 et 1 sinon. Pour un mot de longueur k, le code associé a donc pour paramètres $(k, k+1)$.

Définition 3.10 — Distance d'un code

On appelle distance d'un code (de correction linéaire par bloc) la plus petite distance entre deux mots de $C(k, n)$:

$$d_C = \min\{d(m_1, m_2);\ m_1 \in C, m_2 \in C, m_1 \neq m_2\}.$$

Proposition 8 : *Pour un code linéaire C, on a*

$$d_C = \min\{w(m);\ m \in C, m \neq 0\}.$$

Proposition 9 : *Soit C un code linéaire de paramètre (n, k, d_C), alors*

$$d_C \leq n - k + 1.$$

C'est la borne de Singleton.

Preuve : Il suffit de montrer qu'il existe par exemple un mot du code dont les $k-1$ dernière composantes sont nulles. Soit E le sous espace vectoriel de $\mathbf{F}_2^n$ dont les $k-1$ dernières composantes sont nulles, on a $\dim(E) = n-(k-1)$. Ainsi $\dim(C) + \dim(E) = n+1 > n$ et donc $C \cap E \neq \{\emptyset\}$.

3.2.3 Détection et correction d'erreur

On continue notre formalisation, en s'intéressant maintenant plus précisément à l'ensemble des mots du code, c'est-à-dire $C(k,n)$.

Notion de syndrome

Comme stipulé précédemment, la transmission à travers le canal peut donner lieu à une erreur affectant certains bits du mot. Dans le formalisme précédent, le mot m^* recueilli en sortie de canal sera donc de la forme :

$$m^* = m' + e,$$

où e est un vecteur de $\mathbf{F}_2^n$ comportant des 1 sur les composantes correspondant à celles affectées et des 0 ailleurs.

On introduit alors la capacité de correction e_C d'un code.

Définition 3.11 — Capacité de correction

La capacité de correction e_C d'un code est le plus grand entier tel qu'il soit toujours possible de corriger e_C erreurs ou moins.

Un mot erroné peut donc être corrigé s'il existe un unique mot du code le plus proche.

Ces relations peuvent s'écrire sous la forme matricielle :

$$m' \in C(k,n) \Leftrightarrow Hm' = 0,$$

où H est une matrice de dimensions (r,n).

Définition 3.12 — Matrice de contrôle

La matrice H est appelée *matrice de contrôle* ou matrice de test du code linéaire $C(k,n)$

On a $C(k,n) = \mathrm{Ker}(H)$ et la relation :

$$Hm^* = He,$$

puisque $m \in \mathrm{Ker}(H)$.

Définition 3.13 — Syndrome

Le vecteur $Hm^* = He$ est appelé *syndrome* de m^*.

Si une erreur s'est produite sur la composante i de m', alors le syndrome sera égal à la i-ème colonne de H.

Décodage par maximum de vraisemblance

On introduit encore quelques notions permettant cette fois-ci de décrire $C(k,n)$ et de faire le lien avec la correction d'erreur.

Le lemme suivant donne une méthode simple pour calculer la distance d'un code.

Lemme 3.3 La distance d'un code est égale au nombre de 1 contenus dans le mot non nul du code qui en contient le moins.

Preuve : Ce lemme est une conséquence directe de la formule (3.1). En effet, puisque $C(k,n)$ est un espace vectoriel, on a :

$$C(k,n) - \{0_{\mathbf{F}_2^n}\} = \{m_1 + m_2 / m_1 \neq m_2,\ m_1 \in C(k,n),\ m_2 \in C(k,n)\}.$$

Si la distance d'un code est 1, un mot avec un bit erroné peut également être un mot du code. On ne pourra donc pas à coup sûr détecter 1 erreur. Si la distance est 2, on détectera à coup sûr une erreur, mais on ne pourra pas la corriger dans tous les cas. En effet, il se peut que le mot entaché d'une erreur soit équidistant de deux mots distincts du code, ce qui fait qu'on ne pourra choisir qu'arbitrairement et sans garantie le mot du code qui lui correspondait avant l'erreur. Enfin, si la distance est 3, alors, par un raisonnement analogue, on pourra détecter 2 erreurs et corriger correctement 1 erreur. Tout ceci se généralise dans le lemme suivant dont on ne donnera pas la preuve ici.

Lemme 3.4 Un code par bloc de longueur n utilisant un décodage à distance minimale peut, pour toute paire d'entiers $t \in \{0,\dots,n\}$ et $s \in \{0,\dots,n-t\}$

- corriger les mots contenant t erreurs ou moins,
- détecter les mots contenant de $t+1$ à $t+s$ erreurs,

si et seulement si

$$d_C > 2t + s.$$

Pour illustrer ce résultat, considérons un code par bloc ayant une distance minimale de 8. Un tel code peut être utilisé pour l'un ou l'autre des points suivants :

- corriger tous les schémas d'erreur de moins que 3 (inclus) erreurs et détecter tous les schémas à 4 erreurs ($t = 3, s = 1$) ;
- corriger tous les schémas d'erreur de moins que 2 erreurs et détecter tous les schémas de 3 à 5 erreurs ($t = 2, s = 3$) ;
- corriger tous les schémas d'erreur de 1 erreur et détecter tous les schémas de 2 à 6 erreurs ($t = 1, s = 5$) ;
- détecter tous les schémas de moins que 7 (inclus) erreurs ($t = 0, s = 7$). Un code par bloc ayant une distance minimale de 7 peut (l'un ou l'autre)
- corriger tous les schémas d'erreur de moins que 3 (inclus) erreurs ($t = 3$, $s = 0$) ;
- corriger tous les schémas d'erreur de moins que 2 erreurs et détecter tous les schémas de 3 à 4 erreurs ($t = 2, s = 2$) ;
- corriger tous les schémas d'erreur de 1 erreur et détecter tous les schémas de 2 à 5 erreurs ($t = 1, s = 4$) ;
- détecter tous les schémas de moins que 6 (inclus) erreurs ($t = 0, s = 6$).

Introduisons maintenant les matrices génératrices sous forme systématique.

Définition 3.14 — Forme systématique

Une matrice génératrice G d'un code linéaire (k, n) est dite sous forme systématique si elle est de la forme

$$G = [I_k \; P],$$

où I_k est la matrice identité de taille k et P est une matrice $k \times (n-k)$ souvent appelée *matrice de parité*.

On peut montrer que lorsqu'elle existe pour un code donné, la matrice génératrice systématique est unique.

Exemple

Pour les messages binaires, le *bit de vérification de parité* est le bit qui correspond à la parité du message, c'est-à-dire à la somme (binaire) du message.

Le code par bit de parité consiste simplement à envoyer d'abord le message tel quel, suivi de son bit de parité. En termes de codes, ceci correspond au code binaire linéaire $(k, k+1)$ dont la matrice génératrice est

$$G = \begin{bmatrix} & 1 \\ I_k & \vdots \\ & 1 \end{bmatrix}$$

qui est sous forme systématique.

Théorème 3.2 Pour un code linéaire $C(k, n)$ dont la matrice génératrice est sous forme systématique est

$$G = [I_k \; P]$$

la matrice

$$H = [-P^\top \; I_{n-k}]$$

est une matrice de contrôle.

Preuve : Pour tout message $m \in \mathbf{F}_2^k$, le mot de code $z \in \mathbf{F}_2^n$ correspondant est

$$z = m \cdot G = m \cdot [I_k \; P],$$

c'est-à-dire

$$\begin{cases} (z_1, \dots, z_k) = m, \\ (z_{k+1}, \dots, z_n) = m \cdot P. \end{cases}$$

Ainsi

$$(z_{k+1}, \dots, z_n) = (z_1, \dots, z_k) \cdot P,$$

ce qui sous forme matricielle donne la matrice de contrôle H.

Réciproquement, on montre que tout code $z \in \mathbf{F}_2^n$ tel que $H \cdot z = 0$ vérifie

$$z = (z_1, \dots, z_k) \cdot G$$

et se trouve être un mot de code.

3.2.4 Codes de Hamming

Pour achever la description de la stratégie de correction, il faut spécifier les matrice G et H. C'est l'objet de cette section. On va prendre l'exemple de codes très répandus : les codes de Hamming.

L'idée est de disposer d'un code dont le syndrome indique directement la position de l'erreur, par exemple sous forme de son code binaire.

Choix des paramètres

On se place dans le cadre simple où au plus une erreur peut affecter les mots (de longueur n) du code. Il y a donc $n + 1$ scenari possibles : soit il n'y a pas eu d'erreur pendant la transmission, soit une erreur s'est produite sur l'un des n bits.

Or, d'après les dimensions des espaces considérés, 2^r syndromes possibles. Si l'on veut pouvoir faire correspondre de manière sûre, c'est-à-dire par le biais d'une injection, les vecteurs syndromes observés et le scénario dont il est la conséquence, il faut nécessairement :

$$2^r \geq n + 1.$$

Puisque l'on souhaite optimiser la taille des mots du code, on va chercher à choisir le n le plus petit possible. Le meilleur résultat est donc obtenu lorsque :

$$2^r = n + 1. \tag{3.2}$$

D'autre part, on a :

$$n = k + r. \tag{3.3}$$

Enfin $k \geq 1$, car on travaille sur des blocs de longueur non nulle ! En cherchant quels sont les k pour lesquels (3.2) et (3.3) ont des solutions (k, n) entières, on trouve par exemple :

$$k = 1,\ n = 3,\ r = 2,$$

et ensuite :

$$k = 4,\ n = 7,\ r = 3.$$

Le premier exemple correspond à la stratégie de triplement, décrite à la section 3.2.1. Le second correspond à un code largement utilisé, souvent appelé $C(4, 7)$, ce qui correspond aux notations utilisées dans cette section.

Avec de telles dimensions, la distance minimale de tels codes est dont toujours 3.

Définition 3.15 — Code de Hamming

Un code de Hamming est un code linéaire (k, n) binaire dont la matrice de vérification est

$$H_r = [b_r(1)^\top\ b_r(2)^\top \cdots b_r(n)^\top]$$

où $n = 2^r - 1$ et $k = 2^r - r - 1$, pour $r \geq 2$, et $b_r(i)$ est la représentation binaire de i sur r bits.

Par construction, ces matrices donnent bien un code linéaire et sont bien de rang plein puisqu'il est aisé de construire la matrice indentité I_{n-k} à partir des colonnes. La dimension du noyau est alors k.

Pour construire la matrice de génération du code, il suffit de mettre la matrice H sous forme systématique pour extraire la matrice P et construire la matrice G.

Exemple

Soit la matrice de syndrôme du code de HAMMING $C(4,7)$:

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Pour trouver la matrice génératrice, nous cherchons 4 vecteurs linéairement indépendant z_i tels que $Hz_i^\top = 0$, par exemple :

$$\begin{aligned} z_1 &= 1110000 \\ z_2 &= 1101001 \\ z_3 &= 1000011 \\ z_4 &= 1111111 \end{aligned}$$

ce qui donne

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Supposons maintenant devoir envoyer le message $m = 1001$. Il est codé par G en $z = 0001111$. Faisons maintenant l'hypothèse qu'une erreur soit survenue sur le troisième bit, et donc que $\hat{z} = 0011111$ soit reçu.

Le syndrome d'un tel code reçu sera alors $s = 011$, c'est-à-dire 3 en code binaire, ce qui indique que l'erreur est apparue sur le troisième bit. Le résultat de décodage avec correction d'erreur est alors

$$z = \hat{z} - 0010000.$$

Remarque 3.5 :

On peut montrer que pour un canal ayant une probabilité d'erreur de 5%, la probabilité de transmettre correctement 4 bits est $.95^4 \approx 81\%$.
L'utilisation de $C(4,7)$ permet de faire passer cette valeur à $(1-p)^7 + 7p(1-p)^6 \approx 95\%$. Ce résultat, comparable à la stratégie par triplement, est en fait bien meilleur, car la taille des mots du code n'a même pas doublé. Notons enfin qu'on peut rendre la probabilité d'erreur aussi faible que l'on veut en appliquant un code correcteur plusieurs fois de suite.

3.3 Notes bibliographiques

Ce chapitre s'inspire de notes de cours gracieusement fournies par Yvan Pigeonnat. Pour plus d'informations sur le codage source, on pourra consulter les références [8], [4]. Pour plus d'informations sur les codes correcteurs (de type HAMMING), on pourra également consulter [4], mais bien d'autres documents relatifs à la correction d'erreur se trouvent facilement sur Internet. Enfin, pour des références très complète sur le codage et pour découvrir des codages autres que le codage de HAMMING (il en existe énormément), on pourra consulter [9, 7] et [1].