Théorie de l'information et codage **2010/2011**

Cours 3 — 1er mars

Enseignant: Marc Lelarge *Scribe: Guilhem Gamard*

Pour information

– `http://www.di.ens.fr/~lelarge/info11.html`

3.1 Codage de source universel

On cherche maintenant à trouver un codage pour une une suite quelconque $u^{(n)}$ sans faire aucune hypothèse probabiliste. Nous commençons par voir un exemple simple d'un tel codage pour une suite binaire, avant d'étudier l'algorithme de Lempel-Ziv.

3.1.1 Codage universel pour une suite binaire

Pour coder une suite binaire $u^{(n)} \in \{0,1\}^n$, on étudie l'algorithme offline [1] suivant :
- envoyer le nombre de 1 dans la suite : $\sum_{i=0}^{n} u_i$, en $\lceil \log_2(n+1) \rceil$ bits ;
- envoyer l'indice de la suite parmi toutes les suites ayant k 1, en $\lceil \log \binom{n}{k} \rceil$ bits.

Il faut donc au total :

$$\ell(u^{(n)}) \leq \log(n+1) + \log \binom{n}{k} + 2$$

On relie cette valeur à l'entropie grâce au lemme suivant.

Lemme 3.1.1 *Pour $k \neq 0, n$, on a*

$$\sqrt{\frac{n}{8k(n-k)}} \leq \binom{n}{k} 2^{-nH(\frac{k}{n})} \leq \sqrt{\frac{n}{\pi k(n-k)}}$$

Démonstration. On rappelle la formule de Stirling :

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \exp\left(\frac{1}{12n}\right)$$

On note alors $k = np$ et $q = 1-p$. On rappelle la notation $H(p) = -p\log(p) - q\log(q)$, de telle sorte que $2^{-nH(p)} = p^{np}q^{nq}$.

[1] un algorithme qui doit lire l'ensemble du message avant de pouvoir commencer le codage

On a alors,

$$\begin{aligned}\binom{n}{k} = \binom{n}{np} &\leq \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \exp\left(\frac{1}{12n}\right)}{\sqrt{2\pi np}\left(\frac{np}{e}\right)^{np}\sqrt{2\pi nq}\left(\frac{nq}{e}\right)^{nq}} \\ &= \frac{1}{\sqrt{2\pi npq}} \times \frac{1}{p^{np}q^{nq}} \exp\left(\frac{1}{12n}\right) \\ &< \frac{1}{\sqrt{\pi np}} 2^{nH(p)},\end{aligned}$$

Car $\exp\left(\frac{1}{12n}\right) < \sqrt{2}$.

De même pour la borne inférieure, on a

$$\begin{aligned}\binom{n}{np} &\geq \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{\sqrt{2\pi np}\left(\frac{np}{e}\right)^{np}\sqrt{2\pi nq}\left(\frac{nq}{e}\right)^{nq}} \exp\left(-\frac{1}{12np} - \frac{1}{12nq}\right) \\ &= \frac{1}{\sqrt{2\pi npq}} 2^{nH(p)} \exp\left(-\frac{1}{12np} - \frac{1}{12nq}\right).\end{aligned}$$

On distingue alors plusieurs cas :

- si $np \geq 1$ et $nq \geq 3$, alors $\exp(-(\frac{1}{12np} + \frac{1}{12nq})) \geq \exp(-\frac{1}{9}) > \frac{\sqrt{\pi}}{2}$,
- si $np = 1$ et $nq = 1$ alors $n = 2$ et $p = \frac{1}{2}$, la borne vaut 2 et est correcte.
- si $np = 1$ et $nq = 2$ alors $n = 3$ et $p = \frac{1}{3}$, la borne vaut 2.92 et est correcte.
- si $np = 2$ et $nq = 2$ alors $n = 4$ et $p = \frac{1}{4}$, la borne vaut 5.66 et est correcte.

□

On a donc

$$\begin{aligned}\ell(u^{(n)}) &\leq \log(n+1) + nH\left(\frac{k}{n}\right) - \frac{1}{2}\log(n) - \frac{1}{2}\log(\pi\frac{k}{n}\frac{n-k}{n}) + 2 \\ &\leq nH\left(\frac{k}{n}\right) + \frac{1}{2}\log(n) - \frac{1}{2}\log(\pi\frac{k}{n}\frac{n-k}{n}) + 3\end{aligned}$$

Donc le coût pour décrire cette suite est de $\approx \frac{1}{2}\log(n)$ bits en plus du coût optimal de $nH\left(\frac{k}{n}\right)$ pour une distribution de Bernoulli correspondant à $p = \frac{k}{n}$.

3.1.2 Codage par automates à états finis

Nous allons dans un premier temps étudier la compression d'une suite infinie u par des automates finis. Nous établirons alors une borne sur le taux de compression de tels algorithmes, avant de montrer que l'algorithme de Lempel-Ziv atteint cette borne.

Dans toute la suite, chaque symbole de la source appartient à un alphabet fini ayant J symboles avec $J \geq 2$.

Définition 3.1.1 *Un automate à espace d'états fini est composé d'une tête de lecture se trouvant dans un certain état (parmi un ensemble fini). L'automate lit l'entrée symbole par symbole, chacun d'entre eux entraînant un changement d'état et l'émission d'un mot, éventuellement vide. Les changements sont régis par une table de transitions qui est une caractéristique de l'automate.*

Pour l'entrée $u = u_1u_2u_3\ldots$, l'automate produit $y = y_1y_2y_3,\ldots$ en visitant les états $z = z_1z_2z_3\ldots$ donnés par :
- $y_k = f(z_k, u_k)$ à valeurs $\{0,1\}^*$, pour $k \geq 1$,
- $z_{k+1} = g(z_k, u_k)$ à valeurs dans l'espace d'états (fini), pour $k \geq 1$.

Les fonctions f et g correspondent à une consultation de la table des transitions.

On notera $u_k^j = u_k u_{k+1}\ldots u_j$ et $f(z_k, u_k^j) = y_k^j$, $g(z_k, u_k^j) = z_{j+1}$.

Le décodeur a connaissance de l'automate et de son état initial. Il doit être capable de reconstruire u à partir de y.

Définition 3.1.2 *On dit qu'un encodeur est sans perte d'information, ou SPI, si $u_r^s \neq v_r^t$ alors pour tout z_r on a :*
- *soit $f(z_r, u_r^s) \neq f(z_r, v_r^t)$,*
- *soit $g(z_r, u_r^s) \neq g(z_r, v_r^t)$.*

Pour un encodeur n'étant pas SPI, il est impossible de retrouver u à partir de y. Notons cependant qu'un encodeur SPI n'est pas nécessairement "uniquement décodable", comme le montre l'exemple 3.1.2.

Nous calculons maintenant une borne inférieure sur le nombre de bits utilisés par symbole d'entrée pour tout encodeur SPI. Cette borne s'appliquera également aux encodeurs SPI conçus en connaissant U à l'avance, comme le fait l'algorithme de Huffman.

Définition 3.1.3 *Pour un encodeur E, son ratio de compression pour u est défini par*

$$\rho_E(u_1^n) = \frac{1}{n}\ell\left(y_1^n\right).$$

On définit alors :

$$\rho_s(u_1^n) = \min\{\rho_E(u_1^n), E \text{ encodeur SPI à } s \text{ états}\}$$

On note que $\rho_s(u_1^n) \leq \lceil \log J \rceil$.

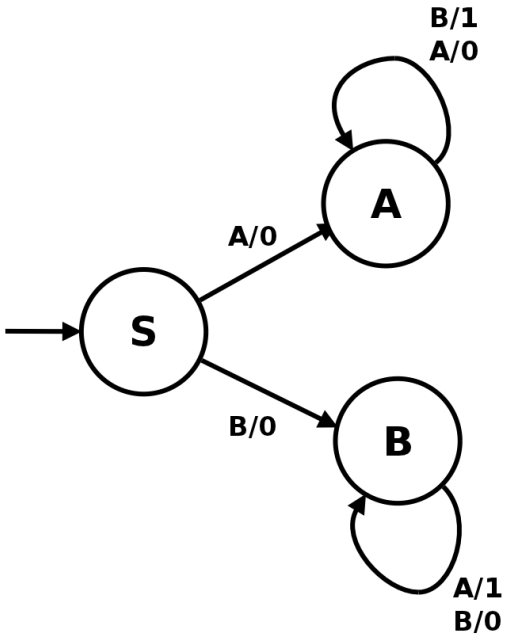

Fig. 3.1. Automate d'encodage. Les suites infinies $AAAA\ldots$ et $BBBB\ldots$ sont codées par $0000\ldots$ et donc indistinguables pourtant l'automate est SPI.

On définit alors la compressibilité de u, notée $\rho(u)$ par :

$$\begin{aligned}\rho_s(u) &= \limsup_{n\to\infty} \rho_s(u_1^n)\\ \rho(u) &= \lim_{s\to\infty} \downarrow \rho_s(u).\end{aligned}$$

Soit $c(u_1^n)$ le nombre maximum de mots distincts en lesquels u_1^n peut être découpé (le mot vide ε inclus). On a donc $c(u_1^n) \geq 1$.

Si u_1^n est découpé en $c \geq 1$ mots distincts, on définit m et $0 \leq r \leq J^m$ tels que

$$c = \sum_{k=0}^{m-1} J^k + r.$$

Si un tel c est donné, le n minimal est obtenu par $ngeq \sum_{k=0}^{m-1} kJ^k + mr$ car il y a J^k mots de longueur k.

Pour $J \geq 2$, on a

$$\sum_{k=0}^{m-1} J^k = \frac{J^m - 1}{J - 1}$$

$$\sum_{k=0}^{m-1} kJ^k = m\frac{J^m}{J-1} - \frac{J}{J-1}\frac{J^m - 1}{J-1}$$

On obtient donc :

$$\begin{aligned} n &\geq m(c - r + \frac{1}{J-1}) - \frac{J}{J-1}(c-r) + mr \\ &\geq (m-2)\left(c + \frac{1}{J-1}\right) - \frac{J}{J-1}c \\ &\geq (m-2)c \end{aligned}$$

De plus, on a $c < \frac{J^{m+1}-1}{J-1}$ donc $c < c(J-1) + 1 < J^{m+1}$ et $m + 1 \geq \log_J(c)$. Au final on obtient :

$$n > c \log_J \left(\frac{c}{J^3}\right). \tag{3.1}$$

Théorème 3.1.1 *Pour tout encodeur SPI à s états,*

$$\ell(y_1^n) \geq c(u_1^n) \log_2 \left(\frac{c(u_1^n)}{8s^2}\right).$$

Démonstration.

On a $u_1^1 = w_1 \dots w_c$ où $c = c(u_1^n)$ mots différents. On pose c_{ij} = le nombre de mots qui trouvent l'encodeur dans l'état i et le laissent dans l'état j. Comme l'encodeur est SPI, les sorties correspondantes sont nécessairement différentes et leur longueur totale ℓ_{ij} doit satisfaire (3.1) avec $J = 2$ puisque y est une suite binaire, donc :

$$\ell_{ij} \geq c_{ij} \log_2 \left(\frac{c_{ij}}{8}\right).$$

Donc $\ell(y_1^n) = \sum_{1 \leq i,j \leq s} c_{ij} \log_2 \left(\frac{c_{ij}}{8}\right)$. Comme $\sum c_{ij} = c(u_1^n)$ et que le minimum du terme de droite sous cette contrainte est atteint à $c_{ij} = \frac{c(u_1^n)}{s^2}$ (fonction convexe symétrique), on obtient le résultat voulu. □

Le lemme suivant est montré en exercie.

Lemme 3.1.2

$$c(u_1^n) = O(\frac{n}{\log(n)})$$

D'après le théorème,

$$\begin{aligned} \rho_s(u) &\geq \limsup \frac{1}{n} c(u_1^n) \log_2(\frac{c(u_1^n))}{8s^2}) \\ &= \limsup_{n\to\infty} \frac{1}{n} c(u_1^n) \log_2(c(u_1^n)) - \limsup_{n\to\infty} \frac{1}{n} c(u_1^n) \log(8s^2) \\ &= \limsup_{n\to\infty} \frac{1}{n} c(u_1^n) \log_2(c(u_1^n)). \end{aligned}$$

Comme cette dernière expression ne dépend pas de s, on a :

$$\rho(u) \geq \limsup_{n\to\infty} \frac{1}{n} c(u_1^n) \log_2(c(u_1^n)). \tag{3.2}$$

3.1.3 Algorithme de Lempel-Ziv

Nous décrivons maintenant l'algorithme de Lempel-Ziv. Le fonctionnement est le suivant :
- initialiser un dictionnaire avec tous les mots de longueur 1 ;
- attribuer à chaque mot un codage en binaire, par ordre lexicographique ; si le dictionnaire a D mots, la longueur des mot-code est $\lceil \log_2 D \rceil$.
- chaque fois qu'un mot de longueur m appartenant au dictionnaire est lu en entrée,
  - émettre le mot-code correspondant,
  - remplacer dans le dictionnaire le mot par l'ensemble des extensions d'une lettre du mot (c'est à dire les mots de longueur $m+1$ ayant comme préfixe le mot lu en entrée).

Un exemple permet de mieux saisir le comportement de l'algorithme. Soit *aaaccb* la chaîne à compresser. La figure 3.1.3 donne alors les différents états du dictionnaire en lisant de gauche à droite. La chaîne émise est 00 000 110 0111.

Le décodage s'effectue de façon parfaitement symétrique. Étant donné que la mise à jour du dictionnaire est faite après l'émission du mot-code, le décodeur peut faire la même mise à jour une fois le mot-code reçu, et donc maintenir le même dictionnaire tout au long de la décompression.

On note qu'il demeure un problème lors de la fin du codage : le dernier mot lu ne correspond pas nécessairement à une feuille de l'arbre. Diverses solutions existent pour pallier à ceci, par exemple en choisisant de numéroter tous les nœuds de l'arbre plutôt que les feuilles.

Voyons maintenant dans quelle mesure le codage de Lempel-Ziv est efficace. Si l'algorithme découpe u_1^n en $c_{LZ}(u_1^n)$ mots $w_1, w_2, \ldots w_{c_{LZ}}$, alors $u_1^n = \varepsilon w_1, w_2, \ldots w_{c_{LZ}}$ et les $c_{LZ}(u_1^n)-1$ premiers mots sont différents. Si l'on concatène les deux derniers mots, on obtient un découpage en $c_{LZ}(u_1^n)$ mots différents. On a alors $c_{LZ}(u_1^n) \leq c(u_1^n)$.

La taille du dictionaire à la fin du découpage de u_1^n est $J + (c_{LZ}(u_1^n) - 1)(J-1)$, et le nombre de nœuds dans l'arbre $c_{lz}(u_1^n)J$.

Même si on attribue à chaque noeud de l'arbre un mot-code, le nombre total de digits binaires envoyés sera :

$$\begin{aligned} \ell_{LZ}(y_1^n) &\leq c_{LZ}(u_1^n)\lceil \log_2(Jc_{LZ}(u_1^n))\rceil \\ &\leq c(u_1^n) \log_2(2Jc(u_1^n) \end{aligned}$$

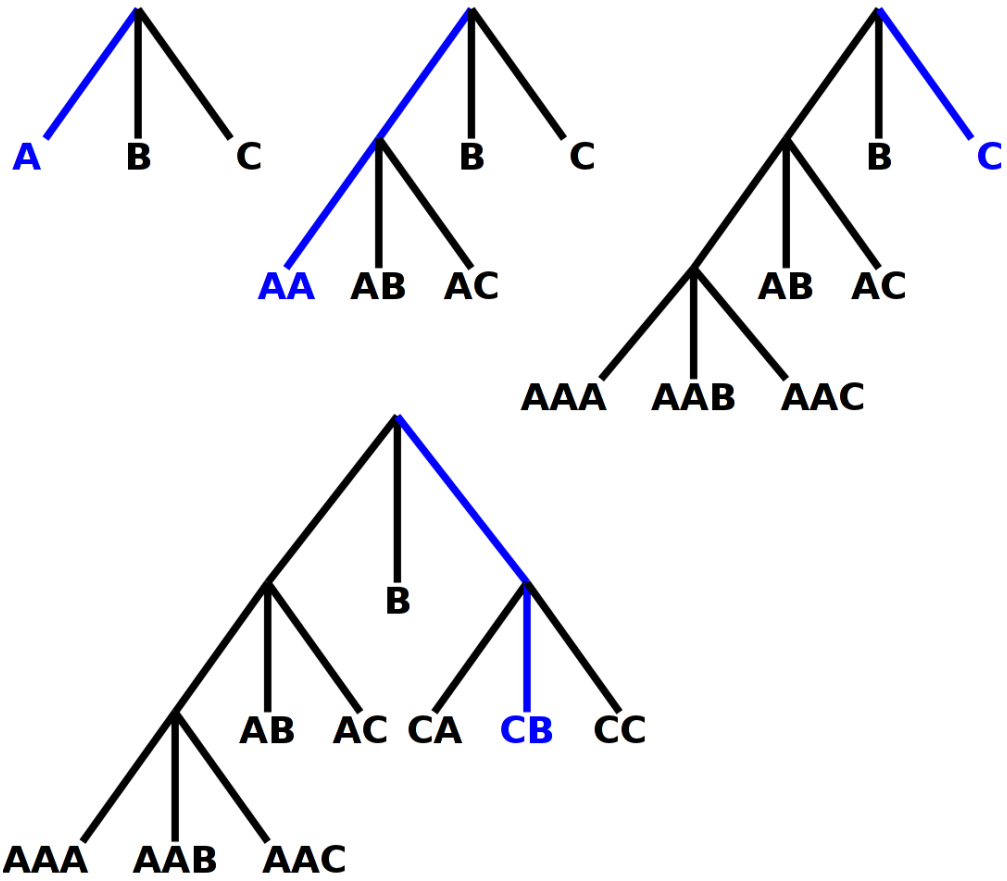

Fig. 3.2. Évolutions successives du dictionnaire

D'où

$$\limsup_{n\to\infty} \frac{1}{n}\ell_{LZ}(y_1^n) \leq \limsup_{n\to\infty} \frac{c(u_1^n)}{n} \log_2(c(u_1^n)) \leq \rho(u),$$

où la dernière inégalité vient de (3.2).

L'algorithme de Lempel-Ziv est donc au moins aussi bon que n'importe quel encodage SPI par automates finis. Noter cependant que l'algorithme de Lempel-Ziv n'est pas un encodage par automate fini. Par contre, le codage de Huffman par blocs fait partie de cette dernière classe et donc l'algorithme de Lempel-Ziv "fait donc aussi bien" que le codage de Huffman par blocs pour toute longueur de bloc.

Dans le cas d'une source sans mémoire, une démonstration rigoureuse de l'optimalité de l'algorithme de Lempel-Ziv est donnée en exercie.