Compression

Guillaume TOCHON

17 mars 2018

Table des matières

1 Compression de données

Site des slides : www.lrde.epita.fr/~gtochon/CODO/
Ces notes sont à travailler en parallèle avec les slides.
Compression + décompression
Sans compression de données, un film 720p d'une heure ferait presque 400Gio.
La naissance naïve de la compression de données remonte aux environs du code morse. Shannon est responsable des fondements mathématiques.
Autres acteurs : - Abraham Lempel - David Huffmann - Terry Welch

Le but de la compression est que lors de la décompression, ce soit le moins visible possible par l'utilisateur.

1.1 Sur la compressabilité

1.1.1 Entropie

On va chercher a eliminer les redondances, et les gaspillages. L'outil que l'on va utiliser est l'entropie.
Du point de vue de Shannon, plus un message est probable, moins il contient d'informations.

Prenons un alphabet Σ de N symboles :

$$\Sigma = \{s_1, s_2, ..., s_N\}$$

Avec leurs probabilités d'occurrence respectives :

$$p_i = p(s_i)$$

Évidemment, on a

$$\sum_{i=1}^{N} p_i = 1$$

Soit F un fichier composé de N_F éléments de Σ.
Statistiquement, s_i est présent $p_i \times N_F$ fois.
On définit q_i la quantité d'information totale d'un symbole :

$$q(s_i) = -log_2(p_i)$$

On a donc Q_{Tot} la quantité d'information propre totale de s_i contenue dans F, d'unité Sh/symbole, abrégée Sh/symb :

$$Q_{Tot}(s_i) = -N_F \cdot p_i \cdot \log_2(p_i) \text{ Sh/symb}$$

On définit donc $Q_{Tot}(F)$ la quantité d'information contenue dans un fichier F, d'unité Sh :

$$\begin{aligned} Q_{Tot}(F) &= \sum_{i=1}^{N_F} (Q_{Tot}(s_i) \\ &= \sum_{i=1}^{N_F} -N_F \cdot p_i \cdot \log_2(p_i) \\ &= -N_F \sum_{i=1}^{N_F} p_i \cdot \log_2(p_i) \text{ Sh} \end{aligned}$$

Or ce $-N_F$ devant la somme pose problème : **la quantité d'information dépend de la taille du fichier**.
Cela implique qu'un gros fichier "probable" contient moins d'information qu'un petit fichier improbable.

On définit donc l'entropie H ainsi :

$$H = \frac{Q_{Tot}(F)}{N_F} = -\sum_{i=1}^{N_F} p_i \cdot \log_2(p_i)$$

On notera en outre que :

$$H \equiv Sh$$

On remarquera que l'entropie H est **indépendante** de la taille du fichier.
H ne dépend donc que de l'alphabet considéré Σ et de la distribution des symboles qui compose le fichier F.

1.1.2 Probabilités

Soit X une variable aléatoire telle que :

$$X = \{x_1, x_2..., x_n\}$$

Et

$$p_i = P(X = x_i)$$

On a donc l'espérance de X :

$$E(X) = \sum_{i=1}^{n} x_i \cdot p_i = \sum_{i=1}^{n} x_i \cdot P(X = x_i)$$

Et on a donc :

$$E(\varphi(X)) = \sum_{i=1}^{n} \varphi(x_i) P(X = x_i)$$

On a donc, pour l'entropie :

$$\begin{aligned} H &= -\sum_{i=1}^{N_F} p_i \cdot \log_2(p_i) \\ &= \sum_{i=1}^{N_F} \underbrace{(-\log_2(p_i))}_{q_i} \cdot p_i \\ &= \sum_{i=1}^{N_F} q_i \cdot p_i \\ &= E(q(s_i)) \end{aligned}$$

1.1.3 Application à l'informatique

On considère donc :

$$\begin{aligned} N_\Sigma &= \{0, 1\} \\ p(0) &= p_0 = p \\ p(1) &= p_1 = 1 - p \end{aligned}$$

On a alors :

$$\begin{aligned} H &= -p \cdot \log_2(p) - (1-p) \log_2(1-p) \\ &= H(p) \end{aligned}$$

Par convention, on pose :

$$H = -\sum_{i=1}^{N_F} p_i \cdot \log_2(p_i)$$

$$p_i \cdot \log_2(p_i) = 0 \quad \text{si} \quad p_i = 0$$

1.1.4 Cas de l'entropie nulle (À corriger avec le prof)

$H(0) = H(1) = 0$
On a des fichiers complètement désordonnés :

$$p = 0 \implies F = \{1111...1\}$$

$$p = 1 \implies F = \{0000...0\}$$

Donc l'entropie est nulle.

Quand $p = \frac{1}{2}$, les symboles sont équiprobables, le fichier est complètement désordonné, donc complètement aléatoire. Dans ce cas, l'entropie est donc **maximale**.

Voir la figure 1.

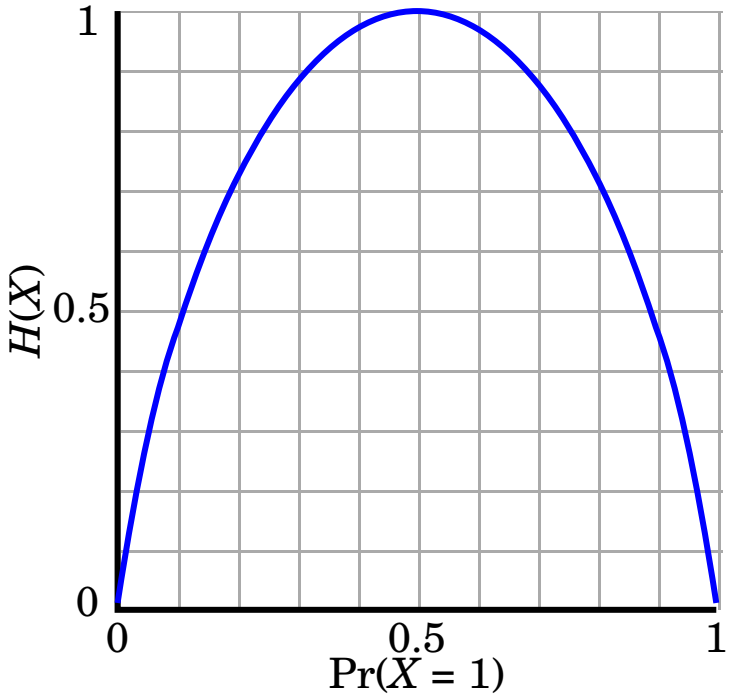

FIGURE 1 – Entropie en fonction de $P(X = 1)$

Attention ! L'entropie ne voit pas les méta-symboles ! Par exemple, ce fichier est considéré comme parfaitement alétaoire :

$$F = \{01010101\}$$

1.1.5 Calcul de l'entropie

Soit l'alphabet

$$\Sigma = \{s_1, s_2, ..., s_N\}$$

De longeur :

$$\forall n \in \mathbb{N}, \quad N_\Sigma = 2^n$$

Avec :

$$\forall s_i \in [\![1, N_\Sigma]\!], \quad p(s_i) = p_i = \frac{1}{N_\Sigma}$$

On a donc :

$$
\begin{aligned}
H &= -\sum_{i=1}^{N_\Sigma} p_i \cdot \log_2(p_i) \\
&= -\sum_{i=1}^{2^n} \frac{1}{2^n} \cdot \underbrace{\log_2\left(\frac{1}{2^n}\right)}_{-n} \\
&= \frac{1}{2^n}(-n) \underbrace{\sum_{i=1}^{2^n} 1}_{2^n} \\
&= n
\end{aligned}
$$

1.1.6 Exercice

VÉRIFIER AVEC LE PROF !

De combien de bits a-t-on besoin pour encoder le fichier suivant ?

$$F = \{ACABBDDBAAABCAAA\}$$

Dans ce cas, on utilisera l'alphabet suivant :

$$\Sigma = \{A, B, C, D\}$$

Comptons le nombre d'occurrences de chacune des lettres de Σ :

$$
\left.
\begin{array}{ll}
A: & 8 \\
B: & 4 \\
C: & 2 \\
D: & 2
\end{array}
\right\} \text{ Longeur : } 16
$$

1. Non compressé ?
   C'est tout simplement, en notant Σ l'alphabet utilisé pour encoder le fichier,

$$
\begin{aligned}
\text{Résultat} &= \text{Longeur} \times \text{Nombre de bits nécessaire pour encoder } \Sigma \\
&= 16 \cdot \log_2(n_\Sigma) \\
&= 16 \cdot \log_2(4) \\
&= 32 \text{ bits}
\end{aligned}
$$

2. Compressé ?
   Calculons l'entropie du fichier :

$$
\begin{aligned}
H &= p_A \cdot \log_2(p_A) + p_B \cdot \log_2(p_B) + p_C \cdot \log_2(p_C) + p_D \cdot \log_2(p_D) \\
&= \frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) + \frac{1}{4} \cdot \log_2\left(\frac{1}{4}\right) + \frac{1}{8} \cdot \log_2\left(\frac{1}{8}\right) + \frac{1}{8} \cdot \log_2\left(\frac{1}{8}\right) \\
&= \frac{7}{4}
\end{aligned}
$$

   Donc la longeur totale minimale du message est :

$$
\begin{aligned}
\text{Résultat} &= \text{Longeur} \times H \\
&= 16 \cdot \frac{7}{4} \\
&= 28 \text{ bits}
\end{aligned}
$$

2 Algorithmes de compression conservatifs

2.1 Run-length encoding

Pas génial pour le texte.
Mieux pour les images de synthèse. C'est utilisé dans bitmap.
Le principe c'est de coder combien de fois on voit chaque caractère.
Pour les nombres, caractère spécial.

2.1.1 Cas du fax

C'est utilisé pour le fax et ça marchait bien car il y a beaucoup de blanc.
On peut même compresser la différence entre deux lignes consécutives pour gagner de la place.
C'est une technique très largement utilisée, par exemple dans JPEG et MPEG.

2.2 Algorithme d'Huffman

Un des algorithmes les plus célèbres.
Code entropique : il est basé sur la distribution statistique des symboles.

Deux étapes :

1. Construction d'un arbre binaire donc les feuilles sont les symboles.
2. Encodage du message avec cet arbre.

Première étape :

1. Trier la table
2. Fusionner les deux symboles

Seconde étape :

1. Parcourir l'arbre de la racine vers les veuiulles (les symboles)
2. Toujours assigner 1 au fils gauche et 1 au fils droit (ou l'inverse)
3. Lire l'encodage sur la branche entière.

C'est un encodage à préfixe donc on peut le décoder à la volée. Il est optimal à la longeur d'encodage moyenne parmi les encodages à préfixe. Il nous donne exactement l'entropie tant que que les symboles sont une puissance de 2.

Problème : c'est de l'encodage symbole par symbole.

De nos jours, on ne s'en sert pas tel quel, mais en conjonction avec des algorithmes plus complexes.

2.3 L'algorithme bzip2

Libre et open-source, créé entre 1996 et 2000. Grosse usine à gaz.
L'idée :

1. Appliquer la transformée de Burrows-Wheeler. Cela change l'ordre des caractères pour former des suites de caractères identiques, tout en restant réversible à la décompression.

   Encodage :
   — Contruire une matrice contenant toutes les rotations de la string à transformer.
   — Trier les colonnes selon l'ordre alphabétique.
   — Stocker la rotation de la matrice et ces colonnes.

Décodage :

— Concaténer les chaîne avec la dernière colonne de la table.

— À céompléter

2. La transformée “Move to front”
3. Algorithme d’Huffman.

Problème : lent à la compression, mais très économe en place. Très efficace pour langage naturel, pas trop pour du code.

2.4 Algorithme LZW

On a intérêt à le refaire sur papier pour bien le comprendre.
C’est le premier algorithme, avec LZ78, qui fasse de l’aprentissage de dictionnaire non-supervisé.
Voir le processus sur les diapos.
C’est utilisé dans GIF.

3 Old

$$Q_{Tot}(S_i) = -N_{FP_i} Sh$$
$$Q_Tot(F) = Q_{Tot}(S_1) + Q_{Tot}(S_2) + ... + Q_{Tot}(SN_2)$$
$$Q_{Tot}(S_1) = -N_F log_2(P_i)$$
$$Q_{Tot}(S_2) = -N_{FP_2} log_2(P_2)$$
$$Q_{Tot}(SN_2) = -N_F P_N log_2(P_N)$$
$$Q_{Tot}(F) = \sum -N_{FP_i} = -N_F \sum P_i log_2(P_i) Sh$$

Un gros fichier "probable" contient plus d’info qu’un petit fichier "improbable" $Q_{Tot} \rightarrow$ Sh
$\rightarrow$ H $\rightarrow$ Sh/Symbole $H = -\sum P_i log_2(P_i) \rightarrow$ ne depend pas du fichier considere, mais uniquement de la distribution de proba des symboles composant le fichier.
X variable aleatoire de valeur $\{x_1, x_2 ... x_n \rightarrow P_i = P(X = x) \sum P_i = 1$
$E[X] = \sum_{i=1}^{N} x_1 P(X = x_i) = \sum_{i=1}^{N} x_i P_i \; E[\phi(X)] = \sum_{i=1}^{N} \phi(x_i) P_i \; H = -\sum_{i=1}^{N} P_i log_2 P_i = \sum_{i=1}^{N} (-log_2(Pi)) P_i = \sum_{i=1}^{N} q_{S_i * P_i} H = E[q(s_i)] \sum = 0,1 P(0) = P_0 = P P(1) = P_1 = 1 - P \; H = -p log_2(p) - (1-p) log_2(1-p) = H(p)$
— Missing things —
$\sum$ avec$N$symboles $s_i, i = 1, ..N \; (2^n)$

$$P(S_i) = \frac{1}{N_{\sum}} = \frac{1}{2^n}$$

$$H = -\sum_{i=1}^{N \sum} log_2(P_i) = -\sum_{i=1}^{2^n} = -\sum_{i=1}^{2^n} \frac{1}{2^n} log_2(\frac{1}{2_n})) = \frac{1}{2_n} * (-n) * \sum_{i=1}^{2^n} * \sum_{i=1}^{2^n} 1 = n 2^n 2^{-n} = n$$

(Sh/Symbole)