Nicolas Sendrier

Majeure d'informatique

Introduction à la théorie de l'information

Cours n°5

Codage de source universel

Codage de source universel

Lorsque qu'un algorithme ne supposera rien sur type de fichier à compresser, c'est-à-dire sur les statistiques de la source, on parlera de codage de source universel.

Nous allons examiner en détail deux algorithmes,
– l'algorithme de Huffman adaptatif,
– l'algorithme de Lempel-Ziv.
Dans les deux cas, il s'agit de construire à chaque instant un modèle dynamique de la source ayant pu produire le texte jusqu'à ce point et de coder la lettre suivante dans ce modèle. L'algorithme de Huffman adaptatif construit un modèle sans mémoire, alors que l'algorithme de Lempel-Ziv prend en compte les dépendances entre les lettres.

Huffman adaptatif – Principe

Supposons que nous ayons déja lu N caractères dans le texte, correspondant à K lettres distinctes.

- nous fabriquons une source de $K+1$ lettres formée de K lettres déjà apparues auxquelles on attribue une probabilité proportionnelle au nombre d'occurences, et d'une $(K+1)$-ème lettre « fourre-tout » (vide) à laquelle on attribue la probabilité 0,
- on construit le code de Huffman de cette source,
- la $N+1$-ème lettre est lue et est codée
  - par son mot de code s'il existe,
  - par le $K+1$-ème mot de code suivi du code ascii sinon.

Huffman adaptatif – Structure de donnée

En pratique nous n'allons pas construire un code de Huffman à chaque fois, mais plutôt essayer de le mettre à jour au fur et à mesure.

Nous allons construire et maintenir un arbre binaire à $K+1$ feuilles, dont chacun des $2K+1$ noeuds $x_i, i = 0 \ldots 2K$, est muni d'un poids $p(x_i)$, et qui vérifie

- la suite des poids $(p(x_i))_i$ est décroissante,
- pour tout $i > 0$, le noeuds x_{2i-1} et x_{2i} sont frères,
- chaque feuille possède comme attribut une lettre et son poids est le nombre d'occurrences de la lettre dans le texte,
- le poids d'un noeud interne est la somme des poids de ses fils.

Huffman adaptatif – Codage

L'arbre initial est constitué d'une unique feuille, celle de la lettre vide.
À chaque fois qu'un caractère x est lu dans le texte source
- s'il est déjà apparu
  - on imprime son mot de code,
  - on met à jour l'arbre,
- sinon
  - on imprime le mot de code de la lettre vide suivi du code ascii (en base 2) de x,
  - on ajoute une feuille dans l'arbre,
  - on met à jour l'arbre.

Huffman adaptatif – Décodage

L'arbre initial est constitué d'une unique feuille, celle de la lettre vide.
On répète à partir de la racine jusqu'à épuisement :

- On parcourt l'arbre en lisant les bits du texte codé ('0' à gauche, '1' à droite) jusqu'à arriver à une feuille
  - s'il s'agit d'une lettre non vide
    - on imprime la lettre,
    - on met à jour l'arbre,
  - sinon, il s'agit de la lettre vide
    - on lit les 8 bits suivants pour obtenir le code ascii d'une lettre que l'on imprime
    - on ajoute une feuille dans l'arbre,
    - on met à jour l'arbre.

Huffman Adaptatif – Optimalité (1)

Soit T un texte contenant des lettres dans un alphabet $A = \{a_1, \ldots, a_K\}$. Pour tout i, nous notons F_i le nombre d'occurrences de a_i dans T et $p_i = F_i/|T|$. L'entropie par symbole du texte T est défini de la façon suivante

$$H_1(T) = H(T) = \sum_{i=1}^{K} p_i \log_2 \frac{1}{p_i}.$$

Pour tout codage du texte T nous pouvons également définir une longueur moyenne par lettre $n(T)$ en divisant la taille totale (en bits) du texte codé par le nombre $|T|$ de lettre dans le texte. Tout codage du texte T codant les lettre une par une indépendamment vérifie

$$n(T) \geq H(T).$$

Huffman Adaptatif – Optimalité (2)

L'algorithme de Huffman adaptatif fournit un codage *localement optimal* du texte. En effet, à tout instant, l'arbre de Huffman adaptatif est (presque) un code de Huffman pour les statistiques mesurées. Seul la lettre la moins fréquente est codée par un mot trop long d'une unité, soit une perte de $\min_{1 \leq i < K} p_i$ pour la longueur moyenne par lettre.

Si l'on suppose que le texte est produit par une source X sans mémoire. On peut montrer à l'aide de la loi des grands nombres que l'efficacité du codage de Huffman adaptatif tends vers 1 lorsque la taille du texte tend vers l'infini.

Preuve (ébauche) : Pour $|T|$ suffisamment grand, p_i est une bonne approximation de $P_X(a_i)$ avec une grande probabilité. On en déduit que $H(T)$ est une bonne approximation de $H(X)$.

Lempel-Ziv 78 – Principe

L'algorithme de Lempel-Ziv (1978) lit un texte constitué de symboles d'un alphabet $\mathcal{A}$. Supposons que nous ayons déja lu N symboles et que nous ayons formé un dictionnaire des mots déjà lus.

- À partir du $(N+1)$-ème symbole on lit les symboles un par un jusqu'à obtenir un mot (de longueur n) absent du dictionnaire, on affiche l'indice du dernier mot reconnu dans le dictionnaire (de longueur $n-1$) ainsi que le dernier symbole lu.
- On ajoute ce mot au dictionnaire et on recommence à partir du $(N+n+1)$-ème symbole.

Lempel-Ziv – Structure de donnée

Il nous faut une façon de représenter efficacement le dictionnaire. Celui-ci possède une propriété intéressante : si un mot est dans le dictionnaire, c'est aussi le cas de tous ses préfixes.

On en déduit que les dictionnaires que nous voulons représenter sont exactement les arbres q-aires. Une telle représentation nous donne une implémentation simple et efficace des fonctionnalités nécessaires, à savoir :

– tester la présence d'un mot,

– ajouter un mot

Lempel-Ziv – Codage

Le dictionnaire (i.e. l'arbre) contient initialement le seul mot vide et sa taille est $K = 1$. On répète à partir de la racine jusqu'à épuisement :

– on parcourt l'arbre en lisant les lettres du texte (la i-ème lettre de $\mathcal{A}$ correspondant à la i-ème branche) tant que c'est possible.

  Soient $b_1, \ldots, b_n, b_{n+1}$ les symboles lus, et soit i, $0 \leq i < K$ (K la taille du dictionnaire), l'indice du mot $(b_1, \ldots, b_n)$ dans le dictionnaire,

– $(b_1, \ldots, b_n, b_{n+1})$ est ajouté au dictionnaire avec l'indice K,

– on imprime i en base 2 sur $\lceil \log_2 K \rceil$ bits suivi du symbole b_{n+1}.

Lempel-Ziv – Exemple

1	0	0 1	0 1 1	1 0	0 0	1 1	1 0 0	1 0 1

Dictionnaire		paire	
indices	mots lu	(indice,symbole)	mot de code
0			
1	**1**	(0,**1**)	**1**
2	**0**	(0,**0**)	**0 0**
3	**01**	(2,**1**)	**10 1**
4	**011**	(3,**1**)	**11 1**
5	**10**	(1,**0**)	**001 0**
6	**00**	(2,**0**)	**010 0**
7	**11**	(1,**1**)	**001 1**
8	**100**	(5,**0**)	**101 0**
9	**101**	(5,**1**)	**0101 1**

Lempel-Ziv 78 – Décodage

Le dictionnaire est cette fois un tableau initialement de taille $K = 1$, sa seule entrée M_0 est le mot vide. On répète jusqu'à épuisement :

- on lit les $\lceil \log_2 K \rceil$ premiers bits du texte codé pour obtenir l'indice i. Soit M_i le mot d'indice i dans le dictionnaire,
- on lit le symbole binaire suivant b,
- on ajoute une K-ème entrée au tableau/dictionnaire $M_K \leftarrow M_i \parallel b$,
- on imprime M_i.

On pourrait également décoder en reconstruisant l'arbre, mais c'est plus complexe.

Lempel-Ziv – Optimalité (1)

Soit T un texte binaire et L un entier positif. Nous découpons le texte par blocs de L bits. Soit $A_L = \{a_1, \ldots, a_K\}$ l'ensemble des mots rencontrés dans le texte ($K \leq 2^L$). Pour tout i, nous notons F_i le nombre d'occurrences de a_i dans T et $p_i = F_i/|T|$. L'entropie par symbole d'ordre L du texte T est défini de la façon suivante

$$H_L(T) = \frac{1}{L} \sum_{i=1}^{K} p_i \log_2 \frac{1}{p_i}$$

Tout codage du texte T codant les bits L à la fois indépendamment vérifie

$$n(T) \geq H_L(T).$$

Lempel-Ziv – Optimalité (2)

L'algorithme de Lempel-Ziv est relativement lent à démarrer. Par contre, on peut montrer que s'il est utilisé pour coder une source stationnaire (ergodique) son efficacité tends vers 1 lorsque la taille du texte tend vers l'infini.

Ce résultat se montre en prouvant pour tout texte T de longueur N une inégalité du type

$$n(T) \leq H_L(T) + \varepsilon(N, L),$$

où $\varepsilon(N, L)$ dépend de L et de N mais pas du texte T, et de plus $\lim_{N\to\infty} \varepsilon(N, L) = 0$.

Donc pour N suffisamment grand, l'algorithme de Lempel-Ziv permet d'atteindre des performances aussi bonne que tout code codant des blocs de L symboles à la fois. En faisant tendre L vers l'infini on atteint l'entropie par lettre.

Variante de Welsh

- Au démarrage les mots 0 et 1 sont déjà dans le dictionnaire.
- Au lieu d'afficher la paire (i, b) on n'affiche que i.
- On ajoute le mot (i, b) dans le dictionnaire.
- On reprend la lecture à partir de b inclus.

$\Rightarrow$ variante légèrement plus efficace.

Lempel-Ziv-Welsh – Exemple

1 0 0 1 0 1 1 1 0 0 0 1 1 1 0 0 1 0 1 ...

Dictionnaire		Mot lu		
indices	mots	valeur	indice	mot de code
0	**0**			
1	**1**			
2	**10**	**1**	1	**1**
3	**00**	**0**	0	**00**
4	**01**	**0**	0	**00**
5	**101**	**10**	2	**010**
6	**11**	**1**	1	**001**
7	**110**	**11**	6	**110**
8	**000**	**00**	3	**011**
9	**011**	**01**	4	**0100**
10	**1100**	**110**	7	**0111**
11	**010**	**01**	5	**0101**
...				

Lempel-Ziv 77

Cette variante est apparue avant celle présentée précédemment (qui date de 78). Elle est plus efficace, mais plus difficile à mettre en œuvre.

On suppose que N bits ont été lus. On lit à partir du $N+1$-ème bit le plus long mot (de longueur n) qui soit un sous-mot commençant au i-ème bit avec $i \leq N$. Ce mot est codé par la paire (i, n).