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 X_n 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 un code de Huffman T_n 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 – Codage

L'arbre initial est constitué d'une unique feuille, celle de la lettre vide.
À chaque fois qu'une lettre x est lue dans le texte source
- si elle est déjà apparue
  - on imprime son mot de code,
  - on met à jour l'arbre,
- sinon
  - on imprime le mot de code de la lettre vide suivi de x non codé (son code ascii en base 2 par exemple),
  - 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.
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 – Définitions préliminaires

Nous identifions un code préfixe d'une source discrète X à un arbre binaire à $|X|$ feuilles étiquetées par les lettres de X. Le code est dit complet si chaque nœud possède 0 ou 2 fils.

Définition Soit un code prefixe d'une source discrète
– le *poids d'une feuille* est la probabilité de sa lettre
– le *poids d'un nœud* est la somme du poids de ses fils

Définition Un *arbre de Huffman* de X est un code préfixe de X qui peut être obtenu par l'algorithme de Huffman.

Définition Un *ordre de Gallager* $u_1, \ldots, u_{2K-1}$ sur les nœuds d'un code préfixe complet (d'une source de cardinal K) vérifie

1. les poids des u_i sont décroissants,
2. u_{2i} et u_{2i+1} sont frères pour tout i, $1 \leq i < K$.

Huffman adaptatif – Propriétés

Théorème [Gallager] Soit T un code préfixe d'une source X.
T est un arbre de Huffman de X si et seulement si il existe un ordre de Gallager sur les nœuds de T.

En reprenant les notations du transparent 2

Proposition Soient X_n la source au rang n et T_n un arbre de Huffman de cette source. Soit x la $n+1$-ème lettre du texte et soit $u_1, \ldots, u_{2K-1}$ un ordre de Gallager sur les nœuds de T_n.
Si $x \in X_n$ et si tous les nœuds $u_{i_1}, u_{i_2}, \ldots, u_{i_\ell}$ du chemin entre x et la racine sont les premiers de leur poids dans l'ordre de Gallager, alors T_n est un arbre de Huffman de X_{n+1}.

Huffman adaptatif – Mettre à jour l'arbre

Soit T_n un arbre de Huffman au rang n et $u_1, \ldots, u_{2K-1}$ un ordre de Gallager sur ses nœuds.

On suppose que la lettre courante x est présente dans l'arbre et que le nœud correspondant est u.

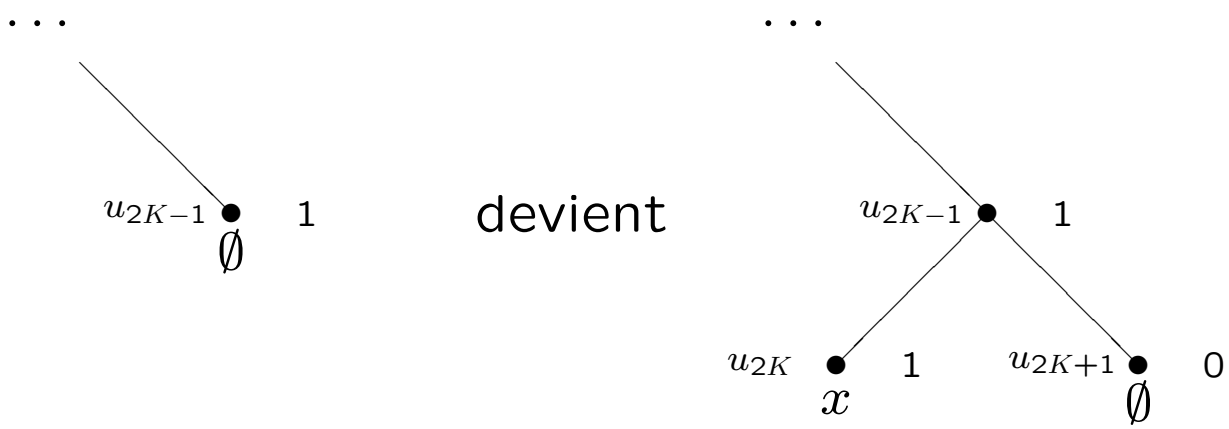
Répéter tant que u n'est pas la racine

- soit $\tilde{u}$ le premier nœud classé avec le poids de u,
- échanger les pères de u et $\tilde{u}$,
- échanger u et $\tilde{u}$ dans l'ordre de Gallager,
- incrémenter le poids de u *(poids = nb occurence)*
- u devient le père de u

Cet algorithme est du à D. Knuth.

Huffman adaptatif – Ajouter une feuille

Lorsque la lettre courante x n'est pas présente dans l'arbre, on effectue la mise à jour à partir de la feuille vide. On remplace ensuite celle-ci par un arbre à deux feuilles, l'une pour x l'autre vide.



Les deux nouveaux nœuds sont ajoutés à la fin de la liste (u_i) qui reste un ordre de Gallager pour le nouvel arbre.

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 nouveau mot (de longueur n) au dictionnaire et on recommence à partir du $(N+n+1)$-ème symbole.

Lempel-Ziv 78 – 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 78 – 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 78 – 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 suivant b,
- on ajoute une K-ème entrée au tableau/dictionnaire $M_K \leftarrow M_i \parallel b$,
- on imprime M_K.

On pourrait également décoder en reconstruisant l'arbre.

Variante de Welsh

– Au démarrage tous les mots de une lettre sont 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).

En pratique, on l'implémente à l'aide d'une fenêtre glissante de taille fixe : le dictionnaire est l'ensemble des sous-mots de cette fenêtre.

Lempel-Ziv – Optimalité

L'algorithme de Lempel-Ziv est asymptotiquement optimal ; pour tout texte produit par une source d'entropie par lettre $\mathcal{H}$, lorsque la taille du texte tend vers l'infini, la longueur par lettre de la suite codée (en binaire) tend vers $\mathcal{H}$.

Idée de la preuve : seuls les mots typiques sont pris en compte.