Compression de données

• Si, au lieu de dépanser 150 euros pour acheter un nouveau disque dur vous pouviez garder cette somme pour vos vacances ?

• Le codage MP3 des données musicales a rendu possible le stokage et l'échange des fichier audio par Internet.

• La sonde *Clementine* lancée par la NASA vers la Lune il y a quelques années emportait un dispositif de compression d'images (dû au CNES et à Matra) qui lui a permis de transmettre un million de photos et non pas seulement 70 000.

• Les techniques de compression semblent si prometteuses que certains se demandent si les réseaux téléphoniques actuels, associéés à quelques satellites de communication, ne pourraint pas suffir à la grande majorité des applications en évitant les dépenses d'infrastructure liées aux fameuses *autoroutes de l'information.*

• JPEG (Joint Photographic Expert Group), MPEG (Moving Picture Expert Group), MP3 : des fichiers audio exploitent les caractéristiques de l'audition humaine et dégrade le son d'une manière quasiment inaudible

La compression de données date du 19e siècle avec l'invention

- du Braille en 1820
- du code Morse en 1838

Elle a été formalisée grâce à la théorie de l'information. La compression fonctionne à *l'inverse*

- des codes correcteurs d'erreurs
- (chiffrement)

Si les codes correcteurs d'erreurs ajoute de la redondance pour transmettre un signal en toute sécurité sur un canal bruité, la compression va, elle, tenter de retirer le plus de redondance possible d'une donnée.

Les principales applications de la compression concernent

- l'archivage des données
- les télécommunications
- les réseaux

L'archivage des données

- sur un disque dur
- CD-ROM, DVD

Certains formats de fichiers intègrent directement de la compression.

- les fichiers d'images (gif ou $jpeg$)
- certains fichiers texte : portable document format (pdf) d'Adobe

Les télécommunications

- dans le fonctionnement des modems (protocole V 42)
- pour les transmissions par télécopie.

Les réseaux : augmenter la bande passante en diminuant le nombre de bits émis

Compression de données = On cherche une représentation alternative des données qui est plus efficace en espace, souvent au détriment du temps d'accès.

Compression de données = à deux algorithmes

- l'algorithme de compression
  - qui prend en entrée une chaîne de caractères B (pour donnée brute)
  - qui calcule une représentation C (pour donnée comprimée) plus courte que B
- l'algorithme de décompression
  - qui prend en entrée une une représentation comprimée C
  - qui calcule R, une donnée reconstruite à partir de C

Compression de données

- compression avec pertes (des signaux audio ou vidéo ...) On tolère que l'image restituée après décompression soit un peu différente, si cela fait gagner de l'espace.
- compression sans perte (données textuelles ou numériques)

Généralement, l'utilisation d'un procédé avec pertes améliore la compression.

Intuitivement, la **compression de données** est réalisée en diminuant la redondance de l'entrée, ce qui a aussi pour effet de rendre la donnée moins fiable, plus sujette aux erreurs.

Le but des **codes correcteurs d'erreurs** est de rendre une donnée plus fiable, au prix d'une redondance accrue.

Les méthodes actuelles de compression permettent de gagner

- 50 pour cent pour les fichier *texte*
- 80 pour cent pour les images *fixes*
- 95 pour cent pour pour un *film*

Définition

- L'algorithme de compression est le programme qui comprime la donnée brute B fournie en entrée et crée en sortie une donnée comprimée C.
- L'algorithme de décompression effectue l'opération inverse, souvent appelée reconstruction.

  La longueur de C est une mesure numerique du contenu en information de B.

- *algorithme symétrique* = l'algorithme de compression et celui de décompression utilisent le même programme qui travaille de manière symétrique (et dans la même complexité en temps).
- *algorithme asymétrique* = les deux algorithmes n'utilisent pas le même programmeet l'un ou l'autre des deux algorithmes effectue un travail plus conséquent. C'est en particulier le cas pour les algorithmes utilisés pour la compression des données sur un DVD où la compression est faite une seule fois à la création et la décompression est utilisée à chaque utilisation.

Définition

- le rapport de compression, $\frac{|C|}{|B|} < 1$. Une valeur de 0,6 signifie que $|B|$ a été réduit de 40%
- le facteur de compression, $\frac{|B|}{|C|} > 1$ (rapport inverse du rapport de compression). Plus la compression est grande, plus le facteur de compression croît.

  Hélas, les méthodes parfaites de compression (mentionnées par la théorie) sont des méthodes idéales dont on démontre qu'elles ne sont pas programmables: aucun algorithme ne permettra jamais de calculer les compressions optimales ! Le recours à des méthodes de compression particulières et imparfaites est donc inévitable

Distance de similarité

= la mesure numerique du contenu commun en information

Applications à la classification des :

- langues
- morceaux de musique
- textes
- images
- données astronomique
- séquence d'ADN
- ...

Soitent X et Y les données à classer et

- $c(X) =$ la longueur de la versions comprimée de X
- $c(Y) =$ la longueur de la versions comprimée de Y
- $c(XY) =$ la longueur de la versions comprimée de XY (X suivi de Y)

$c(X) + c(Y) - c(XY)$ est une mesure numerique du contenu commun en information de X et Y.

La distance de similarité entre X et Y est définie par

$$d(X,Y) = \begin{cases} 1 - \frac{c(X)+c(Y)-c(XY)}{c(X)} & \text{si } c(X) \geq c(Y) \\ 1 - \frac{c(X)+c(Y)-c(XY)}{c(Y)} & \text{si } c(X) \leq c(Y) \end{cases}$$

- plus le contenu commun en information de X et Y est grand, plus petite est la distance $d(X,Y)$
- plus les séquences X et Y sont indépendantes (sans corrélation), plus $d(X,Y)$ s'approche de 1

Méthodes naïves

Lorsqu'un texte comporte un grand nombre d'espaces qui ne sont pas adjacents il est possible de les retirer.

Exemple le texte *Pour réduire la longueur* devient le texte comprimé:

- Pourréduirelalongueur et
- 000010000000100100000000

Compression de tête

Si on travaille sur une liste de mots triés dans l'ordre lexicographique.

Principe : que deux mots successifs dans notre liste partagent souvent un même préfixe p de longueur n. On peut donc remplacer les lettres de p dans le second mot par n, longueur du préfixe commun.

Exemple

Coda	Coda
Codé	3é
Code-Barres	3e- Barres
Coder	4r
Codeur	4ur
Codicille	3icille

Méthode RLE = Run Length Encoding

Comment moyennant quelques conventions de codage bien choisies on exploite les régularités des données informatiques pour en diminuer la taille

Le principe : si une lettre a apparaît n fois successivement dans l'entrée, on peut remplacer les n occurrences de a par le couple $n \cdot a$

Exemple

les chaussettes de l'archiduchesse

et la transforme en les chau@2se@2tes de l'archiduche@2se

les *chau@2se@2tes de l'archiduche@2se*

Il ne faut donc utiliser cette technique que lorsqu'un caractère est répété au moins trois fois.

Pour éviter d'avoir recours à un caractère particulier comme le @ pour indiquer la répétition : si l'entrée comprend n répétitions de c, selon la valeur de n,

- si $n < 3$ l'algorithme de compression écrit cc;
- si $n > 3$, l'algorithme de compression écrit $ccc(n-3)$.

C'est la convention qui est utilisée au sein de la méthode de compression $NP5$ des modems.

Méthode à dictionnaire

Exemple

Le texte :

ainsi font font font les petites marionnettes,

ainsi font font font les petites marionnettes,

de 94 caractères est transformé en :

#ainsi #font #les #petites marionnettes

##1 #2 #2 #2 #3 #1 #2 #2 #2 #3

Le résultat est moins bon sur cet exemple mais la méthode s'applique dans un plus grans nombre de cas, car elle code les répétitions même lorsqu'elles ne sont pas consécutives.

Algorithmes statistiques

Définition Un code est dit optimale pour une source discrète (sans mémoire) s'il minimise la longueur moyenne parmis tous les codes possibles pour cette source.

Définition Soit $A = \{a_1, a_2, \ldots a_r\}$ une source à r symboles (= messages = lettres). Un code pour A est une application

$$\begin{array}{ccc} a_1 & \to & c_1 \\ a_2 & \to & c_2 \\ \ldots & & \\ a_r & \to & c_r \end{array}$$

où $C = \{c_1, c_2, \ldots c_r\} \subset \{0,1\}^*$.

Propriété H Si $C = \{c_1, c_2, \ldots c_r\}$ est un code binaire, préfixe optimal pour une source $A = \{a_1, a_2, \ldots a_r\}$, alors le code binaire $C' = \{c'_1, c'_2, \ldots c'_r, c'_{r+1}\}$ défini par

$$\begin{aligned} c'_1 &= c_1 \\ c'_2 &= c_2 \\ \ldots &= \\ c'_{r-1} &= c_{r-1} \\ c'_r &= c_r \cdot 0 \\ c'_{r+1} &= c_r \cdot 1 \end{aligned}$$

est code préfixe optimal pour la source $A' = \{a_1, a_2, \ldots a_{r-1}, a'_r, a'_{r+1}\}$ si

- $p(a'_{r+1}) + p(a'_r) = p(a_r)$, et
- $p(a'_r), p(a'_{r+1}) \leq p(a_i)$ pour $1 \leq i \leq r-1$

Huffman a proposé un algorithme qui permet de construire un code préfixe, optimal pour une source A' comprenant $r+1$ symboles sur un alphabet binaire fondé sur une procédure de réduction qui permet de transformer A' en une source à r symboles A pour obtenir finalement une source à deux symboles pour laquelle le code préfixe optimal est $\{0,1\}$,

La procédure de réduction :

1. on classe dans une liste les symboles de la source dans l'ordre des probabilités décroissantes;
2. on remplace les deux derniers symboles par un seul symbole, affecté d'une probabilité qui est la somme des probabilités des deux derniers symboles;
3. on réordonne la liste et on répète le processus;
4. on s'arrête lorsque le nombre d'éléments de la liste est égal à deux.

Il ne reste plus qu'à appliquer successivement la propriété H en remontant de la source à 2 symboles codés par 0 et 1 vers A' pour avoir le code préfixe optimal pour A'.

Cet algorithme est utilisé pour la compression de fichiers dans la commande *compact* d'UNIX.

Exemple 1

alphabet source	probabilité
a	$\frac{1}{2}$
b	$\frac{1}{4}$
c	$\frac{1}{4}$

Exemple 2

alphabet source	probabilité
a	$\frac{1}{2}$
b	$\frac{1}{4}$
c	$\frac{1}{8}$
d	$\frac{1}{8}$

lettre	fréquence en anglais	code binaire
E	12.31	011
T	9.59	001
A	8.05	1111
O	7.94	1110
N	7.19	1100
I	7.18	1011
S	6.59	1001
R	6.03	0101
H	5.14	00001
L	4.03	11011
D	3.65	10101
C	3.20	10001
U	3.10	10000
F	2.28	00010

lettre	fréquence en anglais	code binaire
P	2.28	00011
M	2.25	01001
Y	1.88	110100
B	1.62	101001
G	1.61	101000
V	0.93	000000
W	0.25	110101
K	0.52	0000010
Q	2.29	00000110
X	0.20	000001111
Z	0.09	0000011100
J	0.10	0000011101

Algorithmes dynamiques

ou de *substitution de facteurs.*

Principe : le remplacement de facteurs de l'entrée par des codes plus courts. Ces codes représentent les indices des facteurs dans un dictionnaire qui est construit dynamiquement, au fur et à mesure de la compression.

La plupart des programmes de compression dynamiques utilisent une des deux méthodes proposées par Lempel et Ziv en

- 1977 appelé LZ77
- 1978 appelé LZ78

LZ7 et LZ78 : parcourent l'entrée à comprimèr de la gauche vers la droite. Ils remplacent les facteurs répétés par des pointeurs vers l'endroit où ils sont déjà apparus dans le texte.

- *pkzip* sous MS-DOS (LZ77)
- *gzip* sous UNIX (LZ77)
- compress d'UNIX (LZ78)
- format d'images *gif* (LZ78)

LZ77

Principe : l'algorithme utilise une partie de la donnée d'entrée comme dictionnaire. L'algorithme de compression fait glisser une fenêtre de N caractères sur la chaîne d'entrée de la gauche vers la droite. Cette fenêtre est composée de deux parties:

- à gauche, le tampon de recherche de $N - F$ caractères qui constitue le dictionnaire courant des lettres qui ont été lues et comprimées récemment;
- à droite, le tampon de lecture de F caractères dans lequel se trouvent les lettres en attente de compression.

$\ldots \underbrace{a\_ab\_bab\_abab\_ababb}_{tampon\ de\ recherche}\ \underbrace{aba\_bbb\_bab\_bbabb}_{tampon\ de\ lecture}\ldots$

L'algorithme de compression parcourt le tampon de recherche de la droite vers la gauche pour faire correspondre le plus de symboles possibles entre le préfixe du tampon de lecture et le facteur du tampon de recherche.

Il peut mettre en correspondance le facteur ab, de longueur 2 avec un décalage de 3.

$$\ldots \underbrace{a\_ab\_bab\_abab\_ab\underline{ab}b}_{\textit{tampon de recherche}} \quad \underbrace{\underline{ab}a\_bbb\_bab\_bbabb}_{\textit{tampon de lecture}} \ldots$$

Il continue ensuite sa recherche pour essayer de faire correspondre le plus de symboles possibles entre le préfixe du tampon de lecture et un facteur du tampon de recherche.

$$\ldots \underbrace{a\_ab\_bab\_abab\_\underline{aba}bb}_{\textit{tampon de recherche}} \quad \underbrace{\underline{aba}\_bbb\_bab\_bbabb}_{\textit{tampon de lecture}} \ldots$$

La meilleure correspondance est faite entre le facteur *aba* et le préfixe *aba*, avec un décalage de 5, d'une longueur de 3 caractères.

$$\ldots \underbrace{a\_ab\_bab\_\underline{aba}b\_ababb}_{tampon\ de\ recherche}\ \ \underbrace{\underline{aba}\_bbb\_bab\_bbabb}_{tampon\ de\ lecture}\ldots$$

Une correspondance équivalente est trouvée avec un décalage de 10 et une longueur de 3.

L'algorithme choisit la plus longue correspondance et, s'il y en a plusieurs équivalentes, celle qui est le plus à gauche (la dernière trouvée) dans le tampon de recherche.

Dans notre cas, il sélectionne celle correspondant à un décalage de 10 et une longueur de 3 et il écrit le *lexème* correspondant $(10, 3, \_)$.

- 10 est le décalage
- 3 est la longueur
- _ est le premier caractère qui n'est pas en correspondance dans le préfixe du tampon de lecture

Algorithme de compression LZ77

1. mettre le pointeur de codage au début de l'entrée

- **tantque non** vide (tampon de lecture) **faire**

2. trouver la plus longue correspondance entre le tampon de lecture et celui de recherche

3. écrire (p, ℓ, c) où :
   - p : mesure la distance de décalage
   - ℓ : longueur de la correspondance
   - c : 1er caractère de l'entrée qui n'est pas dans le dictionnaire

4. déplacer le pointeur de codage de $\ell + 1$ positions vers la droite;

- **ftq**

A chaque instant, l'algorithme va rechercher dans les $N - F$ premiers caractères du tampon de recherche le plus long facteur qui se répète au début du tampon de lecture. Il doit être de taille maximale F. Cette répétition sera codée (p, ℓ, c) où :

- p est la distance entre le début du tampon de lecture et la position de répétition dans le dictionnaire;
- ℓ est la longueur de la répétition;
- c est le premier caractère du tampon de lecture différent du caractère correspondant dans le tampon de recherche.

Exemple

Avec

- une fenêtre de taille $N = 11$ dont
- le tampon de lecture est de $F = 5$ caractères
- et l'entrée $\_\_\_\_\_le\_mage\_dit\_abracadabra$

$\underbrace{\_\_\_\_\_}\underbrace{le\_ma}ge\_dit\_abracadabra \qquad (0, 0, l)$

$\_\underbrace{\_\_\_\_l}\underbrace{e\_mag}e\_dit\_abracadabra \qquad (0, 0, e)$

$\_\_\underbrace{\_\_\_le}\underbrace{\_mage}\_dit\_abracadabra \qquad (3, 1, m)$

$\_\_\_\_\underbrace{\_le\_m}\underbrace{age\_d}it\_abracadabra \qquad (0, 0, a)$

$\_\_\_\_\_\underbrace{le\_ma}\underbrace{ge\_di}t\_abracadabra \qquad (0, 0, g)$

$\_\_\_\_\_\underbrace{le\_mag}\underbrace{e\_dit}\_abracadabra \quad (5, 2, d)$

$\_\_\_\_\_le\_\underbrace{mage\_d}\underbrace{it\_ab}racadabra \quad (0, 0, i)$

$\_\_\_\_\_le\_m\,\underbrace{age\_di}\,\underbrace{t\_abr}\,acadabra \quad (0, 0, t)$

$\_\_\_\_\_le\_ma\,\underbrace{ge\_dit}\,\underbrace{\_abra}\,cadabra \quad (4, 1, a)$

$\_\_\_\_\_le\_mage\,\underbrace{\_dit\_a}\,\underbrace{braca}\,dabra \quad (0, 0, b)$

$\_\_\_\_\_le\_mage\_\underbrace{dit\_ab}\underbrace{racad}abra$ $(0, 0, r)$

$\_\_\_\_\_le\_mage\_d\underbrace{it\_abr}\underbrace{acada}bra$ $(3, 1, c)$

$\_\_\_\_\_le\_mage\_dit\underbrace{\_abrac}\underbrace{adabr}a$ $(5, 1, d)$

$\_\_\_\_\_le\_mage\_dit\_a\underbrace{bracad}\underbrace{abra}\_\_$ $(4, 1, b)$

$\_\_\_\_\_le\_mage\_dit\_abr\underbrace{acadab}\underbrace{ra}\_\_\_\_$ $(0, 0, r)$

$\_\_\_\_\_le\_mage\_dit\_abra\underbrace{cadabr}\underbrace{a}\_\_\_\_\_$ $(5, 1, '')$

Si on ajoute un codage de Huffman à l'issue de LZ77, on attribuera aux décalages les plus courts des codes plus courts (méthode proposée par B. Herd)= LZH : si on dispose d'un grand nombre de petits décalages, on améliore la compression en utilisant LZH.

En pratique,

- $N - F = 2^{e_1}$
- $F = 2^{e_2}$

On a donc besoin de e_1 bits pour coder p, la position dans le dictionnaire et de e_2 bits pour coder ℓ, la longueur de la répétition.

En pratique, cette méthode fonctionne assez bien pour une taille de fenêtre de l'ordre de $N \leq 8192$ pour les raisons suivantes :

- Beaucoup de mots et de fragments de mots sont suffisamment courants pour apparaître souvent dans une fenêtre. C'est le cas pour: « ...sion », « ...que », « de », « le », « la », « ...ment », « ce » ...
- Les mots rares ou techniques ont tendance à être répétés à des positions très proches. C'est par exemple le cas pour le mot « compression » qui figure souvent dans ces tansparents
- Si un caractère ou une suite de caractères sont répétés un certain nombre de fois consécutivement, le codage peut être très économe parce qu'il autorise la répétition à chevaucher les deux tampons.

La décompression

1. lire un lexème
2. chercher la correspondance dans le tampon de recherche
3. écrire le facteur trouvé au début du tampon de lecture
4. écrire la 3^{e} composante du lexème à la suite
5. décaler le contenu des tampons de $\ell + 1$ cases vers la gauche

Faiblesse de LZ77

- réside dans l'hypothèse que les motifs répétés sont proches dans les données d'entrée.
- un autre inconvénient est que la taille F du tampon de lecture est limitée. De ce fait, la taille de la plus longue correspondance ne peut excéder $F - 1$. F ne peut croître beaucoup, car le temps de compression croît proportionnellement à F. Il en est de même avec la taille du tampon de recherche.

Algorithme de compression LZ78

remédie partiellement aux faiblesses de LZ77. Le dictionnaire n'est plus composé d'une fenêtre coulissante mais est constitué de l'intégralité du texte déjà traité.

Le principe Le texte est comprimé de la gauche vers la droite. Au départ, l'algorithme ne connaît aucun facteur (i.e. le dictionnaire est vide) et va mémoriser au cours du temps la totalité des facteurs rencontrés dans le dictionnaire et leur attribuer un numéro. Cette méthode va rechercher le plus long facteur f inscrit dans le dictionnaire qui coïncide avec le préfixe p du texte restant à traiter.

Deux cas peuvent se produire:

- On ne trouve pas d'entrée au dictionnaire; le texte restant à traiter s'écrit alors comme $c.m$ avec c le caractère inconnu au dictionnaire et m le reste du texte. L'algorithme renvoie le couple $(0, c)$ et ajoute l'entrée c au dictionnaire
- On trouve une entrée f dans le dictionnaire à la position $i > 0$; le texte restant à traiter s'écrit alors comme $f.c.m$ avec f, le facteur trouvé dans le dictionnaire, c le premier caractère qui diffère et m le reste du texte. L'algorithme renvoie le couple (i, c) et ajoute l'entrée fc au dictionnaire

$mot \leftarrow \emptyset$

dictionnaire $\leftarrow \emptyset$

$i \leftarrow 1$

répéter

  lire s, le premier caractère du texte T restant

  (on retire s de T)

  si $mot.s \in dictionnaire$

  alors

  $mot \leftarrow mot.s$

  sinon

  émettre (i, s)

  affecte $(mot.s)$ à l'entrée i du dictionnaire

  $i \leftarrow i + 1$

  $mot \leftarrow \emptyset$

  fin si

jusqu'à fin des données à comprimer

émettre éventuélemet (i, s)

Une implantation rapide de cet algorithme donne lieu à une complexité quadratique sur la longueur du texte à comprimer. Cependant, en utilisant des structures de données adéquates, ce temps de calcul peut devenir linéaire. C'est sur ce point précis que de nombreuses variantes de cet algorithme ont été proposées.

Exemple

Soit la chaîne aaabbabaabaaabab à comprimer

ditionaire		lexème
0	null	
1	a	(0,a)
2	aa	(1,a)
3	b	(0,b)
4	ba	(3,a)
5	baa	(4,a)
6	baaa	(5,a)
7	bab	(4,b)

Résultat : 0**0** **1**0 0**1** 11**0** 100**0** 101**0** 100**1**

Algorithme de décompression

$mot \leftarrow \emptyset$

dictionnaire $\leftarrow \emptyset$

$i \leftarrow 1$

répéter

  lire suivant (i, s), dans le texte comprimé

  si $i = 0$

  alors

  émettre(s)

  dictionnaire[i] $\leftarrow s$

  $i \leftarrow i + 1$

  sinon

  facteur $\leftarrow$ concatène(dictionnaire[i],s)

  émettre facteur

  dictionnaire[i] $\leftarrow$ facteur

  $i \leftarrow i + 1$

  fin si

jusqu'à fin du texte à décomprimer

Une propriété remarquable de cette méthode est que l'algorithme de compression et celui de décompression utilisent le même dictionnaire sans que celuici ne soit transmis : Il est entièrement reconstruit au cours de la décompression.

Exemple

Soit la donnée :
(0, a) (1, b)(0, b)(2, a)(3, a)(3, b)(6, a) (6, b)(2, b)
à décomprimer.

Elle donne lieu à la suite d'émissions suivante:
a ab baba ba bb bba bbb abb

ditionaire		lexème
0	null	
1	a	(0,a)
2	ab	(1,b)
3	b	(0,b)
4	aba	(2,a)
5	ba	(3,a)
6	bb	(3,b)
7	bba	(6,a)
8	bbb	(6,b)
9	abb	(2,b)

En pratique

En pratique, LZ78 ne travaille sur un dictionnaire de taille bornée et lorsque le dictionnaire est complètement rempli, celui-ci est effacé et la compression continue avec un nouveau dictionnaire

La compression sera moins bonne mais cette méthode peut être employée, même si le dictionnaire n'est pas de taille suffisante pour contenir l'ensemble des facteurs du texte

LZ78 a donné lieu à un grand nombre de variantes :

1. **LZW** proposée par T. Welch pour les contrôleurs de disque dur ainsi
2. **LZC** qui est utilisée dans l'utilitaire compress d'UNIX

Limites de la compression

Il existe des données qui ne peuvent être comprimées.

Pour tout entier n, il existe

$$2^n$$

mots binaires différents de longueur n mais seulement

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1$$

descriptions plus courtes (mots comprimés de longueur strictement inférieure à n).

Pour tout n, il existe donc au moins un mot binaire de longueur n qui ne peut être comprimé.

C'est le cas des suites finies (vraiment) aléatoiress. En effet, intuitivement, on ne peut trouver de régularité dans une suite aléatoire. C'est ce qu'atteste la complexité de Kolmogorov.

Comment construire l'algorithme de compression le plus performant du marché. On suppose disposer de mille algorithmes de compression:

$$C_1, C_2, ..., C_{1000}$$

et des algorithmes de décompression correspondants

$$D_1, D_2, ..., D_{1000}$$

On peut alors construire un algorithme de compression capable de comprimer toute suite de symbole aussi bien, à 10 bits près, que le meilleur des mille algorithmes de compression.

En effet, pour comprimer la donnée B, on commence par en effectuer la compression avec chacun des 1000 algorithmes de compression que nous avons à notre disposition. On mémorise k, le numéro de l'algorithme de compression qui nous a fourni le meilleur résultat. Il suffit alors de proposer comme donnée comprimée la suite C' composée du codage en binaire de k suivi du résultat de l'algorithme de compression C_k appliqué à B. Le codage binaire de k nécessite $10\ bits = \lfloor log_2(1000) + 1 \rfloor$

L'algorithme de décompression effectue le travail inverse

Il faut cependant remarquer que le temps de fonctionnement de notre algorithme correspond aux temps cumulés des N algorithmes de compression que nous avons à notre disposition

Journal of Integer sequences, Vol. 7 (2004)

Art 04.2.5 On k-colored Motzkin words A. Sappounakis and Tsikouras

Order

Korsh and LaFollette Loopless generation of Linear extension of a poset

International of computer mathematics

Generation of t-ary trees by ballot sequences Ahrabian and Nowzari-Dalini

La compression de données (Généralités)

Méthode statistique Algorithme naïf 1
Les méthodes réversibles (sans perte)

Compression exacte, permettant la restitution totale des donneés de départ. Lorsqu'il s'agit d'un texte on attend bien sûr de l'opération de décompression qu'elle redonne exactement le texte de départ.

- statistique : algorithmes de Huffman et de Shanon-Fano
- les méthodes dites *à dictionnaire* : algorithme LZW

Généralement, la compression est plus longue que la décompression.

Si la compression fonctionne, c'est que les documents manipulés ne sont pas quelconques. Si les données informatiques étaient quelcoques la compression serait impossible.

Acun algorithme de compression exacte ne peut comprimer tous les textes de longueur n

Pour la recherche d'algorithmes de compression optimaux on a le résultat suivant :

il existe une fonction OPTI de compression

lettre	fréquence en français	fréquence en anglais	alphabet de Morse
a	6.16	8.05	·–
b	0.40	1.62	– ···
c	5.35	3.20	– · –·
d	3.86	3.65	– ··
e	18.61	12.31	·
f	2.24	2.28	·· –·
g	1.79	1.61	– – ·
h	1.48	5.14	····
i	6.35	7.18	··
j	0.04	0.10	· – ––
k	0.13	0.52	– · –
l	5.26	4.03	· – ··
m	1.79	2.25	––
n	6.02	7.19	–·
o	5.12	7.94	– – –
p	2.92	2.28	· – –·
q	0.62	2.29	– – ·–
r	5.35	6.03	· – ·
s	6.96	6.59	···
t	7.41	9.59	–
u	5.03	3.10	·· –
v	1.03	0.93	···
w	0.35	2.30	– ··–
x	0.36	0.20	– ··–
y	1.39	1.88	– · ––
z	0.04	0.09	– – ··