Compression

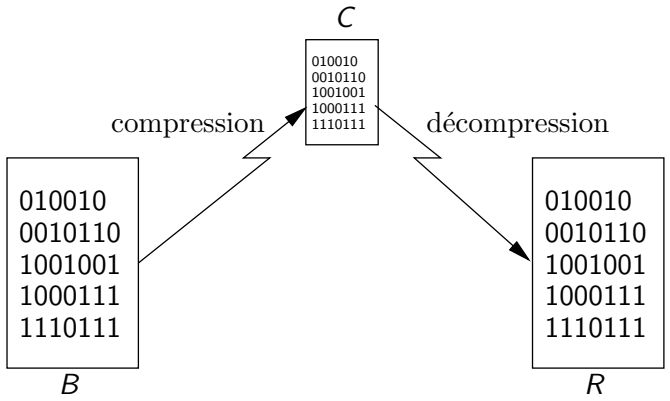
Bruno MARTIN,
Université Côte d'Azur

Plan

Pourquoi la compression de données ?

- les supports de stockage de données se remplissent en même temps que leur taille croît.
- formats de fichiers intègrent la compression :
  - images (gif, jpeg)
  - texte (pdf)
- réseaux : augmenter la bande passante en diminuant le nombre de bits transmis (pb des chiffres)
- télécommunications : utilisée dans le fonctionnement des modems (protocole V42 par exemple) et pour les transmissions par télécopie.

Compression & décompression



R n'est pas forcément identique à B. Quand

- $B = R$ on parle de compression **sans perte**
- $B \neq R$ on parle de compression **avec pertes** (plus efficace)

Types d'algorithmes de compression

- **algorithmes statistiques :** codes de Huffman, construisent un dictionnaire en effectuant une analyse statistique préalable
- **algorithmes dynamiques :** Lempel et Ziv, construisent dynamiquement un dictionnaire qui remplace les données répétées par des liens vers une entrée précédente
- **méthodes heuristiques** essayent de « deviner » les éléments du bloc de données. Ce sont les plus récentes.

Mesures d'efficacité

- **rapport de compression**, $\frac{|C|}{|B|} < 1$. Une valeur de 0,6 signifie que $|B|$ a été réduit de 40%.
- **facteur de compression**, rapport inverse du rapport de compression est normalement > 1. Plus la compression est grande, plus le facteur de compression croît.
- l'expression $100 \times (1 - \text{rapport de compression})$ est souvent utilisée. Une valeur de 40 signifie que $|B|$ à été réduit de 40%.

Méthodes naïves – Compression des espaces

Retirer les espaces d'un texte en mémorisant leur position par un mot binaire : 1 = un espace et 0 = autre caractère
Pour réduire la longueur $\mapsto$ 000010000000100100000000
le texte comprimé est :

000010000000100100000000|Pourréduirelalongueur

Faible nombre d'espaces dans m : $\sharp_1 m << \sharp_0 m$ et m pourra être comprimé, p.e. $0^4 10^7 10^2 10^8$ ou encore 4728.

Méthodes naïves – Compression de tête

Liste de mots triés dans l'ordre lexicographique (p.e. dictionnaire), 2 mots consécutifs partagent souvent un même n-préfixe p. On remplace p dans le second mot par n, longueur du préfixe commun.

Cod a	Coda
Codé	3é
Code -Barres	3e-Barres
Coder	4r
Cod eur	4ur
Codicille	3icille

RLE (pour *Run Length Encoding*)

Idée : répétition n fois de $\ll a \gg$ remplacée par na. (long. de répétition ou *Run length*)

les chaussettes de l'archiduchesse

et la transforme en

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

Observation

Sortie > entrée : dans l'entrée, on n'a que des répétitions de longueur 2 compressées avec 3 caractères !
Tactique valable quand un caractère est répété plus de 3 fois.

Parmi les méthodes dans la compression MNP5 des modems [3].

Méthode statistique, le code de Huffman

But : construire un code **préfixe** optimal pour une source S_r comprenant r symboles sur un alphabet binaire basé sur :

- **réduction :** transforme S_r en une source à $r-1$ symboles S_{r-1} pour obtenir finalement une source à deux symboles : son code préfixe optimal est $\{0,1\}$
- **propriété :** sur $\{0,1\}$ si $C = \{c_1, \ldots, c_r\}$ code préfixe optimal pour une source S_r, alors $C' = \{c'_1, \ldots, c'_r, c'_{r+1}\}$:

$$\begin{cases} c'_i = c_i & 1 \leq i \leq r-1 \\ c'_r = c_r.0 \\ c'_{r+1} = c_r.1 \end{cases}$$

est un code préfixe optimal pour la source S'_{r+1}.

Principe de réduction

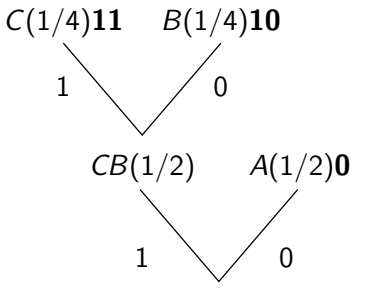
1. ordonner les symboles de source par proba. décroissantes
2. fusionner les 2 derniers symboles, avec proba = somme des proba des 2 derniers symboles
3. réordonner la liste et répéter le processus
4. arrêt quand il n'y a plus que 2 éléments

ensuite, appliquer successivement la propriété en remontant de la source à 2 symboles codés par 0 et 1 vers S_r pour avoir un code préfixe optimal pour S_r.

Exemple de code optimal

A, B et C t.q. $P(A) = 1/2$, $P(B) = P(C) = 1/4$.
après l'étape (4) de l'algorithme, on obtient :

A	$1/2$		$1/2$	$\rightarrow$ 0
B	$1/4$		$1/2$	$\rightarrow$ 1
C	$1/4$	$\nearrow$		



$A \mapsto 0$, $B \mapsto 10$ $C \mapsto 11$.
$ABAACABC \mapsto 010001101011$.

Principe des algorithmes dynamiques

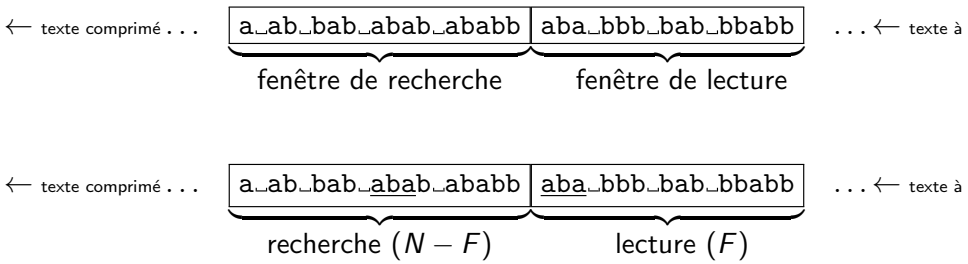
Remplacer des **facteurs** par des codes courts : indices des facteurs dans un dictionnaire construit dynamiquement.
Reposent sur une des méthodes proposées par Lempel et Ziv [5, 6].
LZ77 et LZ78 parcourent l'entrée à compresser de la gauche vers la droite en remplaçant les facteurs répétés par des pointeurs vers l'endroit où ils sont déjà apparus dans le texte.
Nombreuses variantes sur la manière de mémoriser et repérer les facteurs répétés.
LZ77 : utilisé dans `pkzip`, `gzip` ;
LZ78 : utilisé dans `compress` d'UNIX et le format d'images `gif`.

Principe de LZ77

Utilise une partie de la donnée d'entrée comme un dictionnaire.
L'algorithme fait glisser une fenêtre de N caractères sur la chaîne d'entrée de gauche à droite. Fenêtre en deux parties :

- à gauche : fenêtre de recherche, $N - F$ caractères : dictionnaire des lettres lues et comprimées récemment ;
- à droite : fenêtre de lecture, F caractères à comprimer.

Exemple



Algorithme de compression LZ77

Chercher dans les $N - F$ premiers caractères de la fenêtre de recherche le plus long facteur qui est un préfixe de la fenêtre de lecture de taille au plus F. Le codage est (p, ℓ, c) où :

- p : distance entre le début de la fenêtre de lecture et la position du facteur de répétition dans le dictionnaire ;
- ℓ : la longueur de la répétition ;
- c : 1^{er} caractère de la fenêtre de lecture différent du caractère correspondant dans la fenêtre de recherche.

Répétition peut chevaucher le dictionnaire et la fenêtre de lecture.
Après le codage, la fenêtre glisse de $\ell + 1$ caractères vers la droite.
Si on ne trouve pas de répétition, on code $(0, 0, c)$.

␣␣␣␣␣␣	le␣ma	ge␣dit␣abracadabra	(0, 0, l)
␣␣␣␣␣l	e␣mag	e␣dit␣abracadabra	(0, 0, e)
␣␣␣␣le	␣mage	␣dit␣abracadabra	(3, 1, m)
␣␣␣␣␣␣le␣m	age␣d	it␣abracadabra	(0, 0, a)
␣␣␣␣␣␣␣le␣ma	ge␣di	t␣abracadabra	(0, 0, g)
␣␣␣␣␣␣le␣mag	e␣dit	␣abracadabra	(5, 2, d)
␣␣␣␣␣␣le␣	mage␣d	it␣ab	racadabra	(0, 0, i)
␣␣␣␣␣␣le␣m	age␣di	t␣abr	acadabra	(0, 0, t)
␣␣␣␣␣␣le␣ma	ge␣dit	␣abra	cadabra	(4, 1, a)
␣␣␣␣␣␣le␣mage	␣dit␣a	braca	dabra	(0, 0, b)
␣␣␣␣␣␣le␣mage␣	dit␣ab	racad	abra	(0, 0, r)
␣␣␣␣␣␣le␣mage␣d	it␣abr	acada	bra	(3, 1, c)
␣␣␣␣␣␣le␣mage␣dit	␣abrac	adabr	a	(5, 1, d)
␣␣␣␣␣␣le␣mage␣dit␣a	bracad	abra␣	␣	(4, 1, b)
␣␣␣␣␣␣le␣mage␣dit␣abr	acadab	ra␣␣␣	␣	(0, 0, r)
␣␣␣␣␣␣le␣mage␣dit␣abra	cadabr	a␣␣␣␣	␣	(5, 1, "")

Décompression de LZ77

A partir des triplets (p, ℓ, c), le décodage se fait en faisant glisser la fenêtre comme pour le codage. Le dictionnaire est reconstruit.

Algorithme

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

	␣␣␣␣␣␣	l␣␣␣␣	(0, 0, l)
	␣␣␣␣␣l	e␣␣␣␣	(0, 0, e)
	␣␣␣␣le	␣m␣␣␣	(3, 1, m)
	␣␣le␣m	a␣␣␣␣	(0, 0, a)
	␣le␣ma	g␣␣␣␣	(0, 0, g)
	le␣mag	e␣d␣␣	(5, 2, d)
le␣	mage␣d	i␣␣␣␣	(0, 0, i)
le␣m	age␣di	t␣␣␣␣	(0, 0, t)
le␣ma	ge␣dit	␣a␣␣␣	(4, 1, a)
le␣mage	␣dit␣a	b␣␣␣␣	(0, 0, b)
le␣mage␣	dit␣ab	r␣␣␣␣	(0, 0, r)
le␣mage␣d	it␣abr	ac␣␣␣	(3, 1, c)
le␣mage␣dit	␣abrac	ad␣␣␣	(5, 1, d)
le␣mage␣dit␣a	bracad	ab␣␣␣	(4, 1, b)
le␣mage␣dit␣abr	acadab	r␣␣␣␣	(0, 0, r)
le␣mage␣dit␣abra	cadabr	a␣␣␣␣	(5, 1, "")

Faiblesses de LZ77

LZ77 suppose que les motifs répétés sont proches dans l'entrée.

Autre inconvénient : la taille limitée de la fenêtre de lecture F. 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 de la fenêtre de recherche.

Algorithme de compression LZ78

Fonctionnement analogue à LZ77 ; dictionnaire n'est plus une fenêtre glissante. Constitué de l'intégralité du texte déjà traité. Au départ, aucun facteur n'est connu ; ajouter au dictionnaire tous les facteurs rencontrés en les numérotant. Chercher le plus long préfixe p en correspondance avec un facteur f du dictionnaire. Deux cas :

- $f \notin$ dictionnaire ; le texte restant à traiter s'écrit $c.m$ avec c caractère inconnu au dictionnaire et m le reste du texte. L'algorithme rend $(0, c)$ et ajoute c au dictionnaire.
- $f \in$ dictionnaire à l'indice $i > 0$; le texte restant à traiter s'écrit alors comme $f.c.m$. L'algorithme rend (i, c) et ajoute $f.c$ au dictionnaire.

Exemple

Soit la chaîne *aabbababababbbabbbabb* à comprimer.

Dictionnaire		léxème
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	$(2, b)$

Algorithme de décompression de LZ78

Reconstruire le dictionnaire au fur et à mesure du décodage. Les indices des facteurs seront identiques à ceux du codage et les facteurs pourront être interprétés.

Compression et décompression utilisent le même dictionnaire sans que celui-ci soit transmis. Il est entièrement reconstruit au cours de la décompression.

La suite $(0, a)(1, b)(0, b)(2, a)(3, a)(3, b)(6, a)(6, b)(2, b)$ reconstruit le dictionnaire :

Dictionnaire		léxème
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	$(2, b)$

LZ78 en pratique

En pratique, LZ78 a un dictionnaire de taille bornée ; quand il est plein, on l'efface et on continue avec un nouveau dictionnaire.
Compression 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 un grand nombre de variantes, p.e. :

- LZW par par T. Welch [4] (contrôleurs de disque dur)
- LZC dans `compress` d'UNIX.

Le meilleur algorithme de compression

1000 algorithmes de compression et de décompression : $C_1, C_2, \ldots, C_{1000}$ $D_1, D_2, \ldots, D_{1000}$; on construit un algorithme de compression capable de comprimer toute suite de symboles B aussi bien, à 10 bits près que le meilleur des 1000 compresseurs [1].
Comprimer B : essayer chacun des 1000 compresseurs. Mémoriser k, numéro du meilleur algorithme de compression qui a comprimé B en C.
Nouvelle donnée comprimée : suite C' composée du codage en binaire de k suivi du résultat de l'algorithme de compression $C_k(B) = C$. Coder k nécessite 10 bits $= \lfloor \log_2 1000 \rfloor + 1$.
Décomprimer : algorithme de décompression effectue le travail inverse. Il décode k, le nombre binaire porté sur les 10 premiers bits de C' puis applique D_k sur la donnée de C' privé de ses 10 premiers bits.

Limites de la compression : données incompressibles

$\forall n \in \mathbb{N}$:

- 2^n mots binaires distincts de longueur n
- $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ 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 incompressible.
Cas des suites finies (vraiment) aléatoires [2].
On ne peut trouver aucune régularité dans une suite aléatoire (complexité de Chaitin-Kolmogorov).

Généralisation à un nombre N quelconque

Algorithme de compression construit sera aussi efficace que le meilleur des N compresseurs à $= \lfloor \log_2 N \rfloor + 1$ bits près.

Remarque : le temps de fonctionnement de notre algorithme correspond au temps cumulé des N algorithmes de compression


J.P. Delahaye.
La compression des données.
Pour la science, 217 :177–189, 1995.


M. Li and P. Vitányi.
An introduction to Kolmogorov complexity and its applications.
Springer Verlag, 1993.


D. Salomon.
Data compression, the complete reference.
Springer Verlag, 1998.


T.A. Welch.
A technique for high performance data compression.
Computer, 17(6) :8–19, 1984.


J. Ziv and A. Lempel.
A universal algorithm for sequential data compression.
IEEE Transactions on Information Theory, IT–23(3) :337–343, 1977.


J. Ziv and A. Lempel.
Compression of individual sequences via variable-rate coding.
IEEE Transactions on Information Theory, IT–24(5) :530–536, 1978.