UNIVERSITE DES SCIENCES SOCIALES DE GRENOBLE

UNIVERSITE CLAUDE BERNARD, LYON I

MEMOIRE DE D.E.S.S.

EN

INFORMATION SCIENTIFIQUE TECHNIQUE ET ECONOMIQUE

ALGORITHME DE COMPACTAGE DES FICHIERS INVERSES

PRESENTE PAR

BOSUWAN LERSAN

SOUS LA DIRECTION DE

R. CHAPUIS

JUILLET 1979

Table de matières

Introduction.

Dans la chaine documentaire d'un centre automatisé, l'une des principales activités est la recherche rétrospective, c'est-à-dire le mécanisme qui permet d'avoir une réponse la plus rapide et la plus pertinente possible.

Pour atteindre cet objectif, les informaticiens et les chercheurs ont pendant long temps rencontré des difficultés et notamment dans le cas de l'interrogation en conversationnel des grandes bases de données.

Actuellement, de nombreux efforts sont fournis pour résoudre ce problème soit par l'utilisation d'une technologie assez adapté (1,2) soit par des organisations particulières du fichier (3). Celles-ci sont très diverses (4) mais possèdent toutes une caractéristique commune : c'est la création d'un fichier auxiliaire susceptif de permettre d'avoir une réponse rapide. Ce fichier auxiliaire peut se présenter sous deux formes : fichier inversé ou fichier ayant une structure arborescente.

En pratique, il est apparu plus avantageux d'adopter un fichier inversé, malheureusement l'espace pour le stockage du fichier inversé est le double du fichier original (5). C'est cet inconvenient qui a amené les informaticiens à mettre en place des méthodes pour le compactage de ce fichier afin de reduire l'espace nécessaire pour le stockage.

Dans ce mémoire on essayera de donner un aperçu sur ces méthodes. Deux points seront alors à étudier pour effectuer cette approche :

1) Notion du fichier inversé, son role dans le

traitement de l'information et ses représentations ou formes stockées dans la mémoire ou sur le support magnétique.

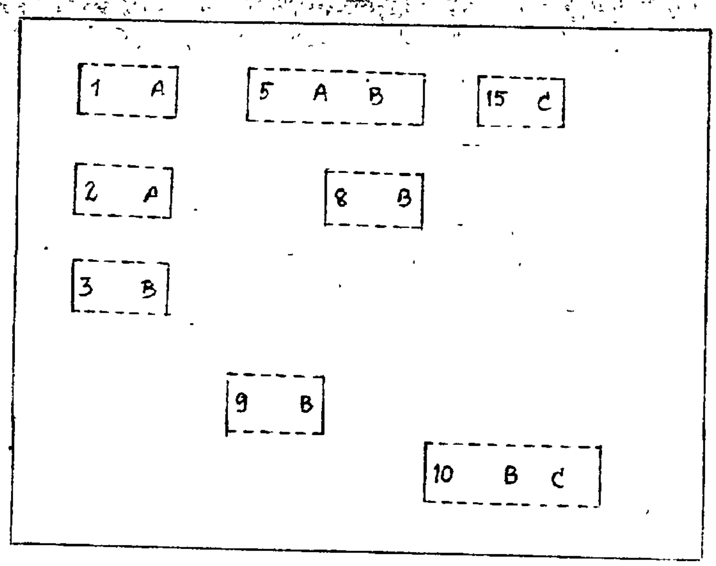
2). Présentation des techniques de compactage des fichiers inversés et les résultats sur le pourcentage de réduction d'espace et de temps d'accès.

Première Partie

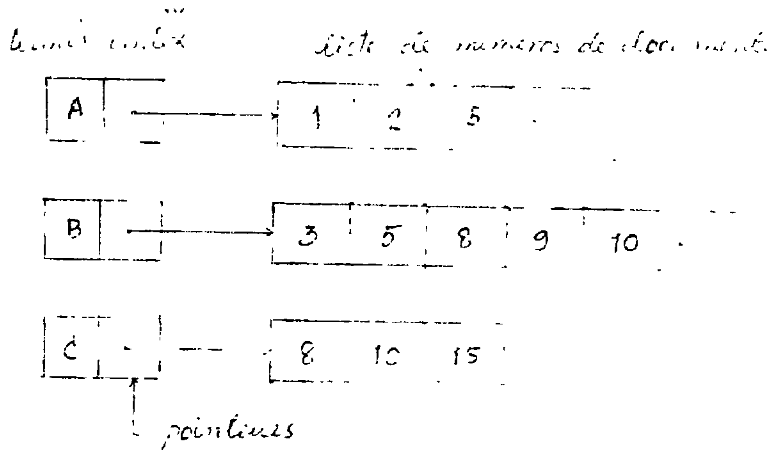
Fichier Inversé

1. Notion du Fichier Inversé:

Fichier inversé est un fichier bien organisé en raison d'être prêt pour le traitement de l'information qu'il contient, il est élaboré, par le système spécial, d'un fichier des notices, qu'on appelle fichier original ou fichier de base. En générale, le fichier inversé est composé d'une liste ou d'un répertoire des termes index utilisés pour décrire le contenu des documents dans la base. A chaque terme index corresponde un enregistrement composé d'une liste d'adresses indiquant l'endroit, dans le support, où se trouve le document indexé par ce terme. La plupart du système de fichier inversé utilisent le numéro de document comme le pointeur, c'est évident que la longueur de chaque liste est différente. On voit plus clair avec les dessins ci-dessus.



a. fichiers de base



6. fichiers inversés

2. Fichier inversé comme un sujet aux traitements de l'information

Les traitements de l'information dans ce domaine informatique peuvent avoir différentes significations. Selon de Jouffroy et Letang (6), ils sont répartis en deux catégories : les traitements fonctionnels et les traitements de servitude ; tandis que Martin (7) les répartit en trois catégories portant cette fois-ci sur leur mode définie par le temps dont on dispose pour effectuer l'opération et le nombre d'enregistrements affectés par chaque opération.

Comme la signification du "traitement des informations" est très variée, on va le définir simplement comme un processus étant capable de fournir des informations répondant aux questions posés par l'utilisateur. La plupart du système d'interrogation utilisent seulement les termes index accompagnés par les opérateurs booléens (ET, OU, SAUF) pour constituer les questions. Dans l'organisation de fichier inversé, quand on va faire la recherche, le système va faire les comparaisons entre la liste des termes index et les termes dans la question, dès qu'il trouve le terme coordonné il va extraire la liste correspondant (à ce terme), et la mémoriser. Après avoir trouvé tous les termes, on aura les listes correspondant à ces termes. De ces listes, grâce à la question booléenne et quelque programme, on aura la liste des numéros de documents répondant à la question. Tous les processus sont exécutés en quelques milli-secondes, mais si il y a plusieurs termes dans la questions il prendra du temps. Plusieurs efforts sont faits soit par la réduction du volume de données, sans diminuer leur pertinences, ce qu'on verra plus tard, soit par la création d'un système de gestion de fichier spécifique qui permet de diminuer [illegible] d'accès (8).

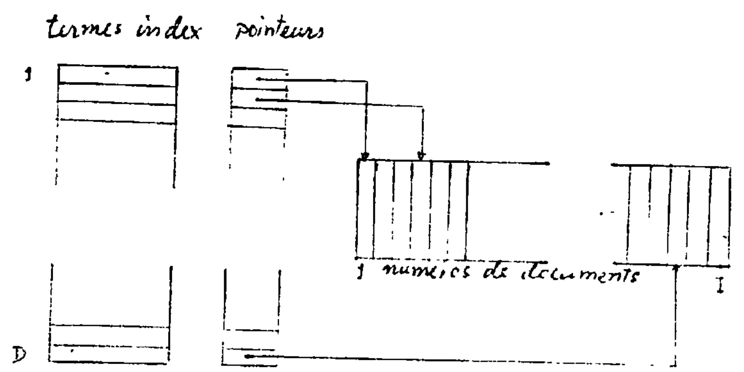
A propos du besoin des gens, il n'y a aucune chose existante dans le monde qui soit tout à fait parfait. Le fichier inversé, lui aussi, a des avantages et des inconvénients. Selon (9, 10, 11, 12), il apparait qu'en fonction des caractéristiques des données à traiter, l'utilisation du fichier inversé convient ou non.

3. Les méthodes de stockage des fichiers inversés

Deux interpretations du fichier inversé sont possible. Ils peuvent affecter à deux formes diferentes de stockage et deux methodes de manipulation (5).

3.1 Listes des numéros de documents

La première interpretations envisage le fichier inversé comme des listes des numéros de documents ce qu'on a déja vu dans la notion du fichier inversé. Si on a D termes index et le nombre total du numéro de documents dans toutes les listes est I, on aura la représentation de ce fichier comme suivante:



Cette représentation est souvent actuellement utilisé car elle est facile à appliquer soit manuellement ou automatiquement.

3.2 Vecteur Binaire ou Vecteur Booléen.

Dans le fichier très grand on voit bien que la liste de numéros de documents consomme beaucoup d'espace sur le support. Pour qu'il soit compacté par quelque techniques, King (12) a représenté la liste de numéros de documents comme un vecteur binaire. Pour chaque terme index, il construit un vecteur binaire zéros de longueur égal au nombre de documents dans le système. Quand un document entre dans le système les zéros dans les positions correspondant au numéro de document dans chaque vecteur de termes index possédant ce document, seront remplacés par les uns. Pour éclaircir on va considérer un exemple en assumant des documents suivants :

Numéro de document	termes pris
1	A, B, D
2	C, E
3	A, C

Les vecteurs binaires pour les termes A, B, C, D et E avant d'entrer dans le système sont comme suivants:

Termes index	Vecteur
A	000 . . 0
B	000 . . 0
C	000 . . . 0
D	000 . 0
E	000 0

Après avoir passé dans le système, les vecteurs deviennent comme suivants:

Termes index	Vecteur
A	101 . 0
B	100 . 0
C	011 0
D	100 . . 0
E	010 . . 0

Le vecteur binaire semble avoir beaucoup plus d'avantages que la liste ou mémoire de documents. D'une part, chaque enregistrement est de longueur fixe car chaque vecteur est égal au nombre de documents, les pointeurs dans la liste peut être éliminé et remplacé par le calcul d'adresse. On peut laisser une espace à la fin du vecteur pour ajouter les nouveaux documents. Et s'il n'y avait pas de limitation de taille de support, on n'aurait besoin qu'une seule accès au enregistrement pour chaque terme. D'autre part, pour la recherche, le plupart d'ordinateurs modernes sont capable de manipuler facilement des chaines binaires et les opérations logiques peuvent se faire par une seule instruction machine.

Hung (12) a dit que la distribution de l'utilisation des [illegible] [illegible] est une fonction logarithmique. Disons simplement que la fréquence d'utilisation d'un terme est d'autant plus faible que le nombre de termes est grand. Ça implique qu'il y aura beaucoup d'octets dans les enregistrements qui sont des bits zéros et par suite les différentes méthodes de compactage étudiées dans la deuxième partie ont été mise en œuvre.

Deuxième Partie

Les méthodes et techniques de compactage des fichiers inversés

Pour compacter le fichier inversé, on va considérer les méthodes par rapport à ses représentations. Dans le cas d'une liste de numéros de documents, le volume du stockage qu'on peut diminuer est seulement les termes index, par exemple, si les index matières sont les termes index on peut les remplacer par quelque code abrégé qui utilise moins d'espace sur le support. Il y a quelques efforts qui sont faits pour résoudre ce problème (13, 14).

Dans le 2ème cas, il y a plusieurs méthodes de compactage de la chaîne de bits-zéros dans le vecteur binaire dont la notion principale est d'essayer de trouver le codage optimal pour ces bits zéros.

1. La méthode de King (12).

King a présenté la méthode élaborée spécialement pour l'ordinateur IBM 360 dont chaque octet contient 8 bits pouvant stocker valeur maximum d'entier de 255. Pour décrire son algorithme il a défini les termes suivants :

sous-vecteur = une chaîne d'octets dans le vecteur binaire

sous-vecteur zéro = un sous vecteur dont chaque octet contient 8 bits-zéros

sous-vecteur nonzéro = un sous vecteur dont chaque octet contient au moins un bit-un.

Processus de Compactage.

1. Partager le vecteur binaire en séries de sous vecteurs zéros et nonzéros, ils peuvent avoir la longueur maximum de 255 octets. Pour le sous-vecteur zéros plus long que 255 octets le 256ème octet va être considéré comme un octet non zéro.

2. Chaque sous-vecteur non-zéro est préfixé par deux octets, le 1er octet contient le nombre d'octet-zéro précédant le sous-vecteur nonzéro de vecteur non-compact, le 2ème octet contient le nombre d'octets nonzéro.

3. Donc le vecteur compact ne compose que les sous-vecteur non-zéro avec les préfixes

4. Le vecteur compact sera terminé par deux octets zéros

Considérez le vecteur binaire suivant comme un exemple:

01100000/10000000/ sept octets zéro/00000001/10000000/...

Ici, on utilise " slash " pour répartir le vecteur en octets. On voit bien que ce vecteur remplace la liste de numéros 2, 3, 9, 80 et 81. Si chaque numéro utilise 3 octets, cette liste donc utilise 15 octets. Avec le processus au-dessus on aura le vecteur compact comme ci-présenté:

00000000/00000010/01100000/10000000/00000111/0000001ϕ/
00000001/10000000/ ..

Dans le vecteur original, les deux premiers octets sont le premier sous-vecteur nonzéro, donc, les 4 premiers octets du vecteur compact peuvent être interprétés comme suivants:

octet 1 : valeur est zéro car aucun octet précédant ce sous-vecteur n'est compacté

octet 2 : valeur deux indique que les deux octets suivants sont le sous-vecteur nonzéro.

octets 3.4 : Ces sont les octets 1 et 2 du vecteur original.

D'octets 3 à 9 du vecteur original sont le sous-vecteur zéro et octets 10 et 11 sont le 2ème sous-vecteur non-zéro. Donc, les 4 deuxièmes octets du vecteur compact sont interprétés comme suivants:

octet 5 : valeur 7 indique que le sous-vecteur zéro de 7 octets est compacté

octet 6 : valeur 2 indique que les deux octets suivants sont le sous-vecteur nonzéro.

octets 7, 8 : Ces sont les octets 10 et 11 du vecteur original.

Le vecteur binaire est diminué de 11 octets à 5 octets, en même temps que la liste est toujours 15 octets. Pour montrer la différence entre la liste et le vecteur binaire, King a pris une base de données de 6121 documents avec 1.484 termes index et le nombre total de numéros de documents dans tous les listes est 94,542.

La liste a utilisé 702,713 octets en même temps que le vecteur binaire compact a utilisé 281,450 octets qui sont égal à 41 pourcent d'espace requise par la liste.

La longueur du vecteur compact va être variable, mais l'effet du compactage n'est pas très grave car on peut le décompacter dans sa forme originale quand on va faire la recherche. En effet, le processus d'expansion du vecteur est relativement simple et parce que les termes index dans la question seulement qui ont besoin de l'expansion, donc le temps de la recherche n'est pas significativement affecté.

Avec la fonction "TIME" de PL/1 il a comparé le temps de la recherche entre le fichier inversé standard et le fichier vecteur binaire. Il a pris 40 questions pour faire l'analyse, il y a 22 questions pour lesquelles le temps de recherche du fichier vecteur binaire est plus rapide, dont une est un facteur de 35. Et 18 questions pour lesquelles le temps de recherche du fichier standard est au plus 6.17 fois plus rapide.

Enfin, il a conclu que le fichier vecteur binaire pourrait économiser l'espace necessaire, le temps de la recherche et également les efforts de programmation.

---

2. La méthode de Schuegraf (15)

Schuegraf a utilisé la méthode de codage des bits-zéro s'appelle "run-length coding" Avant de discuter de son méthode on va expliquer brièvement le "run-length coding".

2.1 Run-length coding.

Le mot "run" signifie une chaîne des bits-zéro terminée par un bit-un, la longueur de "run" est le nombre de ses bits-zéro. Par exemple : 0001 est un run de longueur 3, 000001 est de longueur 5, etc. Selon de (15), 3 méthodes de codage ce run sont discutés, ces sont le méthode de Huffman, de Golomb et "conventional run-length encoding". Je vais prendre le 3ème comme un exemple car c'est facile à comprendre et présenter seulement le tableau de 2ème méthode qui est le plus efficace.

Le processus de codage "conventional run-length" est divisé en 2 étapes (voir le tableau 1).

étape première Chaque run est transformé en sa longueur (L), colonnes 1 et 2 dans le tableau 1 montrent la correspondance entre le run et sa longueur. M étant un nombre entier en forme de 2^N-1 (N est un nombre entier) est choisi d'être la longueur maximum de run (M = 7 dans le tableau 1). Si L est égal à M ou plus longue que M, chaque groupe de M zéros va être considéré comme un run de longueur M et les bits qui restent vont être un autre run. (N est choisi selon le mot)

étape deuxième : Chaque longueur L aura un code qui est la représentation binaire de N-bit de L comme dans la colonne 3.

Tableau 1 : Exemple de codage "conventionnel run-length" avec M = 7

Run	Longueur (L)	Code
1	0	000
01	1	001
001	2	010
0001	3	011
00001	4	100
000001	5	101
0000001	6	110
0000000	7	111

Tableau 2. Exemple de codage de Golomb, m = 4

Longueur	Code
0	0 000
1	001
2	010
3	011
4	4 1000
5	1001
6	1010
7	2^5 1011
8	11000
9	11001
10	11010

Pour le codage de Golomb (Tableau 2), m est un nombre entier tel que $p^m \simeq 0.5$ (p est une probabilité de bit-1). Cet exemple a pris de (16), son méthode de codage est simplifié dans (15).

Avec ces méthodes de codage, Scheugraf a présenté une valeur b, un nombre de bits utilisés pour codifier la longueur du run qui rendre le stockage minimum. b est une fonction de longueur moyenne des run, $r(n,k)$, tel que

$$r(n,k) = \frac{n}{k+1} \quad ;$$

n longueur de vecteur binaire.

k : nombre de bit-1 dans ce vecteur.

Donc, $b(n,k) = \lceil \log_2 \frac{n}{k+1}$

Cependant, les résultats obtenus sont optimum seulement avec un vecteur de composant de n contenant k uns, et pour le fichier inversé dont chaque terme index est utilisé pour décrire le même nombre de documents.

Par suite, il a pris en considération la distribution des termes index en impliquant Loi de Zipf, " Si les termes index D sont rangés dans l'ordre des fréquences descendantes, la fréquence du terme du rang r est

$$f(r) = \frac{C}{r}$$

tel que C est une constante de valeur approximative

$$\frac{I}{\log_e D + 0.577}$$; I : nombre total de numéros de documents.

Dans ce cas, le vecteur binaire n'est pas toujours économique, il est convenable seulement pour les termes de haute fréquence. Mais, pour les termes de fréquence basse les numéros de documents est plus convenable. Schuegraf a réalisé que la combinaison entre les deux représentation serait meilleur. Donc, il doit y avoir un rang, r_0, où la représentation de run-length codage sera remplacée par les numéros de documents.

Avec le calcul mathématique, l'équation pour r_0 est

$$r_0 = \frac{C \log_2 N}{\left(1 - \frac{C}{r_0}\right)\log_2 \frac{N r_0}{C}} - 1$$; N : nombre de documents

La solution peut être résolue par la méthode numérique. Les valeurs de r_0 sont montrées dans les tableaux 3 et 4, avec les graphes 1 et 2. Dans le cas de combinaison entre les deux représentation, le nombre optimum de bits b pour run-length coding est comme suivant :

$$b = \frac{1}{\log 2}\left\{\log\frac{N}{C} - \frac{\frac{\log^2 r_0}{2} + r_0 \cdot C \cdot [\log r_0 - 1] + C}{r_0 + C\cdot(\log r_0 + 0.577)}\right\}$$

Les valeurs de b sont également montrées dans les tableaux 3 et 4, comme les fonctions de C et N, et les graphes 3 et 4. Dans ces graphes, on voit que le nombre de bits pour le codage du run augmente quand le nombre de documents augmente, mais diminue quand le nombre de terme index augmente. Comme b doit être un entier, les valeurs dans les tableaux 3 et 4 doivent être arrondies au entier le plus haut suivant. Toutefois, à cause de la longueur fixe du caractère ou mot de l'ordinateur qu'on va utiliser, la valeur de b va être forcé d'être autre choses. Par exemple, la choix de 8 ou 12 bits pour codifier un run va être convenable pour l'ordinateur ayant un caractère de 8 bits, car tous les ordinateurs ont des instructions spéciales pour manipuler les caractères. Si cette approche est choisi, l'espace de stockage sera sacrifié pour avoir de facilité de la programmation et du traitement.

L'avantage de cette combinaison sera réalisé quand le taux de compactage R, défini par une proportion de la longueur du fichier compact et fichier non-compact, aura été calculé. L'équation de R est suivante

$$R = 1 - \left[1 - \frac{b}{\lceil \log_2 N} \right] \frac{\log r_0}{\log D} \quad ,$$

et ses valeurs sont dans les tableaux 3 et 4. Avec ces valeurs, on voit que pour le fichier inversé contenant un grand nombre de documents et vocabulaires on peut economiser l'espace de stockage de 30 à 40 pour cent.

Dans sa méthode de compactage, Schuegraf n'a rien discuté du temps d'accès pour la recherche. Cependant, l'application de sa méthode ne semble pas très difficile, les gens qui sont habitués au traitement de l'information peuvent faire eux-même la comparaison du temps utilisé entre les deux représentation.

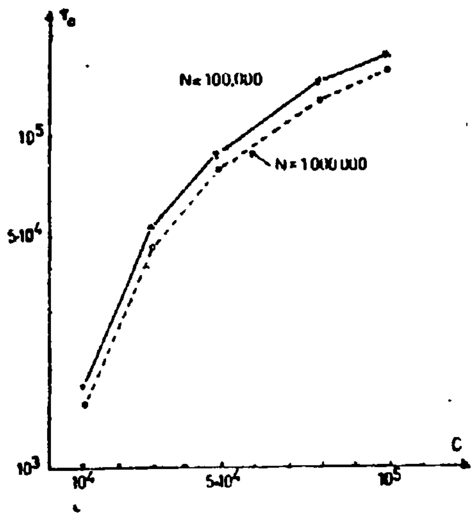
Tableau 3

ZIPF C	N = 100,000 r_0	b	R	N = 1,000,000 r_0	b	R
10,000	1773	8.43	0.707	1561	11 64	0.750
30,000	5314	7.63	0.641	4673	10.85	0 695
50,000	8855	7.26	0.603	7785	10.48	0.654
80,000	14,165	6.92	0.537	12,454	10 14	0.602
100,000	17,706	6.76	0.516	15,566	9.98	0.588

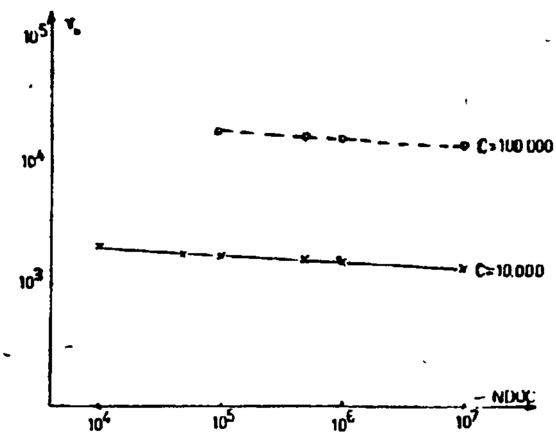
Tableau 4

N	C = 10,000 r_0	b	R	C = 100,000 r_0	b	R
10,000	2069	5.23	0 642	—	—	—
50,000	1852	7 46	0.686	—	—	—
100,000	1773	8 43	0.707	17,706	6 76	0 516
500,000	1618	10 67	0 735	16,145	9.01	0 565
1,000,000	1560	11.64	0 750	15,566	9 98	0 588
10,000,000	1398	14 88	0 780	13,937	13.21	0 675

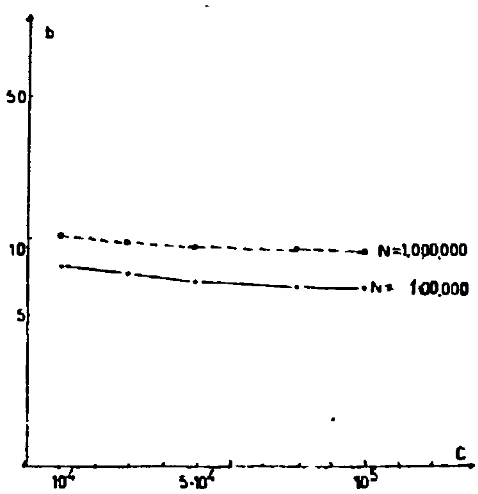
Graphe 1



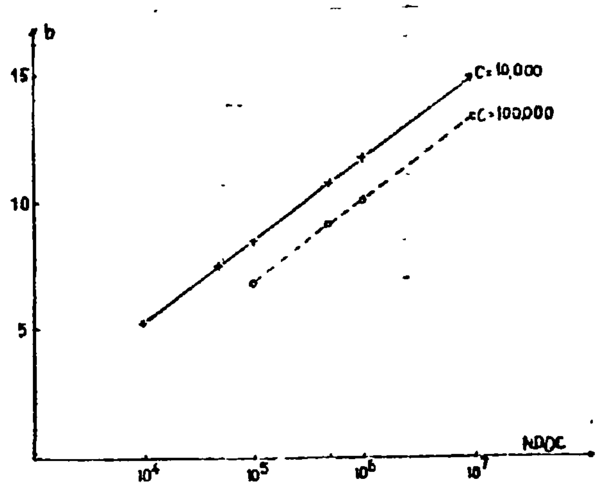
Graphe 2



Graphe 3



Graphe 4



3 Les autres méthodes

Les travaux de comparaisons des performances des techniques de compactage du vecteur binaire montrent que ces techniques sont proches de la technique de run-length coding (17). Parmi ces techniques on trouve la méthode de King, le codage "conventional run-length" et le codage de Golomb, que nous avons déjà expliqués dans les section 1 et 2 de deuxième partie, les autres sont

- Codage de Bradley
- Codage de Wedekind et Härder en un-mot (WH1)
- Modification de codage WH1 (WH1M)
- Codage de Wedekind et Härder en 2-mot (WH2)

et Codage de Thiel et Heaps.

Les chercheurs expliquent ces 8 techniques donnent un modèle stochastique de n états (n-state stochastic model) en utilisant de chaîne de Markov pour générer le vecteur binaire. Les gains résultant de compactage de chaque technique ($C = \frac{\text{vecteur binaire noncompacté}}{\text{vecteur binaire compacté}}$) sont comparés entre eux en supposant qu'il sont une fonction de p (probabilité de bit zéro). La comparaison s'effectue en outre au moyen d'autre paramètre, comme par exemple, la longueur optimale de bits pour le code.

Ainsi dans le tableau 5, les gains sont montrés en supposant que $n = 1$, et la longueur du code équivaut à k bits.

D'après ce tableau, le gain de WH2 est supérieur à 1 lorsque p est petit, tandis que les 3 premiers codages donnent des bons résultats lorsque p est grand. Ces -

Tableau 5.

Technique	p = 0.05	0.50	0.75	0.90	0.95	0.99
Run-length coding	0.53	0.87	1.15	1.98	2.90	11.24
Golomb	0.35	0.62	1.07	1.88	3.36	12.25
Bradley	0.53	0.87	1.15	2.01	3.20	11.59
WH1	0.94	0.94	0.94	1.34	1.93	5.38
WH1M	0.94	0.94	0.94	1.69	2.69	9.80
WH2	$\boxed{1.68}$	0.87	0.88	1.47	2.63	9.52
Thiel et Heape	0.94	0.94	0.94	1.24	2.04	7.37
King	0.94	0.94	0.93	1.01	1.52	5.13

propriétés sont combinées dans WH1M et WH2 qui semblent être efficaces pour toutes valeurs de p.

Conclusion

Ces différentes techniques de compactage sont incontestablement un apport important dans l'interrogation des grands fichiers.

En effet, le volume de ces derniers devenant à la suite du développement de l'automatisation, il est apparu nécessaire d'utiliser des moyens appropriés à la fois à réduire ces volumes et rendre rapide la réponse.

Toutefois, un inconvénient du fichier inversé reste à surmonter. Il s'agit de faire en sorte que le nombre des numéros des documents correspondant aux termes index soit toujours le même.

La solution à cet inconvénient pourrait être trouvé grâce à la méthode de "equifrequent fragments" (18) qui consiste à découper les termes index en plusieurs éléments ; la recherche pouvant alors s'effectuer en combinant ces éléments entre eux au lieu d'utiliser les termes index.

Bibliographie

(1) J. L. PIRNER, A logic per track retrieval system. Proceedings IFIP Congress, P. 711-716, North Holland, Amsterdam (1971)

(2) B. PARHAMI, A highly parallel computing system for information retrieval, Proceedings Fall Joint Computer Conference, P. 681-690 (1972)

(3) J. J. DIMSDALE and H. S. Heaps, File structure for an on-line catalog of one million titles, J. Lib. Autom., 1973, 6, 37

(4) D. Lefkovitz, File Structures for On-line Systems. Spartan Books, New York, (1969)

(5) E. J. Schuegraf, Compression of large inverted files with hyperbolic term distribution, Inform. Processing & Management. 12, P. 377-384 (1976)

(6) C. JOUFFROY, C. LETANG, Les fichiers, DUNOD informatique P. 18-26 (1972)

(7) D. MARTIN, Base de données. DUNOD informatique, P. 89 (1977)

(8) W. H. STELLHORN, An inverted file Processor for Information Retrieval, IEEE Trans. Computers, 26 (12), P. 1258-1267 (1977)

(9) C. T. MEADOW, An Introduction to Information Retrieval, The analysis of Information System, P. 281 (1967)

(10) A. F. CARDENAS. Analysis and Performance of inverted data base structures. Comm. ACM., 1975, 18 (5), P. 253

(11) A. F. CARDENAS, Evaluation and Selection of file organisation - a model and System. Comm. ACM. 1973, 16 (9), P. 540

(12) D. R. KING, The binary vector as the basis of an inverted index file, J. Lib. Automation. 1974, 7 (4), P. 307.

(13) M. CHABRE-PECCOUD. Le Projet Abrégé, Thèse, Université de Grenoble III.

(14) G. LOUIS-GAVET, *Compactage des données structurées*. Thèse, Université Claude Bernard, Lyon I.

(15) L.R. BAHL, H. KOBAYASHI, Image Data Compression by Predictive Coding II, Encoding Algorithms. *IBM. J. Res. Devel.* 1974, 18, P. 172-179

(16) S.W. GOLOMB. Run-length encoding. *IEEE Trans Inform. Theory*. IT-12, 1966, P 399-401

(17) O. NEVALAINEN, M. JAKOBSSON and R. BERG, Compression of Clustered inverted files, *Mathematical Foundations of Computer Science* 1978, P. 393-402, 4-8 SEPT. 1978, Zakopane, Poland.

(18) E.J. SCHUEGRAF and H.S. HEAPS, Selection of equifrequent word fragments for information retrieval. *Inform. Stor.-Retr.* 1973, 9, P. 697