Contribution a l'étude des techniques de compression des données et de leurs domaines d'application

Salwa Abdel Razek

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR-INGENIEUR

CONTRIBUTION A L'ETUDE DES TECHNIQUES DE COMPRESSION DES DONNEES ET DE LEURS DOMAINES D'APPLICATION

Salwa Abdel RAZEK (née ZAKARIA A.W. EL SHARAWY)

Thèse soutenue le 9 novembre 1976 devant la Commission d'Examen

Président : L. BOLLIET

Examinateurs : J.P. ANSART
G. CULLMAN
C. DELOBEL
G. LOUIS-GAVET

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : M. Philippe TRAYNARD

Vice-Président : M. Pierre-Jean LAURENT

PROFESSEURS TITULAIRES

MM. BENOIT Jean	Radioélectricité
BESSON Jean	Electrochimie
BLOCH Daniel	Physique du solide
BONNETAIN Lucien	Chimie Minérale
BONNIER Etienne	Electrochimie et Electrométallurgie
BOUDOURIS Georges	Radioélectricité
BRISSONNEAU Pierre	Physique du solide
BUYLE-BODIN Maurice	Electronique
COUMES André	Radioélectricité
DURAND Francis	Métallurgie
FELICI Noël	Electrostatique
FOULARD Claude	Automatique
LESPINARD Georges	Mécanique
MOREAU René	Mécanique
PARIAUD Jean-Charles	Chimie-Physique
PAUTHENET René	Physique du solide
PERRET René	Servomécanismes
POLOUJADOFF Michel	Electrotechnique
SILBER Robert	Mécanique des Fluides

PROFESSEUR ASSOCIE

M. ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

MM. BLIMAN Samuel	Electronique
BOUVARD Maurice	Génie Mécanique
COHEN Joseph	Electrotechnique
LACOUME Jean-Louis	Géophysique
LANCIA Roland	Electronique
ROBERT François	Analyse numérique
VEILLON Gérard	Informatique Fondamentale et Appliquée
ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM. ANCEAU François	Mathématiques Appliquées
CHARTIER Germain	Electronique
GUYOT Pierre	Chimie Minérale
IVANES Marcel	Electrotechnique
JOUBERT Jean-Claude	Physique du solide
MORET Roger	Electrotechnique Nucléaire
PIERRARD Jean-Marie	Mécanique
SABONNADIERE Jean-Claude	Informatique Fondamentale et Appliquée
Mme SAUCIER Gabrièle	Informatique Fondamentale et Appliquée

MAITRE DE CONFERENCES ASSOCIE

M. LANDAU Ioan	Automatique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

MM. FRUCHART Robert	Directeur de Recherche
ANSARA Ibrahim	Maître de Recherche
CARRE René	Maître de Recherche
DRIOLE Jean	Maître de Recherche
MATHIEU Jean-Claude	Maître de Recherche
MUNIER Jacques	Maître de Recherche

Ce travail a été réalisé au sein de l'Equipe Réseaux d'Ordinateurs et Télétraitement du Centre Interuniversitaire de Calcul de Grenoble.

à ma mère,
à mon mari Adel
qui m'a toujours encouragée,
à ma fille Lobna.

Je tiens avant tout à exprimer ma grande reconnaissance et à rendre un sincère hommage à Monsieur Jean DU MASLE.

C'est Monsieur DU MASLE qui est à l'origine de mon sujet de recherche. Il a guidé mes premiers pas dans le domaine de la recherche, m'a toujours encouragée et a supervisé ce travail sans jamais ménager son temps.

Sa disparition a été une grande douleur pour moi, mais son souvenir m'a permis de garder toujours le courage nécessaire pour continuer mon travail. Je n'aurais pu trahir son sens aigü du travail bien fait qu'il a toujours placé avant tout.

Je tiens à remercier :

Monsieur le Professeur Louis BOLLIET qui m'a toujours apporté son aide durant mon séjour en France et a bien voulu me faire l'honneur de présider mon jury,

Monsieur Guy LOUIS-GAVET, maître assistant à l'IUT de Lyon, qui a bien voulu accepter de superviser mon travail à la disparition de Monsieur DU MASLE ; il n'a ménagé ni son temps ni ses conseils ; son aide efficace m'a redonné confiance et m'a permis de mettre un point final à mon travail,

Monsieur Jean Pierre ANSART, ingénieur au CNRS : je lui dois toute ma gratitude ; il a toujours mis toute sa compétence à répondre à mes nombreuses questions en m'apportant les conseils et les critiques indispensables à la réalisation de cette thèse ; mon travail doit beaucoup à son aide aux moments difficiles,

Monsieur le Professeur Georges CULLMAN et Monsieur le Professeur Claude DELOBEL qui ont bien voulu faire partie de mon jury : je leur présente mes vifs remerciements.

Je voudrais également remercier :

Madame Clotilde CHALAND qui par sa sympathie, ses conseils et son goût du travail bien fait a su rendre à mon travail la lisibilité que ma méconnaissance de la langue française rendait indispensable,

Madame J. DU MASLE, Madame P.J. LAURENT et Madame F. PERRET qui ont bien voulu assurer la traduction française d'une partie de ma thèse,

tous mes collègues du CICG et plus particulièrement Madame FREDENUCCI, Madame SIRET, Monsieur GUILLOU, Monsieur REY et Monsieur SUTY qui m'ont entourée de leur amitié durant mon séjour en France,

et enfin Monsieur IGLESIAS et toute son équipe qui ont réalisé matériellement cet ouvrage.

S O M M A I R E

CHAPITRE I

INTRODUCTION

. INTÉRÊT DU COMPACTAGE.

. LES DIFFÉRENTES MÉTHODES UTILISÉES POUR COMPACTER LES DONNÉES.

. BUTS DE L'ÉTUDE ENVISAGÉE.

. CONCLUSION.

1. INTÉRÊT DU COMPACTAGE

Dans le cadre du traitement de données, celles-ci sont appelées à subir de nombreuses opérations que l'on peut ranger en trois grandes catégories :

1) fonction d'accès par l'utilisateur (c'est-à-dire consultation, mise à jour, suppression, tri, ..) ;

2) fonction de stockage (permanent ou temporaire, rendu nécessaire par la conception des systèmes, comme par exemple les stockages effectués dans un réseau de commutation de messages) ;

3) fonction de transfert de l'information (par exemple, transfert de fichiers dans un réseau, ou plus simplement transfert de la mémoire centrale vers un terminal).

Si nous prenons pour exemple une base de données stockée sur un site d'un réseau d'ordinateurs, les données peuvent :

- être transférées vers un autre noeud,
- y être stockées d'une manière temporaire,
- être utilisées par des utilisateurs grâce à des terminaux.

Ainsi, s'il est évident que l'on doive accéder aux données originelles, il sera possible, grâce au compactage, de manipuler, durant les stockages intermédiaires et les transferts, des données beaucoup moins volumineuses.

On voit clairement à travers cet exemple que le compactage a un champ d'action très étendu.

En l'utilisant dans les fonctions de transfert et de stockage temporaire ou permanent, le compactage permet :

- une économie de place,
- une économie de temps grâce à la possibilité d'augmenter le transfert via une machine de communication (ligne, réseau, canal, ..).

Cela nous donne une idée des domaines variés dans lesquels le compactage peut s'appliquer :

- bases de données,
- réseaux généraux d'ordinateurs,
- réseaux de terminaux légers ou lourds,
- communication sur ligne synchrone,
- systèmes à spool.
- ...

2. LES DIFFÉRENTES MÉTHODES UTILISÉES POUR COMPACTER LES DONNÉES

Les différentes recherches consacrées au problème du compactage ont eu pour but de tirer avantage du fait que :

- d'une part les données présentaient des caractéristiques propres susceptibles de compression (espaces, liens logiques entre elles, probabilités d'apparition, ..),

- et d'autre part, on se servait de codes dont la représentation était bien supérieure au nombre de caractères généralement employés (ASCII : 128 symboles possibles, EBCDIC : 256 combinaisons possibles).

C'est pourquoi les différentes méthodes que l'on essaie d'appliquer pour le compactage de textes consistent à :

1) supprimer les zéros et les blancs dans un texte ; les méthodes les plus utilisées consistent à employer des délimiteurs [20] [17] [1], des segments et des tables binaires descriptives [12], des chaînes [19] et des tabulations logiques [15].

2) tirer avantage des divers liens logiques existant entre les données, permettant ainsi de supprimer certaines redondances [22] ;

3) remplacer des groupes de caractères (bigrammes par exemple) ou l'ensemble des monogrammes par un code [2], [8] ;

4) coder les différentes parties de données suivant leur probabilité d'apparition, le code étant d'autant plus court que la fréquence d'apparition de la donnée considérée est plus grande [6], [18], [24], [22] ;

5) coder les données par groupes de longueur fixe ; celles-ci sont remplacées par des combinaisons chiffrées et additionnées entre elles [13], [14] ;

6) appliquer la théorie de l'information, c'est-à-dire transposer la notion de quantité d'information moyenne de l'état d'un système pouvant prendre n états avec des probabilités différentes, à la notion de quantité d'information moyenne par caractère contenu dans une information écrite dans un langage naturel [26], [27], [4], [24] ;

7) trouver un nouveau code de transmission de caractères, par exemple en représentant tout caractère numérique et signe d'opération par un code à 4 bits, les autres caractères par un code à 5 bits [10] ;

8) élaborer une fonction de compactage très simple en partant d'une étude mathématique théorique sur la répartition des caractères dans un langage structuré [16] ;

9) compacter les données par des codes mnémoniques ou consonnants à l'aide de plusieurs règles appliquées consécutivement [9] ou par des codes phonétiques, chaque code représentant un phonème [11].

Il est évident que cette liste de méthodes n'est pas exhaustive, ne serait-ce que du fait que l'on peut en combiner plusieurs [3], [29].

3. BUTS DE L'ÉTUDE ENVISAGÉE

Nous nous intéressons à mener une étude expérimentale pour donner une base réaliste à une étude théorique sur quelques aspects des méthodes de compactage. Pour ce faire, nous étudierons les méthodes les plus classiques afin de les améliorer et nous expérimenterons de nouvelles méthodes.

En comparant toutes ces méthodes, basées sur une reconnaissance automatique des données, cela nous permettra de trouver la meilleure méthode de compactage pour une classe déterminée de données.

Par ailleurs, nous espérons tirer de cette étude des méthodes de compactage permettant d'atteindre les buts suivants :

1) Utilisation optimale des lignes de transmission. [21]

Par exemple, dans le réseau CYCLADES, les lignes les plus rapides sont de 48 K bits/seconde, ce qui est un handicap important pour les transferts de fichiers ; l'idée est d'augmenter la vitesse apparente de transfert sur ces lignes par un compactage très important.

2) Réalisation d'un stockage géant (bandothègue) sur disque.

L'ordinateur aura deux sortes de mémoire secondaire. La première sur disques classiques, directement accessible par le système d'exploitation (son temps d'accès sera court, sa capacité faible). La deuxième est celle proposée d'un stockage sur disque de données d'un volume très important ; la densité des données étant augmentée par l'utilisation d'algorithmes de compactage très efficaces, l'accès se fera par l'intermédiaire d'un logiciel approprié. Ce système de stockage sera manipulé par un mini-ordinateur et il aura l'avantage entre autres, que les données seront disponibles en permanence (évitant ainsi les pertes de temps dues aux montages ou démontages des bandes magnétiques). La charge de l'unité centrale due au compactage sera déportée sur le mini-ordinateur.

4. CONCLUSION

Pour concrétiser les propos qui précèdent, nous nous proposons

- de décrire brièvement les principales méthodes de compactage utilisées sur le marché (chapitre II) ;

- d'évaluer les performances des plus caractéristiques d'entre elles, de les améliorer et d'en expérimenter de nouvelles, ceci sur tous les langages possibles (chapitre III) ;

- de faire une synthèse des résultats obtenus et une comparaison entre toutes les méthodes expérimentées (chapitre IV) ;

- de créer une relation biunivoque entre un langage déterminé et une méthode de compactage, ceci par une reconnaissance automatique dudit langage (Conclusion).

CHAPITRE II

DESCRIPTION GENERALE

DES PRINCIPALES METHODES DE COMPACTAGE

. INTRODUCTION.

. MÉTHODES BASÉES SUR LA SUPPRESSION DE CARACTÈRES REDONDANTS.

. MÉTHODES BASÉES SUR UNE ÉTUDE STATISTIQUE DU CORPUS DES DONNÉES.

. MÉTHODES BASÉES SUR L'ÉTUDE DES LANGAGES STRUCTURÉS.

. MÉTHODES BASÉES SUR DES CALCULS ARITHMÉTIQUES DE CODES.

1. INTRODUCTION

Pour bien centrer notre propos, nous survolerons dans ce chapitre quelques unes des nombreuses méthodes de compression de données utilisées sur le marché.

Pour plus de clarté, nous les avons regroupées en quatre classes différentes :

1) les méthodes basées sur la suppression des caractères redondants,

2) les méthodes basées sur une étude statistique du corpus des données,

3) les méthodes basées sur l'étude des langages structurés,

4) les méthodes basées sur des calculs arithmétiques de codes.

2. MÉTHODES BASÉES SUR LA SUPPRESSION DES CARACTÈRES REDONDANTS

2.1. Présentation générale de ces méthodes

Cette classe de compacteurs comprend une grande variété de programmes qui suppriment soit les blancs, soit les zéros, soit les blancs et les zéros. Ce sont les plus simples des programmes de compactage et leur usage est largement répandu.

La suppression des valeurs nulles exploite une des caractéristiques de la plupart des fichiers de données, à savoir la prépondérance des blancs et des zéros. Elle est facile à concevoir, aisée à mettre en oeuvre et réussit généralement à atteindre un taux raisonnable de compression pour un coût relativement bas en temps unité centrale.

La suppression des valeurs nulles peut être organisée facilement comme programme standard pouvant s'appliquer à des fichiers de nature très différente. Par contre, cette méthode n'atteint en général pas un taux de compression aussi élevé que d'autres techniques.

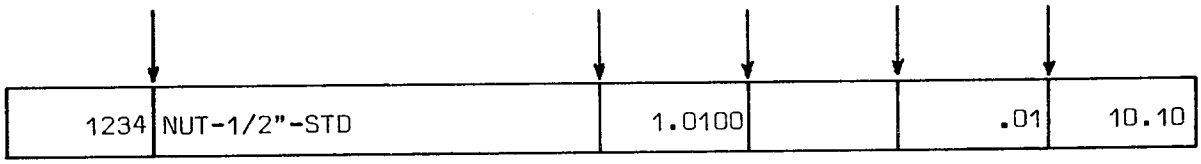
Notons cependant qu'une prépondérance de blancs et de zéros n'est pas forcément signe d'une mauvaise mise en forme (organisation) des données. Elle apparaît parce que :

- il a fallu prévoir de la place pour une valeur maximale, mais la valeur effectivement rencontrée n'était pas de taille maximale,
- de la place a été prévue pour une information qui, au moment précis, n'est pas applicable, ou bien n'est pas disponible, mais peut être utilisable soit plus tôt, soit plus tard,
- de la place a été prévue pour une information qui ne s'applique pas à tous les enregistrements.

2.2. Méthode des tabulations logiques

Cette technique de compactage [15] essaie de tirer profit du fait qu'une proportion élevée de caractères faisant défaut peut se rencontrer. Un tel programme est utilisé par l'IBM 2780. Dans celui-ci des tabulations sont disposées à intervalles réguliers le long de l'enregistrement.

Considérons par exemple l'image de l'enregistrement suivant, où des tabulations sont effectuées aux endroits désignés par une flèche.



Les blancs à gauche d'un marqueur de tabulation sont remplacés par un caractère spécial, par exemple Ø. Dans le cas ci-dessus, on obtiendra alors le format compacté suivant :

␣␣ 1234ØNUT-1/2"-STDØ ␣ 1,0100ØØ ␣␣␣ .01Ø ␣ 10.10

Pour retrouver le format original de l'enregistrement, le marqueur de tabulation (qui est un caractère spécial de contrôle) est interprété en combinaison avec un format de contrôle horizontal de l'imprimante, de sorte que :

a) si le marqueur apparaît en tête de l'enregistrement et est suivi par un autre caractère particulier de contrôle, cela signifie que le reste de l'enregistrement sera un format de contrôle horizontal de l'imprimante.

b) Si le marqueur apparaît à l'intérieur du format de contrôle de l'imprimante, il provoque un stop électronique tabulé de l'imprimante.

c) Si le marqueur apparaît dans un enregistrement ordinaire, les données qui le suivant seront imprimées à partir de la tabulation suivante, effectuée par l'enregistrement de contrôle de l'imprimante.

Par rapport à d'autres méthodes, celle-ci présente l'inconvénient :

1) de particulariser le caractère "blanc", seul compacté,

2) de se servir d'un caractère spécial pour la tabulation logique (Ø).

2.3. Méthode par troncation

Une autre méthode de compression supprimant les "blancs" est celle utilisée par le Michigan Terminal System [17].

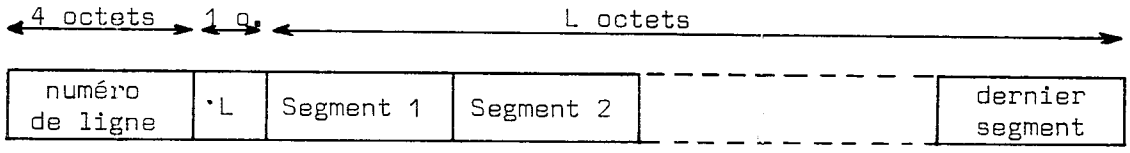
Ici on supprime tous les blancs à la fin d'un enregistrement. Cette méthode part de l'idée que dans les enregistrements à longueur fixe (par exemple, les 80 caractères d'une image de carte perforée), pratiquement tous les caractères sur la droite de la carte sont identiques, à savoir des blancs.

On remarque toute de suite que cette méthode ne réalise pas d'économie de place dans le cas d'enregistrements sérialisés (la sérialisation des cartes se trouve dans les colonnes 73 à 80).

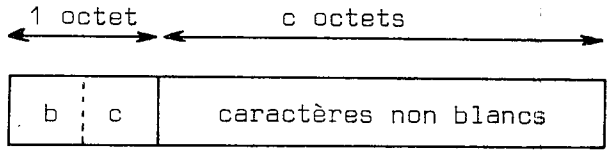
2.4. Méthode par segmentation

Dans le système d'édition de textes (WYLBUR) [12], la compression s'applique aux blancs à l'intérieur, aussi bien qu'à la fin des enregistrements. Les blancs à l'intérieur sont comprimés, tandis que les blancs en fin d'enregistrement sont tronqués. Chaque ligne de texte est divisée en un nombre de segments nécessaires. Un segment peut décrire jusqu'à 15 blancs consécutifs suivis, en plus, de 15 caractères non blancs ; si plus de 15 blancs consécutifs ou non blancs consécutifs apparaissent, alors des segments multiples sont créés. La figure a. montre la technique de compression.

a. Image d'un enregistrement compacté :



b. Constitution d'un segment :



Un segment consiste en un octet compteur éclaté en un compteur à 4 bits pour les blancs et un compteur à 4 bits pour les non blancs, suivi des caractères non blancs (voir figure b.).

Le numéro de la ligne et le compteur du nombre total d'octets de tous les segments sont placés en tête de chaque ligne. Dans certains cas, le format compacté peut en fait être plus long que la ligne elle-même. Dans ce cas là, la ligne n'est pas compactée et un indicateur spécial est placé pour le signaler.

Ceci semble une bonne méthode, où manque cependant la compression des caractères non blancs. Elle est adéquate pour la compression de langages machine, pourvu qu'il n'y ait pas beaucoup de commentaires.

2.5. Méthode par délimiteur

La méthode HASP [1, 19] qui fait partie du système opérationnel de l'IBM 360, utilise un algorithme proche de celui de WYLBUR, mais sait reconnaître des chaînes de symboles identiques et des longues chaînes de blancs, et les traite de façon particulière.

Le système HASP est utilisé pour les lignes téléphoniques (procédure par multi-entrelacements, "multi-leaving procedure").

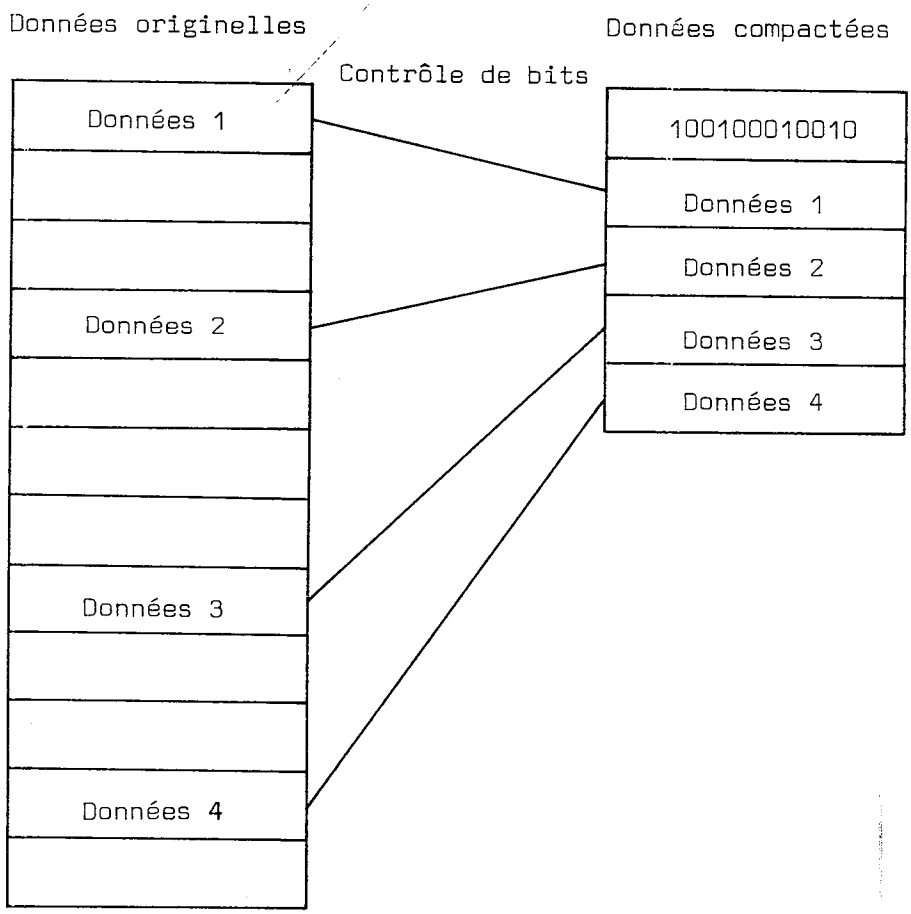
2.6. Méthode par représentation de tables binaires

La compression par suppression des zéros pourrait se faire par "bit mapping" (contrôle de la correspondance des bits) [18] : on fait précéder d'un contrôle de bit le début de l'enregistrement : un 1 dans le contrôle de bit indique qu'il y a des données dans cette position de l'enregistrement ; un zéro signale l'absence de données.

Les unités contenant uniquement des zéros sont supprimées de l'enregistrement pendant la compression et ré-insérées au cours de la décompression. Les unités de taille fixe représentées par chaque bit dans le contrôle des bits peuvent être de longueur quelconque.

Sur les machines à mots, il est commode d'utiliser des mots comme unités, puisque cela permet une compression extrêmement rapide. Des unités plus petites, telles que les demi-mots, caractères hexadécimaux ou octaux, améliorent certes légèrement le taux de compression, mais coûtent cher en temps unité centrale à cause du nombre plus important d'itérations.

Il est également possible d'utiliser la compression par contrôle de la correspondance des bits là où les bits représentent des champs de données. Dans ce cas, le compresseur doit avoir accès à une table donnant la longueur des champs successifs à l'intérieur de l'enregistrement.
La figure qui suit montre un exemple de suppression de zéros par contrôle de la correspondance des bits.



Compression des données par "bit mapping"

La technique de la longueur courante, bien connue, constitue un autre type de suppression des valeurs nulles ("null suppressor"). La figure qui suit illustre cette technique : un signe de ponctuation est inséré afin de signaler une longueur courante de zéros, suivi d'un nombre précisant sa longueur. #N se substitue dans la chaîne de données là où il y avait une chaîne de caractères nuls plus longue que deux - n représentant le nombre de zéros ou de blancs remplacés, moins deux, pourvu que l'ensemble des caractères comprenne des caractères qui n'apparaissent pas dans les données.

Si l'ensemble des caractères ne comprend pas de caractère inutilisé, comme c'est le cas pour les machines à caractères à 6 bits, la technique peut quand même être employée à condition de choisir comme caractère de ponctuation un caractère utilisé rarement et de le doubler chaque fois où il apparaît faisant partie des données.

Codage des longueurs courantes :

Données originales :

A1E0000X000456000000N00000

Données compactées :

A1E#2X#1456#4N#3

3. MÉTHODES BASÉES SUR UNE ÉTUDE STATISTIQUE DU CORPUS DES DONNÉES

3.1. Introduction

Ces méthodes sont basées sur le fait que si l'on considère un langage déterminé, il ne sera construit qu'à partir d'un nombre fini d'éléments suivant une structure déterminée, soit dans celui du langage lui-même, soit dans la représentation globale au niveau de l'enchaînement des données. Ce qui fait que nous sommes sûrs d'avoir des redondances (groupes de caractères qui se répètent).

Ainsi, l'ensemble de ces méthodes de compactage est-il basé sur l'étude de ces redondances. L'ennnui de ces méthodes se situe dans le fait que, pour chaque nouveau fichier, il faut construire la table des redondances, ce qui demande beaucoup de temps.

Cependant, comme nous le verrons dans la conclusion, on pourra créer a priori une table de référence de ces redondances. Ainsi, nous ne calculerons plus systématiquement cette table des fréquences. Il suffira alors de ne soustraire que le temps de transfert de ces tables d'un volume très réduit (256 mots).

3.2. Méthode par substitution

La substitution des schémas ("pattern substitution") [18] est un type de compacteur indiqué particulièrement pour les fichiers de gestion, puisqu'ils comportent souvent des schémas stéréotypés réapparaissant sans cesse dans les enregistrements. Ils peuvent contenir des informations numériques aussi bien qu'alphabétiques, combinées avec, ou bien ajoutées, aux chaînes inévitables de zéros et de blancs.

Le compacteur peut être conçu de telle sorte qu'il parcourt les données et construit sa propre table de schémas en conséquence, ou bien la table peut être préparée d'avance.

La table des schémas peut être stockée, ou bien être transmise ensemble avec les données compactées, ou encore elle peut être conçue comme formant partie permanente aussi bien du compacteur que du dé-compacteur.

Comme c'était le cas pour la suppression des valeurs nulles, il s'avère pratique ici de se servir des caractères inutilisés de l'ensemble des caractères, s'il y en a.

La figure suivante montre comment une telle table de schémas apparaissant souvent, peut être utilisée pour le compactage des données.

Données originales :

AE10004MFQ00000F320006BCX4

AE20000DBF00000F300000BCX1

AE30002RBA00000F301214BCX7

AE40006MDC00000F373000BCX4

Table des schémas :

AE = #

000 = $

00000F3 = %

BCX = @

Données comprimées :

#1$4MFQ%2$6@4

#2$0DBF%$00@1

#3$2RBA%01214@7

#4$6MDC%73$@4

Par la substitution de schémas, on atteint un taux de compactage plus grand que par la suppression des valeurs nulles, puisque les schémas contribuent également à la compression des chaînes de zéros et de blancs.

Toutefois, le rendement effectif constitue un problème sérieux en ce qui concerne les compacteurs par substitution des schémas, car il faut effectuer de nombreuses comparaisons entre les données et la table des schémas, avant de pouvoir identifier un schéma.

Les schémas sont de longueur variable et peuvent être des sous-ensembles d'autres schémas.

La table des schémas peut être organisée sous forme arborescente pour réduire le temps nécessaire à l'identification d'un schéma.

3.3. Méthode de Brillouin

Dans les différents systèmes, les caractères sont toujours représentés par un nombre invariable ou constant de bits. Du point de vue économique cela n'est pas pratique, car la fréquence statistique d'utilisation peut être mise à profit en choisissant des représentations ayant un nombre variable de bits.

Le codage statistique exploite la probabilité de rencontre d'une unité de message, de sorte à utiliser des représentations courtes pour des caractères rencontrés fréquemment et des représentations longues pour des caractères rencontrés peu souvent.

Le code MORSE, par exemple, utilise des groupes de codes courts pour les lettres courantes et des groupes de codes plus longs pour celles qui sont plus rares.

La sténographie digitale de Bemer [2] est une version du même concept où les unités sont des mots plutôt que des caractères.

Quand on utilise des uns et des zéros binaires pour la représentation d'un message en code de longueur variable, il faut avoir un moyen d'indiquer la fin d'un caractère et le début du prochain. Ceci peut se faire, au choix, par :

1) auto-définition, où les n premiers bits d'un groupe indiquent sa longueur,

2) détermination par contrôle de chaque $n^{ième}$ bit (ou des $n^{ièmes}$ bits), dans un seul track,

3) détermination par reconnaissance d'un schéma à bits fixe, ne faisant normalement pas partie du code,

4) un code ayant des propriétés préfixables.

La troisième méthode est dûe à Brilloin : chaque représentation est reconnaissable par au moins deux bits zéro consécutifs, suivis d'un bit 1. L'exemple de la chaîne ci-dessous illustre cette méthode :

100111 00 101101110100 00 1001010 001

Cette méthode a l'avantage de posséder une possibilité d'autoréparation après une erreur de transmission (sauf pour le mot dans lequel l'erreur s'est produite).

Elle ne convient guère pour la transformation par calcul de grands dictionnaires, parce que toutes les combinaisons d'adressage ayant deux zéros consécutifs sont exclues a priori.

3.4. Méthodes de Shannon - Fano et Huffman [26], [28]

Les codes Shannon - Fano et Huffman sont des codes ayant des propriétés préfixables. Ils sont construits de manière à satisfaire des conditions de longueur de code moyenne minimale, donc de codes optimaux.

Les codes Shannon - Fano s'assemblent par arrangements de l'ensemble des caractères selon leur probabilité d'occurence. Cet ensemble est divisé en deux parties, de sorte que la somme des probabilités de la première partie soit égale à celle de la deuxième partie. Ensuite, un bit zéro est attribué à l'une des parties, et un bit un à l'autre. Cette opération est répétée séparément pour chaque partie jusqu'à l'épuisement des divisions (chaque division ne contient pas plus d'un élément). Le code qui en résulte a la propriété d'être préfixable, c'est-à-dire qu'aucun groupe de codes courts n'est reproduit en tant que début d'un groupe plus long.

Les codes Huffman ont également la propriété d'être préfixaux et ils sont en plus des codes à redondance minimale. Ils sont optimaux dans le sens que des données ainsi codées ne sauraient être exprimées en moins de bits.

Nous introduirons des bases théoriques pour les codes binaires de Shannon - Fano et de Huffman au chapitre suivant. Toutefois, rappelons brièvement que des codes de Huffman sont établis en commençant par une liste de caractères arangés selon leur probabilité d'occurence. Les deux caractères ayant les probabilités les plus faibles sont alors sélectionnés et on attribue zéro à l'un et un à l'autre. Leurs probabilités sont ensuite additionnées et le groupe combiné est incorporé dans la liste restreinte, en se basant sur leur nouvelle probabilité combinée. La procédure se poursuit jusqu'à ce qu'il ne reste qu'un seul groupe.

Les codes Huffman produisent une compression modérée avec une dépense en temps unité centrale relativement modérée et ils utilisent peu de mémoire centrale.

Du côté négatif, il faut noter que les codes de Huffman perdent de leur efficacité si les propriétés statistiques du fichier changent en cours de route. Ils sont donc surtout appropriés pour des fichiers permanents à contenu stable, comme nous le verrons dans le chapitre III.

3.5. Méthode de Snyderman et Hunt

Snyderman et Hunt [25] ont conçu un compacteur qui se base sur les propriétés statistiques des caractères.

Leur conception se sert du fait que la plupart des données n'utilisent que 88 sur les 256 représentations possibles dans le code IBM EBCDIC à 8 bits, à savoir 52 caractères alphabétiques minuscules ou majuscules, 10 caractères numériques et 26 symboles spéciaux tels que virgule, signe dollar, point, ...

L'ensemble des 88 caractères utilisables est divisé en trois catégories, dont la première comprend les caractères les plus fréquents, appelés "caractères maîtres". La deuxième catégorie comprend des caractères moins fréquents, appelés "caractères combinés", parce que, au cours du processus de compactage, ces caractères seront combinés avec des caractères maîtres. Notons que les caractères maîtres participent aussi bien en tant que caractères maîtres que de caractères combinés. La troisième catégorie comprend les caractères les moins utilisés, il s'appellent "caractères non combinés" et ne seront pas compactés.

Le compacteur est une combinaison d'opérations en trois étapes :

1) chaque caractère du texte est traduit dans un code hexadécimal alterné,

2) certains couples de lettres sont combinés de telle manière qu'une seule valeur hexadécimale représente les deux,

3) la représentation en caractères simples qui en résulte remplace l'originale dans le texte qu'on stocke en mémoire.

Au cours de la première étape, chaque caractère EBCDIC dans le texte est traduit en son code compacté correspondant à des symboles simples. Ceci utilisera les codes de hex00 jusqu'à 57, laissant donc 168 représentations contigües à la fin de la gamme, à savoir 58 jusqu'à FF.

La véritable compression s'obtient alors en testant, l'un après l'autre, chaque caractère traduit pour déterminer s'il fait partie des "caractères maîtres". Ceux-ci sont utilisés comme premier élément des couples de lettres combinées, tandis que les caractères combinés constituent le deuxième élément du couple.

Si, dans le texte d'entrée, un caractère est un caractère maître, le programme détermine si le caractère qui suit immédiatement est un des caractères combinés.

Après avoir détecté un caractère maître suivi d'un caractère combiné, la valeur hexadécimale représentera le couple de caractères. Ce couple est rapidement déterminé en additionnant la valeur de base du caractère maître à la valeur hexadécimale attribuée au caractère combiné. La configuration hexadécimale obtenue est alors stockée dans l'emplacement disponible le plus proche dans le texte de sortie.

Si le caractère qui suit immédiatement un caractère maître n'est pas un caractère combiné, alors les représentations hexadécimales traduites pour chacun sont placées, l'une après l'autre, dans les emplacements disponibles les plus proches dans le texte de sortie, sans qu'il y ait de combinaison. C'est alors la position suivante qui est testée en vue de déterminer un caractère maître.

Si le caractère examiné n'est pas un caractère maître, sa représentation hexadécimale est transférée dans l'emplacement disponible le plus proche dans le texte de sortie, sans aucun changement.

Le tableau qui suit (page suivante) montre un exemple des différents ensembles de caractères avec les valeurs qui leur ont été attribué.

La base de cet algorithme est intéressante, mais le taux d'économie de place qu'il réalise est relativement faible. La raison en est qu'il dépend de la reconnaissance d'un couple de caractères "maitre - combiné", même si tout le fichier est constitué de caractères maîtres suivis de caractères combinés. Le gain en place théorique ne dépassera pas 50 % du volume initial (en pratique, il n'atteindra pas plus de 35 %).

Dans un même ordre d'idée, mais d'une façon beaucoup plus simple, Donio [10] a créé un nouveau code de caractères, en constatant que dans tous les fichiers de gestion, un code à 5 bits était suffisant. Il a créé un ensemble de sous-classes (numérique, alphabétique, ..) avec un nombre de caractères inférieur à 32 dans chaque sous-classe. Les bits de début nous permettant de reconnaître la sous-classe que nous manipulons.

Caractères maîtres		Caractères combinés		Caractères non-combinés						Couples combinés			
Symbole	Valeur de base	Symbole	Hexa code	Symbole	Hexa code	Symbole	Hexa code	Symbole	Hexa code	Symbole	Hexa code	Symbole	Hexa code
Ø	58	Ø	ØØ	J	15	q	2B	<	41	ØØ	58	AØ	6D
A	6D	A	Ø1	K	16	r	2C	(	42	ØA	59	AA	6E
E	82	B	Ø2	Q	17	s	2D	+	43	ØB	5A	AB	6F
I	97	C	Ø3	X	18	t	2E	&	44	ØC	5B	AC	7Ø
O	AC	D	Ø4	Y	19	u	2F	!	45	ØD	5C	↓	↓
N	C1	E	Ø5	Z	1A	v	3O	$	46	ØE	5D	AW	81
T	D6	F	Ø6	a	1B	w	31	*	47	ØF	5E	EØ	82
U	EB	G	Ø7	b	1C	x	32	)	48	ØG	5F	EA	83
		H	Ø8	c	1D	y	33	;	49	ØH	6O	↓	↓
		I	Ø9	d	1E	z	34	-	4A	ØI	61	EW	96
		L	ØA	e	1F	Ø	35	/	4B	ØL	62	IØ	97
		M	ØB	f	2O	1	36	,	4C	ØM	63	↓	↓
		N	ØC	g	21	2	37	%	4D	ØN	64	Ob	AC
		O	ØD	h	22	3	38	_	4E	ØO	65	↓	↓
		P	ØE	i	23	4	39	>	4F	ØP	66	NØ	C1
		R	ØF	j	24	5	3A	?	5O	ØR	67	↓	↓
		S	1O	k	25	6	3B	:	51	ØS	68	TØ	D6
		T	11	l	26	7	3C	#	52	ØT	69	↓	↓
		U	12	m	27	8	3D	@	53	ØU	6A	UØ	EB
		V	13	n	28	9	3F	'	54	ØV	6B	↓	↓
		W	14	o	29	$	3F	=	55	ØW	6C	VW	FF
				p	2A	.	4O	"	56				
								<	57				

4. MÉTHODE BASÉE SUR L'ÉTUDE DES LANGAGES STRUCTURÉS

Méthode de Louis-Gavet

Toutes les méthodes décrites précédemment (§ 3), sont dépendantes du corpus des données. Cela implique que, pour chaque fichier, nous avons à construire la table des redondances liées à une étude statistique des données contenues dans ce fichier.

La méthode de Louis-Gavet [16] consiste à lier la fonction de compactage à la structure du langage. Ainsi, cette méthode de compactage n'est-elle plus liée à un corpus déterminé.

Pour ce faire, il a démontré que la répartition des caractères (monogrammes), ou groupes de caractères (bigrammes, trigrammes), dans les langages structurés, suivait la répartition des lois statistiques classiques.

De cette façon, la fonction de compactage sera identique quelles que soient les données du fichier. Cette fonction de compactage consiste à prélever des caractères à intervalles réguliers.

Les résultats obtenus sont très performants lorsque l'information recherchée est connue au départ (interrogation des bases de données, par exemple). Le gain de compactage peut être de l'ordre de 95 % ; cela permet de trouver des structures de fichier originales.

En ce qui nous concerne, nous avons étudié cette méthode dans le cas où nous voulions retrouver les données originelles.

Comme nous le verrons, c'est une méthode intéressante pour les gros fichiers.

5. MÉTHODE BASÉE SUR DES CALCULS ARITHMÉTIQUES DE CODES

Méthode de Hahn

Cette méthode [14] combine trois avantages :

. suppression de caractères redondants,

. calcul des occurences des différents caractères pour construire un dictionnaire,

. liaison entre les caractères et leur place dans le dictionnaire.

De cette façon, il suffit pour compacter un ensemble de caractères, de créer une formule simple de calculs arithmétiques entre la représentation numérique des caractères donnée par leur position dans le dictionnaire ; cette formule aboutit à un nombre de longueur fixe.

Ainsi, un groupe de caractères ayant les positions P1, P2,...,Pn sera codé comme un nombre unique par la formule suivante (B étant le nombre de caractères dans le dictionnaire) :

$$(P1 \times B^{N-1}) + (P2 \times B^{N-2}) + \ldots + P_{n-1} \times B + P_N$$

Cette méthode est élégante et donne de bons résultats.

CHAPITRE III

ETUDE DES EXPERIENCES ET EVALUATION DES METHODES DE COMPACTAGE

RÉALISÉES EN VUE DE LEUR APPLICATION SUR DES LANGAGES DÉTERMINÉS

. INTRODUCTION. LES DIFFÉRENTES MESURES EFFECTUÉES.

. ÉVALUATION SUR LES MÉTHODES CLASSIQUES.

. ÉVALUATION SUR LES MÉTHODES COMBINÉES.

. ÉVALUATION SUR LA MÉTHODE DE CHANGEMENT DE BASE DE NUMÉRATION.

. CONCLUSION.

1. INTRODUCTION. LES DIFFÉRENTES MESURES EFFECTUÉES

Ce chapitre présente une étude expérimentale des algorithmes se rapportant aux différents schémas de compactage.

Nous décrivons les différentes techniques et nous évaluons l'efficacité, les performances, les avantages ainsi que les inconvénients :

1) des méthodes de compactage les plus performantes utilisées à l'heure actuelle,

2) des nouvelles méthodes, dites "combinées", utilisant et améliorant différentes méthodes que nous avons jugées complémentaires,

3) une nouvelle méthode basée sur le changement de base de numération.

Remargue :

Ces mesures sont effectuées sur un IBM 360/67.

Afin de comparer les différentes méthodes de compactage avec le plus de rigueur possible, nous calculerons :

- d'une part le temps moyen de compactage (TCM) et le temps moyen de décompactage (TDM),

- d'autre part le rapport $\frac{\text{gain réel}}{\text{gain théorique}}$ $(\frac{GR}{GT})$ pour les différents fichiers.

N.B. La méthode la meilleure est celle qui donne le meilleur rapport $\frac{GR}{GT}$ pour le temps moyen de compactage le plus faible possible.

Temps de compactage moyen (TCM) :

On mesure, sous ASP/MVT, le temps de passage des deux programmes suivants :

P1 :

```
┌──►
│ compacter enregistrement IN
│ écrire enregistrement (OUT)
└──┘
```

P2 :

```
┌──►
│ MVC OUT, IN
│ écrire enregistrement (OUT)
└──┘
```

$$\frac{\text{temps (P1) - temps (P2)}}{\text{nombre d'enregistrements du fichier}} = \text{temps de compactage moyen}$$

Pour un algorithme particulier, le temps de compactage moyen dépend bien sûr du type de fichier et de la longueur de ses enregistrements.

Temps de décompactage moyen (TDM) :

On le mesure de la même façon que le temps de compactage moyen.

Gain de place :

Considérons le fichier dont (IN) représente l'enregistrement à compacter, et (OUT) l'image de l'enregistrement compacté ; (i) est le nombre d'enregistrements dans ce fichier ; le gain réel GR en pourcentage est :

$$GR\ (\text{en } \%) = \frac{\sum_{i=1}^{i=nb\ enreg.} [\text{longueur}(IN_i) - \text{longueur}\ (OUT)_i]}{\sum_{i=1}^{i=nb\ enreg.} \text{longueur}\ (IN_i)} \times 100$$

Gain maximal théorique :

L'entropie de la source, ou information moyenne par symbole (I_i), est, par définition de Shannon & Wiener [24] :

$$I_i = - \sum_{i=a}^{i=n} P(i) \log P(i) \text{ bits par symbole}$$

$i = a, b, c, \ldots, n$

D'où l'information contenue dans un message constitué d'un nombre M de symboles :

$$I = - M \sum_{i=a}^{i=n} P(i) \log P(i) \text{ bits}$$

Comme I représente la quantité d'information dans ce message, l'information ne peut pas être représentée par un nombre de bits inférieur à I ; on considère donc que I est la taille minimale du message.

De point de vue compactage, si l'on peut compacter l'information à sa taille minimale théorique (V_{mnt}), on obtient le gain maximal possible ou le gain maximal théorique (G_{mxt}).

$$\frac{\text{taille initiale - taille minimale théorique}}{\text{taille initiale}} \times 100 = \text{gain maximal théorique}$$

Mesure de l'efficacité d'une méthode :

Nous comparons le gain réel au gain maximal théorique pour connaître l'efficacité de compactage. Ainsi, pour chaque méthode, nous calculerons

V_i	la taille initiale des données
V_{mnt}	la taille minimale théorique des données
G_{mxt}	le gain maximal théorique (en pourcentage)
V_c	la taille des données compactées
G_R	le gain réel en place (en pourcentage)
$\frac{G_R}{G_{mxt}}$	le facteur d'efficacité
TCM	le temps de compactage moyen par enregistrement (en micro.sec)
TDM	le temps de décompactage moyen par enregistrement (en micro secondes).

2. ÉVALUATION SUR LES MÉTHODES CLASSIQUES

2.1. Méthode de compactage par troncation : CHOPPING

2.1.1. Description de la méthode

Dans cette méthode, un enregistrement compacté a le format suivant :

<enregistrement> := <chaîne>|<chaîne><enregistrement>
<chaîne > := <SCB1><SCB2><chaîne de caractères>
SCB1 = 'xxxxxxxx' longueur initiale de l'enregistrement en cas de longueur variable
SCB2 = 'xxxxxxxx' enregistrement compacté ; contient xxxxxxxx caractères (8 bits) ; longueur maximale égale à 255 caractères.

Par exemple, partant d'un enregistrement de caractères de longueur fixe (par exemple 80 caractères d'une image de carte), on constate qu'en pratique tous les caractères de droite de la carte sont identiques.

1 10 20 28 80
CECI EST UNE IMAGE DE CARTE ␣ | | ␣ ␣

L'information contenue dans la carte ci-dessus est conservée si l'on garde seulement les 28 premiers caractères et le compte effectif des caractères conservés.

28 | CECI EST UNE IMAGE DE CARTE ␣

Sachant que la longueur originelle de la carte est de 80 caractères, on conçoit qu'il est très facile de reconstituer la carte origine par duplication du 28ème caractère 52 fois.

Pour appliquer cette méthode à des enregistrements de longueur variable, il est nécessaire de garder en mémoire la longueur origine de l'enregistrement (ici 80)

| 80 | 28 | CECI␣EST␣UNE␣IMAGE␣DE␣CARTE␣ |

Cette méthode, dans cette forme, n'apporte aucun profit dans le cas d'une image de carte sérialisée en colonnes 73 à 80.

80

CECI EST UNE IMAGE DE CARTE␣ || XYZ12345

sérialisation

Pour rendre la méthode utilisable dans ce cas, il vient immédiatement à l'idée de modifier l'ordre des colonnes de façon, par exemple, à reporter la sérialisation à gauche :

| XYZ12345 CECI EST UNE IMAGE DE CARTE␣ |

Un fichier d'image de carte sérialisée n'est qu'un cas bien particulier de la disposition de l'information sur une carte, puisqu'on remarque qu'à chaque langage, pour chaque programmeur, correspond une distribution particulière des caractères dans la carte, et comme résultat, il y a des colonnes qui sont très peu utilisées. Ceci nous a conduit à étudier la façon de réordonner les colonnes de la carte en fonction de leur "utilisation". Cette étude sera développée plus loin (p. 104).

2.1.2. Algorithme de compactage / décompactage

Voir Annexe 1.

2.1.3. Résultats obtenus

Voir tableau page suivante.

La méthode s'applique bien avec les fichiers de type source (Assembleur, Cobol, ..), avec un temps de compactage et décompactage relativement bas.

Cette méthoce n'est pas intéressante pour des fichiers de type numérique ou module objet, la raison étant, bien entendu, la non existence du caractère blanc dans ces fichiers ; elle ne l'est guère pour des fichiers de gestion.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	337824	4.08	49 %	0.62	304	183	80
Source Algol W	457600	62066	1.09	86 %	178464	3.12	61 %	0.70	302	186	80
Source Cobol	763520	156589	1.64	79.5 %	358240	3.80	53 %	0.67	352	190	80
Module Objet	48000	18214	3.03	62 %	44210	7.30	8 %	0.1	297	248	80
Numérique	583040	140183	1.92	76 %	583064	8.00	0 %	0.0	278	187	80
Gestion	730000	226213	2.48	69 %	612160	6.70	16 %	0.21	279	192	80

Résultats de la méthode de troncation "chopping"

2.2. Compactage par réduction des caractères répétés

2.2.1. Description de la méthode

L'idée de base de cette méthode consiste à remplacer une succession de n caractères identiques par deux octets ; le premier contient la longueur de la chaîne de caractères identiques et un délimiteur qui définit la nature de ces caractères ; le deuxième octet contient le caractère répété.

Dans cette méthode, un enregistrement compacté a la forme suivante :

```
<enregistrement> = <chaîne>|<chaîne><enregistrement>
<chaîne>         = <SCB1>|
                 = <SCB3><caractère>|
                 = <SCB2><chaîne de caractères>|
                 = <SCB4>

SCB1 ::= '00000000'   fin d'enregistrement
SCB2 ::= '11XXXXXX' chaîne de caractères de longueur XXXXXX (6 bits) ;
         longueur maximale : 63 caractères
SCB3 ::= '101XXXXX' chaîne de XXXXX caractères, le caractère répété étant
         le caractère suivant ; longueur maximale : 31 caractères
SCB4 ::= '100XXXXX' chaîne de XXXXX blancs ; longueur maximale : 31 blancs.
```

2.2.2. Algorithme de compactage / décompactage

Voir Annexe 2.

2.2.3. Résultats obtenus

Voir tableau page suivante.

On note que la méthode de réduction de caractères identiques mène à un gain considérable sur des fichiers de type source.

Elle mène à un gain faible avec des fichiers "module objet" ou de gestion.

Comme nous pourrions le penser, elle n'est pas utilisable avec des fichiers de type numérique où elle cause une perte de place.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	271576	3.28	58 %	0.75	802	442	80
Source Algol W	457600	62066	1.09	86 %	161248	2.80	63 %	0.74	791	427	80
Source Cobol	763520	156589	1.64	79.5 %	274936	2.80	62 %	0.79	810	425	80
Module Objet	48000	18214	3.03	62 %	42280	7.05	12 %	0.30	799	515	80
Numérique	583040	140183	1.92	76 %	604904	8.30	- 3 %	- 0.04	732	421	80
Gestion	730000	226213	2.48	69 %	561000	6.18	23 %	0.30	787	458	80

Résultats de la méthode d'élimination de caractères identiques (HASP)

2.3. Méthodes liées à la théorie de l'information

2.3.1. Notions théorigues [26, 28]

Le codage sert à adapter statistiquement la source au canal. Par l'intermédiaire du codage, la source primaire donnée avec un ensemble de probabilités déterminé, est transformée en une source que nous allons désigner sous le nom de source secondaire, qui a le même alphabet que le canal et dont les probabilités d'apparition des caractères sont telles que la transformation en est maximalisée :

la relation qui donne la capacité du canal sans bruits est :

$$C = \max H(X) = \log D$$

où D est le nombre des caractères de l'alphabet du canal, c'est-à-dire du code et H(X) est l'entropie du code du canal.

Il s'ensuit que pour atteindre la capacité du canal, la source primaire d'information donnée doit être codée de façon à ce que l'entropie de la source secondaire atteigne sa valeur maximale, c'est-à-dire qu'elle soit égale à log D.

Pour atteindre ce but, nous allons présenter les codes utilisés.

a) Codes à décodage unique

Soit une source discrète, sans mémoire, qui fournit des symboles pris dans un ensemble (S), désigné sous le nom d'alphabet de la source :

(S) = (S1, S2, ..., SN)

qui possèdent les probabilités :

(P) = (P(S1), P(S2), ..., P(SN)).

Soit (X) l'alphabet de la source (vu du canal) :

(X) = (X1, X2, ..., XD)

Avec ces lettres on constitue un nombre N de mots-code :

(C) = (C1, C2, ..., CN)

Les mots-code sont des successions finies de lettres appartenant à l'alphabet (X).

Le codage est l'opération qui consiste à établir une correspondance biunivoque entre les symboles $S_K \varepsilon$ S et les mots $C_K \varepsilon$ C. La totalité des mots C_K constitue un code.

Définition :

Un code s'appelle non singulier lorsque tous les mots-code sont distincts. Le code du tableau ci-dessous, par exemple, est un code non-singulier.

Si on considère toutefois une succession de lettres (symboles) de la source, telle que S2 S3 et S1 S4 S1, on constatera qu'elle se traduit par la même succession de mots code :

S2 S3 → 1011

S1 S4 S1 → 1011

messages S_K	Mots-code C_K
S1	1
S2	10
S3	11
S4	01

Ce qui veut dire que, dans son ensemble, le code n'est pas non-singulier. Ce code est dépourvu d'utilité pratique car la réception de la succession 1011 peut être interprétée comme provenant soit de S2 S3 soit de S1 S4 S1. Cette ambiguité pourrait être évitée par l'introduction d'un signe indiquant le commencement ou la fin de chaque mot-code.

En choisissant convenablement les mots-code, on peut faire en sorte qu'à chaque succession de mots-code corresponde une seule succession de lettres (mots) de la source. Dans ce cas, le code s'appelle *à décodage unique*. Lorsqu'un code à décodage unique n'emploie pas de démarcation entre les mots, il s'appelle *séparable*.

b) Codes instantanés

Les codes A et B du tableau ci-dessous sont à décodage unique, parce qu'ils sont non-singuliers et, pour le code A, il y a une marque de fin de mot (symbole 0) et pour le code B, il y a un symbole (0) qui marque le début de chaque mot. Mais il existe entre les deux codes une différence importante : dans le cas du code A, à mesure que la succession de lettres de l'alphabet du code est reçue, les mots du code peuvent être déterminés sans aucune référence aux mots suivants. Ainsi :

Code A	Code B
0	0
10	01
110	011
1110	0111

0 10 110 0 1110 0
51 52 53 51 54 51

Ce qui veut dire que si la succession des caractères de l'alphabet du code forme un mot du vocabulaire du code, celui-ci est unique puisqu'en ajoutant des lettres à un mot de code, on ne peut pas obtenir un nouveau mot-code. Un tel code s'appelle *instantané*.

Le code B n'est pas instantané puisque, pour le décodage, il faut qu'après avoir reçu un mot du vocabulaire du code, on reçoive encore quelques lettres qui suivent afin de pouvoir décider quel est le mot qui a été transmis. Par exemple, à la réception de la suite :

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

on ne peut prendre aucune décision à la réception de <<0>> car cela pourrait signifier aussi bien les symboles S1, ou S2, ou S3, ou encore S4, et pour cette raison, le décodage devra être retardé jusqu'à la réception des lettres suivantes. L'apparition de <<1>> indique avec certitude que la seule interprétation possible est 01, voire S2, et ainsi de suite.

Définition :

Soit (Ci) = (X_{i1} X_{i2} ... X_{im}) un mot du vocabulaire d'un code. La suite de lettres X_{i1} X_{i2} ... X_{ik}, où k < m, se nomme le préfixe du mot Ci.

Donc, pour qu'un code soit instantané, il faut qu'il ne contienne aucun mot qui puisse être le préfixe d'un autre mot-code.

La condition nécessaire et suffisante pour l'existence de codes intantanés, ayant des mots de longueur l_k déterminée, est posée par l'inégalité de KRAFT :

$$K = \sum_{K=1}^{N} D^{-l_k} \leq 1 \qquad (b)$$

où D est le nombre de lettres de l'alphabet du code.

Les codes binaires instantanés peuvent être aisément construits d'après l'algorithme suivant :

1. L'ensemble (S) des messages de la source est divisé en deux sous-ensembles S0 et S1 de la manière suivante :

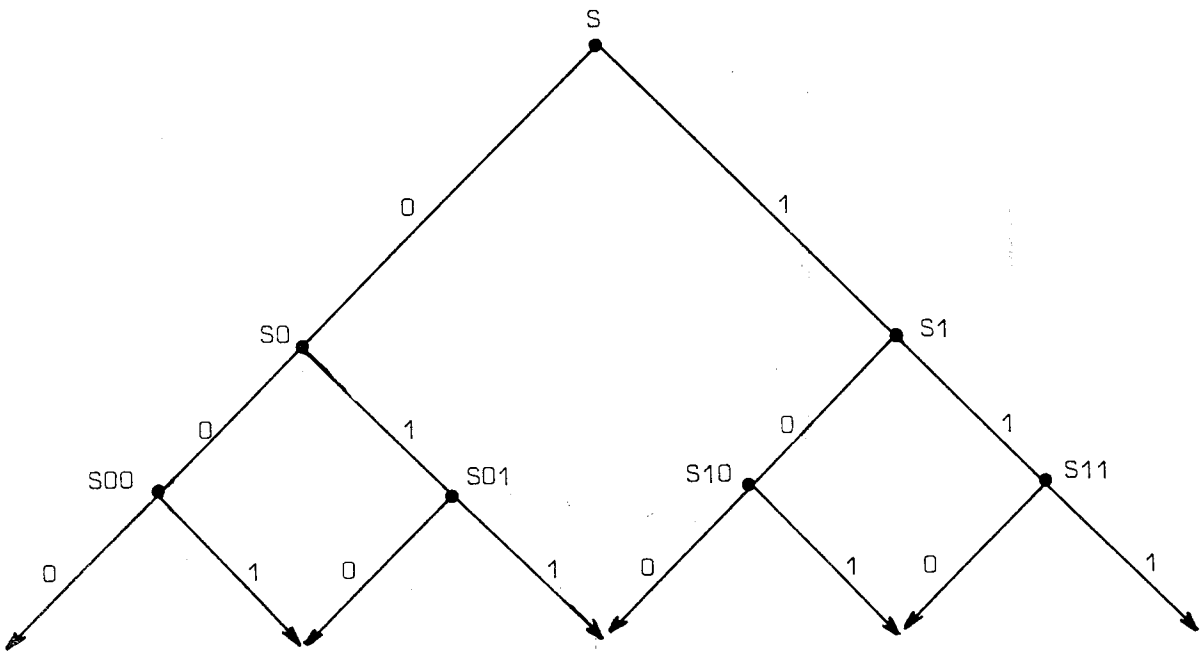
$$\underbrace{S1,\ S2,\ \ldots,\ Sk}_{S0} \quad \underbrace{Sk+1,\ \ldots,\ Sn}_{S1}$$

On attribue la lettre "0" à tous les messages de S0 et la lettre "1" à tous les messages de S1.

2. On fait la même opération pour S0 et S1 en les divisant en S00 et S01, ou bien en S10 et S11. On attribue à chaque ensemble la lettre 0 ou 1, de sorte que les messages de S00 auront en début de mot 00, tandis que ceux de S01 auront 01, et ainsi de suite.

3. On poursuit l'opération jusqu'à ce que de chaque sous-ensemble il n'en reste qu'un seul élément Sj à qui devra correspondre un mot-code.

L'opération peut être représentée graphiquement par la figure ci-dessous : représentation de l'opération de codage par un graphe arborescent :



Dans le code constitué, aucun mot du code ne peut être formé en y ajoutant une ou plusieurs lettres (0 ou 1) où un autre mot, pour la bonne raison que seuls les noeuds terminaux représentent des mots-code, tandis que les ramifications représentent des préfixes.

Longueur moyenne d'un mot-code

Ce que l'on recherche en général par le codage, c'est d'accroître l'efficacité de la transmission de l'information. Dans les canaux sans bruits, quand on dit qu'on accroît l'efficacité, on pense généralement à la minimisation d'une certaine fonction de coût.

Si on associe à chaque mot Ci un certain coefficient de coût, celui-ci peut être la durée du mot Ci, puisque l'on peut supposer que le coût de l'exploitation d'un système de transmission croît de façon approximativement linéaire avec le temps, donc avec la durée moyenne d'un mot code. Comme la durée d'un mot-code dépend de sa longueur (nombre de lettres constituant le mot), le coût moyen de transmission dépend de la longueur moyenne des mots-code :

$$\bar{l} = \sum_{i=1}^{N} P(Si)\ li \qquad (c)$$

où $\bar{l}$ = longueur moyenne du mot-code,

li = nombre de lettres constituant le mot-code Ci,

$P(Si)$ = probabilité du message i.

La marge inférieure de la longueur moyenne est donnée par la relation :

$$\bar{l}_{min} = \frac{H(S)}{\log D} \qquad (d)$$

où $H(S)$ est l'entropie de la source,

$\log D$ est la valeur maximale de l'entropie de l'alphabet du code.

Définitions :

La capacité du code :

c'est la valeur maximale de l'entropie de l'alphabet du code :

$C = \max H(X) = \log D.$

L'efficacité du code :

est égale au rapport de la longueur moyenne minimale à la longueur moyenne d'un mot code :

$$\zeta = \frac{\bar{l}_{min}}{\bar{l}}$$

La redondance du code :

c'est la grandeur complémentaire de l'efficacité :

$\rho = 1 - \zeta$

c) Codes optimaux

Un code optimal est celui qui mène à l'efficacité maximale. Dans le cas des codes d'efficacité maximale, la relation (b) de KRAFT devient l'égalité :

$$\sum_{i=1}^{N} D^{-l_i} = 1$$

Les codes dont l'efficacité est égale à l'unité, s'appellent "codes optimaux absolus" ; leur longueur moyenne de mot-code est donc minimale.

Le premier théorème de Shannon prouve que pour minimiser la longueur moyenne, le codage doit être fait par groupes de n symboles, au lieu de le faire symbole par symbole. En pratique, il ne peut pas être question que n tende vers l'infini ; n aura toujours une valeur finie qui sera, dans la plupart des cas, égale à l'unité, puisqu'il y a souvent des cas où l'accroissement de l'efficacité n'a plus d'importance lorsque n est plus grand que 1. Dans ce cas, la complication des opérations de codage n'est plus justifiée et l'on préfère le cas simple avec n = 1.
Un exemple de codage symbole par symbole est celui de Shannon-Fano et le codage de Huffman.

2.3.2. Méthode de Shannon - Fano

2.3.2.1. Description de la méthode

Supposons que les symboles de la source soient codés individuellement un par un. Dans le cas particulier où l'ensemble (S) = (S1, S2, ..,Sn), les messages de la source peuvent être partagés en deux ensembles S0 et S1 de même probabilité $P(S0) = P(S1) = 2^{-1}$, et les ensembles S0 et S1 peuvent à leur tour être divisés en ensembles S00 et S01, et S10 et S11, de même probabilité $P(S00) = P(S01) = P(S10) = P(S11) = 2^{-2}$. Cette opération est continuée jusqu'à ce que les ensembles en question ne contiennent plus qu'un seul élément. On en arrive à :

$$P(S1) = 2^{-l1},\ P(S2) = 2^{-l2},\ \ldots,\ P(Sn) = 2^{-ln},$$

et

$$\sum_{i=1}^{n} P(Si) = \sum_{i=1}^{n} 2^{-li} = 1$$

Dans ce cas, le code binaire, dont les longueurs l1, l2, ..., ln sont obtenues par le procédé de division en ensembles de probabilités égales, est un code à maximum d'efficacité. Les mots du code peuvent être obtenus selon le diagramme de l'exemple suivant.

Lorsque l'ensemble (S) des messages ne peut pas être partagé successivement en ensembles à probabilités égales, le code obtenu avec le diagramme indiqué ne sera pas optimal, mais il sera d'autant plus proche du code optimal que les probabilités des divisions seront moins différentes.

On constate que ce code est optimal en remarquant que pour obtenir l'efficacité maximale, il faudrait que :

1) toutes les lettres de l'alphabet soient utilisées avec les mêmes probabilités, quelle que soit leur position,

2) les probabilités d'apparition de lettres en n'importe quelle position dans le mot-code soient indépendantes des lettres qui précèdent.

source symboles S_K	probabilités a priori $P(S_K)$	ETAPE 1	ETAPE 2	ETAPE 3	ETAPE 4	Code final
S1	0.40	0				0
S2	0.20	1	0	0		1 0 0
S3	0.12	1	0	1		1 0 1
S4	0.08	1	1	0	0	1 1 0 0
S5	0.08	1	1	0	1	1 1 0 1
S6	0.08	1	1	1	0	1 1 1 0
S7	0.04	1	1	1	1	1 1 1 1

Lorsque les deux conditions précédentes sont remplies, l'entropie du code H(X) atteint la valeur maximale H(X) = log D et la longueur moyenne :

$$l = \frac{H(S)}{H(X)}$$

aura la valeur minimale, donc le code sera compact, et l'efficacité :

$$\zeta = \frac{H(X)}{\log D}$$

sera égale à l'unité, ce qui veut dire que le code sera optimal.

Le mode de construction du code de Shannon - Fano (tableau page précédente), montre que les caractères 0 et 1 sont utilisés avec la même probabilité (étant attribués à des divisions ayant la même probabilité) et que lorsqu'on alloue les caractères 0 et 1 on ne tient pas compte de ceux qui précèdent ; en d'autres termes, les probabilités des caractères ne dépendent pas de la position occupée par le mot-code (ils sont donc indépendants). Il s'ensuit que les conditions 1 et 2 sont remplies et, par conséquent, le code Shannon - Fano est optimal.

2.3.2.2. Algorithme de construction du code Shannon - Fano

A partir d'un tableau "PROB" de probabilités de caractères, où

$$PROB(i) = \frac{FREQ(i)}{\sum_{i=0}^{255} FREQ(i)}$$

, FREQ(i) est la fréquence du caractère i,

on range les probabilités dans un ordre décroissant tel que :
PROB(0) $\geq$ PROB(1) $\geq$, ..., $\geq$ PROB(255).

Dans un tableau "CHAR" on classe les 256 caractères EBCDIC selon le nouveau rangement de leurs probabilités, par exemple le caractère le plus fréquent a la position 0, ..., et le caractère le moins fréquent a la position 255.

Pour construire le code Shannon - Fano, considérons les variables suivantes :

PROB tableau des probabilités classées,

MM index de déplacement dans un groupe particulier de probabilités (dans PROB),

NN nombre de probabilités dans un groupe particulier (En cas d'initialisation, MM = 0 et NN = 256, si tous les caractères ont des probabilités supérieures à 0, sinon, NN < 256)

SOM contient $\sum_{i=MM}^{MM+NN} PROB(i)$

FRONT contient la valeur SOM/2,

K compteur du nombre de probabilités dans le nouveau sous-ensemble généré de l'ensemble ayant la longueur NN ; K sera plus tard le "NN" du sous-ensemble à générer ;

GRP(20::1) adresse où on note le nombre de sous-ensembles générés (il contient zéro au démarrage) ;

U index de déplacement dans GRP,

SUB1(20::1) adresse où on garde les NN et MM des ensembles à procéder,

SUB2(20::1) adresse où on garde les NN et MM des sous-ensembles générés,

POS contient les index de déplacement dans SUB2,

DCMPCT(256::20) adresse où les mots-code seront cédés aux caractères. Au départ, ce tableau ne contient que des blancs. Ce tableau sera utilisable en cas de décompactage pour raison de vitesse de l'opération. Un autre tableau sera fourni pour le compactage.

G variable.

```
/ charger valeurs d'initialisation ; /
SUB1(0,0) := 256 ; / NN du départ /
SUB1(0,1) := 0; / MM du départ /
        U := 0;
   GRP(U) := 1; / un seul ensemble à procéder /
1.      G := 0; / compteur des ensembles déjà procédés /
2.     NN := SUB1(G,0);
       MM := SUB1(G,1);
/ Pour un ensemble de longueur = NN et index de déplacement = MM, chercher
  la frontière (K) où une division de l'ensemble satisfait la condition :
            MM+K            MM+NN
             Σ  PROB(i) =    Σ     PROB(i) /
            i=MM            i=MM+K+1
3.     SOM := 0; FRONT := 0; K := 0;
       For i := MM until MM+NN-1 DO
       Begin SOM := SOM+PROB(i) end;
           i := MM;
       While FRONT < SOM/2 DO
       Begin
         FRONT := FRONT + PROB(i);
              K := K+1;
              i := i+1;
       End;
/ Chercher la position qui donne une meilleure exactitude de l'égalité
            K              NN
            Σ  PROB(i) = Σ    PROB(i) ; /
           i=MM            i=K+1
FRONTI := FRONT - PROB(i) ; / i ici = MM+K /
    Δ1 := FRONT - SOM/2 ;
    Δ2 := SOM/2 - FRONTI;
Begin
   if Δ1 > Δ2 then
      K := K - 1;
End
```

```
/ Deux sous-ensembles sont générés : (SO) de longueur NN0 = K et adresse
  MM0 = MM, et (S1) de longueur NN1 = NN-K et adresse MM1 = MM+K ; marquer
  dans GRP(U+1) la génération du sous-ensemble au-dessus (SO), marquer
  aussi ses NN et MM dans SUB2 /

SUB2(GRP(U+1),0) := K ;
SUB2(GRP(U+1),1) := MM ;
GRP(U+1) := GRP(U+1) +1 ;

/ Céder 0 aux caractères  du sous-ensemble (SO) et 1 aux caractères du
  sous-ensemble (S1), puis procéder immédiatement (S1) (s'il contient plus
  de un élément, sinon prendre dans SUB1 un autre ensemble à procéder) ;
  quand tous les ensembles dont le nombre est marqué dans GRP(U) sont pro-
  cédés, attaquer les sous-ensembles générés dans SUB2 /

/ Obtenir la position verticale du 0 à céder ; pour un mot-code particulier,
  la bonne position est le premier "blanc" après un 0 ou un 1 précédent /

j := 0 ; i := MM ;
while DCMPCT(i,j) ≠ code (64) do
begin j := j+1 end ;
for i := MM until MM+K-1 do
begin DCMPCT(i,j) := 0 end ;
for i := MM+K until MM+K+NN1-1 do
begin DCMPCT(i,j) := 1 end;

/ Procéder le sous-ensemble (S1) dont NN = NN - K et MM = MM + K /

MM := MM1 ;
NN := NN1 ;
if NN > 1 then go to 3
else G := G + 1 ; continue ;
```

```
begin if G < GRP(U) then go to 2
                    else U := U + 1 ;
end ;
      if GRP(U) := 0 then go to stop ;
else / tous les ensembles dans SUB1 sont procédés, garder les ensembles de
       SUB2 dans SUB1 et recommencer la procédure /
      for j := 0 until 1 do
      begin for i := 0 until 20 do
         begin SUB1(i,j) := SUB2(i,j) ;
               SUB2(i,j) := 0 ;
         end ;
      end ;
      go to 1 ;
Stop : / tous les caractères sont encodés par le mot-code de longueur varia-
         ble ; le code est optimal /
```

Description de l'enregistrement compacté :

Un enregistrement compacté contient deux octets d'en-tête suivis par une chaîne de bits exprimant l'information compactée.

Comme les mots-code du code Shannon - Fano sont de longueur variable (en bits), un enregistrement compacté peut avoir :

1) soit une longueur : L octets entiers + zéro fraction d'un octet,
2) soit une longueur : L octets entiers + l bits (l < 8).

Les deux octets d'en-tête servent à indiquer la longueur du buffer compacté (en-tête non compris) : le premier indique L et le deuxième indique l. Dans le cas 1) on indique le premier octet L et dans le deuxième 0 ; dans le cas 2) on indique dans le premier L+1 et dans le deuxième, l = nombre de bits utilisables du $(L+1)^{ème}$ octet.

2.3.2.3. Algorithme de compactage/décompactage

Voir Annexe 3.

2.3.2.4. Résultats obtenus

Voir tableau page suivante.

Le codage binaire de Shannon - Fano permet un gain considérable sur tous les types de fichiers.

Le temps moyen de compactage n'est pas élevé par rapport aux autres méthodes, mais le temps de décompactage est élevé.

Comme les mots-code sont de longueur variable, on perd du temps en détectant chaque bit pour retrouver le symbole équivalent. Il faut cependant remarquer que ce temps peut fortement diminuer d'un ordinateur à l'autre. Par exemple, comme ces résultats sont obtenus sur un IBM 360, nous sommes obligés, dans la routine de décompactage, de comparer d'une façon séquentielle chaque série de bits par chaque mot-code dans le groupe particulier de mots-code ; or il s'avère qu'un ordinateur comme l'IRIS 80 donne la possibilité de comparer, d'un seul coup, la série de bits avec les différents mots-code.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	226425	2.76	66 %	0.84	1456	5396	80
Source Algol W	457600	62066	1.09	86 %	136692	2.45	70 %	0.81	1278	4702	80
Source Cobol	763520	156589	1.64	79.5 %	235961	2.48	69 %	0.81	1701	4958	80
Module Objet	48000	18214	3.03	62 %	26438	4.40	45 %	0.72	1508	9782	80
Numérique	583040	140183	1.92	76 %	232364	3.22	60 %	0.76	1414	5122	80
Gestion	730000	226213	2.48	69 %	357385	3.92	51 %	0.74	1482	5462	80

Résultats de la méthode Shannon - Fano

On remarque que, avec un fichier de type module object, le temps de décompactage est presque le double du temps de décompactage des autres types de fichiers. L'explication est la suivante : quand le nombre de symboles dans le dictionnaire (symboles utilisés dans le texte ayant des fréquences supérieures à 0) est très grand (dans le cas de ce module object 200 symboles), les mots-code codés sont plus longs ; ainsi, le temps de détection pour retrouver le symbole est bien plus grand.

2.3.3. Méthode de Huffman

2.3.3.1. Introduction. Description de la méthode

Si les symboles de la source sont codés un par un, la question qui se pose alors est de trouver une méthode de codage qui donne, dans tous les cas (pour toute distribution des probabilités), un codage optimal, en d'autres termes, que l'on ne puisse pas trouver un autre procédé de codage (avec les symboles de la source pris individuellement) qui conduise à une meilleure efficacité.

Dans ce cas, les mots-code obtenus ont une plus petite longueur moyenne par rapport aux longueurs que l'on aurait pu obtenir par d'autres procédés de codage ; le codage est donc compact.

Les codes compacts ont les deux propriétés suivantes :

1) Pour que le codage soit compact, il faut que le mot-code le plus court soit attribué au symbole à la plus grande probabilité. A cet effet, les messages seront rangés dans l'ordre décroissant de probabilités, soit :

$$\left.\begin{array}{l} P(S1) \geqslant P(S2) \geqslant \ldots \geqslant P(SN) \\ l1 < l2 < \ldots < lN \end{array}\right\} \; A$$

Si cet ordre n'est pas observé (par exemple $l1 \rightarrow S2$ et $l2 \rightarrow S1$), la longueur moyenne $\bar{l}$ du mot-code diminuera lorsque l'ordre est rétabli. Par conséquent, l'ordre donné par les relations A aussure l'existence d'une longueur moyenne minimale.

2) Pour un codage compact, il faut que les longueurs des deux derniers messages soient égales :

$$l_{N-1} = l_N$$

Supposons que cela ne soit pas vrai et que :

$$l_N = l_{N-1} + 1$$

Sachant que l'on ne considère que des codes qui ont la propriété de préfixe, en écartant la dernière lettre du mot-code de longueur $l_N = l_{N-1} + 1$ correspondant au symbole S_N, on n'obtiendra pas un mot qui puisse être attribué à un message autre que S_N. Cette lettre est donc superflue, car elle accroît la longueur moyenne $\bar{l}$. Aussi, pour avoir un code compact, faut-il que les deux derniers messages se voient attribuer des mots-code de la même longueur.

Un procédé général de codage compact a été trouvé par Huffman. Ce codage est basé sur l'idée de la division des ensembles des messages (S) = (S1, S2, ..., Sn) en sous-ensembles S0 et S1, à probabilités aussi proches que possible, les sous-ensembles S0 et S1 étant divisés à leur tour en ensembles S00 et S01, ou S10 et S11, aux probabilités aussi proches que possible, et ainsi de suite.

Dans le cas particulier où les probabilités des ensembles divisés sont égales, le codage est absolument optimal. Mais, les probabilités des ensembles divisés ne sont généralement pas égales et le problème consiste notamment à les diviser de façon que ces probabilités soient aussi proches que possible de l'égalité, ce qui assurerait les conditions d'un code optimal.

Les messages (S) seront groupés en ensembles de probabilités aussi proches que possible de la manière suivante :

1. Ordonner l'ensemble des messages (S) dans la succession des probabilités décroissantes :

$$\left.\begin{array}{l}(S) = (R0) = (S1, S2, \ldots, Sn) \\ P(S1) \geq P(S2) \geq , \ldots, \geq P(Sn)\end{array}\right\} \quad (B)$$

2. Former des ensembles de messages qui puissent être divisés en deux sous-ensembles de probabilités aussi proches que possible.

a) Considérer d'abord un ensemble constitué de deux éléments seulement et supposer que l'un est Sn. Dans ce cas, l'autre élément sera obligatoirement S_{n-1} puisque, conformément à la relation (B), sa probabilité $P(S_{n-1})$ est la plus proche de la probabilité P(Sn) de l'élément Sn.

Notons par

$$r1 = S_{n-1} \cup S_n$$

l'ensemble constitué de S_{n-1} et S_n, en lui attribuant la probabilité :

$$P(S_{n-1} \cup S_n) = P(S_{n-1}) + P(S_n) = P(r1)$$

b) Considérer l'ensemble constitué des deux éléments S_{n-1} et S_n comme un nouveau message r1, inclus dans l'ensemble des autres messages, dans l'ordre des probabilités décroissantes :

$$(R1) = (S1, S2, \ldots, r1, \ldots)$$

$$P(S1) \geq P(S2) \geq , \ldots, \geq P(r1) \geq , \ldots \qquad (C)$$

Grouper les deux derniers messages de cette suite comme avant, en formant un nouveau message r2 qui sera inclus dans l'ensemble des autres messages, dans l'ordre des probabilités décroissantes, en constituant la suite ordonnée (R2). Obtenir de la même manière la suite ordonnée (R3) de la suite ordonnée (R2) et continuer l'opération jusqu'à ce que l'on arrive à la suite (Rn), constituée de deux éléments seulement :

$$(Rn) = (r_n, r_{n-1}).$$

c) On déduit de ce qui précède que l'ensemble (S) peut être divisé en deux ensembles r_n et r_{n-1} de façon que leurs probabilités $P(r_n)$ et $P(r_{n-1})$ soient aussi proches l'une de l'autre que possible et que r_n et r_{n-1} puissent à leur tour être chacun divisé en deux ensembles de probabilités aussi proches l'une de l'autre que possible et ainsi de suite, jusqu'à ce que l'on arrive à des ensembles d'un seul élément : S1, S2, ... Sn.

Lorsque les probabilités des ensembles obtenus par division sont égales, on peut alors réaliser un codage absolu optimal. Si tel n'est pas le cas, le codage ne sera pas absolu optimal ; il sera optimal et se rapprochera d'autant plus du codage absolu optimal que la différence entre les probabilités des ensembles divisés sera moindre.

d) Les mots-code correspondants à chaque message sont tirés de la façon suivante :

. attribuer le symbole 0 à l'ensemble r_n ;

. attribuer le symbole 1 à l'ensemble r_{n-1} ;

. attribuer le symbole 0 ou le symbole 1 à chaque division par deux et continuer l'opération jusqu'à ce que l'on arrive à un ensemble qui contient un seul élément S_K (K = 1, 2, ..., n).

L'exemple suivant montre les étapes successives pour construire le code Huffman :

considérons la source (S) avec la distribution suivante :

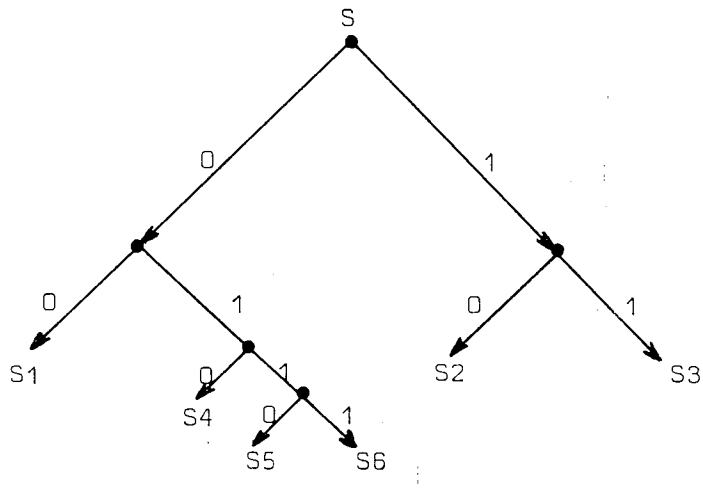
(S) (S1, S2, S3, S4, S5, S6)

(P) (0.3; 0.25; 0.15; 0.15; 0.10; 0.05)

Rangeons sous forme de tableau les probabilités de la source (S) et celles des sources restreintes (R) constituées par la réunion des deux derniers messages de la source précédente :

Message S_K	Probabilités $P(S_K)$	Mots-code C_K	(R1)	Sources restreintes (R2)	(R3)	(R4)
S1	0.30	00	0.30 (00)	0.30 (00)	0.30 (00)	0.60 (0)
S2	0.25	10	0.25 (10)	0.25 (10)	0.30 (01)	0.40 (1)
S3	0.15	11	0.15 (11)	0.15 (11)	0.40 (1)	
S4	0.15	010	0.15 (010)	0.30 (01)		
S5	0.10	0110	0.15 (011)			
S6	0.05	0111				

Graphe correspondant au procédé de codage binaire de Huffman, du tableau ci-dessus :



2.3.3.2. Algorithme de construction du code Huffman :

PROB(1::N) contient les probabilités de source (S) rangées dans un ordre décroissant,

N nombre de probabilités dans PROB,

PR contient la valeur PROB(x) + PROB(x+1)

POS contient l'index de déplacement de PR dans PROB,

INDCTR contient tous les POS, c'est-à-dire sera une référence des positions de tous les PR,

j index de déplacement dans INDCTR,

CODHUF domaine des mots-codes construits,

TOTAL nombre des éléments dans une source restreinte,

i index de déplacement dans PROB et CODHUF,

K entier

zone(1::20) zone de longueur = longueur maximale d'un mot-code,

S entier

CODMAXL longueur maximale d'un mot-code ; on la considère égale à 20 ; par experience : 13 < CODMAXL < 18.

On dispose de deux procédures :

(1) Procédure RESTREIN(N) :

/ Cette procédure constitue les probabilités des sources restreintes (R) où $P(rx) = P(S_{n-x}) + P(S_{n-x+1})$. Cette probabilité est donc insérée dans PROB de telle sorte que les probabilités restent dans leur ordre décroissant ; les positions d'insertion de P(r) sont marquées régulièrement dans INDCTR(j)/

```
Procedure RESTREIN(N)
begin j := 1;
   for i:= N-1 step -1 until 2 do
   begin PR := PROB(i) + PROB(i+1) ;
      for K := i-1 step -1 until 1 do
      begin if PR > PROB(K) then
         begin PROB(K+1) := PROB(K) ;
               PROB(K) := Pr ;
               POS := K ;
         end ;
      end ;
      INDCTR(j) := POS ;
      j := j+1 ;
   end ;
end ;
```

(2) Procédure HUFFMAN (N.CODMAXL) :

/ Chercher le code non complet d'ensembles à étendre, le garder dans ZONE, classer le tableau de codes CODHUF, générer deux images du code stocké dans ZONE, stocker ces deux images dans leurs propres places de CODHUF, ajouter à la première image le symbole 0 et à la deuxième le symbole 1, continuer l'opération jusqu'à ce que les codes à céder soient complets pour tous les messages (S_K) /

/ Le dernier ensemble restreint contient deux éléments A et B (cas initial) ; on commence par attribuer 0 à l'ensemble A et 1 à l'ensemble B /

```
Begin
   TOTAL := 2 ;
   CODHUF(1,1) := 0 ;
   CODHUF(2,1) := 1 ;
   for j := N-2 step -1 until 1 do
   begin POS := INDCTR(j)  / position de l'ensemble à procéder /
      for i := 1 until CODMAXL do
      begin ZONE(1,i) := CODHUF(POS,i) ;
      end ;
```

```
/ Réordonner CODHUF selon le nouveau processus /
          For S := POS until TOTAL-1 do
          begin for i := 1 until CODMAXL do
             begin CODHUF(s,i) := CODHUF(s+1,i) ;
             end ;
          end ;
/ générer deux images du mot-code à étendre, stocker les images dans leurs
  propres places de CODHUF, en même temps détecter la position du nouveau
  symbole à attribuer /
          while ZONE(1,i) ≠ X do
          begin
             CODHUF(TOTAL,i) := ZONE(1,i) ;
             CODHUF(TOTAL+1,i) := ZONE(1,i) ;
             i := i+1 ;
          end ;
/ Position du prochain 0 ou 1 à attribuer, tombe dans la colonne i, donc /
          CODHUF(TOTAL,i) := 0 ;
          CODHUF(TOTAL+1,i) := 1 ;
          TOTAL := TOTAL + 1 ;
        end ;
      end ;
/ Le code Huffman est construit pour les messages S_K (K = 1, 2, ..., n) /
```

2.3.3.3. Algorithme de compactage - décompactage

L'algorithme de compactage et décompactage est le même que celui de la méthode Shannon - Fano. Voir Annexe 3.

2.3.3.4. Résultats obtenus

Voir tableau ci-après.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	224824	2.74	66 %	0.84	1534	5330	80
Source Algol W	457600	62066	1.09	86 %	136422	2.44	70 %	0.81	1186	4658	80
Source Cobol	763520	156589	1.64	79.5 %	235632	2.47	69 %	0.81	1634	4940	80
Module objet	48000	18214	3.03	62 %	26432	4.40	45 %	0.72	1428	9738	80
Numérique	583040	140183	1.92	76 %	232026	3.22	60 %	0.76	1324	5008	80
Gestion	730000	226213	2.48	69 %	357362	3.92	51 %	0.74	1456	5402	80

Résultats de la méthode de Huffman

Il est à remarquer que les méthodes Huffman et Shannon - Fano aboutissent aux mêmes résultats.

Aussi pouvons-nous dire que l'amélioration apportée par Huffman n'est perceptible qu'au niveau des temps de compactage/décompactage.

2.4. Méthode de Hahn

2.4.1. Introduction. Description de la méthode

Tout en appliquant la compression des blancs, Hahn a en outre comprimé les caractères non-triviaux en les codant en groupes de longueur fixe en tant que nombres *uniques à virgule fixe.*

Le codage numérique se fait comme suit : les symboles non-triviaux sont considérés en groupes de longueur N. Chacun des symboles d'un groupe est recherché dans un dictionnaire et sa position est notée. Les symboles sont codés selon leur position dans le dictionnaire. Par la suite, au décodage, on se servira des positions numériques aussi bien que du dictionnaire pour reproduire le texte initial.

Soit B le nombre de symboles dans le dictionnaire et Pi la position du $i^{ème}$ symbole d'un groupe ($1 \leqslant i \leqslant N$) dans le dictionnaire ($1 \leqslant Pi \leqslant (B-1)$).

La $B^{ième}$ position est réservée à un caractère fictif destiné à rendre possible l'extension de la taille du dictionnaire. Un groupe de symboles ayant les positions P1, P2, ... PN est codé comme un nombre unique à virgule fixe :

$$P_1 \times B^{N-1} + P_2 \times B^{N-2} + \ldots + P_{N-1} \times B + P_N$$

Dans la mise en oeuvre effectivement réalisée, les valeurs B^{N-1}, ... B^2 peuvent être calculées une seule fois et stockées ensuite pour éliminer des opérations arithmétiques redondantes. On peut utiliser plus de B symboles différents en se servant de la $B^{ième}$ position dans le dictionnaire comme caractère d'extension.

Si la $B^{ième}$ position n'a pas été réservée, la valeur $B \times B^i$ pourrait apparaître, ce qui prêterait à ambiguité. Le caractère d'extension signifie que le symbole suivant tient une position dans la zone B+1 à 2B-1. On peut prévoir plus d'un caractère d'extension pour pouvoir étendre encore davantage la longueur du dictionnaire. Par définition, les B premiers symboles comprennent le dictionnaire primaire.

Ainsi, en général, si P est la position d'un symbole donné du dictionnaire, ce symbole est codé comme $\lfloor P/B \rfloor$ caractères d'extension suivi de MOD(P,B) où $\lfloor x \rfloor$ désigne la partie entière de X.

Le caractère d'extension est codé comme zéro. Chaque fois que l'on rencontre un zéro au cours du processus de décodage, on ajoute la valeur B au prochain nombre de positions non-zéro ; ainsi, on trouve la véritable position du symbole codé dans le dictionnaire.

Par exemple, supposons que la longueur du dictionnaire primaire soit 21 (c'est-à-dire B = 21) et que les symboles doivent être codés en groupes de 7 (c'est-à-dire n = 7). Les symboles à coder auraient les positions 5, 7, 20, 25, 45, 6, 9, 22, 10 et 15 dans le dictionnaire. Ces symboles seraient codés comme deux nombres à virgule fixe, le premier ayant pour valeurs : $5 \times 21^6 + 7 \times 21^5 + 20 \times 21^4 + 0 \times 21^3 + 4 \times 21^2 + 0 \times 21 + 0 =$ 461, 310, 696 ; et le second ayant pour valeurs : $3 \times 21^6 + 6 \times 21^5 + 9 \times 21^4 + 0 \times 21^3 + 1 \times 21^2 + 10 \times 21 + 15 =$ 283, 553, 949.

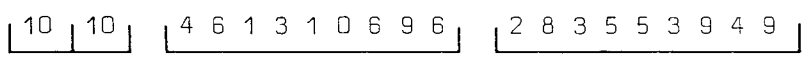
On notera que le symbole à la position 25 est codé 0,4 et le symbole à la position 45 est codé 0,0,3.

Pour minimiser le nombre de caractères d'extension nécessaires, une partie du texte est parcourue préalablement pour estimer la probabilité d'occurence de chaque symbole. Ensuite, le dictionnaire est établi avec les symboles dans l'ordre descendant d'occurence prédite.

L'enregistrement compacté a un format à trois zones : les deux premières qui sont enregistrées dans les moitiés gauche et droite du même 1/2 mot machine, contiennent le nombre de blancs "en tête" et le nombre de symboles non-triviaux qui suivent le dernier blanc "en tête" ; la troisième zone est de longueur variable et consiste en un nombre suffisant de mots pour pouvoir stocker tous les symboles non blancs dans l'enregistrement.

Exemples :

1) Reprenons l'exemple codé ci-dessus. Si un enregistrement d'entrée à 80 caractères se compose de 10 symboles suivis de 10 symboles dont la position dans le dictionnaire serait identique à celle donnée ci-dessus, suivis à leur tour de 60 blancs, l'enregistrement compacté serait stocké de la façon suivante (chaque boîte oblongue représentant un mot machine) :



2) Un enregistrement de 80 blancs serait compacté de la façon suivante :

| 80 | 0 |

Optimisation du compactage

L'utilisateur du système peut contrôler la valeur des variables B (longueur du dictionnaire primaire) et N (taille des groupes de codage). Dans l'idéal, les valeurs de B et de N sont choisies de façon à essayer de minimiser le nombre moyen de bits par caractère nécessaires pour coder le texte d'entrée. Pour chaque valeur de N il existe une valeur optimale de B. Cette valeur optimale de B est l'entier le plus grand de B tel que $B^N - 1 \leq L$ où L est le nombre le plus grand à virgule fixe pouvant être stocké dans un seul mot. Car le nombre le plus grand qui peut ainsi être produit est sans aucun doute $B \times B^{N-1} + B \times B^{N-2} + \ldots + B = B^N - 1$. Sur l'IBM 360, $L = 2^{31} - 1$. Ceci engendre des valeurs de B correspondant à des valeurs de N raisonnables. La valeur de B s'obtient à partir de l'équation :

$$B = \lfloor (2^{31} - 1)^{1/N} \rfloor$$

Ainsi, B est-il choisi le plus grand possible sans que toutefois il puisse y avoir débordement pour une valeur donnée de N.

2.4.2. Algorithme de compactage - décompactage

Voir Annexe 4.

2.4.3. Recherche du meilleur couple (B,N)

B étant la longueur du dictionnaire et N le nombre de caractères par groupe.

Comme la méthode de Hahn est en fait une méthode combinée (suppression des redondances et calculs numériques), nous avons voulu connaître ce que cette méthode seule apportait.

Le tableau qui suit est le tableau récapitulatif de l'algorithme des combinaisons numériques (sans appliquer l'élimination des blancs).

Les paramètres sont :

B	longueur apparente du dictionnaire,
N	nombre de caractères par groupe (selon B)
b	nombre de bits par caractère
P	perte (en bits par mot)
TCM	temps de compactage moyen (en micro-secondes)
GR%	gain de place en pourcentage

Le test est fait pour deux types de données :

1. langage naturel,
2. source Assembleur.

B	N	b	P	TCM		GR%	
				Langage naturel	Source Assembleur	Langage naturel	Source Assembleur
2	30	1	2	13936	9490	9 %	34 %
3	19	1	13	9074	6110	21 %	38 %
4	15	2	2	7254	5200	26 %	39 %
5	13	2	6	6266	4576	31 %	40 %
6	11	2	10	5668	4212	29 %	37 %
7	11	2	10	5044	3978	36 %	42 %
8	10	3	2	4888	3796	35 %	40 %
9	9	3	5	4758	3718	33 %	36 %
10	9	3	5	4394	3666	37 %	39 %
11	8	4	00	4238	3588	32 %	33 %
12	8	4	00	4290	3562	35 %	35 %
13	8	4	00	4186	3545	36 %	36 %
14	8	4	00	4134	3614	38 %	38 %
15	7	4	4	4004	3588	31 %	30 %
16	7	4	4	3978	3718	33 %	31 %
17	7	4	4	4056	3666	34 %	32 %
18	7	4	4	3926	3666	35 %	33 %
19	7	4	4	3848	3588	35 %	33 %
20	7	4	4	3848	3536	36 %	34 %
21	7	4	4	3822	3614	36 %	35 %
22	6	5	2	4108	3484	27 %	25 %
23	6	5	2	3900	3458	27 %	25 %
24	6	5	2	3874	3458	27 %	26 %
25	6	5	2	4056	3458	28 %	26 %
26	6	5	2	3874	3406	28 %	26 %
27	6	5	2	3796	3406	28 %	27 %

B	N	b	P	TCM		GR%	
				Langage naturel	Source Assembleur	Langage naturel	Source Assembleur
28	6	5	2	3848	3380	28 %	27 %
29	6	5	2	3952	3380	28 %	27 %
30	6	5	2	3874	3406	28 %	27 %
31	6	5	2	3822	3458	28 %	27 %
32	6	5	2	3796	3354	28 %	28 %
33	6	5	2	3796	3458	28 %	28 %
34	6	5	2	3796	3484	28 %	28 %
35	6	5	2	3796	3354	28 %	28 %
36	5	6	2	3822	3406	17 %	15 %
37	5	6	2	3822	3380	17 %	15 %
38	5	6	2	3796	3380	18 %	16 %
39	5	6	2	3796	3406	18 %	16 %
40	5	6	2	3822	3458	18 %	16 %
41	5	6	2	3874	3380	18 %	16 %
42	5	6	2	3822	3432	18 %	17 %
43	5	6	2	3822	3458	18 %	17 %
44	5	6	2	3822	3458	18 %	17 %
45	5	6	2	3796	3454	18 %	17 %
46	5	6	2	3822	3446	18 %	17 %
47	5	6	2	3796	3354	18 %	17 %
48	5	6	2	3796	3346	18 %	17 %
49	5	6	2	3796	3336	18 %	18 %
50	5	6	2	3796	3332	18 %	18 %
51	5	6	2	3874	3332	18 %	18 %
52	5	6	2	3796	3328	18 %	18 %

B	N	b	P	TCM		GR%	
				langage naturel	Source Assembleur	langage naturel	Source Assembleur
53	5	6	2	3796	3302	18 %	18 %
54	5	6	2	3822	3302	18 %	18 %
55	5	6	2	3796	3302	18 %	18 %
56	5	6	2	3822	3302	18 %	18 %
57	5	6	2	3822	3302	18 %	18 %
58	5	6	2	3900	3328	18 %	18 %
59	5	6	2	3900	3302	18 %	18 %
60	5	6	2	3874	3302	18 %	18 %
61	5	6	2	3848	3354	18 %	18 %
62	5	6	2	3822	3302	18 %	18 %
63	5	6	2	3796	3224	18 %	18 %
64	5	6	2	3796	3276	18 %	18 %
65	5	6	2	3822	3250	18 %	18 %
66	5	6	2	3796	3224	18 %	18 %
67	5	6	2	3770	3276	18 %	18 %
68	5	6	2	3796	3250	18 %	18 %
69	5	6	2	3796	3250	18 %	18 %
70	5	5	2	3770	3276	18 %	18 %
71	5	6	2	3796	3250	18 %	18 %
72	5	6	2	3796	3224	18 %	18 %
73	5	6	2	3770	3224	18 %	18 %
74	4	8	0	3874	3351	- 2 %	- 2 %
75							
76							
77							

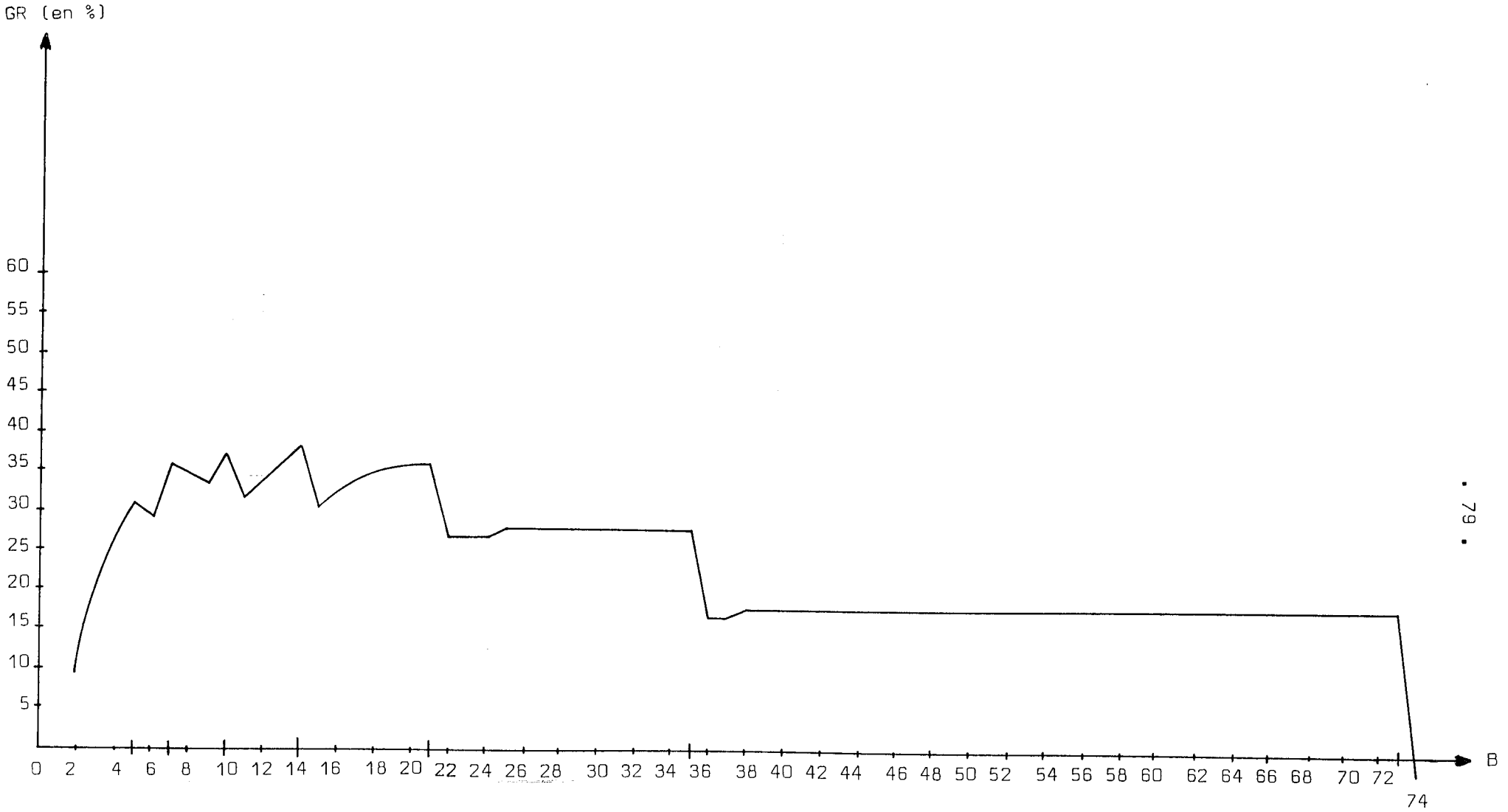


Diagramme indiquant la relation entre B (longueur apparente du dictionnaire) et GR (gain en place (en %)).

Nous avons vu que la méthode de Hahn comporte deux parties : la première utilise des caractères réduits, nous l'appellerons "méthode des combinaisons numériques", la seconde utilise la réduction et l'élimination des blancs.

On voit dans le tableau qui précède, les résultats obtenus en utilisant seulement les combinaisons numériques sans appliquer la seconde méthode pour mesurer l'efficacité réelle de cet algorithme.

Nous avons mené cette expérimentation sur deux types de données :

. le langage naturel, en particulier la langue française,

. les langages ordinateurs, en particulier l'Assembleur.

L'expérience a été faite avec plusieurs valeurs des paramètres :

. B : longueur apparente du dictionnaire,

. N : nombre correspondant de symboles compressibles dans un mot d'ordinateur (4 octets).

Les résultats sont exprimés en pourcentage du gain de place GR% et en temps nécessaire pour comprimer un enregistrement, TCM.

Nous considérons les résultats obtenus par le test du langage naturel comme plus exacts pour mesurer les performances de cet algorithme, tandis que les résultats obtenus par le test de l'assembleur sont considérés comme un cas spécial, surtout ceux du sommet du tableau (pour $2 \leqslant B \leqslant 6$).

Nous remarquons que le gain obtenu en compactant les sources assembleur est nettement plus grand que celui que l'on obtient en compactant un langage naturel. Ce phénomène est probablement dû au rôle du blanc. Le blanc est un caractère qui possède une très haute probabilité d'apparition dans les langages ordinateurs, ainsi occupe-t-il la plupart du temps la première entrée du dictionnaire.

En fait, il y a trois alternatives qui mènent à ces résultats :

- la première est que quand B est très faible (de 2 à 6), N est très grand (30 à 11) ;

- la seconde alternative est que le blanc représente à peu près 50 % de l'information totale dans les programmes source ;

- enfin, comme le blanc occupe toujours la première entrée du dictionnaire, il n'est pas représenté de façon plus étendue. Les blancs seront compressés par grands nombres, donnant lieu à un gain de place exceptionnel sur 50 % de l'information totale. Ce débit de gain diminue suivant l'augmentation de la valeur de B, parce que la première alternative n'est plus valable, c'est-à-dire que les blancs sont regroupés mais en petit nombre.

Discutons à présent les résultats des expérimentations sur le langage naturel.

La série des résultats peut être divisée en trois parties :

- la première, où B est très bas ($2 \leqslant B \leqslant 6$) : le gain est bas et le temps moyen de compactage est grand,

- la seconde ($7 \leqslant B \leqslant 21$) : le gain augmente et reste relativement élevé pendant que le temps diminue progressivement,

- la troisième ($21 \leqslant B \leqslant 73$) : le gain diminue après des intervalles de stabilisation et le temps décroît plus légèrement.

Pour comprendre ce qui se passe dans la première partie, nous donnons l'exemple suivant :

Considérons le dictionnaire D illustré par la figure ci-dessous, qui contient 64 entrées, ces entrées étant rangées par ordre décroissant :

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R		+	-	%	><

Ces entrées auront les positions suivantes (P) :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18		61	62	63	64

Appliquons la première partie de l'algorithme de Hahn en utilisant ce dictionnaire. Si la longueur apparente B du dictionnaire est choisie égale à 2, on obtient une nouvelle forme du dictionnaire D' et les nouvelles positions des entrées P', indiquées dans la figure qui suit :

D' :

A		B		C		D		E		F		G		H		I		J		K		L		M		N		O				><

P' :

1	0	3	0	5	0	7	0	9	0	11	0	13	0	15	0	17	0	19	0	21	0	23	0	25	0	27	0	29			0	127

Supposons que la séquence des caractères "GIN" fait partie des informations à compacter ; en représentant les caractères par leurs positions P', on obtient :

- la position du caractère G ($P'_G = 13 = 6B + 1$) sera développée ainsi : 0 0 0 0 0 0 1
- la position du caractère I ($P'_I = 17 = 8B + 1$) deviendra : 0 0 0 0 0 0 0 0 1
- la position de N ($P'_N = 27 = 13B + 1$) sera : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

avec substitution dans la formule de base (valeur numérique des informations compactées = $B^{N-1} \times P_1 + B^{N-2} \times P_2 + \ldots + P_n$)

Rappelons que pour B = 2 et N = 30, on a : valeur numérique de GIN compactée :

$$2^{29}X0 + 2^{28}X0 + 2^{27}X0 + 2^{26}X0 + 2^{25}X0 + 2^{24}X0 + 2^{23}X1 + 2^{22}X0$$
$$+ 2^{21}X0 + 2^{20}X0 + 2^{19}X0 + 2^{18}X0 + 2^{17}X0 + 2^{16}X0 + 2^{15}X0 + 2^{14}X1$$
$$+ 2^{13}X0 + 2^{12}X0 + 2^{11}X0 + 2^{10}X0 + 2^{9}X0 + 2^{8}X0 + 2^{7}X0 + 2^{6}X0$$
$$+ 2^{5}X0 + 2^{4}X0 + 2^{3}X0 + 2^{2}X0 + 2^{1}X0 + 2^{0}X1.$$

Il est clair que les positions de ces trois caractères vont jusqu'à 30 et occupent les 30 termes de la formule de compression, c'est-à-dire que seulement les trois caractères pourraient être compactés en un mot complet.

Bien que ces caractères atteignent presque le sommet du dictionnaire initial (dans les 7ème, 9ème et 14ème positions), le résultat de leur compression entraîne une perte de place de 25 %, parce que quand B est très petit, les positions des caractères sont rapidement éjectées vers des ordres plus hauts, donc sont représentés par un plus grand nombre de bits.

En ce qui concerne le temps de compactage, il est normal qu'il soit élevé dans cette partie, car lorsque B est bas, les symboles occupent les entrées d'un ordre plus élevé, aussi le temps d'analyse de chaque position est-il plus long.

Pour la deuxième partie des résultats, B a des valeurs moyennes et ainsi les positions des caractères ne sont-elles pas trop développées ; de même, le nombre N n'est pas très bas, ce qui entraîne un gain considérable de place.

La perte de gain à travers la troisième partie des résultats est normalement due à la diminution du paramètre N. La diminution du temps de compactage est moins importante parce que quand la valeur de B est grande, la forme du dictionnaire change peu.

Le tableau qui suit est le tableau récapitulatif de l'algorithme de combinaisons numériques, en éliminant les blancs d'en-tête et de fin. Les paramètres sont les mêmes que ceux employés précédemment (tableau p.76), les données sont de type source assembleur.

B	N	TCM	GR%
6	11	2860	55 %
7	11	2626	60 %
8	10	2418	60 %
9	9	2314	59 %
10	9	2210	61 %
11	8	2210	60 %
12	8	2132	61 %
13	8	2158	63 %
14	8	2028	64 %
15	7	2002	61 %
16	7	1976	62 %
17	7	1950	63 %
18	7	1924	63 %
19	7	1898	64 %
20	7	1872	64 %
21	7	1872	65 %
22	6	1898	61 %
23	6	1872	61 %
24	6	1898	62 %
25	6	1898	62 %
26	6	1898	62 %
27	6	1872	63 %
28	6	1846	63 %
29	6	1820	63 %
30	6	1950	63 %
31	6	1950	64 %
32	6	2002	64 %
33	6	1950	64 %
34	6	2002	64 %
35	6	1950	64 %
36	5	1924	59 %
37	5	2002	59 %
38	5	2054	59 %
39	5	2002	59 %
40	5	1976	60 %

B	N	TCM	GR%
41	5	1846	60 %
42	5	1820	60 %
43	5	1846	60 %
44	5	1846	60 %
45	5	1846	60 %
46	5	1976	60 %
47	5	1898	60 %
48	5	1872	60 %
49	5	1924	60 %
50	5	1898	60 %
51	5	1898	60 %
52	5	2002	60 %
53	5	1924	60 %
54	5	1976	60 %
55	5	1898	60 %
56	5	1924	60 %
57	5	1898	60 %
58	5	2106	60 %
59	5	1898	60 %
60	5	1898	60 %
61	5	1872	60 %
62	5	1976	60 %
63	5	1898	60 %
64	5	1846	60 %
65	5	1820	60 %
66	5	1820	60 %
67	5	1820	60 %
68	5	2002	60 %
69	5	1924	60 %
70	5	1872	60 %
71	5	1872	60 %
72	5	1872	60 %
73	5	1846	60 %
74	4	1872	41 %

2.4.4. Résultats obtenus avec le meilleur couple (B,N)

Voir tableau ci-après.

Par la méthode de Hahn, du point de vue gain de place, on obtient :

- un gain élevé pour des fichiers de type source ou numérique,
- un gain très faible pour des fichiers de type binaire (module object).

Nous pouvons dire que ce n'est pas la forme de construction de l'enregistrement qui fait varier le gain obtenu par cette méthode, mais la longueur du dictionnaire.

En effet, lorsque le dictionnaire contient un nombre modéré de symboles (jusqu'à 73), la méthode mène à un gain considérable (par exemple, pour des fichiers de type source, le dictionnaire ne dépasse pas 64 entrées et avec les fichiers numériques nous avons obtenu un gain élevé lorsque le dictionnaire contient 14 entrées seulement), tandis qu'avec le "module object", le gain tombe à 3 % car le dictionnaire contient alors 193 entrées. Dans ce cas-là, beaucoup de symboles occupent des positions d'ordre élevé dans le dictionnaire, ceux-ci étant représentés par un grand nombre de bits.

La différence que l'on remarque entre les temps de compactage et décompactage des fichiers de type source, numérique ou binaire, provient de la disparition des blancs.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	218520	2.65	67 %	0.88	962	780	80
Source Algol W	457600	62066	1.09	86 %	109824	1.92	76 %	0.88	910	858	80
Source Cobol	763520	156589	1.64	79.5 %	190880	2.01	75 %	0.93	936	862	80
Module objet	48000	18214	3.03	62 %	46520	7.74	3 %	0.04	2114	2142	80
Numérique	583040	140183	1.92	76 %	274408	3.75	53 %	0.68	2054	2136	80
Gestion	730000	226213	2.48	69 %	424336	4.65	42 %	0.61	1820	2058	80

Résultats de la méthode de combinaisons numériques (Hahn)

Valeurs choisies de B selon la classe de données :

Texte source : B = 32, N = 6
données binaires : B = 73, N = 5
données numériques : B = 8, N = 10
données de gestion : B = 32, N = 6

2.5. Méthode de Louis-Gavet

2.5.1. Description de la méthode

Cette méthode est originale dans le fait que la fonction de compactage n'est plus liée au corpus des données, mais à la structure du langage. Louis-Gavet a montré que dans tout langage structuré, les caractères (ou groupes de caractères) suivaient les répartitions des lois statistiques classiques. Aussi peut-on dire qu'à partir d'un certain nombre de caractères, il y a indépendance entre eux.

D'où une règle de compactage (représentant la fonction) très simple. Celle-ci consiste à prendre un certain nombre de caractères dans la chaîne originelle, à intervalles réguliers, pour former "l'empreinte".

Si nous ne nous intéressons qu'à la présence ou l'absence d'une donnée (interrogation d'une base de données, par exemple), on peut atteindre des taux de compactage très élevés (de l'ordre de 90 %). Cette fonction de compactage peut très facilement être une fonction de "hash coding". Ceci a fait l'objet de nombreuses applications.

Pour nous, ce qui nous intéressait était de savoir si nous pouvions appliquer cette méthode dans le transfert des données dans un réseau, ou pour le stockage de données sur disque.

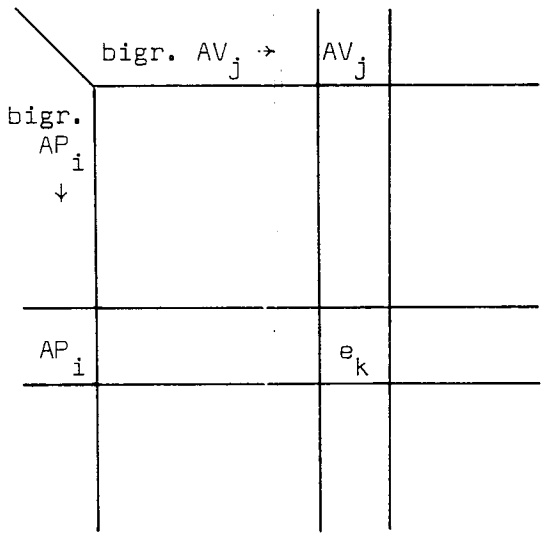
Malgré le tirage au hasard des caractères, il semble utopique de retrouver l'information d'origine. Or, on montre, comme nous travaillons sur des langages structurés, que l'information manquante ne peut être que structurée. Ce qui fait que le nombre de bigrammes ou de trigrammes est en nombre limité, ainsi le complément de l'information compactée est-il relativement peu volumineux.

Ainsi, si on prend un caractère sur trois, il suffira de raccrocher à chaque caractère de l'empreinte le bigramme précédent et suivant. Ceci peut se faire à l'aide d'une matrice où devra figurer l'ensemble des bigrammes avant (AV) et après (AP). Il suffit alors, pour deux caractères donnés de l'empreinte, de trouver le même bigramme, qui sera le bigramme après pour le premier et le bigramme avant pour le deuxième.

Si l'empreinte est e_1, e_2, e_3, on aura le schéma suivant :

$e_1 \rightarrow$ [AP1 = AV2] $\rightarrow e_2$ [AP2 = AV3] $\rightarrow e_3$ [AP3 = AV4]

Cette recherche est rapide puisque pour un e_K déterminé, on connaît le bigramme avant et le bigramme après.



Le taux théorique maximum de compactage peut être de 75 %, mais cela est illusoire, vue la dimension de la matrice. Aussi cette méthode ne peut-elle s'appliquer que si nous ne considérons que les bigrammes ; le taux théorique est alors de 66,6 %.

Remarque :

Cette méthode, comme nous l'avons vu, ne peut s'appliquer que sur un langage structuré, c'est pourquoi nous ne l'avons expérimenté que sur des fichiers de type Algol, Cobol et Gestion.

2.5.2. Algorithme de compactage/décompactage

Voir Annexe 4.

2.5.3. Résultats obtenus

Voir tableau page suivante.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Algol W	457600	62066	1.09	86 %	152496	2.66	66.66 %	0.77	961	2125	80
Source Cobol	763520	156589	1.64	79.5 %	254448	2.66	66.66 %	0.83	948	2020	80
Gestion	730000	226213	2.48	69 %	243216	2.66	66.66 %	0.96	941	1994	80

Résultats de la méthode de Louis-Gavet

En fait, ce taux de compactage est très théorique puisqu'il faut ajouter le volume de la matrice des bigrammes (250 000 caractères), car, contrairement aux méthodes statistiques où nous pouvons, comme nous le verrons dans la conclusion, trouver une méthode qui permettra de ne pas recalculer systématiquement les tables statistiques de référence, ici cette matrice existera quel que soit le volume du fichier.

Aussi cette méthode devient-elle intéressante lorsque le volume des fichiers est important, car dans ces conditions le volume de la matrice devient négligeable. Nous le montrons dans le tableau qui suit pour un fichier de gestion (les volumes sont donnés en caractères).

Volume du fichier	Volume du fichier compacté	Dimension de la matrice	Volume total fichier compacté	Gain réel (en %)
80 000	26 664	250 000	276 664	- 245,8
160 000	53 328	250 000	303 328	- 89,58
320 000	106 656	250 000	356 656	- 11,45
480 000	159 984	250 000	409 984	14,58
960 000	319 968	250 000	569 968	40,62
1 440 000	479 952	250 000	729 952	49,30
1 920 000	639 936	250 000	889 936	53,64
3 840 000	1 279 872	250 000	1 529 872	60,15
7 680 000	2 559 744	250 000	2 809 744	63,41
15 360 000	5 118 488	250 000	5 368 488	65,04

Ainsi voyons-nous que cette méthode ne devient intéressante que pour des fichiers dépassant 1,5 millions de caractères. En fait, elle sera très performante pour les fichiers gestion.

3. ÉVALUATION SUR DES MÉTHODES COMBINÉES

3.1. Introduction

L'idée est de pouvoir évaluer le compactage de données en combinant plusieurs méthodes complémentaires, ceci permettant d'atteindre, théoriquement les taux de compactage les plus élevés possibles dans les limites d'un temps de compactage/décompactage raisonnable. Cela peut être faite de manière successive ou parallèle.

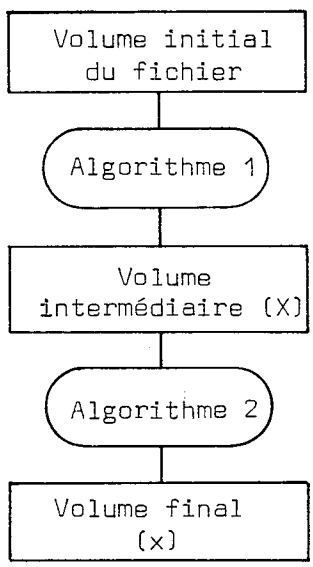
a) Successivement

Si on applique une méthode 1 sur un texte, le texte sera compacté à un volume intermédiaire, par exemple X % du volume initial. En comptactant ce X % à son tour par une autre méthode 2, on peut effectuer un autre pourcentage du gain de place : (100 - x) %,

où x = volume final du fichier compacté,

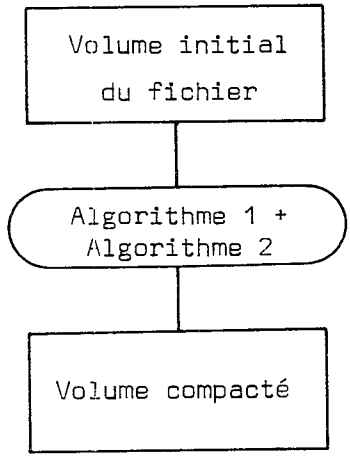
x < X

x << volume initial du fichier



b) Parallèlement

L'enregistrement est traité par plusieurs algorithmes en même temps.



3.2. Méthode basée sur les calculs arithmétiques de code et les délimiteurs

On a expérimenté successivement la méthode HASP et la méthode de HAHN. On compacte d'abord par la méthode d'élimination des caractères répétés, puis par la méthode des combinaisons numériques.

Le buffer compacté produit par la première étape a la même construction que le buffer compacté décrit dans la méthode HASP et le buffer compacté final a une en-tête d'un seul octet (il contient le nombre de caractères compactés), suivi des blocs des valeurs numériques.

Voir le tableau des résultats à la page suivante.

La combinaison des deux méthodes (réduction des caractères répétés et combinaison numérique) n'apporte pas un gain important par rapport à la méthode de Hahn. Cela vient du fait qu'en appliquant la première méthode (HASP), celle-ci crée de nouveaux caractères (les SCB). Ces nouveaux caractères ont les effets suivants :

1) ils rajoutent des symboles au dictionnaire qui devient donc plus long ; nous en avons vu les effets nefastes au § 2.4.4. p. 87.

2) ils possèdent de faibles fréquences, donc occupent toute la fin du dictionnaire, ils sont donc représentés par des symboles plus longs.

Le temps est très élevé parce que le temps de compactage est égal au TCM de la méthode HASP + le TCM de la méthode de combinaisons numériques. Il en est de même pour le temps de décompactage : TDM1 + TDM2.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	212920	2.56	68 %	0.88	1885	1344	80
Source Algol W	457600	62066	1.09	86 %	115208	1.95	77 %	0.88	1823	1396	80
Source Cobol	763520	156589	1.64	79.5 %	214416	2.25	72 %	0.91	1856	1396	80
Module objet	48000	18214	3.03	62 %	52808	8.95	- 10 %	- 0.16	3022	2964	80
Numérique	583040	140183	1.92	76 %	297528	4.06	49 %	0.64	2995	2948	80
Gestion	730000	226213	2.48	69 %	434616	4.78	41 %	0.59	2819	2954	80

Résultats des méthodes Hasp + Hahn

3.3. Méthode basée sur des statistiques et des délimiteurs

On compacte d'abord en employant la méthode d'élimination des caractères répétés, puis la méthode de codage binaire compact (code Huffman). Le buffer compacté intermédiaire a la même construction que s'il était produit par HASP seul et le buffer final a la même construction que s'il était produit par la méthode de Huffman.

Voir le tableau des résultats à la page suivante.

La combinaison de la méthode HASP avec le codage de Huffman conduit à un gain plus grand que celui obtenu par le codage de Huffman seul, mais les temps de compactage/décompactage sont très élevés.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	219762	2.65	67 %	0.87	2375	5980	80
Source Algol W	457600	62066	1.09	86 %	118335	2.08	74 %	0.86	2280	5792	80
Source Cobol	763520	156589	1.64	79.5 %	206464	2.16	73 %	0.92	2223	5776	80
Module objet	48000	18214	3.03	62 %	24062	4.00	50 %	0.80	2415	10465	80
Numérique	583040	140183	1.92	76 %	221923	3.04	62 %	0.81	2254	5726	80
Gestion	730000	226213	2.48	69 %	352485	3.85	52 %	0.78	2468	5982	80

Résultats des méthodes Hasp + Huffman

3.4. Méthode basée sur des statistiques et des troncations

Nous avons combiné la méthode de Huffman et celle de la troncation des blancs en fin d'enregistrement.

Pour un enregistrement donné, les blancs de fin d'enregistrement sont tronqués et le reste du texte est codé par le code de Huffman. Le buffer compacté a la même structure que celle décrite dans le codage binaire de Huffman.

Voir le tableau des résultats à la page suivante.

Les résultats obtenus en testant des fichiers source montrent que cette méthode est plus performante que le codage binaire de Huffman seul.

On remarque, par rapport aux résultats obtenus par le codage de Huffman, une augmentation du gain de place accompagné par une baisse du temps de compactage et décompactage. Cela provient du fait que la troncation des blancs économise de la place pour les mots-code et du temps de codage et décompactage.

On n'obtient pas le même résultat avec des fichiers de type binaire ou numérique, car il y a peu de blancs. Ainsi, le gain reste le même que dans le cas du code Huffman, tandis que le temps augmente un peu à cause de la recherche des blancs.

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM (en m.s.)	TDM (en m.s.)	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	185324	2.24	72 %	0.92	983	3230	80
Source Algol W	457600	62066	1.09	86 %	100865	1.76	78 %	0.91	806	3001	80
Source Cobol	763520	156588	1.64	79.5 %	183514	1.93	76 %	0.96	992	3310	80
Module objet	48000	18214	3.03	62 %	25952	4.32	46 %	0.74	1876	9044	80
Numérique	583040	140183	1.92	76 %	227589	3.12	61 %	0.76	1480	5372	80
Gestion	730000	226213	2.48	69 %	314356	3.45	57 %	0.82	991	3328	80

Résultats des méthodes Huffman + Chopping

3.5. Méthode basée sur les troncations et la méthode Louis-Gavet

Il est évident que cette méthode a sa pleine signification, car cela permet de ne pas stocker dans la matrice complémentaire des données redondantes.

On applique tout d'abord la méthode de troncation, puis celle de Louis-Gavet.

Nous obtenons les résultats portés dans le tableau qui suit, pour les trois sortes de fichiers choisis (cf. raisons expliquées au § 2.5.).

Type données	V_i	V_{mnt}	Entropie	V_c	Nb bits p/carac. compacté	GR	TCM	TDM	Longueur enregist
S.Algol W	457600	62066	1.09	59480	1.03	87.00	723	1632	80
S.Cobol	763520	156589	1.64	119400	1.26	84.36	741	1653	80
Gestion	730000	226213	2.48	204032	2.23	72.05	811	1732	80

Il faut souligner, comme au § 2.5., que ce gain est très théorique du fait de la dimension de la matrice des bigrammes.

Pour ces différents fichiers, nous aurons les gains réels suivants, en fonction de la taille des fichiers en caractères :

Volume du fichier	Gain réel en %		
	Algol	Cobol	Gestion
320 000	8.89	6.25	- 5.06
480 000	34.92	32.28	20.97
960 000	60.96	58.32	47.01
1 440 000	69.64	67.00	55.69
1 920 000	73.98	71.34	60.03
3 840 000	80.49	77.85	66.54
7 680 000	83.75	81.11	69.80
15 360 000	85.38	82.74	71.43

4. MÉTHODE DE CHANGEMENT DE BASE DE NUMÉRATION

4.1. Description de la méthode

L'idée de cette méthode vient du fait que le code EBCDIC a 256 éléments, par contre le jeu de caractères imprimables est réduit à seulement 64 ; il devient donc possible de représenter l'ensemble des caractères imprimables sur 6 bits au lieu de 8 par une recodification appropriée du code EBCDIC.

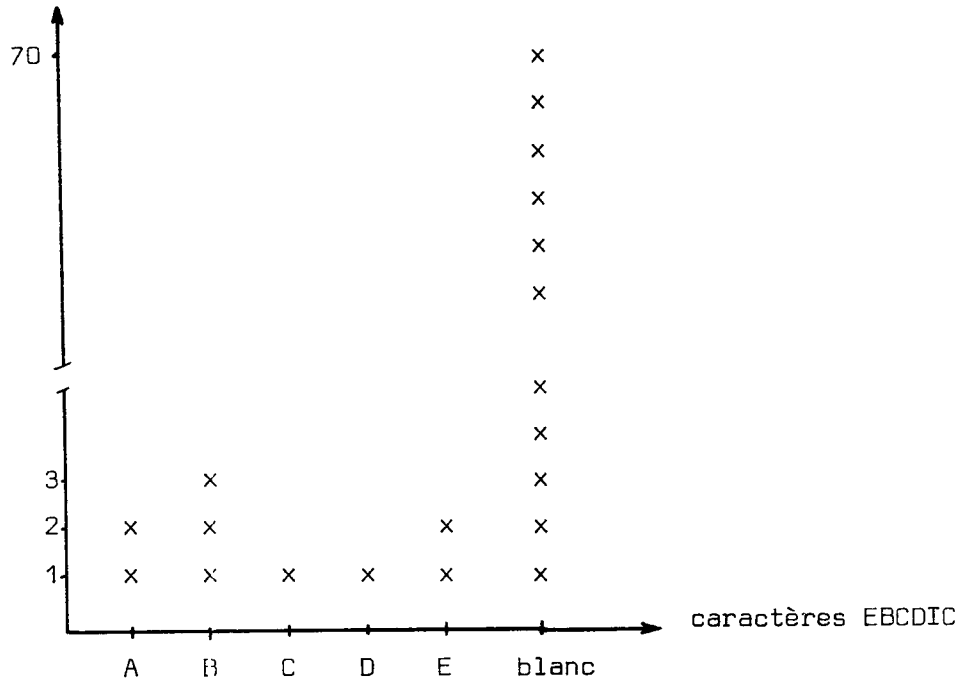
La généralisation de cette méthode consiste à calculer la fréquence des caractères d'un fichier et d'en déduire un code ordonné tel que le caractère le plus utilisé soit représenté par 0 et le moins utilisé par 255.

Disposant d'un code ordonné, on peut rapidement trouver pour un enregistrement le code maximal utilisé dans cet enregistrement. La comparaison de ce code maximal aux puissances de 2 permet de connaître le nombre maximum de bits nécessaires pour représenter chaque caractère de l'enregistrement.

Soit, par exemple, l'image d'une carte :

```
1 2 3 4 5 6 7 8 9 10                    80
|A A B B B C D E E ␣ ␣ ␣ ..... ␣ ␣ ␣ ␣ ␣ ␣|
```

L'histogramme des fréquences de caractères est le suivant :

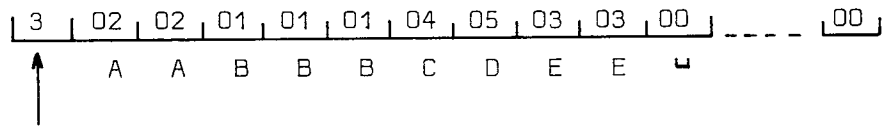


Ce qui donne les codes internes ordonnés :

blanc	00
B	01
A	02
E	03
C	04
D	05

En utilisant ce code, on trouve que l'enregistrement ci-dessus utilise des caractères dont le code ordonné maximal est 5 ; on peut donc représenter l'enregistrement avec des caractères réduits à 3 bits au lieu de 8.

La carte sera alors représentée par :



nombre de bits par caractère

La carte qui comporte 80 x 8 bits = 640 bits est donc désormais représentée par 80 x 3 bits = 240 bits + 1 octet de contrôle en tête.

Cette méthode est très sensible à la présence isolée d'un seul caractère peu utilisé dans un enregistrement. Ceci nous a conduit à l'utiliser en découpant l'enregistrement en zones, de façon à regrouper dans une zone les caractères peu utilisés (à code interne élevé) et dans une autre zone les caractères très utilisés. Dans ce cas, il y aura deux bases différentes pour un seul enregistrement (un pour chaque zone).

Pour regrouper les caractères peu utilisés d'une part, et les caractères très utilisés d'autre part, il a fallu étudier une façon de réordonner les colonnes de l'enregistrement en fonction de leur utilisation.

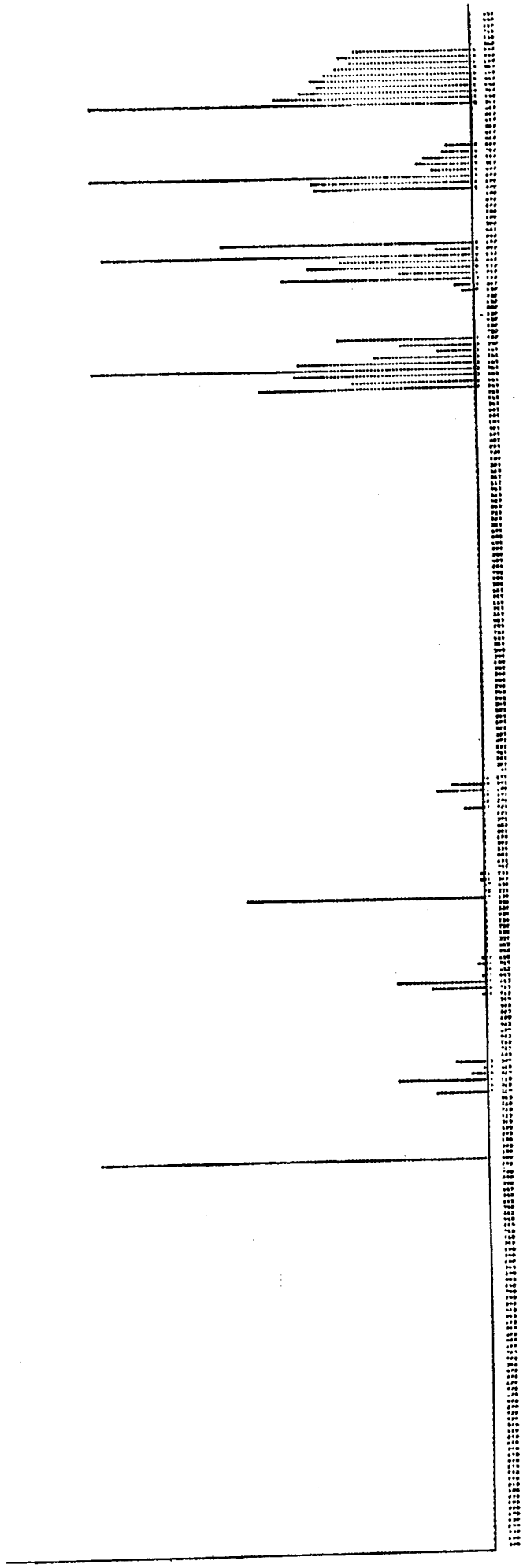
Etude de fréquence

Par une lecture séquentielle, on peut compter le nombre d'occurences des caractères :

a) soit globalement,

b) soit de façon exclusive, c'est-à-dire q'un caractère apparaissant n fois par enregistrement compte pour 1 au lieu de n.

De cette analyse on tire un histogramme des nombres d'apparitions des caractères. La figure qui suite représente un histogramme de fréquence de caractères dans un source Assembleur ; il est pris dans le cas b).



Histogramme des fréquences de caractères dans un texte source

A partir de cet histogramme, il devient possible d'associer un poids W à chaque caractère.

Le choix de la fonction qui fait correspondre les poids de chaque caractère selon l'histogramme de fréquence est très important. Nous avons examiné trois cas :

a) Poids consécutif :

le caractère le plus utilisé a le poids 0, puis le suivant le poids 1, ..., le caractère le moins utilisé a le poids 255.

b) Poids logarithmique :

le caractère a un poids égal au logarithme de poids consécutif (dans ce cas, le poids est exprimé en bits).

c) Poids proportionnel :

le poids d'un caractère est proportionnel à sa fréquence. Ce poids est encodé par le nombre au point fixe : $\frac{f(i) \times 255}{fmax}$

où f(i) = fréquence du caractère i

fmax = fréquence maximale

Disposant du poids W des caractères, il devient possible de calculer le poids Ω de chaque colonne de l'enregistrement. Pour obtenir le poids d'une colonne particulière, il suffit d'encoder par le code ordonné, tout le long du fichier, les caractères qui apparaissent dans cette colonne. Le poids de la colonne est égal à la somme des poids de ses caractères. La figure qui suit représente l'histogramme des poids des colonnes d'un source assembleur. On remarque une vraisemblance entre l'image de la carte représentée par cet histogramme et l'utilisation des colonnes en langage assembleur (en colonne 1, étiquette, en colonnes 10 et 16, instructions, à partir de la colonne 36, le commentaire, et de 73 à 80, la sérialisation).

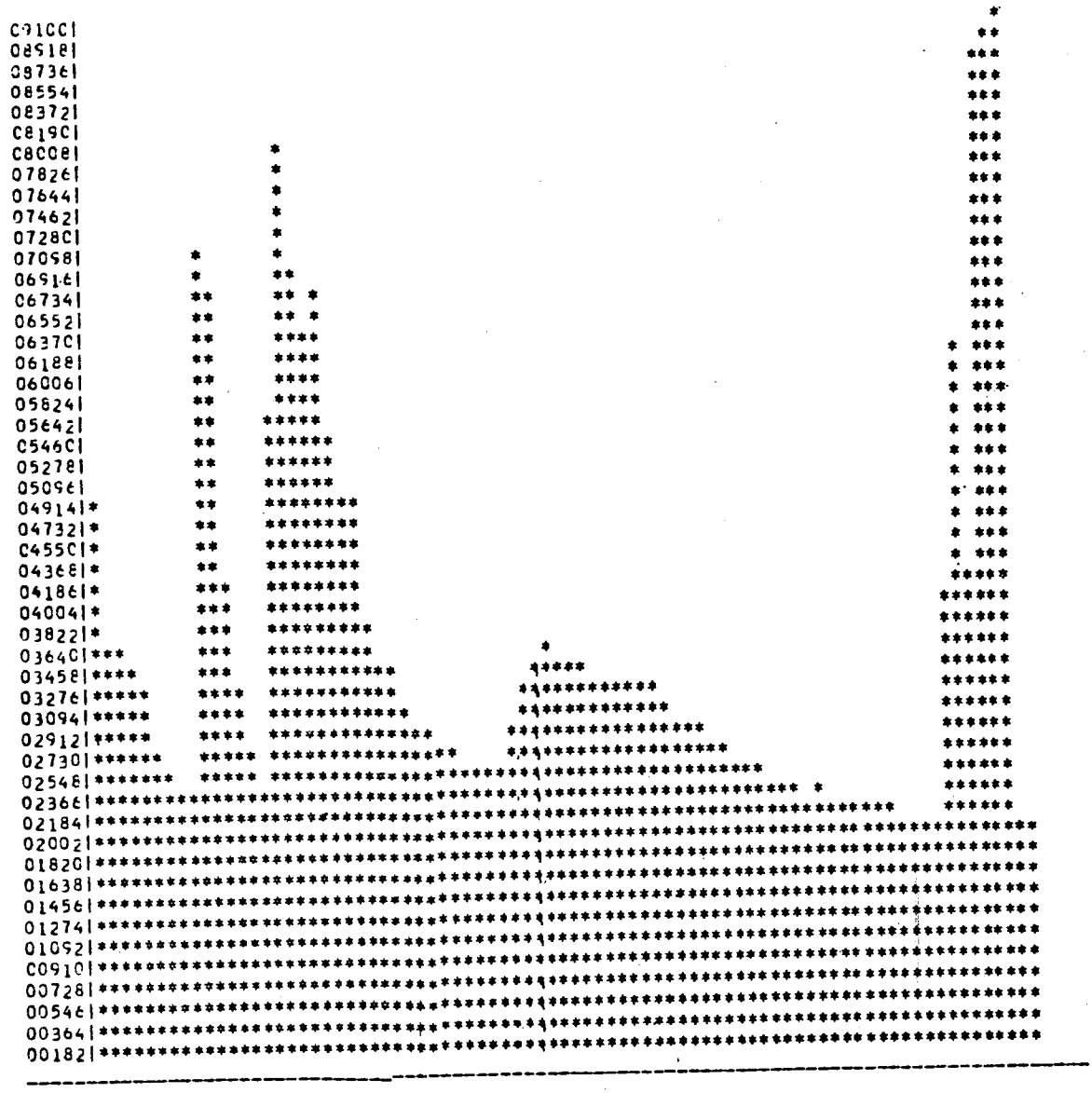


Image d'une carte de source Assembleur

Ceci permet d'ordonner les colonnes par poids décroissant et d'obtenir l'image de carte représentée dans la figure qui suit où les colonnes de poids "lourd" sont regroupées à gauche et les colonnes de poids "léger" sont regroupées à droite.

On peut garder le module des nouvelles positions des colonnes ordonnées et l'utiliser comme image globale des enregistrements de ce fichier.

Une fois que l'on a obtenu cette image, on peut chercher la base maximale dans chacune des deux zones et représenter les caractères par le nombre de bits équivalent à la base utilisée.

L'algorithme en pseudo-Algol est le suivant :

```
Etape 1 : /obtenir les fréquences de caractères dans un fichier F, stocker
          ces fréquences dans la table FREQ/
          Read CARD(RCRD) ;
          for i := 0 until LRECL-1 do
              C := decode (RCRD(i|1)) ;
              FREQ(C) := FREQ(C) + 1 ;

Etape 2 : /céder un code interne aux caractères tel que le caractère le plus
          utilisé soit représenté par 0 et le moins utilisé par 255 ;
          stocker le code dans la table CODE ; MAXC contient le caractère
          à fréquence maximale /
          for i := 0 until 254 do
          begin MAXC := 0 ;
             for j := 0 until 254 do
             begin
                if FREQ(MAXC) < FREQ(j+1) then MAXC := j+1 ;
             end ;
                if FREQ(MAXC) < 0 then go to STOP
                                  else FREQ(MAXC) := -FREQ(MAXC) ;
                                       CODE(MAXC) := i ;
          end ;
```

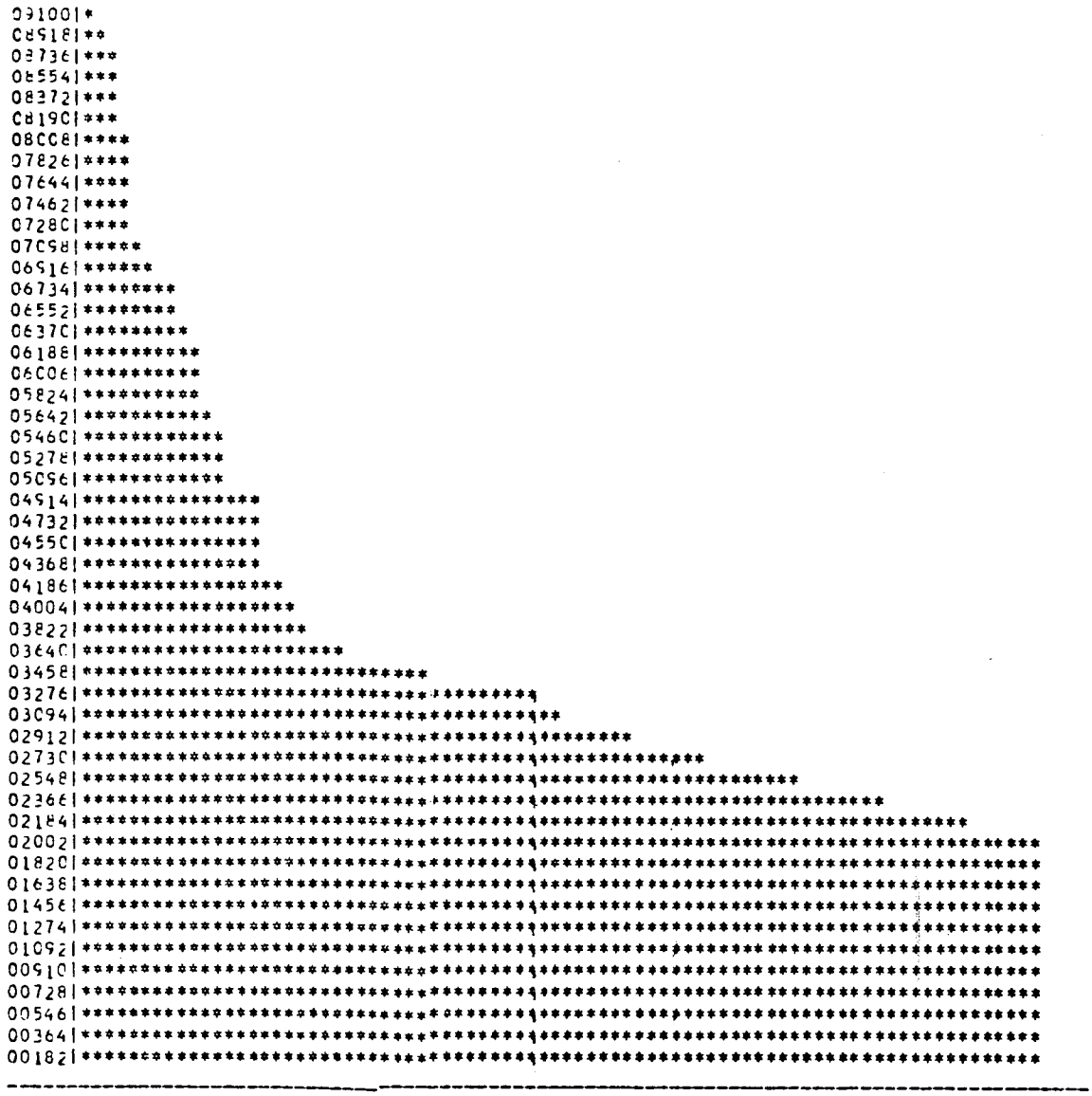


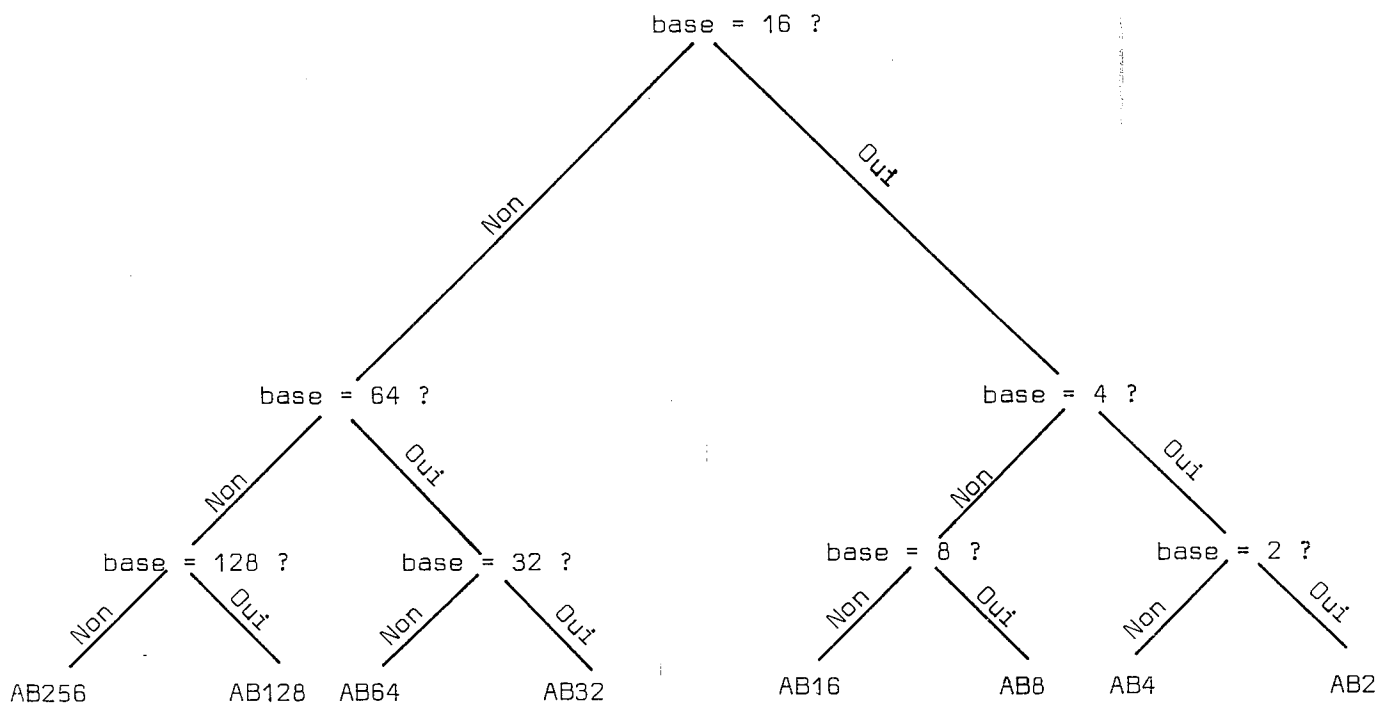
Image de la carte après regroupement des colonnes
les plus utilisées d'une part et les moins utilisées d'autre part

```
            STOP : /CODE contient les codes ordonnés des caractères/
Etape 3 : /à partir du CODE, obtenir les poids logarithmiques des caractères,
            les stocker dans la table POIDS/
            for i := 0 until 255 do
                C := CODE(i) ;
                POIDS(i) := lg(C) ;
Etape 4 : /obtenir les poids des colonnes tout le long du fichier/
            Read CARD(RCRD) ;
           /représenter les caractères par leur poids/
            for i := 0 until LRECL-1 do
            begin
                C := decode (RCRD(i|1));
                RCRD(i/1) := POIDS(C) ;
           /cumul des poids des colonnes ; les stocker dans la table POIDCOL/
            for i := 0 until LRECL-1 do
            POIDCOL(i) := POIDCOL(i) + RCRD(i|1) ;
Etape 5 : /déduire le module de réordonnancement des colonnes, le stocker
            dans la table GRID/
            for i := 0 until LRECL-2 do
            begin MAXCOL := 0 ; /colonne à poids maximal/
               for j := 0 until LRECL-2 do
               begin if POIDCOL(MAXCOL) < POIDCOL(j+1) then MAXCOL := j+1 ;
               end ;
                  if POIDCOL(MAXCOL) < 0 then go to STOP
                                         else GRID(i) := MAXCOL,
                                              POIDCOL(MAXCOL) := -POIDCOL(MAXCOL);
            end ;
            STOP : /GRID contient le module de réordonnancement des colonnes
                     des fichiers F/
```

Etape 6 : /la division de l'enregistrement en deux zones doit être faite de manière à apporter un gain maximal. Alors, pour un type particulier de fichier (les programmes source, par exemple, de Fortran et Algol ayant une structure proche l'une de l'autre), on peut tester une seule fois le rapport entre la longueur de la zone gauche et la longueur de la zone droite qui donne le meilleur résultat. Ce rapport peut être fixe pour un certain type de fichier et on peut le garder comme GRID pour l'utilisation du compactage et du décompactage/

```
/détecter la meilleure position de coupage, changement de longueur par intervalles de 4/
for i := 4 step 4 until LRECL-4 do
begin
   L'zone gauche := Asize := i ;
   L'zone droite := Bsize := LRECL-i ;
```

/PROCEDURE BASE(K,SIZE) chercher la base optimale dans la zone en question ; le graphe ci-après montre le chemin suivi pour détecter la base/



/Les deux paramètres de procédure BASE : K et SIZE, sont successivement la première position d'une zone et la longueur de la zone. La procédure utilise une table MIDDLE dont la structure est illustrée dans la figure ci-dessous. Les zéros et FF de la table servent de fonction pour une instruction TRT. Si tous les caractères de la fonction sont zéros, cela signifie que la zone peut être complètement représentée avec cette base ; si une des fonctions est FF, la zone contient du code dont la base est plus grande que la base testée.

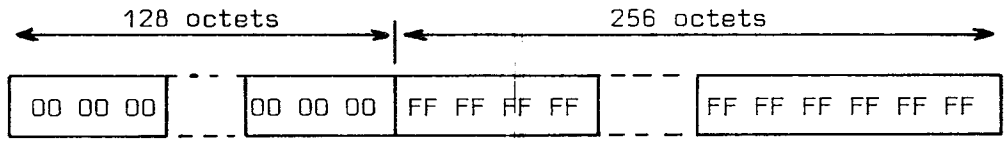


Table MIDDLE

```
Begin read CARD(RCRD)
/convertir les caractères en code ordonné puis ordonner les colon-
 nes selon GRID ; le format réordonné sera stocké dans IMAGE/
for j := 0 until LRECL-1 do
begin C := RCRD(j|1)
      RCRD(j/1) := CODE(C);
end ;
for j := 0 until LRECL-1 do
begin C := GRID(j) ;
      IMAGE(j) := RCRD(C|1);
end;
K := 0 ;
BASE(K,ASIZE)
```

```
AT16 :      for j := 0 until K + ASIZE - 1 do
            begin X1 := 128 - 16 ;
                  X2 := X1 + IMAGE(j)
                  if MIDDLE(X2) ≠ 0 then go to ATT64
            end ;
ATT4 :      for j := 0 until K + ASIZE - 1 do
            begin X1 := 128 - 4 ;
                  X2 := X1 + IMAGE(j) ;
                  if MIDDLE(X2) ≠ 0 then go to ATT8
            end ;
                  Go to ATTT2
ATT64 :     for j := 0 until K + ASIZE - 1 do
            begin X1 := 128 - 64 ;
                  X2 := X1 + IMAGE(j) ;
                  if MIDDLE(X2) ≠ 0 then go to ATTT128
            end ;
                  Go to ATTT32
ATTT2 :     for j := 0 until K + ASIZE - 1 do
            begin X1 := 128 - 2 ;
                  X2 := X1 + IMAGE(j) ;
                  if MIDDLE(X2) ≠ 0 then go to AB4
            end ;
                  Go to AB2

          /de même pour ATTT8, ATTT32, ATTT128/
AB2       /la base de zone gauche est '2' go to PART2/
AB4       /la base de zone gauche est '4' go to PART2/
AB8       /la base de zone gauche est '8' go to PART2/
AB16      /la base de zone gauche est '16' go to PART2/
AB32      /la base de zone gauche est '32' go to PART2/
AB64      /la base de zone gauche est '64' go to PART2/
AB128     /la base de zone gauche est '128' go to PART2/
AB256     /la base de zone gauche est '256' go to PART2/
```

PART2 : /ayant trouvé la base optimale de la zone gauche, répéter pour la zone droite ; on appelle la procédure BASE(K,BSIZE), où K = ASIZE - 1. Quand tous les enregistrements du fichier F sont procédés, on calcule le gain de place puis on recommence pour une autre valeur de ASIZE et BSIZE.

END;

4.2. Résultats obtenus

Voir tableau page suivante.

Les résultats obtenus sont moyens, aussi bien en taux qu'en temps de compactage/décompactage. Cependant cette méthode est très intéressante pour les fichiers de type numérique et "module objet".

Type des données	V_i (en bits)	V_{mnt} (en bits)	Entropie (en bits)	G_{mxt} (en %)	V_c (en bits)	Nb bits par caractère compacté	GR (en %)	$\frac{GR}{G_{mxt}}$	TCM	TDM	Longueur enregist.
Source Assembleur	662400	157433	1.9	78 %	351536	4.25	47 %	0.60	1664	1378	80
Source Algol W	547600	62066	1.09	86 %	274640	4.81	40 %	0.47	1664	1425	80
Source Cobol	763520	156589	1.64	75.9 %	350408	3.68	54 %	0.67	1616	1352	80
Module objet	48000	18214	3.03	62 %	34136	5.67	29 %	0.47	1653	1334	80
Numérique	583040	140183	1.92	76 %	245616	3.36	58 %	0.76	1720	1270	80
Gestion	730000	226213	2.48	69 %	453320	4.97	38 %	0.55	1716	1248	80

Résultats de la méthode de changement de base de numération

5. CONCLUSION

Après avoir évalué intrinsèquement les performances ces différentes méthodes de compactage (classiques ou créées par nous) en fonction des différents langages utilisés, il nous faut les comparer. Ce sera l'objet du chapitre IV.

CHAPITRE IV

SYNTHESE ET CRITIQUE DES DIFFERENTES METHODES DE COMPACTAGE EXPERIMENTEES

. INTRODUCTION.

. SYNTHÈSE DES RÉSULTATS OBTENUS.

. ÉTUDE DES GAINS RÉELS RESPECTIFS DES DIFFÉRENTES MÉTHODES DANS LE CAS D'UN TRANSFERT DE DONNÉES.

. CONCLUSION.

1. INTRODUCTION

Pour que cette étude soit utile et afin d'obtenir des chiffres représentant la performance réelle des différentes méthodes de compactage, il nous reste à étudier ces résultats, à les comparer entre eux pour pouvoir définir la méthode la plus performante, et à trouver le meilleur domaine d'application.

Cependant il nous fait faire une dichotomie entre les résultats obtenus sans considérer le temps de compactage/décompactage, et les résultats obtenus si nous considérons ce dernier.

Le premier point correspond au souci de ne considérer l'efficacité d'une méthode que par son taux de compactage, c'est-à-dire par le gain de place pris sur les supports mémoire.

Le deuxième point considère le temps global de transfert des données, par exemple dans un réseau. Aussi dans ces conditions doit-on tenir compte des temps de compactage/décompactage.

2. SYNTHÈSE DES RÉSULTATS OBTENUS

Pour que les comparaisons soient plus aisées, on introduit tableaux comparatifs

- le premier tableau permet de comparer les méthodes suivant leur taux de compactage et leur facteur d'efficacité,

- les trois tableaux suivants permettent de comparer les temps de compactage/décompactage des différentes méthodes.

2.1. Comparaison des taux de compactage (tableau 1)

D'une façon générale, et cela est normal, les méthodes combinées que nous avons créées, sont plus efficaces que les méthodes simples. Le taux de compactage est exceptionnel (de l'ordre de 80 %) pour certaines d'entre elles pour les langages de type source. Cependant il faut noter que certaines méthodes, comme celle de Hahn, perdent de leur efficacité pour certaines données. Cela vient du fait qu'elles utilisent un grand nombre de symboles.

Par contre, d'autres méthodes, comme celles de Huffman, Shannon-Fano, sont globalement les plus performantes puisqu'elles gardent leur efficacité quel que soit le type de donnée.

Enfin, il faut noter que certaines méthodes sont plus efficaces, comparativement aux autres, par rapport à une classe déterminée de données ; c'est le cas de la méthode de Louis-Gavet pour les fichiers de type gestion, et de la méthode de changement de base de numération pour les fichiers de type numérique et binaire.

Méthode de compactage / type de fichier	Chopping		Hasp		Huffman		Shannon Fano		Hahn		Change. base numérat.		Louis-Gavet		Hasp + Hahn		Hasp + Huffman		Huffman + Chopping		L.Gavet + Chopping	
	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$	GR	$\frac{GR}{G_{mxt}}$
Assembleur	49	0.62	58	0.75	66	0.84	66	0.84	67	0.88	47	0.60			68	0.88	67	0.87	72	0.92		
Algol W	61	0.70	62	0.74	70	0.81	70	0.81	76	0.88	40	0.47	53.6	0.62	77	0.88	74	0.86	78	0.91	73.9	0.85
Cobol	53	0.67	62	0.79	69	0.81	69	0.81	75	0.93	54	0.67	53.6	0.67	72	0.91	73	0.92	76	0.96	71.3	0.89
Module objet	8	0.1	12	0.30	45	0.72	45	0.72	3	0.04	29	0.47			-10	-0.16	50	0.80	46	0.74		
Numérique	0	0.0	- 3	-0.04	60	0.76	60	0.76	53	0.68	58	0.76			49	0.64	62	0.81	61	0.76		
Gestion	16	0.21	23	0.30	51	0.74	51	0.74	42	0.61	38	0.55	53.6	0.77	41	0.59	52	0.78	57	0.82	60.0	0.85

Tableau 1

Taux de compactage (GR) en pourcentage et facteurs d'efficacité

2.2. Comparaison des temps de compactage/décompactage (tableaux 2, 3, 4)

D'une façon générale, les méthodes combinées ont un temps de compactage/décompactage plus élevé car théoriquement les temps de chaque méthode combinée devraient s'additionner.

Il faut noter certaines exceptions (Huffman + Chopping, Louis-Gavet + Chopping), où les temps de compactage/décompactage sont inférieurs à celui des méthodes prises individuellement (Huffman, Louis-Gavet). Cela s'explique fort bien par le fait que la suppression des caractères redondants permet de travailler sur un corpus de données moins volumineux.

Nous remarquons, et cela est normal, que ce sont les méthodes qui font appel aux statistiques qui ont les temps de compactage/décompactage les plus élevés ; ces temps sont d'autant plus élevés que le nombre de symboles que constitue le langage est important.

Comme nous pouvions le penser, ce sont les méthodes les moins sophistiquées (Chopping, Hasp) qui ont les temps les plus bas.

Enfin, remarquons que pour toutes les méthodes, le temps de compactage/décompactage le plus élevé est lorsque l'on travaille sur des fichiers de type binaire, sauf pour la méthode de changement de base.

Méthode / Type de fichier	Chopping	Hasp	Shannon Fano	Huffman	Hahn	Changemt. Base Numérato	Louis-Gavet	Hasp + Hahn	Huffman + Hasp	Huffman + Chopping	L.Gavet + Chopping
Assembleur	487	1244	6852	6864	1742	3042		3229	8355	4213	
Algol W	488	1218	5980	5844	1768	3090	3086	3219	8072	3807	2355
Cobol	542	1235	6658	6574	1798	2968	2968	3254	7983	4302	2394
Module Objet	545	1314	11290	11166	4256	2987		5986	12877	10920	
Numérique	465	1153	6556	6332	4190	2990		5943	7980	6852	
Gestion	471	1245	6944	6858	3878	2964	2935	5773	8450	4319	2543

Tableau 2

Temps total de compactage/décompactage moyen par enregistrement (en micro-secondes)

Méthode / Type de données	Chopping	Hasp	Shannon Fano	Huffman	Hahn	Changem.t base Numérat.o	Louis-Gavet	Hasp + Hahn	Hasp + Huffman	Huffman + Chopping	L.Gavet + Chopping
Assembleur	183	442	5396	5330	780	1378		1344	5980	3230	
Algol W	186	427	4702	4658	858	1426	2125	1396	5792	3001	1632
Cobol	190	425	4958	4940	862	1352	2020	1396	5770	3310	1653
Module objet	248	515	9782	9738	2142	1334		2964	10462	9044	
Numérique	187	421	5122	5008	2136	1270		2948	5726	5372	
Gestion	192	458	5462	5402	2058	1248	1994	2954	5982	3328	1732

Tableau 3

Temps de décompactage moyen par enregistrement (en micro-secondes)

Méthode / Type de données	Chopping	Hasp	Shannon Fano	Huffman	Hahn	Changem.t base Numérat.o	Louis-Gavet	Hasp + Hahn	Hasp + Huffman	Huffman + Chopping	L.Gavet + Chopping
Assembleur	304	802	1456	1534	962	1664		1885	2375	983	
Algol W	302	791	1278	1186	910	1664	961	1823	2280	806	723
Cobol	352	810	1701	1634	936	1616	948	1858	2213	992	741
Module objet	297	799	1508	1428	2114	1653		3022	2415	1876	
Numérique	278	732	1434	1324	2054	1720		2995	2254	1480	
Gestion	279	787	1482	1456	1820	1716	941	2819	2468	991	811

Tableau 4

Temps de compactage moyen par enregistrement (en micro-secondes)

3. ÉTUDE DES GAINS RÉELS RESPECTIFS DES DIFFÉRENTES MÉTHODES DANS LE CAS D'UN TRANSFERT DE DONNÉES

Lorsque l'on transfère des données sur des lignes de communication, le gain de place n'est pas le seul argument du jugement de l'efficacité d'une méthode, car il ne représente pas le gain réel de temps de transfert, à cause du temps perdu en compactage/décompactage.

Aussi, afin de donner une mesure vraisemblable et homogène, partons-nous des hypothèses suivantes :

- soit une ligne de communication d'un débit de 24 K bits/seconde équivalent à 3 octets/milliseconde. Le temps nécessaire pour le transfert de 100 000 caractères est :

$$t_{100\ 000} = \frac{100\ 000}{3} = 33\ 333{,}33 \text{ millisecondes}$$
$$= 33\ 333\ 330 \text{ microsecondes.}$$

- On dit que le gain apparent en temps de transfert de 100 000 caractères est :

$$Gt_{ap} = t_{100\ 000} \times GR\ \%$$

où GR % est le gain réel en place (en pourcentage).

- Le vrai gain en temps de transfert de 100 000 caractères est alors :

$$Gt_{vr} = Gt_{ap} - (TMC_{100\ 000} + TMD_{100\ 000})$$

où $TMC_{100\ 000}$ = temps moyen de compactage de 100 000 octets
$TMD_{100\ 000}$ = temps moyen de décompactage de 100 000 octets.

- Auquel il faudra soustraire éventuellement le temps de transfert de la table des occurences dans les méthodes dites statistiques (cette table variant de 256 à 1024 caractères), d'où :

$$Gt_{vr} = Gt_{ap} - ((TMC_{100\ 000} + TDM_{100\ 000}) + T_{occur})$$

Remarque importante :

On considérera, pour les méthodes liées à des calculs statistiques, que les tables de référence sont normalisées, donc non recalculables à chaque fichier (voir conclusion).

Les tableaux qui suivent montrent les trois valeurs précitées : gain apparent en temps Gt_{ap}, perte de temps en compactage/décompactage (TMC + TMD), gain réel en temps Gt_{vr} et en pourcentage. Toutes ces valeurs sont données en millisecondes pour 100 000 caractères.

Méthode de Chopping

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	16 330	610	15 920	47.5 %
Algol W	20 330	610	19 720	59 %
Cobol	17 660	700	16 960	51 %
Module objet	2 660	700	2 960	5.8 %
Numérique	0	600	- 600	- 2 %
Gestion	5 330	600	4 730	14 %

Méthode de Hasp

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	19 660	1 560	18 100	54 %
Algol W	21 660	1 530	20 130	60 %
Cobol	21 330	1 550	19 780	59 %
Module objet	4 000	1 650	2 350	7 %
Numérique	- 1 000	1 450	- 2 450	- 7.4 %
Gestion	7 660	1 560	6 100	18 %

Méthode de Shannon-Fano

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	21 670	8 560	13 110	39.5 %
Algol W	23 330	7 480	15 850	47.5 %
Cobol	23 000	8 320	14 680	44.5 %
Module objet	15 000	14 110	890	2.7 %
Numérique	20 000	8 170	11 830	35 %
Gestion	17 000	8 300	8 700	26 %

Méthode de Huffman

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	22 100	8 580	13 520	40.5 %
Algol W	23 330	7 300	16 030	48 %
Cobol	23 000	8 210	14 790	44.7 %
Module objet	15 000	13 950	1 050	3 %
Numérique	20 000	7 910	12 090	36 %
Gestion	17 000	8 270	8 730	26 %

Méthode de Hahn

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	22 330	2 170	20 160	60 %
Algol W	25 330	2 210	23 120	69 %
Cobol	24 660	2 240	22 420	67 %
Module objet	1 000	5 320	- 4 320	- 13 %
Numérique	17 660	5 230	12 430	37.5 %
Gestion	14 000	4 840	9 160	27.5 %

Méthode du changement de base de numération

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	16 000	3 800	12 200	36.5 %
Algol W	13 670	3 860	9 810	29.5 %
Cobol	18 000	3 710	14 290	43 %
Module objet	10 000	3 730	6 270	19 %
Numérique	19 670	3 730	15 940	48 %
Gestion	12 670	3 700	8 970	27 %

Méthodes de Hasp + Hahn

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	22 670	4 030	17 270	52 %
Algol W	25 660	4 020	21 640	65 %
Cobol	24 000	4 070	17 200	52 %
Module objet	- 3 330	7 480	- 10 810	- 32 %
Numérique	13 000	7 430	5 570	16.5 %
Gestion	12 330	7 220	5 110	15.5 %

Méthodes de Hasp + Huffman

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	22 000	10 440	11 560	34.5 %
Algol W	24 660	10 090	14 570	43.5 %
Cobol	24 330	9 990	13 170	39 %
Module objet	16 670	16 090	470	1.5 %
Numérique	20 660	9 980	10 680	32 %
Gestion	17 330	10 560	6 770	20 %

Méthodes de Huffman + Chopping

	Gt_{ap}	TMC+TMD	Gt_{vr}	Gt_{vr} en %
Assembleur	24 000	5 260	18 740	56 %
Algol W	26 000	4 760	21 240	64 %
Cobol	25 330	5 370	19 960	60 %
Module objet	15 330	13 650	1 680	5 %
Numérique	20 330	8 560	11 770	35 %
Gestion	19 000	5 390	13 610	41 %

En ce qui concerne la méthode de Louis-Gavet, les taux variant en fonction du volume des données, nous ne ferons apparaître que les gains vrais en pourcentage pour chaque fichier et ceci en fonction du volume des fichiers.

Méthode de Louis-Gavet

Volume fichier (en caractères)	Gain réel en %		
	Algol W	Cobol	Gestion
480 000	5.33	5.68	5.78
960 000	31.37	31.72	31.82
1 440 000	40.15	40.45	40.55
1 920 000	44.39	44.69	44.79
3 840 000	50.90	51.25	51.35
7 680 000	54.16	54.51	54.61
15 360 000	55.79	56.14	56.24

Méthodes de Louis-Gavet + Chopping

Volume fichier (en caractères)	Gain réel en %		
	Algol W	Cobol	Gestion
480 000	27.30	25.10	13.91
960 000	53.34	51.14	39.95
1 440 000	62.02	59.92	48.63
1 920 000	66.36	64.16	52.97
3 840 000	72.87	70.67	59.48
7 680 000	76.13	73.93	62.74
15 360 000	77.76	75.56	64.37

4. CONCLUSION

Pour conclure à partir du tableau récapitulatif qui suit, nous pouvons donner les méthodes les plus performantes pour un langage déterminé dans les deux cas suivants :

- en ne considérant comme critère que le taux de compactage (gain de place),

- en considérant le gain réel de transfert à travers un réseau.

Tableau récapitulatif des gains obtenus (en pourcentage) en fonction du gain de transfert et de stockage

Méthode / Type de fichier	Chopping	Hasp	Shannon Fano	Huffman	Hahn	Changem.t de base	Louis-Gavet	Hasp + Huffman	Hasp + Hahn	L.Gavet + Chopping	Huffman + Chopping
Assembleur	47.5 / 49	54 / 58	39.5 / 66	40.5 / 66	60 / 67	36.5 / 47		34.5 / 67	52 / 68		56 / 72
Algol W	59 / 61	60 / 63	47.5 / 70	48 / 70	69 / 76	29.5 / 40	44.39 / 53.64	43.5 / 74	65 / 77	66.36 / 73.98	64 / 78
Cobol	51 / 53	59 / 62	44.5 / 69	44.7 / 69	67 / 75	43 / 54	44.69 / 53.64	39 / 73	52 / 72	64.16 / 71.34	60 / 76
Module objet	5.8 / 8	7 / 12	2.7 / 45	3 / 45	- 13 / 3	19 / 29		1.5 / 50	- 32 / - 10		5 / 46
Numérique	- 2 / 0	- 7.4 / - 3	35 / 60	36 / 60	37.5 / 53	48 / 58		32 / 62	16.5 / 49		35 / 61
Gestion	14 / 16	18 / 23	26 / 51	26 / 51	27.5 / 42	27 / 38	44.79 / 53.64	20 / 52	15.5 / 41	52.97 / 60.03	41 / 57

N.B. 1. Le chiffre du haut de chaque case indique le gain en fonction du transfert ; le chiffre du bas de chaque case indique le gain en fonction de la place.

2. Les résultats donnés pour la méthode de Louis-Gavet et Louis-Gavet + Chopping le sont pour des fichiers dont le volume est compris entre 1,5 et 2 millions de caractères.

3. Pour les méthodes faisant appel aux statistiques, nous ne tenons pas compte du temps pour créer les dictionnaires de référence (voir conclusion).

Méthodes les plus performantes en fonction du gain de place pour un langage déterminé :

Assembleur	Huffman + Chopping ; Hasp + Hahn ; Hasp + Huffman ; Hahn ; Huffman.
Algol W	Huffman + Chopping ; Hasp + Hahn ; Hahn ; Hasp + Huffman.
Cobol	Huffman + Chopping ; Hahn ; Hasp + Huffman ; Hasp + Hahn ; Huffman.
Module objet	Hasp + Huffman ; Huffman + Chopping ; Huffman.
Numérique	Hasp + Huffman ; Huffman + Chopping ; changement de base.
Gestion	Louis-Gavet + Chopping ; Huffman + Chopping ; Louis-Gavet.

Méthodes les plus performantes en fonction du gain de transfert pour un langage déterminé :

Assembleur	Hahn ; Huffman + Chopping ; Hasp ; Hahn + Hasp.
Algol W	Hahn ; Hahn + Hasp ; Huffman + Chopping ; Hasp.
Cobol	Hahn ; Huffman + Chopping ; Hasp ; Chopping.
Module objet	Changement de base.
Numérique	Changement de base ; Hahn.
Gestion	Louis-Gavet + Chopping ; Louis-Gavet ; Huff.+Chop

En résumé, nous préconisons les méthodes suivantes en fonction des langages et des critères de gain de stockage et de transfert :

Langage	Méthode préconisée en fonction :	
	du gain de stockage	du gain de transfert
Assembleur	Huffman + Chopping (72 %)	Hahn (60 %)
Algol W	Huffman + Chopping (78 %)	Hahn (69 %)
Cobol	Huffman + Chopping (76 %)	Hahn (67 %)
Module objet	Huffman + Hasp (50 %)	Changement de base (19 %)
Numérique	Huffman + Hasp (62 %)	Changement de base (48 %)
Gestion	Huffman + Chopping (57 %) pour volume inférieur à 1,5 millions caractères	Huffman + Chopping (41 %) pour volume inférieur à 1,5 millions caractères
	Louis-Gavet + Chopping (de 60.03 à 71.43 %) pour un volume supérieur à 1,5 millions caractères	Louis-Gavet + Chopping (de 52.97 à 64.37 %) pour un volume supérieur à 1,5 millions caractères

CHAPITRE V

C O N C L U S I O N

. INTRODUCTION.

. CLASSIFICATIONS POSSIBLES.

. MÉTHODE DE DIFFÉRENCIATION DES CLASSES DE DONNÉES.

. VERS UN SYSTÈME ÉVOLUTIF "INTELLIGENT".

. EXEMPLES D'APPLICATION.

. CONCLUSION.

1. INTRODUCTION

Après l'implémentation et le test de différentes méthodes de compactage sur des types variés de fichiers, nous allons essayer d'introduire une classification - ou plutôt des classifications - entre les différentes méthodes et leur domaine d'application.

Le but de cette classification est d'offrir à un utilisateur qui se propose de résoudre un problème donné, de disposer de critères suffisants pour savoir quelle méthode de compactage utiliser pour un corpus déterminé de données.

Par ailleurs, basé sur cette classification des données, nous verrons que le système proposé peut être évolutif.

2. CLASSIFICATIONS POSSIBLES DES MÉTHODES

2.1. Classification selon le caractère de généralité de la méthode

Nous appelons "caractère de généralité de la méthode" sa capacité à conserver son efficacité quel que soit le type de données sur lequel on travaille.

De ce point de vue, il existe deux classes de méthodes :

a) celles qui font une hypothèse précise sur les données manipulées. Ce sont les méthodes de Chopping, Hasp, Louis-Gavet.

Ces méthodes présentent les caractéristiques suivantes :

- elles coûtent peu cher en temps de compactage/décompactage,
- elles permettent un gain important sur les fichiers de type source,
- le gain obtenu est faible (parfois même négatif) sur des fichiers numériques ou des fichiers binaires,
- elles sont généralement faciles à implémenter.

b) Celles qui ne font des hypothèses que sur des caractères particuliers. Ce sont les méthodes de Shannon-Fano, Hahn, Huffman et la méthode de changement de base de numération.

Ces méthodes présentent les caractéristiques suivantes :

- elles sont plus difficiles à implémenter ;
- dans certains cas, le temps de compactage/décompactage est élevé ;
- elles s'adaptent assez bien à tous les types de fichiers et présentent donc un caractère de généralité plus grand que les méthodes de la classe a).

On peut remarquer que cette classification, déduite de l'analyse des résultats, recoupe exactement la classification introduite au début de l'étude entre des méthodes basées sur la suppression de caractères redondants et celles basées sur une étude statistique du corpus des données.

2.2. Classification suivant le mode d'élaboration de la méthode

La classification introduite précédemment fait apparaître que :

- les méthodes de la classe a) s'élaborent sur chaque enregistrement indépendemment du précédent ou du suivant ; ce qui est logique, puisque, par hypothèse de départ, tous les enregistrements sont censés répondre à la même caractéristique : blancs en fin d'enregistrement ou caractères répétés, par exemple ;

- les méthodes de la classe b) ne s'élaborent qu'à partir d'un corpus entier de données (par exemple un fichier) sur lequel un travail préalable doit être fait pour déterminer soit un tableau de fréquences, soit un dictionnaire, soit un nouveau codage.

Dans une première approche nous pouvons faire les remarques suivantes :

. il n'existe pas de méthode "générale" applicable partout et toujours ;

. connaissant la classe d'un enregistrement ou d'un fichier (c'est-à-dire les caractéristiques du corpus auquel il appartient), on peut alors le compacter de manière optimale dans un temps minimum.

Cette dernière remarque nous a amené à étudier des méthodes de comparaison de données différentes afin que, d'une part,

. on puisse créer une liaison biunivoque entre une méthode de compactage et un langage donné,

et que d'autre part,

. on puisse se servir d'une même table de références pour l'élaboration des méthodes de compactage basées sur une étude statistique (classe b), applicable pour plusieurs langages.
Ceci nous permet alors de ne plus recalculer, chaque fois pour un fichier déterminé, les tables de fréquences (Shannon-Fano, Hahn, ...).

3. MÉTHODE DE DIFFÉRENCIATION DES CLASSES DE DONNÉES

3.1. Introduction

Nous avons vu que les résultats obtenus (chapitres III et IV) variaient en fonction de la méthode et du langage.

Pour pouvoir appliquer automatiquement la meilleure méthode, il faut, à partir d'un enregistrement quelconque pris indépendemment de tout corpus de données, que l'on soit capable de déterminer sa classe, c'est-à-dire la nature des données qu'il contient. On pourra alors déterminer la méthode qui donnera le compactage optimal et, s'il doit être compacté par une méthode de classe b), choisir dans une table précalculée, le paramètre à appliquer (tableau de fréquences ou de mots-code, ou dictionnaire).

3.2. Différenciation des langages par application d'une méthode statistique

On peut définir le type des données en calculant l'espérance mathématique et le coefficient de corrélation entre deux échantillons de données différents. Pour cela nous emploierons la méthode de calcul du coefficient de corrélation Ce test est rapide et efficace. On peut arriver à discerner, à partir de cinq enregistrements seulement, la nature des données.

On calculera :

. l'espérance mathématique (X) = $\sum_{i=0}^{i=255} P(i) * i$

. le coefficient de corrélation (CCR) = $\dfrac{\sum_{i=0}^{i=255} (i-x_1) * (i-x_2)}{\sqrt{\sum_{i=0}^{i=255} (i-x_1)^2 * \sum_{i=0}^{255} (i-x_2)^2}}$

où i est le caractère même,

x_1 l'espérance mathématique de l'échantillon 1,

x_2 l'espérance mathématique de l'échantillon 2,

P(i) la probabilité du caractère i.

Les résultats obtenus montrent que le coefficient de corrélation a une valeur particulière selon le type de donnée. (voir Annexe p. 177).

Nous avons fixé la nature du premier échantillon de type source (échantillon 1) et le coefficient de corrélation a été calculé pour reconnaître la nature de l'échantillon 2 inconnu au départ.

Tableau des résultats obtenus :

Echantillon 2	Coefficient de corrélation
Source	1 ⩾ CCR > 0.9
Numérique	0.9 > CCR > 0.7
Gestion	0.8 > CCR > 0.5
Module objet	0.4 > CCR > 0.1

Remarque :

On peut avoir une ambiguité si le coefficient de corrélation calculé varie entre 0.7 et 0.8 puisque l'échantillon 2 peut être considéré de type numérique ou de gestion.

Dans ce cas, on peut différencier les données en utilisant un petit test qui permet, sur un seul enregistrement, de détecter les caractères alphabétiques (instruction TRT).

3.3. Conclusion

Les différentes données à compacter peuvent donc se décomposer en classes, chaque classe étant caractérisée par une valeur déterminée de paramètres. Il y aura ainsi :

- la classe texte source (assembleur, PL/1, ..),
- la classe des modules objets liés à une machine donnée,
- la classe des données numériques,
- la classe des données de gestion.

Ainsi, à partir d'un enregistrement, on est capable par ces méthodes de savoir sur quel langage, sur quel texte on travaille, d'où la possibilité de créer automatiquement une liaison biunivoque "classe de données - méthode de compactage".

Ce qui permettra de créer un logiciel indépendant et transparent vis à vis de l'utilisateur potentiel.

4. ÉLABORATION DE RÉFÉRENCES COMMUNES POUR LES MÉTHODES BASÉES SUR UNE ÉTUDE STATISTIQUE PRÉALABLE

Pour appliquer efficacement une méthode de compactage basée sur une analyse statistique du corpus des données, il faut faire préalablement une analyse statistique qui permet de déduire soit un dictionnaire, soit une table de fréquences ou de codes qui représente en quelque sorte un paramètre de la méthode lié aux caractéristiques du fichier.

Il est évident que ceci représente le défaut majeur de ces méthodes. Aussi avons-nous recherché, pour une classe déterminée de données, si nous ne pouvions pas, a priori, nous servir de paramètres communs. Or, on constate dans la pratique que, pour une classe donnée, ces paramètres sont sensiblement identiques et que, par conséquent, il n'est pas nécessaire de les recalculer pour chaque fichier de cette classe.

Citons un exemple : nous avons utilisé un seul dictionnaire, déduit des données source assembleur, pour compacter d'autres données source : assembleur, cobol et algol W. Nous avons obtenu des temps de compactage/décompactage identiques et une très faible différence de gain en compactage par rapport aux gains réalisés avec leur propre table, comme nous pouvons le voir sur le tableau suivant (cf. aussi p.184)

Fichier	Gain obtenu avec une table liée au fichier	Gain obtenu avec une table référence
Cobol	75 %	74 %
Algol W	76 %	73 %

(Voir en annexe les dictionnaires correspondant aux classes de données, p. 178-182).

5. VERS UN SYSTÈME ÉVOLUTIF "INTELLIGENT"

Il peut arriver que, pour une classe déterminée de données, le système détecte une baisse anormale du taux de gain habituel. Dans ce cas, le système effectuera la procédure normale pour l'élaboration du dictionnaire.

Ainsi, en conservant la trace des dictionnaires produits par ces données "inconnues", on pourra rendre le système évolutif en les rapprochant des dictionnaires normalisés des classes de données connues par le système. De cette façon, il sera alors possible de déterminer de nouvelles classes de données que le système reconnaîtra par la suite.

Supposons, par exemple, que l'on dispose d'une classe "module objet" que l'on compacte par la méthode de Huffman avec un dictionnaire de codes fixes. Si l'analyse des statistiques de compactage montre que le taux baisse à la suite de l'introduction d'un nouveau langage, le système peut décider de transformer la classe "module objet" en classe "inconnue" et donc de recalculer le dictionnaire des codes pour chaque fichier. L'analyse de l'ensemble de ces dictionnaires peut amener à conclure qu'il faut créer (dans notre exemple) deux sous-classes :

"module objet langage X, Y, Z"

"module objet langage Z, T, U".

6. EXEMPLES D'APPLICATION

6.1. Introduction

Pour conclure, il nous a semblé intéressant de montrer deux exemples très différents d'application de nos travaux.

Ceux-ci nous permettront, comme nous l'avions envisagé dans l'introduction (§ 3) :

- une utilisation optimale des lignes de transmission,
- la réalisation d'un stockage géant (bandothèque) sur disque.

Il est à remarquer que ces exemples recoupent parfaitement la dichotomie que nous avons faite entre les performances des méthodes, suivant que nous considérions ou non le temps de compactage/décompactage (chapitre IV, § 4).

6.2. Implémentation d'un compactage dans un réseau d'ordinateurs

Pour introduire le compactage dans un réseau d'ordinateurs, il faut tenir compte des caractéristiques générales de ce réseau. Prenons, par exemple, le réseau CYCLADES [30].

CYCLADES, réseau hétérogène général d'ordinateurs, relie entre elles des machines de marque et de type multiples. Il peut se décomposer de la manière suivante :

- CIGALE, machine de communication, est un sous-réseau indépendant (commutation de paquets) composé de CII Mitra 15 reliés entre eux par des lignes téléphoniques louées. Chaque calculateur participant dialogue avec ses correspondants via CIGALE en utilisant le protocole de transport host - host qui permet l'ouverture de flots entre deux correspondants, un flot étant un lien bidirectionnel qui sert à échanger des données entre portes.

Le protocole appareil virtuel définit les règles d'échange et de dialogue entre deux appareils virtuels standard. Le protocole appareil virtuel peut être utilisé pour un transfert de fichiers comme pour une liaison terminal - ordinateur.

Le réseau est bâti sur une hiérarchie de protocoles telle qu'à un niveau donné il n'est pas besoin de faire d'hypothèse sur la façon dont les niveaux plus externes utiliseront ce service. Il y a donc indépendance des niveaux.

- Au niveau le plus interne, la machine de communication CIGALE assure le transport d'un paquet d'une origine vers une destination. La machine de communication ne fait aucune hypothèse et ne connaît pas le contenu des paquets qu'on lui donne à transporter.

- Le protocole host - host quant à lui :

. ignore le fonctionnement interne de CIGALE,

. ne fait aucune hypothèse sur la nature des données qui seront échangées sur les flots.

On en arrive alors à la question : à quel niveau du réseau CYCLADES faut-il introduire le compactage ?

- Nous avons montré que la structure du réseau impose que la machine de communication ignore tout des données qu'elle transporte. Il est donc exclu d'y inclure le compactage. Par ailleurs, cela serait incompatible avec la philosophie générale de séparation et de hiérarchisation des fonctions du réseau.

- Pour les mêmes raisons il est impossible d'implémenter le compactage au niveau du protocole de transport.

Le protocole d'appareil virtuel paraît le mieux adapté pour supporter le compactage, ceci pour deux raisons :

. d'une part, il offre la possibilité de négocier des options, le compactage pourra être une de ces options ; en effet, on ne saurait obliger "tout le monde" à implémenter les différents types de compactage,

. d'autre part, un échange de données entre deux appareils virtuels a lieu en général sur un type de donnée bien précis auquel on pourra appliquer un certain type de compactage avec la table de référence associée pour une méthode de classe B.

On peut donc imaginer que la solution optimale consiste à demander l'application d'un certain compactage pour un dialogue AV-AV sur un flot. Par exemple, dans une liaison terminal interactif ↔ serveur time-sharing, c'est le *programme* gérant le terminal qui, lors de l'établissement de la liaison AV-AV demandera d'appliquer le compactage correspondant à la classe de données 'texte source'.

Pour un transfert de fichiers utilisant le PAV, c'est le protocole de transfert de fichier qui préciserait la classe de données à partir soit d'une connaissance a priori de ce que contient le fichier, soit d'une analyse faite à partir de quelques enregistrements de celui-ci par la méthode des corrélations décrite au § 3.2.

Remarques :

a) La solution de l'implémentation du compactage au niveau "le plus extérieur" du réseau amène une économie de place lors des stockages intermédiaires.

b) Il est évident que tous les appareils virtuels ne pourront pas disposer de l'ensemble des méthodes de compactage. Le paramètre "méthode de compactage sur la classe des données" devra donc être négocié entre les deux extrêmités.

Le tableau X1 ci-après donne les classes des différents types de données et le tableau X2 montre notre choix des méthodes de compactage selon la classe des données, conformément aux résultats obtenus (chapitre IV § 4 ; Conclusion § 4).

Classe de données	Type de données
0	binaire (module objet)
1	texte source (assembleur, cobol)
2	données numériques
3	données de gestion

Tableau X1

Classe de données	Méthode de compactage à utiliser
0	changement de base
1	Hahn
2	Changement de base
3	Huffman + Chopping ou Louis-Gavet pour les volumes supérieurs à 1,5 millions.

Tableau X2

6.3. Application du compactage pour la réalisation d'un stockage géant de données

L'idée de base est la même que pour un réseau : appliquer le compactage en fonction d'une classe de données.

Le tableau suivant montre notre choix des méthodes de compactage selon la classe des données, conformément aux résultats obtenus (chapitre IV § 4, Conclusion § 4).

Classe de données	Méthode de compactage à utiliser
0	Huffman
1	Huffman + Chopping
2	Huffman
3	Huffman + Chopping ou Louis-Gavet + Chopping (pour les volumes supérieurs à 1,5 M)

Dans cette application, le système de gestion du stockage est transparent vis à vis de l'utilisateur potentiel ; c'est-à-dire que conformément aux outils développés précédemment, ce système est capable :

- de connaître la classe des données,
- d'appliquer la méthode de compactage la plus performante,
- de créer une sous-classe de données (Conclusion § 5) en la dotant d'un nouveau dictionnaire de référence pour avoir les meilleurs résultats possibles.

7. CONCLUSION

7.1. Résumé des résultats

Nous avons développé un certain nombre d'expériences et obtenu des résultats dont la portée dépasse le cadre strict de certaines applications. Revenons brièvement sur ceux-ci :

a) Etude expérimentale de méthodes utilisées sur le marché

Nous avons essayé de faire une étude exhaustive des méthodes de compactage utilisées sur le marché et tenté de les améliorer et de voir leur meilleur domaine d'application.

b) Etude expérimentale de nouvelles méthodes

Nous avons créé de nouvelles méthodes en combinant les caractéristiques de certaines d'entre elles. Les résultats sont relativement spectaculaires, puisque nous obtenons des gains réels de stockage de l'ordre de 75 à 80 % pour les langages les plus utilisés.

c) Création d'une méthode basée sur le changement de base de numération

Cette méthode est conçue pour compacter les fichiers binaires et numériques pour lesquels nous n'avions obtenu que des résultats relativement médiocres avec les autres méthodes.

d) Création de dictionnaires de référence communs

Comme nous l'avons vu, le très gros handicap des méthodes basées sur les statistiques était que, pour tout nouveau fichier, il fallait recalculer les dictionnaires de référence où sont notées les probabilités d'apparition d'un caractère ou d'un groupe de caractères. En nous appuyant sur une méthode de reconnaissance automatique d'une classe de données, nous avons créé des dictionnaires communs, évitant ainsi tout re-calcul systématique, d'où un gain de temps important.

e) Création d'un système évolutif "intelligent"

Comme nous l'avons souligné précédemment, le système basé sur une reconnaissance automatique des données peut donc employer la meilleure méthode possible de compactage pour un langage déterminé. De plus, si le système détecte, pour une classe déterminée de données, une baisse anormale dans le gain, il pourra créer d'autres dictionnaires de référence correspondant à d'autres classes de données, permettant ainsi d'obtenir le meilleur taux de compactage quel que soit le type des données.

7.2. Perspectives d'avenir

Elles se situent dans la mise en pratique du point e). Il est évident que les outils que nous avons développés nous permettront de construire un logiciel indépendant, transparent vis à vis des utilisateurs.

ANNEXES

MÉTHODE DE CHOPPING

Algorithme de compactage en pseudo-Algol

In : contient le texte à compacter
OUT : adresse de rangement du texte compacté
i : index de déplacement dans le buffer IN
j : index de déplacement dans le buffer OUT
n : nombre de caractères à compacter

On dispose de la procédure suivante :

COMPCT(i,l,j)

qui crée le SCB2 et une chaîne de longueur l à partir des caractères d'indice i de IN ; la chaîne produite est rangée à l'adresse OUT(j).

L'algorithme de compactage est alors le suivant :

1. i = n-1 ; j = 0 ; K = 0 ;
2. si (IN(i) := IN(i-1) et (i > 0) alors go to 4
   sinon continue
3. i := 0 ; l := n-K+1 ; j = 0 ;
   COMPCT(0,n-K+1,0) ; STOP
4. K := K+1 ; i := i-1 ; go to 2
5. STOP/OUT contient le résultat de longueur l.

Algorithme de décompactage :

Soit la chaîne IN de longueur n à décompacter et soit N la longueur initiale des enregistrements non compactés.

1. i := 0 ; j := 0 ;
2. si i ⩽ n alors continue
   sinon go to 4
3. OUT(j) = IN(i) ; i := i+1 ; j := j+1 ; go to 2
4. si j ⩽ N alors continue
   sinon STOP
5. OUT(j) := IN(n) ; j := j+1 ; go to 4
6. STOP

MÉTHODE HASP

Algorithme de compactage en pseudo-Algol

IN : contient le texte à compacter

OUT : adresse de rangement du texte compacté

i : index de déplacement dans le buffer IN (i varie de zéro à n-1)

j : index de déplacement dans le buffer OUT

n : nombre de caractères à compacter.

On dispose de deux procédures :

1) COPY(i,l,j) :

/la procédure COPY crée une chaîne de longueur l à partir du caractère d'indice i de IN ; la chaîne produite est rangée à l'adresse OUT(j) et j progresse sur le prochain caractère de OUT à remplir/

1. créer le SCB2 et la chaîne de caractères (de longueur maximale lgr = 63)
2. faire progresser j, (j = j+l+1)
3. si lgr ⩽ 63 alors STOP
   sinon l = l-63 ; i = i+63 ; go to 1

2) BLOC(i,l,j) :

/la procédure BLOC est analogue à COPY, mais pour des caractères répétés/

```
1. si IN(i) = blanc alors OUT(j) = SCB4 ; j = j+1 ; go to 3
                    sinon OUT(j) = SCB3 ; OUT(j+1) = IN(i) ; j = j+2
2. si lgr > 63 alors l = l-63 ; i = i+63 ; go to 1
3. si lgr > 31 alors l = l-31 ; i = i+31 ; go to 1
```

L'algorithme de compactage est alors le suivant :

```
1. i = 0 ; j = 0 ; K = 0 ;
2. si i ≥ n-1 alors go to 11
              sinon continue
3. si IN(i) ≠ IN(i+1) alors go to 10
                    sinon continue
4. si K ≠ 0 alors COPY(i-K,K,j)
            sinon continue
5. K = 2
6. si ((IN(i) = IN(i+K)) et (i+K < n)) alors K = K+1 go to 6
                                       sinon continue
7. BLOC(i,K,j)
8. i = i+K
9. si i = n-1 alors COPY(n-1,1,j) ; go to 12
              sinon K = 0 ; go to 1
10. i = i+1 ; K = K+1 ; go to 1
11. si K ≠ 0 alors COPY(i+1-K,K,j)
            sinon continue
12. si j < n-1 alors STOP
              sinon j = n ; OUT(0,n-1) = IN(0,n-1)
13. STOP
```

/OUT contient le résultat de longueur j/

Algorithme de décompactage :

Soit la chaîne IN à décompacter :

```
1. i = 0 ; j = 0 ;
2. si i < n alors continue
              sinon STOP
3. lgr = IN(i)
4. si SCB = SCB2 alors OUT(j,lgr) = IN(i+1,lgr)
                       j = j+lgr
                       i = i+1+lgr
                       go to 2
                  sinon continue
5. si SCB = SCB3 alors OUT(j,1) = IN(i+1,1)
                       OUT(j+1,lgr-1) = OUT(j,lgr-1)
                       i = i+2
                       j = j+lgr
                       go to 2
                  sinon OUT(j,1) = blanc
                       OUT(j+1,lgr-1) = OUT(j,lgr-1)
                       i = i+1
                       j = j+lgr
                       go to 2
6. STOP
```

CODE SHANNON - FANO

Algorithme de compactage :

IN : contient le texte à compacter

OUT : adresse de rangement de l'information compactée

i : index de déplacement dans le buffer IN (i varie de zéro à n-1)

j : index de déplacement dans le buffer OUT

n : nombre de caractères à compacter par enregistrement

COD : contient les mots-code classés selon la suite normale des caractères EBCDIC (0, 1, ..., 255) ; les mots-code sont de longueur variable (expérimentalement, la longueur maximale d'un mot-code se situe entre 14 et 17 bits) ; ces mots-code sont stockés dans la table COD par unité de mot ordinateur complet (32 bits) pour chacun

NBITS : contient les longueurs (en bits) des mots-code de tous les caractères

COUNTBIT : compteur du nombre de bits de plusieurs mots-code

C : domaine du caractère à coder

MOTCOD : domaine du code du caractère dans C

NB : contient la longueur du code dans MOTCOD

K : compteur de caractères par enregistrement à compacter

INTERMID : buffer intermédiaire de longueur 32 bits ; contient le groupe de mots-code

BACK : contient le nombre de bits supérieure à 32 dans INTERMID.

Comme les codes ont une longueur de bits variable, la manipulation de stockage des mots-code dans le buffer OUT doit être faite bit par bit ; mais pour accélérer l'opération de compactage, on dispose des quatre procédures suivantes :

1) MANIPREG(NB,MOTCOD,INTERMID)

cette procédure regroupe un nombre de mots-code dans un buffer intermédiaire d'un mot ordinateur (32 bits) pour pouvoir les stocker dans OUT, mot par mot, ce qui est beaucoup plus rapide que bit par bit ; par exemple :

- R8 est le buffer INTERMID qui contient quelques bits de mots-code précédents (< 32 bits)
- R9 est MOTCOD
- R3 est NB

donc MOTCOD est réjouté à INTERMID comme suit :

```
SLL R8,0(R3) vider la place pour le prochain code
OR R8,R9
```

Si INTERMID est rempli, mais déborde de quelques bits, la procédure :

2) BCKSHIFT(BACK,COUNTBIT,j,MOTCOD,INTERMID)

tient compte de cela ; elle enlève les bits en plus, de telle sorte que INTERMID ne contient plus que les 32 bits utilisables ; INTERMID est stocké dans OUT(j), j progresse de 4 octets sur OUT, puis COUNTBIT est réglé sur le nombre de bits précédemment enlevés, c'est-à-dire COUNTBIT = COUNTBIT-32.

3) La procédure MANIPOCT(COUNTBIT,j) fonctionne quand 8 < COUNTBIT < 32 ; elle stocke les bits de mots-code octet par octet ; j progresse sur le prochain octet de OUT à remplir.

4) La procédure MANIPBIT(COUNTBIT,j) fonctionne lorsque COUNTBIT < 8 elle stocke les codes bit par bit.

L'algorithme de compactage est alors le suivant :

```
1. i := 0 ; j := 2 ; K := 0 ; COUNTBIT := 0 ; INTERMID := 0 ;
2. C := IN(i) ; i := i+1 ; K := K+1 ; NB := NBITS(C) ; MOTCOD := COD(C);
   COUNTBIT := COUNTBIT+NB ;
3. si COUNTBIT < 32 alors MANIPREG(NB,MOTCOD,INTERMID) ; go to 4
                    sinon go to 5
4. si K < n alors go to 2
            sinon go to 6
5. BACK := 32-NB ; BCKSHIFT(BACK,COUNTBIT,j,MOTCOD,INTERMID) ;
   si K < n   alors go to 2
              sinon continue
6. si COUNTBIT < 8 alors go to 7
                   sinon MANIPOCT(COUNTBIT,j) ; continue ;
7. si COUNTBIT := 0 alors go to 8
                    sinon OUT(0) := j+1 ;
                          OUT(1) := COUNTBIT ;
                    MANIPBIT(COUNTBIT,j) go to STOP
8. OUT(0) := j ;
   OUT(1) := 0 ;
9. STOP    /l'enregistrement IN est compacté dans le buffer OUT/
```

Algorithme de décompactage :

Pour décompacter un enregistrement codé en code Shannon-Fano, on a besoin des quatre tables suivantes :

1) DCMPCT : contient les mots-code de caractères, classés selon l'ordre de probabilité des caractères ; la raison du choix de ce classement est de diminuer le temps de recherche du mot-code adéquat.

2) CHAR : contient les caractères eux-mêmes, classés selon leur probabilité.

3) ADRS : contient les index de déplacement dans DCMPCT ; chaque index pointe une série de mots-code ayant une même longueur définie.

4) LOOP : contient le nombre de mots code dans chaque série de mots-code.

Pour décompacter l'information reçue sous forme d'un ensemble de bits, il faudrait d'abord séparer les mots-code les uns des autres, puis trouver le caractère correspondant.

Nous sommes obligés de tester bit par bit ; si l'information correspond à un mot-code dans la table DCMPCT, le caractère correspondant existe dans CHAR avec le même déplacement du mot-code dans DCMPCT ; sinon on rajoute un deuxième bit de l'information compactée et on recommence le test pour deux bits au lieu de un, et ainsi de suite.

En pratique, s'il y a i bits à tester, on commence d'abord par chercher s'il existe un mot code ayant la longueur (i). Par exemple, si loop(i) := 0, cela signifie qu'il n'existe pas de mot-code de longueur i ; on rajoute alors un $(i+1)^{ème}$ bit et on recommence à chercher dans loop pour (i+1). Si loop(i+1) = 3, cela signifie qu'il existe trois mots-code ayant la même longueur (i+1). Il faut alors chercher la position de ces codes dans DCMPCT, puis regarder dans ADRS(i+1) pour avoir l'adresse de ces trois codes.

Il devient alors facile de les comparer avec les (i+1) bits à tester. Si la comparaison avec un de ces trois bits est bonne, on cherche le caractère équivalent dans CHAR et on le transforme dans un buffer OUT. Sinon, on rajoute un $(i+2)^{ème}$ bit et on recommence à chercher dans loop(i+2), et ainsi de suite.

Le fait que la longueur du buffer compacté soit stockée dans l'en-tête détermine la fin du décompactage.

IN	: buffer à décompacter
i	: index de déplacement dans IN
OUT	: adresse du texte décompacté
j	: index de déplacement dans OUT
INTERMID	: domaine des bits à tester
CMTR	: nombre de bits à tester
LP	: entier
POINT	: entier
L,S,X,Z,m	: entiers
Loop	: on divise les codes en sous-groupes égaux en longueur ; loop indique le n° de code dans chaque sous-groupe
ADRS	: indique les adresses de chaque sous-groupe de codes
CHAR	: domaine d'un caractère.

On dispose de la procédure SHIFTBITS(i,INTERMID,CMTR) qui fait bouger une série de 32 bits de IN dans le buffer (un mot ordinateur), puis ces bits sont décalés dans INTERMID avec un débit d'un bit à la fois.

Cet algorithme profite de l'instruction SLDL pour accélérer le temps de comparaison entre les bits reçus et les mots-code.

Supposons que le buffer IN ait la longueur :

$$32 * X + x \text{ bits}$$

où X est le nombre de groupes de 32 bits

x est le nombre de bits inférieur à 32.

L'algorithme de décompactage est le suivant :

```
1.  m := 32 ;
2.  i := 2 ; j := 0 ; INTERMID := 0 ; CMTR := 0 ;
3.  for Z := 0 until X-1 do
4.  begin for L := 0 until m-1 do
5.  begin SHIFTBITS(i,INTERMID,CMTR) ;
        LP := LOOP(CMTR) ;
        if LP = 0 then go to 7
        else POINT := ADRS(CMTR) ;
        for S := 0 until LP-1 do
6.      begin
           if INTERMID = DCMPCT(POINT) then
              OUT(j) := CHAR(POINT)
              j := j+1 ; INTERMID := 0 ;
              CMTR := 0 ;
7.      end ;
8.  end ;
9.  end ;
10. if x > 0 then m := x ; x := 0 ; go to 4
              else STOP
11. STOP
```

MÉTHODE DE COMBINAISONS NUMÉRIQUES
(MÉTHODE DE HAHN)

Il s'agit de la combinaison de trois tables : la première combinée à la troisième servira au compactage, la première combinée à la deuxième servira au décompactage.

1) POWER :
contient les valeurs B^{n-1}, B^{n-2}, ..., B°

2) DIC (dictionnaire) :
contient les caractères rangés selon deux modes :

a) les caractères sont ordonnés dans l'ordre décroissant de leurs probabilités,

b) si la longueur apparante du dictionnaire B est supérieure à la longueur réelle, il faut vider la $(xB)^{ième}$ position $(1 \geqslant x \leqslant 256)$ en décalant les caractères qui existent dans des index supérieurs ou égaux à B. Considérons par exemple le cas d'un dictionnaire de 15 caractères (longueur réelle du dictionnaire). D'après la condition a), la forme de DIC est la suivante :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	E	M	A	Ø	S	L	O	D	(	G	I	)	1	,

Si B > 15 les caractères auront les positions indiquées dans le schéma ci-dessus. Mais si, par exemple, on choisit comme longueur apparente B = 7 (B < 15), DIC aura la forme suivante :

						1B							2B			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
R	E	M	A	Ø	S	00	L	O	D	(	G	I	00	)	1	,

Chaque $(xB)^{ième}$ position contient "O" ; notons que le caractère L, par exemple, a la position $P_L = 7$ dans le premier cas et $P_L = 8$ dans le deuxième cas. Cette forme de dictionnaire (DIC) sera utilisable pour le décompactage.

3) Tandis que la table POSITION est établie pour l'algorithme de compactage. Cette table contient les positions des caractères telles que décrite dans le second schéma ci-dessus.

Algorithme de compactage :

IN : contient le texte à compacter

i : index de déplacement dans IN

L : longueur fixe de IN

OUT : adresse de rangement des combinaisons numériques équivalentes au texte à compacter

j : index de déplacement dans OUT

m : compteur de blancs dans le texte à compacter

adress1 : point du premier caractère à compacter sous forme de valeur numérique

adress2 : point du dernier caractère à compacter sous forme de valeur numérique

LC : nombre de caractères à compacter sous forme de valeur numérique
COUNTC : compteur de caractères
POSC : contient la position d'un caractère particulier
C : contient le caractère dont la position est dans POSC
B : longueur apparente de DIC
N : nombre de caractères à compacter par une seule valeur numérique
COMPVALU : contient une valeur numérique qui représente N caractères
NT : index de déplacement dans POWER

L'algorithme de compactage est le suivant :

```
1. i := 0 ; j := 0 ; m := 0 ;
2. /tronquer les blancs en-tête de IN, indiquer leur nombre dans le premier
    octet de OUT/
   for i := 0 until L do
   begin
      if IN(i) = ␣ then m := m+1 ;
                   else go to 3
   end ;
3. ADRESS1 := i ;
   OUT(j) := m ;
4. /tronquer les blancs de fin/
   m := 0 ;
   for i := L-1 step -1 until ADRESS 1 do
   begin
      if IN(i) = ␣ then m := m+1 ;
                   else go to 5
5. ADRESS2 := i ;
   LC := ADRESS2 - ADRESS1 ;
   OUT(j+1) := LC ;
   COUNTC := 0 ;
   i := ADRESS1 ; j := 2 ; NT := 0 ;
```

```
    /compacter les caractères de longueur LC ; chaque n caractère est repré-
     senté par une valeur numérique/
 6. if COUNTC ≮ LC then STOP
                    else C := IN(i) ;
                         POSC := POSITION(C) ; continue
 7. if POSC > B then go to 8
                 else go to 9
 8. NT := NT+1 ;
    POSC := POSC-B ; go to 10
 9. NT := NT+1 ;
    COMPVALU := COMPVALU + POWER(NT) ⋇ POSC ;
    COUNTC := COUNTC + 1 ;
    i := i+1 ; continue ;
10. if NT ≮ n then OUT(j) := COMPVALU ;
                   j := j+1 ;
                   NT := 0 ; go to 6
               else continue
11. if COUNTC < LC then go to 6
                   else OUT(j) := COMPVALU ;
12. STOP /le texte dans IN est compacté dans OUT/
```

Algorithme de décompactage :

L	: longueur fixe de l'enregistrement initial (non compacté)
NBLNK	: contient le nombre de blancs d'en-tête
NCHAR	: contient le nombre de caractères compactés sous forme de combinaisons numériques
IN	: buffer à décompacter
i	: index de déplacement dans IN
OUT	: adresse d'un buffer où on range les caractères décompactés
j	: index de déplacement dans OUT
NT	: index de déplacement dans POWER
POS1	: portion de l'index d'un caractère dans DIC (valeur réelle)
POS	: contient la valeur entière de POS1
INDEX	: index exact de la position d'un caractère dans DIC
C	: compteur de caractères décompactés
VALUE	: variable réelle
K	: variable entière
LBC	: variable entière

Le buffer à décompacter IN contient un en-tête de deux octets : le premier octet indique le nombre de blancs d'en-tête tronqués dans l'enregistrement initial, le deuxième octet indique le nombre de caractères compactés sous forme de combinaisons numériques ; donc :

```
NBLNK := IN(0) ;
NCHAR := IN(1) ;
```

```
1. /décompacter les blancs d'en-tête/
   for j := 0 until NBLNK-1 do
   OUT(j) := ␣ ;
2. /décompacter les caractères non triviaux/
   i := 2 ;
   C := 0 ;
   INDEX := 0 ;
```

```
3. /prendre une combinaison numérique/
   COMPVALU := IN(i) ;
   NT := 0 ;
4. /analyser, par une série de N divisions, la valeur numérique ; obtenir
    de chaque division (ou de plusieurs divisions) la position d'un carac-
    tère compacté dans le dictionnaire DIC/
   POS1 := COMPVALU/POWER(NT) ;
   POS := INTEGER(POS1) ;
   VALUE := POS * POWER(NT) ;
   COMPVALU := COMPVALU-VALUE ;
   /si la position obtenue est zéro, le caractère a une position supérieure
    à B/
   if POS := 0 then INDEX := INDEX + B ;
                    NT := NT+1 ;
               else /position exacte du caractère obtenue/
                    INDEX := INDEX+POS ;
                    NT := NT+1 ;
                    OUT(j) := DIC(INDEX) ; /stocker caractère dans OUT(j)/
                    INDEX := 0 ; j := j+1 ;
                    C := C+1 ; continue
5. /tester si tous les caractères sont décompactés/
   if C ≮ NCHAR then go to 7
                else continue
6. /tester si cette combinaison numérique est complètement analysée/
   if NT < N then go to 4
             else i := i+1 ; go to 3
7. /décompacter les blancs en fin d'enregistrement/
   LBC := NBLNK + NCHAR ;
   for j := LBC until L do
       OUT(j) := ␣ ;
8. STOP /le buffer IN est décompacté/
```

ALGORITHME DE COMPACTAGE
DE LA MÉTHODE DE LOUIS-GAVET

IN : contient le texte à compacter

i : index de déplacement dans IN

n : nombre de caractères dans IN

MAT(0::500,0::500) : table des

j,k : index de déplacement dans MAT

TABAV(0::900) : table des adresses des bigrammes "avant"

TABAP(0::900) : table des adresses des bigrammes "après"

AV : bigramme avant

AP : bigramme après

ADR : contient l'adresse du bigramme courant

INCAR : contient le caractère courant.

L'algorithme de compactage est :

```
(1)     i := 0 ; j := 0 ; k := 0 ;
(2)     MAT(j,k) := IN(i)
(3)     /calcul de l'adresse du bigramme (IN(i+1),IN(i+2))
         donc AP ≡ (IN(i+1),IN(i+2));
          et ADR ≡ adresse de AP/
(4)     TABAP(ADR) := k ;
(5)     i := i+3 ;
        if i > n then go to 12
                 else INCAR := IN(i) ; continue ;
(6)     if TABAV(ADR) := 0 then to go 7
                           else j := TABAV(ADR);
                                     go to 8
(7)     j := j+1 ; TABAV(ADR) := j ;
(8)     /calcul de l'adresse du bigramme (IN(i+1),IN(i+2))
         ADR := adresse du bigramme (IN(i+1),IN(i+2)/
        if TABAP(ADR) := 0 then go to 9
                           else k := TABAP(ADR);
                                go to 10
(9)     k := k+1 ; TABAP(ADR) := k ;
(10)    MAT(j,k) := INCAR ;
(11)    go to 5
(12)    STOP /le texte dans IN est compacté dans MAT/
```

L'algorithme de décompactage est :

```
(1)     f := 0 ; h := 0 ; j := 0 ; k := 0 ; i := 0 ; l := 0 ; m := 0 ;
        count := 0 ;
(2)     OUT(i) := IN(m) ; L := IN(M) ; count := count+1 ;
        if count ≠ n then go to (3) else go to STOP
(3)     for j ;= h until 500 DO
        begin for K := f until 500 DO
           begin if MAT(j,k) = L then go to (5)
           end ;
        end ;
(4)     erreur 1 ; STOP
(5)     f := k+1 ; h := j ;
        for l := 0 until 900 DO
        begin if TABAP(l) = k then go to (7)
        end ;
(6)     erreur 2 ; STOP
(7)     ADR := l ; /reconnaissance du bigramme après (AP)/
        OUT(i+1) := AP1 ; OUT(i+2) := AP2 ; m := m+1 ;
(8)     OUT(i+3) := IN(m) ; L := IN(m) ; i := i+3 ;
        count := count+1 ; if count ≠ n then go to (9) else go to STOP
(9)     j := TABAV(l) ;
        for k := 0 until 500 DO
        begin if MAT(j,k) = L then go to (11)
        end ;
(10)    /détecte erreur/
(11)    for l := 0 until 900 DO
        begin if TABAP(l) = k then go to (13)
(12)    /détecte erreur/
```

```
(13)    ADR := 1 ; OUT(i+1) := AP1 ; OUT(i+2) := AP2 ;
        m := m+1 ; i := i+3 ; count := count+1 ;
        if count / n then go to (14) else go to STOP
(14)    OUT(i) := IN(m) ; L := IN(m) ;
(15)    j := TABAV(1) ;
        for k := 0 until 500 DO
        begin if MAT(j,k) = L then go to (11)
        end ;
(16)    /détecte erreur/
(17)    STOP /le texte dans IN est décompacté dans OUT/
```

Coefficients de correlation entre des échantillons de différentes classes de données

C C R	Echantillon 1	Echantillon 2
1.000000	Source assembleur	Source assembleur
0.9992339	Source assembleur	Source assembleur
0.9996431	Source assembleur	Source fortran
0.9580218	Source assembleur	Source fortran
0.9992160	Source assembleur	Source cobol
0.9993102	Source assembleur	Source cobol
0.9942854	Source assembleur	Source algol W
0.9961532	Source assembleur	Source algol W
0.7486664	Source assembleur	données numériques
0.7543102	Source assembleur	données numériques
0.2023085	Source assembleur	module objet
0.1919286	Source assembleur	module objet
0.5120414	Source assembleur	langage naturel
0.7053574	Source assembleur	langage naturel
1.000000	Source fortran	source fortran
0.9500985	Source fortran	source fortran
0.9978196	Source fortran	source cobol
0.9589440	Source fortran	source cobol
0.9911112	Source fortran	source algol W
0.9934862	Source fortran	source algol W
0.7308597	Source fortran	données numériques
0.7366722	Source fortran	données numériques
0.1763195	Source fortran	module objet
0.1653879	Source fortran	module objet
0.4891238	Source fortran	langage naturel
0.6863505	Source fortran	langage naturel
1.000000	Source cobol	Source cobol
0.9997976	Source cobol	source cobol
0.9977328	Source cobol	source algol W
0.9988422	Source cobol	source algol W
0.7743266	Source cobol	données numériques
0.7797131	Source cobol	données numériques
0.2402232	Source cobol	module objet
0.2306338	Source cobol	module objet
0.5456484	Source cobol	langage naturel
0.7328634	Source cobol	langage naturel
1.000000	Source algol W	source algol W
0.9998159	Source algol W	source algol W
0.8151602	Source algol W	données numériques
0.8200867	Source algol W	données numériques
0.3057011	Source algol W	module objet
0.2956035	Source algol W	module objet
0.6068161	Source algol W	langage naturel
0.7770017	Source algol W	langage naturel
1.000000	Données numériques	données numériques
0.9999640	Données numériques	données numériques
0.8006730	Données numériques	module objet
0.7942722	Données numériques	module objet
0.9527719	Données numériques	langage naturel
0.9680171	Données numériques	langage naturel
1.000000	Module objet	module objet
0.9994434	Module objet	module objet
0.9447935	Module objet	langage naturel
0.9368419	Module objet	langage naturel
1.000000	Langage naturel	langage naturel
0.9700432	Langage naturel	langage naturel

CF	10	11	12	13	14	15	16	17	1F	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E
2F	30	31	32	33	34	35	36	37	3E	39	3A	3B	3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E
CC	4F	5C	51	52	53	54	55	56	57	58	C2	59	5A	0D	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	68	69	6A	6B
04	6C	6D	6F	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81	82	83	84	85	0C	86	87	8C	89
EA	8B	8C	8D	8E	8F	9C	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
AA	AB	AC	AD	AE	AF	BC	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
CA	CB	CC	CD	CE	CF	DC	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9
EA	EB	EC	ED	EE	EF	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	01	03	05	06	07	08	09	CA	0B	0C	FA	FB	FC	FD	FE	FF

(1)

36	57	03	64	29	43	22	08	50	15	02	01	49	21	35	28	42	14	63	07	56	46	60	04	18	48	55	06	13	20	27	34	41	62	11	25	32	39	66	72
53	76	33	19	47	40	61	12	26	05	54	00	09	10	16	17	23	24	30	31	37	38	44	45	51	52	58	59	65	66	67	69	70	71	73	74	75	77	78	79

(2)

Code interne (1)

et modèle de nouveau rangement de colonnes d'une image de carte (2)

(pour la classe des données numériques)

par la méthode de changement de base de numération

␣	0
C	110 1
D	100 11
R	10 111
P	110 01
U	111 01
E	1000 10
A	1000 11
.	101 000
T	101 010
I	10 101
L	101 100
O	110 000
N	111 000
2	111 001
S	1111 00
D	11111 0
4	111111
I	1000 001
3	1000 010
5	1001 001
E	100 101 0
é	100 101 1
a	101 001 1
7	10 1110
F	100 111
S	110 001
)	1111 011
(	1000 0000
'	1001 0001
H	1001 0000
M	1003 002 0
.	100 100 1
W	110 0000
*	1111 010
X	1000 0001
=	1001 0000
V	100 100 110
G	100 100 111
É	101 001 001
C	110 0000
Y	110 0001
Z	1111 011
K	100 100 1000
:	1111 010 1
+	101 001 0000
J	1111 010 000
-	1111 010 001
$	101 001 0001 11
?	101 001 0001 001
!	101 001 0001 000
/	101 001 0000 101
>	101 001 0000 100
;	101 001 0000 011
<	101 001 0000 010
_	101 001 0000 000
%	101 001 0000 111
"	101 001 0000 110
(	101 001 0000 101
#	101 001 0000 011
¬	101 001 0000 010

Table des codes de Huffman
pour les textes source

L	1
E	0011
R	00100
A	01000
N	01001
L	01011
O	01101
U	01110
I	01111
4	000011
S	000101
T	000111
e	001000
D	001001
9	001100
C	001101
M	0000000
C	0000001
3	0000011
1	0001000
Y	0001001
F	0001111
5	0010100
2	0010101
P	0011011
E	00000000
F	00000101
G	00010000
8	00010001
7	00011100
V	00011101
¬	000000000
J	000000111
.	00011011
X	00011101
G	00011111
,	000000001
-	000000011
'	000001011
/	000111111
"	000001100
+	000001101
K	0001111110
:	0001111111
?	00111111101
W	001111111110
(	001111111100
)	00111111111
/	001111111111
@	0011111111110
:	0011111111101

Table des codes de Huffman
pour les données de gestion

␣	1
0	C1C
.	C11
1	C011
-	CCCC1
2	C01C0
3	CC1C1
4	CCCCCC
5	CCCCC1
6	CCC1CC
7	CCC1C1
8	CCC11C
9	CCC111C
+	CCC1111C
#	CCC11111

Table de codes de Huffman
pour les données numériques

␣ O D R P U E A , T 1 L C N 2 S O 4 I 3 5 B 6 8 7 F 9) (' H M . W * = X V Q & G Y Z K : + J - $? | / < ; > ¬ % _ # @ "

Dictionnaire de texte source

B = 32

␣ E R A N L O U I 4 S T 6 D 9 C M 0 3 1 Y F 5 2 P B H G 8 7 V ¬ J . X Q , - ' Z " + K : ? W ( ) / % ;

Dictionnaire de langage naturel et de données de gestion

B = 32

␣ 0 . 1 - 2 3 4 5 6 7 8 9 + #

Dictionnaire des données numériques

B = 8

Ces dictionnaires appartiennent à la méthode de combinaisons numériques (méthode de Hahn). (L'espace entre deux groupes de caractères indique la position de la Bième entrée).

La longueur moyenne (LM) a été calculée pour mesurer l'efficacité du code de Huffman. Pour ce faire, une étude statistique sur toutes les données stockées sur les disques IBM 360 a été réalisée pour obtenir des tables de probabilité de caractères à partir desquelles on peut construire le code de Huffman. Ces tables de probabilités sont obtenues soit à partir de fichiers séquentiels (PS), soit à partir de fichiers partitionnés (PO), ou les deux à la fois. Les résultats obtenus sont portés dans le tableau ci-dessous. On remarquera que la longueur moyenne du mot code obtenue à partir des fichiers partitionnés est plus grande que celle obtenue à partir de fichiers séquentiels ; ceci montre qu'il est difficile de compacter des données binaires.

disques manipulés	LM appartenant aux fichiers PS	LM appartenant aux fichiers PO	LM appartenant à PS et PO
IMAG 80	4.93	5.57	5.68
IMAG 81	2.34	2.98	2.95
IMAG 83	3.46	5.50	4.86
IMAG 85	2.33	2.89	2.97
Tous disques	3.30	5.30	3.71

Longueur moyenne du mot-code selon le type de fichier

Nous avons utilisé un dictionnaire unique pour compacter les différentes données (cf. p. 179, 180 et 185). Dans le cas des méthodes de Hahn, Huffman et Shannon-Fano, le gain de place obtenu reste toujours à un taux correct (voir tableaux ci-après), tandis que la méthode de changement de base de numération n'a pas admis ce dictionnaire unique et nécessite donc le pré-calcul du code interne pour chaque donnée à compacter.

Type de donnée	GR %
S. Assembleur	77 %
S. Assembleur	78 %
S. Algol W	78 %
S. Algol W	70 %
S. Cobol	71 %
S. Fortran	72 %
S. PL/1	64 %
Données gestion	41 %
Données gestion	42 %

La méthode de Hahn compacte des textes source et des données de gestion en employant les deux dictionnaires proposés.

Type de données	GR %
S. Assembleur	71 %
S. Assembleur	66 %
S. Assembleur	71 %
S. Assembleur	73 %
S. Algol W	69 %
S. cobol	66 %
Données gestion	52 %
Données gestion	49 %

La méthode de Huffman compacte des textes source et des données de gestion en employant les deux tables de code proposées.

	Code de Huffman	Code Shannon-Fano
␣	0	0
O	1101	1000
D	10011	10010
P	10111	10011
R	11000	10100
U	11101	10101
E	100010	101100
A	100011	101101
,	101000	101110
T	101010	101111
1	101011	110000
L	101100	110001
2	110011	110010
C	111000	110011
N	111001	110100
S	111100	110101
0	111101	110110
4	111111	110111
3	1000001	111000
I	1000010	111001
5	1000011	111010
B	1001010	111011
6	1001011	111100
8	1010010	111101
7	1011010	111110
9	1011011	111111
F	1100100	1111000
)	1111101	11110010
(	10000000	11110011
'	10010000	11110100
H	10010001	11110110
M	10010010	11110111
"	10100111	11111000
W	11001011	11111001
*	11111001	11111010
X	100000010	111110110
=	100000011	111110111
V	100100110	111111000
Q	100100111	111111001
&	101001100	111111010
G	101001101	111111011
Y	110010101	111111100
Z	111110001	111111101
K	1100101001	1111111100
:	1111100000	1111111101
+	11001010000	11111111100
J	11001010001	11111111101
-	11111000010	111111111110
?	111110000011101	1111111111100
$	111110000011110	1111111111101
|	1111100000111000	11111111111100
/	1111100000111001	11111111111101
;	1111100000111110	111111111111110
>	1111100000111111	111111111111111

Les deux codes ci-dessus sont obtenus à partir des mêmes données. Les caractères imprimés sont rangés dans un ordre décroissant selon leur probabilité d'apparition. On remarque que dans le code Shannon-Fano, il arrive qu'un caractère moins probable soit représenté par un code plus court. Par exemple, le caractère "I" est représenté par un code plus court (6 bits) que les caractères qui le précèdent (7 bits) ; il en est de même pour le caractère "'", respectivement 7 et 8 bits. Un tel problème ne se pose pas avec le code Huffman.

BIBLIOGRAPHIE

[1] ANSART J.P.
COMPACT
Note Cyclades, SCH 535.

[2] BEMER R.W.
Do it by the numbers-digital shorthand
Communications ACM, October 1960, pp. 530-536.

[3] BOURNE G. & FORD D.
A study of methods for systematically English words and names
Journal ACM, October 1961.

[4] BRILLOIN C.
La science et la théorie de l'information
Masson, Paris, 1960, pp. 11-35.

[5] CLAVIER J., NIQUIL M., COFFINET G., BEHR F.
Théorie et technique de la transmission des données
Masson & Cie, Paris, 1972, pp. 7-28.

[6] CULLUM R.D.
A method for the removal of redundancy in printed text
GSL

[7] CULMAN G. & KAUFMANN A.
Cours de calcul informationnel appliqué
Albin Michel, Paris, 1970.

[8] DE MAINE & MARRON
Automatic data compression
Communications ACM, 1967, pp. 711-715.

[9] DEWEZE A.
Documentation automatique. Quelques aspects d'indexation de saisie et de composition automatique
Revue internationale du traitement automatique de l'information, Avril 1971, pp. 16-25.

[10] DONIO J.
Conception et équilibrage d'un petit système informatique sur des problèmes de gestion à nature interactive.
Application au projet A.I.D.E.,
Rapport A.N.P.E., Paris, Décembre 1973.

[11] CORGIER H.
Une méthode phonétique de recherche et de mise à jour sur fichiers multiples ou de masse
Travail et Méthodes, Novembre 1970, pp. 39-49.

[12] FAJMAN, BORGELT, WYLBUR
An interactive text editing and remote job entry system
Communications ACM, 1973, pp. 314-322.

[13] HAGAMEN W.
Encoding verbal information as unique numbers
IBM Systems J 11, 4, 1974.

[14] HAHN B.
A new technique for compression and storage data
Communisations ACM, 1974, pp. 434-436.

[15] Sans auteur
IBM 2780 Data transmission, pp. 42-44.

[16] LOUIS-GAVET G.
Compactage de données structurées. Contribution à la conception d'un système d'informations composé de fichiers multiples et volumineux.
Thèse, Lyon, Juin 1974.

[17] Sans auteur
Le Michigan Terminal System.

[18] MULFORD J.B. & RIDALL R.K.
Data compression technique for economic processing of large commercial files,
ACM Symposium on Information Storage and Retrieval, 1971, pp. 207-215.

[19] Sans auteur
Multileaving. The HASP system.
IBM Publication, February 1971, pp. 1139-1153.

[20] OLIVER B.M.
Efficient coding
Bell System Techniques, J.V. 21, n° 4, July 1952, pp. 724-750.

[21] POUZIN L.
Présentation du réseau CYCLADES
Note Cyclades, GAL 506.

[22] RUTH S. & KNEUTZER P.
Data compression for large business files
Datamation, Septembre 1972, pp. 62-66.

[23] SALTON G.
Automation, organisation and retrieval
McGraw Hill, New York, 1968, Chap. I, II, III.

[24] SHANNON C.
A mathematical theory of communication
Bell System techniques, 1948, pp. 379-423, 623-656.

[25] SYNDERMAN M. & HUNT B.
The myriad virtues of text compaction
Datamation, vol. 16, 1970, pp. 36-40.

[26] SPATARU A.
Théorie de la transmission de l'information,
Vol. 2, Masson & Cie, Paris, 1973, pp. 11-52.

[27] THEIL H.
Statistical decomposition analysis.
North Holland Publishing Company, Amsterdam, 1972,
pp. 1-6.

[28] THOMAS J.B.
An introduction to statistical communication theory
John Wiley & Sons, Inc. New York, 1968, pp. 477-532.

[29] WAGNER R.A.
Common phrases and minimum space text storage
Communications ACM, 1973, pp. 148-152.

[30] ZIMMERMANN H.

Host - host protocol

Note Cyclades, SCH 519.1

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU les rapports de présentation de

M.M. J.P. ANSART, Docteur-Ingénieur C.N.R.S. BURES S/YVETTE

G. LOUIS GAVET, Maître Assistant Université de LYON I

Madame ABDEL RAZEK née ZAKARIA EL SHARAWY Salwa

est autorisée à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "GENIE INFORMATIQUE".-

Fait à Grenoble, le 26 Octobre 1976

Le Président,

Ph. TRAYNARD
Président
de l'Institut National Polytechnique