Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Étude des problèmes d'optimisation du volume mémoire occupé par les représentations de tables de valeurs

Pâris, Jehan F.

Award date:
1975

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. Sep. 2024

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX – NAMUR
Institut d'Informatique

ÉTUDE DES PROBLÈMES D'OPTIMISATION DU VOLUME MÉMOIRE OCCUPÉ PAR LES REPRÉSENTATIONS DE TABLES DE VALEURS

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique.

Jehan F. Pâris

Août 1975

Année académique 1974 - 1975

Je tiens tout d'abord à remercier le Professeur F. BODART, directeur de l'Institut d'Informatique des Facultés Universitaires de Namur, pour toute l'aide et les encouragements qu'il n'a cessé de me prodiguer en tant que directeur de mon mémoire.

Je tiens tout spécialement à remercier le Professeur J.C. SIMON de l'Institut de Programmation de l'Université de Paris VI pour tout ce que m'ont apporté ses enseignements, ses conseils, les nombreuses discussions que j'ai pu avoir avec lui, le stimulant constant d'une direction attentive et bienveillante.

Je ne saurais trop remercier Mademoiselle M.C. LOYO pour toute la part qu'elle a prise dans ce travail, tant au stade de l'étude des techniques d'inférence que lors de l'établissement de l'algorithme de construction de représentations arborescentes.

Je tiens à remercier très sincèrement Monsieur G. GUIHO qui a bien voulu superviser notre travail à l'Institut de Programmation et avec qui j'aurai eu de nombreux et fructueux échanges d'idées.

I N T R O D U C T I O N

Rares sont les domaines de l'informatique où les tables n'occupent pas une place importante.

Qu'il nous suffise d'évoquer ici le rôle joué en compilation par les tables des symboles, de citer, pêle-mêle, les tables de décisions, les tables de messages, les tables des contenus des volumes, les tables d'adresses et les pointeurs de toutes sortes sans parler des multiples tables utilisées par les systèmes d'exploitation.

C'est donc dire toute l'importance pratique du problème de la représentation en mémoire des tables, du choix d'une structure de représentation minimisant les temps d'accès aux informations tout en restant économe de l'espace mémoire.

Il n'y a donc pas lieu de s'étonner du nombre de travaux qui ont été consacrés à ce sujet, travaux qui se sont surtout attachés à l'optimisation des temps d'accès en accès aléatoire. Cet intérêt est pleinement justifié, ne serait-ce qu'en raison de l'importance que revêt l'optimisation des temps d'accès dans un cas aussi important que celui des tables des symboles des comilateurs.

Remarquons toutefois que dans d'autres applications, le volume occupé par les tables peut lui aussi devenir un facteur critique. Ce sera le cas pour certaines tables de messages, pour lesquelles des techniques spéciales ont été d'ailleurs mises au point (25) (28), pour les tables contenant un grand nombre d'articles, pour les tables comportant des articles fort longs.

C'est pourquoi, il nous a paru intéressant d'étudier de façon plus approfondie l'optimisation du volume mémoire occupé par la représentation des tables. Il était difficile, sous peine d'être superficiel, de considérer toutes les structures de tables possibles; nous nous sommes donc limités au cas des tables dont les articles sont des entités élémentaires, tables que nous appellerons tables de valeurs. Nous évitions ainsi tous les problèmes qu'aurait posés l'étude de la structure des articles tout en ne perdant rien de ceux liés à la structure tabulaire elle-même.

Notre étude nous aura amené dans de nombreux cas à considérer des structures de représentation inhabituelles ou lourdes à manier pour l'usager. Il nous a semblé qu'il devenait alors intéressant de définir un interface entre les structures de représentation et leurs usagers : nous avons donc voulu introduire le concept de système de gestion de données et à définir des structures de données qui seraient les seules à être connues par l'usager.

A partir de cette notion de structure de données, il nous a été par ailleurs possible de donner une définition formelle des tables de valeurs et d'en déduire une typologie des structures de tables significatives du point de vue du volume occupé. Elle nous servira notamment lors de l'établissement des critères de performances pour les représentations de tables de valeurs.

Nous disposions ainsi de bases solides pour débuter l'étude d'optimisation proprement dite.

Elle comporte tout d'abord une partie théorique originale consacrée à établir les algorithmes permettant de construire les représentations minimales correspondant à des tables de X valeurs choisies arbitrairement parmi N valeurs possibles.

Elle se poursuivra par une étude des techniques d'optimisation locale : Nous y présenterons un certain nombre de techniques de compression de chaînes de caractères qui permettent d'important gains de mémoire sans avoir à modifier l'architecture de la structure de représentation.

Nous serons ensuite amenés à examiner des techniques d'optimisation globale et à discuter la possibilité d'appliquer les techniques d'inférence à la recherche de représentations globales pour les ensembles de valeurs.

Comme illustration de ces techniques, nous proposerons un algorithme original d'inférence de représentations arborescentes. Nous présenterons brièvement une implémentation de l'algorithme ainsi que les structures de représentation des tables et des opérateurs avant de discuter les performances de l'algorithme.

En guise de conclusion, nous confronterons les différentes méthodes d'optimisation de manière à en préciser les domaines d'application.

CHAPITRE UN

LA REPRESENTATION EN MEMOIRE D'ENSEMBLES DE DONNEES.

1.1. L'APPARITION DES SYSTEMES DE GESTION DE BASES DE DONNEES.

Tous les ordinateurs actuels sont susceptibles d'être munis d'organes de mémoire leur permettant d'y enregistrer d'importantes quantités d'informations, segments de code ou données proprement dites.

Tant que les capacités de leurs organes de mémoire étaient modestes, les ordinateurs étaient avant tout des machines à traiter l'information et les informations présentes en mémoire étaient soit des segments de code soit des données ne reprenant le plus souvent que des résultats intermédiaires.

Aujourd'hui, avec la diminution du coût des mémoires et l'amélioration constante de leurs caractéristiques, il devient rentable d'utiliser l'ordinateur comme gestionnaires de banques d'informations, reprenant toutes les informations se rapportant à la gestion d'un système qu'il s'agisse d'une entreprise privée, d'une administration publique ou d'une collectivité régionale.

Ces nouvelles possibilités des ordinateurs n'auront pas été sans influencer notre façon de voir le problème de la représentation en mémoire de données. Tant que les capacités des mémoires à accès direct étaient limitées, c'était l'utilisateur qui définissait lui-même le mode de représentation de ces données. Certes, les systèmes d'exploitation mettaient à la disposition des utilisateurs différents moyens de définir des modes de représentation "logiques" grâce auxquels l'utilisateur se trouvait libéré de contraintes telles que la gestion des entrées-sorties et l'allocation des tampons physiques.

Il n'en demeurait pas moins vrai que la structure logique ainsi définie par l'utilisateur n'était qu'une abstraction idéalisée et stylisée de la structure physique effectivement réalisée par le système.

Au fur et à mesure que les volumes de données à représenter sont allés en croissant, la complexité des relations pouvant exister entre les données à représenter a crû elle aussi. Il est donc devenu de plus en plus impraticable d'exiger des utilisateurs de la base de données la connaissance, de la structure de représentation des données, même sous une forme simplifiée. D'autre part, il importait de permettre aux utilisateurs d'accéder aux informations de la manière la plus simple et la plus naturelle possible telle qu'elle découle des relations sémantiques pouvant exister entre les entités de la base de données.

On en est donc venu petit à petit à définir des structures de données de plus en plus indépendantes des considérations de représentation en mémoire. Cette évolution est aisément perceptible si l'on compare par exemple le prototype qu'était l'INTEGRATED DATA STORE (IDS) de General Electric (20) au langage de définition et de manipulation de bases de données du groupe CODASYL (3). Alors que l'IDS comportait encore de nombreuses références à des structures physiques telles les chaînes, concept directement hérité de l'accès séquentiel chaîné, c'est-à-dire géré par pointeurs, le langage CODASYL qui a été fortement influencé par IDS évite soigneusement de telles homonymies.

A ce niveau, les structures de données définies par les usagers deviennent des entités abstraites découlant des relations sémantiques existant entre les données et non plus de considérations liées à la représentation en mémoire. Apparait alors la nécessité d'un interface spécialisé établissant tout au cours de l'existence de la base de données, la liaison entre la structure de données définies par l'usager et les structures de représentation effectivement présentes en mémoire. Lors de la création de la base de données, il faudra choisir

la structure de représentation la mieux appropriée aux exigences de l'utilisateur telles qu'elles découlent notamment de la structure de données. Il faudra ensuite initialiser la base de données et entrer en mémoire toutes les informations qui la constituent. Enfin, il faudra assurer, pendant toute l'existence de la base de données, les transferts d'informations entre l'usager et sa base de données. Cette tâche est loin d'être la plus simple. L'usager ne connait en effet sa base de données qu'à travers la structure de données qu'il a lui-même définie et c'est en se référant à cette structure qu'il va construire ses procédures de mise à jour et d'interrogation. L'interface devra donc traduire en opérations sur la structure de représentation les procédures de l'utilisateur et retraduire ensuite dans un langage connu de l'utilisateur les réponses fournies à ses interrogations.

La complexité des tâches assurées et leur permanence au cours de la vie de la base de données nous feront dire que l'interface que nous venons de définir est en fait le gestionnaire des informations contenues dans la base de données. L'ensemble interface et organes de mémoire constitue un véritable système : le système de gestion de bases de données.

Pour ses usagers, le système se comporte comme une "boite noire" : l'ensemble des flux d'informations entre le système et les usagers est entièrement défini dans le cadre de la structure de données définie par l'usager. Celui-ci n'aura donc jamais à se soucier des structures de représentation des données effectivement utilisées en mémoire. Nous verrons plus loin toutes les conséquences de ce fait pour les structures de représentation en insistant sur celles qui concernent plus précisement les questions d'optimisation des structures de représentations.

1.2 CONSEQUENCES POUR LES STRUCTURES DE REPRESENTATION.

Tels qu'ils sont apparus les systèmes de gestion de bases de données répondaient à un besoin des utilisateurs : il fallait permettre à ces utilisateurs de manipuler les structures d'informations fort complexes pouvant exister dans les bases de données sous une forme plus naturelle que celles choisies pour leur représentation.

C'est de là qu'est né le concept de la structure de données indépendante de toute considération de gestion de mémoire. Il en est résulté une conséquence pour les structures de représentation : n'étant plus connues de l'utilisateur, elles deviennent une simple question de conventions internes au sein du système.

Les structures de représentation n'ont plus à être le reflet de la vision qu'a l'utilisateur de sa base de données. Chaque fois que des considérations de volume mémoire occupé ou de temps d'accès entrent en ligne de compte le système de gestion pourra choisir une structure de représentation adéquate ; c'est ainsi qu'il pourra faire appel à des techniques de compactage pour diminuer le volume mémoire occupé ou de calcul d'adresse pour réduire le temps d'accès en accès aléatoire. Grâce à l'interface que fournit le système de gestion de la base de données, l'utilisateur ne s'apercevra de rien, sinon d'une amélioration des performances globales du système; ce qui bien sûr n'est pas le cas lorsque l'utilisateur garde la responsabilité de la gestion de la mémoire.

Pour être moins directement sensible, une autre conséquence de cette dissociation entre structure de données et structure de représentation n'en est pas moins importante.

Telle qu'elle est définie par l'utilisateur, la structure de données reprend les caractéristiques principales des entités constitutives de la base de données et explicite notamment les relations sémantiques qui apparaissent significatives à l'utilisateur et qui lui serviront à construire ses procédures de consultation et de mise

à jour. Nous disposons ainsi d'une description formelle de la base de données, description qui constitue un véritable cahier des charges pour la construction de la structure de représentation.

1.3. APPLICATION A L'OPTIMISATION DES REPRESENTATIONS DE TABLES DE VALEURS.

En permettant de définir la structure des données indépendamment de toute considération d'organisation mémoire et en rendant la structure de représentation transparente, les systèmes de gestion de bases de données permettent de mieux délimiter les problèmes que pose le choix de la structure de représentation.

En particulier, l'existence d'une définition formelle de la structure de données à représenter est un apport précieux : même dans le cas de structures aussi simples que celles des tables, le choix d'une représentation suppose une définition extrêmement précise de la structure des données à représenter. Par exemple, on utilise beaucoup de techniques de calcul d'adresse pour réduire le temps d'accès aux éléments d'une table. Ces techniques offrent cependant un inconvénient : elles ne respectent pas l'ordre des éléments au sein de la table. Avant de pouvoir évaluer les performances d'une représentation utilisant le calcul d'adresse, il importe de savoir si l'ordre des éléments au sein de la table est significatif ou arbitraire. Dans la construction de la représentation minimale d'une table de valeurs, le fait que tous les éléments de la tables soient distincts ou qu'on puisse rencontrer plusieurs fois la même valeur, conditionne le choix d'un algorithme plutôt qu'un autre.

On voit donc toute l'importance d'une bonne définition de la structure de données à représenter : lorsqu'on désire une optimisation poussée de la structure de représentation, les structures de données telles qu'elles apparaissent dans les langages de description et de manipulation de données propres aux systèmes de gestion de bases de données, peuvent même devenir insuffisantes (si, par exemple, on désire tenir compte du fait que tous les éléments d'une table de valeurs sont distincts).

Ces considérations nous ont tout naturellement amenés à considérer que la modélisation des problèmes de représentation des données qui avait été adoptée dans les systèmes de gestion de bases de données, pourrait être étendue à d'autres cas et en particulier au problème de la représentation des tables.

Notre point de vue est toutefois différent. Si nous adoptons le principe de la dissociation de la structure de données définies par l'utilisateur et de la structure de représentation effectivement réalisée en mémoire, ce ne sera pas en vue de permettre la manipulation de structures de données plus complexes que celles des fichiers classiques mais pour pouvoir définir des structures de représentation qui soient mieux adaptées et plus efficaces que celles qui sont employées habituellement par les utilisateurs. Ce faisant, nous nous assurerons d'un autre avantage non négligeable : la transparence des structures de représentation aux yeux des usagers, chose d'autant plus importante que la structure de représentation s'écarte des habitudes des usagers, ce qui sera souvent le cas pour les représentations résultant des algorithmes d'optimisation du volume occupé.

Utiliser tel quel le cadre des systèmes de gestion de bases de données et adopter l'un ou l'autre langage de définition et de manipulation de données eut cependant posé quelques problèmes. Nous en avons déjà laisse entrevoir un : le formalisme de ces langages de définition et de manipulation de données est plus adapté à la description de bases de données qu'à la définition précise de structures plus simples en vue d'une optimisation.

Nous avons voulu cependant éviter de traiter le problème de la construction de représentations de longueur minimale pour les tables de valeurs dans un cadre trop étroit, ce qui eut été le cas si nous avions adopté un formalisme uniquement adapté aux tables de valeurs.

C'est pourquoi, il nous a paru intéressant de traiter les tables de valeurs comme une classe de structures de données parmi d'autres et de conserver la même modélisation du problème de la représentation en mémoire que dans les systèmes de gestion de bases de données. Cette démarche nous a conduit au concept de système de gestinn de données qui constituera le cadre dans lequel nous étudierons les structures de données qui correspondent au concept de table de valeurs et nous poursuivrons notre étude des algorithmes conduisant à des représentations de longueur minimale.

1.4. LE CONCEPT DE SYSTEME DE GESTION DE DONNEES.

Nous appellerons système de gestion de données un ensemble de ressources mises en oeuvre en vue d'assurer la gestion d'un ensemble de données pour le compte d'utilisateurs.

Ce qui caractérise donc les systèmes de gestion de données et les différencie des autres systèmes informatiques, c'est la nature du service rendu aux usagers, à savoir la gestion d'un ensemble de données. Avant d'énumérer l'ensemble des tâches que recouvre ce concept, il nous faut d'abord préciser que nous entendons exactement par ensemble de données.

En vue de réaliser un tâche qui lui est particulière, un usager peut avoir besoin de mémoriser un certain nombre d'informations, de données. Il aura tout naturellement tendance à regrouper les informations présentant entre elles une forte connexité. Parmi les entités ainsi constituées, un certain nombre ont la propriété de posséder une structure se prêtant à une description formelle : nous appellerons ces entités ensembles de données.

Ces ensembles de données peuvent être tout aussi bien la table des identificateurs d'un compilateur ou la matrice des coëfficients d'un programme linéaire que l'ensemble des informations décrivant le comportement d'un système. Comme on le voit, les ensemble de données

peuvent être de tailles fort différentes et présenter des structures allant des plus simples aux plus compliquées. La seule restriction qui subsiste est que ces structures doivent se prêter à une description formelle, ce qui nous permettra de les exprimer dans le cadre d'un modèle formel de description de structures.

Ce modèle formel de description de structures fait partie du système de gestion de données. Il fournit à l'usager le cadre dans lequel celui-ci aura d'abord à définir sa structure de données et spécifiera ensuite tous les transferts d'informations qu'il souhaite établir avec le système : initialisation de l'ensemble de données, consultations et mises à jour des informations qu'il contient, suppression lorsque les informations cessent d'être nécessaires à l'usager.

La mission du système sera d'assurer la réalisation de ces transferts d'informations. A l'initialisation de l'ensemble de données, correspondent l'allocation des organes de mémoire et le choix d'une structure de représentation. A chaque mise à jour et à chaque consultation correspondent un certain nombre d'opérations sur la structure de représentation traduisant les actions spécifiées par l'utilisateur dans le cadre de la structure de données telle qu'il l'a définie grâce au modèle formel de description de structures. A la suppression de l'ensemble de données, correspondent des tâches de libération des ressources et de réinitialisation du système. A d'autres moments de la vue de l'ensemble de données, le système devra aussi réorganiser les informations présentes en mémoire : ce sera par exemple le cas lorsque les modifications apportées par une série de mises à jour risquent d'affecter les performances du système.

Cette énumération des tâches à effectuer, pour brève qu'elle est, permet de mieux entrevoir la complexité du processus de gestion d'un ensemble de données.

Pour réaliser sa mission le système dispose, nous l'avons dit, d'un certain nombre de ressources. Ces ressources peuvent être réelles ou virtuelles mais une d'entre elles sera nécessairement un

processeur. Elle devra accueillir la partie du système ayant en charge les intéractions avec les usagers et la gestion des organes de mémoire, nous lui donnerons le nom de gestionnaire. Les autres ressources seront en principe des organes de mémoire et accueilleront les représentations des entités constitutives de l'ensemble de données telles qu'elles ont été définies par le gestionnaire dans la structure de représentation.

Il importe de remarquer ici que cette structure de représentation peut être radicalement différente de la structure de données définie par l'usager. En particulier, on peut très bien envisager de remplacer une liste de valeurs par une représentation conventionnelle d'un algorithme construisant les éléments de la liste. Pour paradoxale que cette idée puisse paraître, elle ne s'intègre pas moins bien dans le formalisme des systèmes de gestion de données : elle implique simplement que le gestionnaire du système soit capable de construire l'algorithme en question et de l'interpréter par la suite lors de chaque consultation de la liste.

Comme on le voit, le concept de système de gestion de données permet d'étendre considérablement la notion de représentation d'un ensemble de données : nous dirons que, pour un système donné, toute représentation interprétable par le système, d'un algorithme lui permettant d'énumérer les éléments d'un ensemble de données tels qu'ils sont définis dans la structure de données de l'ensmble, est une représentation valide et complète de l'ensemble considéré.

Cette définition est quelque peu inhabituelle : nous sommes accoutumés à ne considérer que des représentations où chaque entité définie dans la structure de données a son correspondant dans la structure de représentation. Il n'en demeure pas moins que même dans ce cas, la structure de représentation n'a de sens que si elle est interprétable soit directement par l'usager soit à travers un quelconque système. Ce qui permet donc d'affirmer qu'un ensemble de bits en mé-

moire est bien la représentation d'un ensemble de données, c'est le fait qu'il est possible moyennant un décodage approprié de cette structure, de reconstituer une énumération des éléments de l'ensemble considéré tels qu'ils sont définis dans sa structure de données.

Nous aurons l'occasion d'illustrer ce concept dans le cadre de la recherche de représentations minimales pour les tables de valeur, plus particulièrement lorsque nous nous intéresserons aux techniques d'inférence.

CHAPITRE DEUX

FORMALISATION DU CONCEPT DE TABLE DE VALEURS

2.1. PRESENTATION INTUITIVE.

Tel qu'il est défini dans la littérature (13) (21), le concept de table recouvre à la fois un type de structure de données et une structure de représentation.

On peut dire,cependant, que ce ne sont là que deux aspects d'une seule et même réalité : Une table n'est, en effet, pas autre chose qu'une collection d'articles tous de structure identique, rangés ou non dans un certain ordre mais toujours susceptibles d'être consultés d'une manière aléatoire, c'est-à-dire indépendamment de toute séquence préétablie.

Cette notion d'accès aléatoire est fort importante tant pour l'organisation de la mémoire que pour la structure de données. Elle nécessite le plus souvent que l'organe de mémoire utilisé par la structure de représentation soit lui-même une mémoire à accès direct, c'est-à-dire une mémoire pour laquelle les temps d'accès aux informations ne dépendent pas de leur emplacement mémoire centrale, tambour ou disques magnétiques. C'est ainsi que l'on a pris l'habitude d'appeler "table" tout fichier implanté en mémoire centrale.

Une autre conséquence de la notion d'accès aléatoire est la nécessité de disposer d'un moyen de désigner les articles, tant au niveau de la structure de données qu'au niveau de la structure de représentation.

Tant que l'ordre de consultation des articles correspondait à une séquence préétablie, il était toujours possible de faire correspondre cet ordre à la séquence physique ou logique des articles au sein de la table, éventuellement au prix d'une réorganisation de celle-ci. On accédait ainsi aux différents articles d'une manière purement séquentielle sans qu'il soit jamais besoin de les désigner individuellement.

En accès aléatoire, par contre, nous devons disposer d'un moyen d'identifier l'article auquel nous voulons accéder de manière à permettre au système de gestion de données de le retrouver.

Une première solution serait de repérer chaque article par sa position dans la structure de données en lui affectant un indice, c'est-à-dire une valeur entière définissant de façon unique sa position. Cette méthode offre l'avantage de bien s'adapter à la structure de représentation tabulaire : comme tous les articles ont la même structure, ils seront normalement représentés sur une longueur fixe et il sera facile de calculer l'adresse physique correspondant à chaque valeur de l'indice. Cependant peut-on encore dire que la structure de données que nous venons de définir soit encore à proprement parler une structure tabulaire ? L'introduction de la notion d'indice y apporte quelque chose de nouveau et il serait sans doute plus juste de parler de structure de tableau à un indice ou plus simplement de vecteur (cfr. la nomenclature de l'IFIP (13) et plus spécialement les définitions C45, C46 et C49).

Aussi utilisera-t-on surtout une autre méthode, associative celle-là : on conviendra que chaque article de la table possèdera une rubrique spéciale, appelée clé, qui serviera à identifier l'article en question. Logiquement cette clé fait partie de l'article qu'elle identifie et elle apparaîtra, dans la structure de données de la table, au milieu des autres rubriques. Toutefois, cette clé ne sera pas nécessairement rangée en mémoire à côté des autres rubriques relatives à l'article considéré : ce sera

un exemple des différences pouvant exister à ce niveau entre structure de données et structure de représentation.

La définition du concept de table, telle que nous venons de l'esquisser, ne dit que fort peu de choses sur la structure interne des articles de la table : nous savons seulement qu'ils peuvent compter une ou plusieurs rubriques et que tous les articles d'une même table sont caractérisés par la même structure, mais rien n'est dit sur le nombre et la nature de ces rubriques.

Si nous nous intéressions aux problèmes d'optimisation des temps d'accès aux informations contenues dans la table, cette définition serait cependant suffisante. Certes les procédures de consultation et de mise à jour des tables concernent l'ensemble des rubriques de chaque article mais les processus de sélection des articles ne font jamais appel qu'à des tests sur les clés.

Au contraire, l'étude des problèmes d'optimisation du volume mémoire occupé par la représentation des tables nécessite une connaissance approfondie de la structure des articles, du nombre et du contenu des rubriques, des corrélations possibles entre les valeurs des différentes rubriques car ce sera à partir de ces informations que sera construite la structure de représentation optimale.

Ne désirant pas entrer dans ces considérations, qui nous auraient mené trop loin, nous avons voulu limiter notre étude aux tables dont les articles ne comportent qu'une seule rubrique. Dans ce cas, nous pouvons, en effet, considérer les articles de la table à représenter comme des entités élémentaires choisies sur un ensemble de valeurs possibles ; ce qui vaudra d'ailleurs à ces tables le nom de tables de valeurs.

Une table de valeurs sera donc une collection d'entités élémentaires, appelées valeurs, rangées ou non dans un certain ordre et susceptibles d'être consultées dans un ordre aléatoire.

Comme les articles d'une table de valeurs ne peuvent comporter qu'une seule rubrique, la clé d'accès aux articles en accès aléatoire sera nécessairement la valeur de l'article.

De cette définition intuitive du concept de table de valeurs nous pourrons déduire une définition formelle qui nous permettra d'esquisser une typologie des structures de données propres aux tables de valeurs.

2.2 DEFINITION FORMELLE.

Soit N un univers d'objets de cardinal N, qui constituera l'ensemble des valeurs possibles pour les éléments de la table : nous supposenons toujours cet ensemble fini.

Soit X un ensemble d'occurrence d'éléments de N, ensemble dont nous noterons le cardinal X. Dans cette définition, le même élément de N est donc susceptible de figurer plusieurs fois dans X.

Nous dirons que l'ensemble X est une table de valeurs si et seulement si la seule relation pouvant être définie sur X est une relation d'ordre total (8) (ou éventuellement une relation de préordre total (8) définissant un ordre total sur l'ensemble des valeurs de N figurant dans X).

Cette définition s'applique donc à toutes les tables constituées d'une collection d'entités élémentaires, que ces entités soient distinctes ou non, rangées dans un certain ordre ou non. L'extension de la définition contenue dans la parenthèse permet d'y inclure le cas limite d'une table comprenant des occurrences multiples d'un sous-ensemble de N totalement ordonné : dans ce cas, il n'y aurait qu'un préordre total sur l'ensemble des éléments de la table. A notre connaissance, cette structure de données ne semble correspondre à aucune structure sémantique particulière ; c'est pourquoi nous ne la citons que pour mémoire.

2.3. TYPOLOGIE DES STRUCTURES DE DONNEES POSSIBLES POUR LES TABLES DE VALEURS.

A partir de la définition que nous venons de présenter, il est possible d'établir une typologie des structures de données possibles pour les tables de valeurs.

Nous distinguerons quatre types de structures de données en considérant respectivement les deux critères de classification suivants :

- l'existence d'un ordre sur l'ensemble des éléments de la table,
- la présente ou l'absence d'occurrences multiples d'une même valeur.

Les types de structures de données que nous distinguerons sont :

- les ensembles de valeurs distinctes,
- les ensembles de valeurs quelconques,
- les ensembles ordonnés de valeurs distinctes,
- les ensembles ordonnés de valeurs quelconques.

Les raisons de cette classification sont doubles. Nous verrons plus loin que les quatre types de structures correspondent à quatre contenus informationnels différents et donc à quatre valeurs distinctes du volume minimal théorique (cfr III.2). Nous allons montrer maintenant que les critères de classification adoptés sont également significatifs au niveau des techniques dd manipulation de données qu'elles permettent de définir et donc au niveau de la définition du langage de manipulation des données correspondant à la structure de données considérée.

S'il existe un ordre total sur l'ensemble des éléments de la table, nous pourrons en effet considérer la table comme une liste linéaire ((13) définition G 53) ou liste simple (27) . X,étant fini et totalement ordonné, possèdera un maximum et un

minimum ; tout élément de X distinct du maximum aura un et un seul successeur défini par l'ordre total : tout élement distinct du minimum aura de même un et un seul prédécesseur.

On pourra donc définir sur la structure de données toutes les opérations de traitement des listes linéaires, toutes les opérations de traitement de files ou de piles (cfr. (27) : certaines des opérations reprises par SIMON pour les listes linéaires sont d'ailleurs identiques à celles définies sur les vecteurs). Autre conséquence : à chaque insertion d'un nouvel élément dans la table, il faudra lui assigner un prédécesseur et/ou un successeur, de manière que la table reste totalement ordonnée.

Au contraire, si la table n'est pas ordonnée, le seul mode d'accès possible aux éléments de la table sera le mode d'accès aléatoire via les valeurs de ces éléments. Il sera toujours possible d'obtenir l'énumération des éléments de la table mais l'ordre de présentation des éléments sera non significatif voire imprévisible au niveau de la structure de données.

Si la table ne comporte pas d'occurrences multiples d'une même valeur, il s'agira donc table de valeurs distinctes. A chacune des N clés possibles, correspondra tout au plus un élément de la table. En accès aléatoire, chaque consultation sera donc du type "la valeur x appartient-elle ou non à l'ensemble X". La représentation de la table sera donc aussi une représentation du prédicat d'appartenance à X. La réciproque n'est pas nécessairement vraie: toute représentation du prédicat d'appartenance à l'ensemble X ne sera pas une représentation de la table dans la mesure où elle ne permettra pas nécessairement l'énumération des éléments de X.

Par contre, si la table comporte des occurrences multiples d'une même valeur, il y aura, en accès aléatoire, plusieurs articles de la table qui pourront correspondre à la même valeur de la clé. Dans ce cas, on n'accèdera pas à un élément de la table, mais à un groupe d'éléments.

Remarquons incidemment que, contrairement aux opérations de traitement des listes, les opérations possibles en accès aléatoire seront fort simples et peu nombreuses. Ce seront essentiellement :
- retrouver le ou les éléments correspondant à la valeur x,
- insérer un élément de valeur x,
- supprimer un élément de valeur x.

2.4 REMARQUES.

La typologie que nous venons d'introduire serait incomplète si nous n'y ajoutions pas un troisième critère se superposant aux deux premiers : celui du degré de permanence des informations contenues dans la table.

Il est des tables dont le contenu varie considérablement avec le temps, ces valeurs étant sans cesse ajoutées ou retranchées. D'autres tables auront un contenu pratiquement invariable durant toute leur durée d'existence. Il va sans dire qu'il faudra adopter les structures de représentation en conséquence. Dans le premier cas, la structure de représentation choisie devra permettre de procéder efficacement aux mises à jour, c'est-à-dire sans pertes de temps et sans détérioration des performances ultérieures de la représentation. Dans le second cas, on pourra éventuellement choisir pour la table une représentation conçue spécifiquement pour l'ensemble des valeurs à représenter si cela s'avérait intéressant.

Un autre facteur qui ne sera pas à négliger dans la détermination d'une structure de représentation pour une table de valeurs sera l'ensemble des renseignements que l'usager possède à priori sur les informations contenues dans la table.

Si nous avons à représenter un ensemble ordonné de valeurs distinctes, il n'est pas indifférent de savoir que l'ordre défini sur l'ensemble des valeurs sera nécessairement la séquence ascendante définie sur les chaînes de caractères correspondant aux éléments de la table : dans beaucoup de cas, il pourra être représenté de manière plus efficace qu'un ordre quelconque.

Si nous avons à représenter une table comportant un nombre fixe de valeurs, il importe de savoir si ce nombre est connu de l'usager et donc du système de gestion de données où s'il fait partie des informations reprises dans la structure de données. (On retrouve ici le problème de l'existence d'un contrôle du dépassement des bornes d'un tableau avec les deux solutions de FORTRAN et d'ALGOL.)

D'une manière plus générale, il sera toujours nécessaire de bien spécifier les informations intérieures et extérieures à la structure de représentation.

CHAPITRE TROIS

CRITERES DE PERFORMANCES POUR LES STRUCTURES DE REPRESENTATION DE TABLES DE VALEURS

3.1. ROLE DES CRITERES DE PERFORMANCE.

Avant d'aborder l'étude d'optimisation proprement dite et de discuter les différents algorithmes de réduction du volume occupé, il importe de disposer au préalable de critères de performances nous permettant d'évaluer de façon précise les performances des structures de représentation et de pouvoir comparer les avantages et inconvénients de deux structures différentes.

La première qualité que nous allons exiger de ces critères sera donc l'indépendance vis à vis de l'environnement matériel et logiciel des structures de représentation : dans le cas contraire, toute comparaison entre deux structures de représentation réalisées dans deux systèmes différents serait possible. Nous allons donc ne prendre comme mesures de performances que des mesures indépendantes de l'environnement de la structure considérée. Cette exigence est d'ailleurs réalisée par tous les critères rencontrés dans la littérature, notamment ceux qui ont été recensés par GUIHO (15).

Nous allons cependant y ajouter une autre condition et exiger des critères de performances qu'ils mesurent les performances des structures de représentation en les comparant à des étalons de performances bien définis et universels. Cette exigence ne se trouve pas toujours réalisée, spécialement par les critères relatifs au volume mémoire occupé . Habituellement, on mesure en effet les performances d'un algorithme de réduction du volume mémoire occupé par son pourcentage de gain par rapport

à la structure de représentation initiale. Si ce critère est parlant, il a pour défaut de ne donner du volume occupé par la nouvelle structure de représentation qu'une mesure relative, par rapport à la structure initiale. Le critère du gain de mémoire est parfaitement justifié lorsqu'il s'agit de choisir entre plusieurs algorithmes de réduction s'appliquant à une même structure de représentation. Il n'est plus satisfaisant lorsqu'il s'agit d'estimer les mérites d'une structure de représentation en tant que telle.

Se pose alors le problème du choix des étalons de performance. Le poids de la tradition nous amènerait à choisir comme étalon de performance la performance d'une structure de représentation plus classique que les autres. C'est ainsi, par exemple, dans l'étude des méthodes de code haché:on définit le facteur d'occupation de la mémoire par le rapport entre la mémoire occupée par la méthode choisie et la mémoire occupée par une structure purement séquentielle.

Nous nous sommes cependant tournés vers un autre mode de définition des étalons de performance. Lorsqu'on s'intéresse à l'optimisation bien précise d'une performance particulière, telle le volume occupé ou le temps de consultation en accès aléatoire, on se pose souvent la question de savoir dans quelle mesure la performance d'une structure de représentation donnée est susceptible d'être améliorée. En particulier, si la performance de la structure est fort voisine de la valeur optimale, on sera peut être amené à considérer que les possibilités d'amélioration ultérieure de cette performance ne justifient plus la poursuite de la procédure d'optimisation. Nous pourrions donc choisir comme étalon de performance cette valeur optimale telle qu'on peut la déduire de la structure de données.

Cette définition soulève toutefois une difficulté : dans de nombreux cas pratiques, nous possédons certaines informations sur la structure des informations contenues dans la table. Nous pouvons par exemple savoir que les éléments de la table seront tous des mots de la

langue anglaise ou des chaînes de caractères obéissant toutes à certaines lois de formation ce qui est souvent le cas pour les éléments d'une nomenclature.

Dans ces cas, il est parfaitement possible de concevoir des représentations dont certaines performances seront supérieures aux valeurs considérées jusque là comme optimales. Pour ne prendre qu'un exemple fort simple, si l'ensemble à représenter est l'ensemble de tous les nombre entiers divisibles par 4 et inférieurs à 10.000, il suffit de fort peu de place pour représenter un algorithme permettant d'engendrer tous les éléments de l'ensemble.

Nous nous trouvons donc amenés à choisir, chaque fois que ce sera possible, des étalons de performances tenant compte de toutes les informations que nous possédons à priori sur la structure des informations présentes dans la table. Ce sera notamment le cas lorsque les éléments de la table seront des mots ou des phrases d'un langage pour lequel nous connaissons certaines données statistiques, comme les taux de fréquence des lettres par exemple.

Outre les problèmes de définition d'une mesure de performances, indépendante de l'environnement et le choix d'un étalon de performance, la définition d'un critère de performances pose encore un troisème problème, sans doute plus important pour l'usager que les deux premiers : l'adéquation du critère aux besoins des usagers. Il importe en effet que la performance mesurée par le critère soit significative au niveau du comportement du système de gestion de données, tel qu'il est perçu par ses usagers. Nous aurons l'occasion d'y revenir en examinant les deux grandes classes de critères de performances : ceux relatifs au volume occupé et ceux relatifs aux temps d'accès.

3.2. LES CRITERES RELATIFS AU VOLUME OCCUPE.

Ces critères retiendront tout particulièrement notre attention car ils sont relatifs à l'objet même de notre étude : l'optimisation du volume mémoire occupe la structure de représentation.

Ici, le problème du choix d'une mesure de performances, indépendant de l'environnement système ne se pose pas : nous prendront en effet la quantité de mémoire occupée par la représentation que nous nous proposons d'étudier. Nous l'exprimerons dans une unité indépendante du matériel utilisé, le bit. Cette unité offre le double avantage d'être commune à tous les matériels utilisant la représentation binaire et d'être aussi l'unité de quantité d'information utilisée en théorie de l'information.

Avant d'aborder l'étude des critères de performances proprement dites, remarquons que, dans notre évaluation de la quantité de mémoire utilisée pour représenter une table de valeurs, nous ne tenons pas compte de la quantité de mémoire occupée par le système de gestion de données. Dans la mesure où ce coût peut être partagé entre toutes les tables de valeurs gérées par le système, nous sommes en droit de le négliger. Ce ne serait plus le cas si le système de gestion de données avait été particularisé en vue d'une optimisation plus poussée d'un problème spécifique. Il faudrait alors mettre en balance le gain de mémoire obtenu, grâce à une sophistication de la structure de représentation et le coût de représentation en mémoire de toutes les procédures du système qui auraient été écrites spécialement pour la structure de représentation considérée.

Le critère du pourcentage de gain.

Nous avons déjà dit que le critère de performances vient le plus immédiatement à l'esprit pour caractériser un

algorithme de réduction de la quantité de mémoire occupée par la représentation d'un ensemble de données est le pourcentage de gain par rapport au volume mémoire initialement occupé.

Si V est le volume mémoire initialement occupé par la représentation de l'ensemble de données et si V' est le volume moyen occupé par la nouvelle représentation qui résulte de l'application de l'algorithme, le pourcentage de gain g sera défini par

$$g = \frac{V-V'}{V} . 100\%$$

Nous avons toutefois déjà signalé les deux inconvénients de ce critère. Tout d'abord, il ne mesure pas à proprement parler les performances de la représentation engendrée par l'algorihtme de réduction mais celles de l'algorithme de réduction proprement dit quand il est appliqué à une structure de représentation donnée. Ensuite il ne permet pas de mesurer l'écart entre le volume occupé par la représentation auquel il s'applique et le volume minimal théoriquement nécessaire pour représenter l'ensemble de données considéré. Cet écart est en effet la meilleure mesure de l'efficacité avec laquelle la mémoire est gérée et donc le meilleur critère de performances pour le volume mémoire occupé par une structure de représentation.

Le critère du taux d'occupation de la mémoire.

Pour répondre à ces critiques, nous allons introduire un nouveau critère de performances, basé lui sur la mesure de l'efficacité de la gestion de l'espace mémoire. Ce sera le taux d'occupation de la mémoire.

Soit V le volume moyen occupé par la représentation d'une table de données.

Si V_M désigne le volume minimal moyen théoriquement nécessaire pour représenter la table de données, compte tenu de tous les renseignements que nous possédons sur la structure de la table de valeurs et les informations qu'elle contient, le taux d'occupation de la mémoire α vaudra par définition.

$$\alpha = \frac{V_M}{V}$$

On remarquera que l'on a nécessairement $\alpha \leq 1$.

La grandeur que nous venons de définir exprime le rendement de la représentation considérée en ce qui concerne la gestion de l'espace mémoire : elle mesure en effet le rapport de la quantité de mémoire effectivement utile à la quantité de mémoire réellement dépensée. La seule difficulté que pose le critère réside dans l'évaluation de la quantité V_M.

Rappelons en effet un résultat bien connu de la théorie de l'information (26). La quantité minimale de mémoire H nécessaire pour représenter un état au sein d'un ensemble de n états possibles dépend des probabilités d'occurrence Pi de chacun des n états par la relation

$$H = -\sum_{1}^{n} p_i \, lb \, p_i$$

lb Pi désignant le logarithme en base 2 de Pi.

En particulier, pour le cas qui nous intéresse, le calcul du volume mémoire théoriquement nécessaire pour représenter une table de valeurs suppose que nous connaissions non seulement l'ensemble de toutes les tables compatibles avec ce que nous savons sur la structure de la table et sur les informations qu'elle contient mais aussi les probabilités d'occurrence de chacune de ces tables.

Dans la plupart des cas pratiques, cette démarche s'avère malheureusement impraticable soit que nous ne possédions pas toutes les données numériques nécessaires soit que les calculs deviennent inextricables. Nous serons alors amenés à ne pas exploiter la totalité des renseignements que nous possédons sur le contenu de la table, de manière à nous ramener à des cas plus simples pour lesquels la détermination du volume minimal n'offrira pas de difficultés.

Nous examinerons plus en détail deux de ces cas. Le premier sera celui des tables comportant X valeurs choisies arbitrairement parmi N et le second celui des tables dont les éléments sont des textes d'un langage.

Il va sans dire que toutes les fois que nous nous ramènerons à un de ces cas en négligeant une partie des renseignements que nous possédons sur le contenu de la table, nous obtiendrons une estimation du volume minimal qui sera nécessairement supérieur à la valeur exacte. Il en résultera un taux d'occupation de la mémoire n'exprimant plus exactement l'efficacité de la gestion de l'espace mémoire et pouvant même, dans certains cas limites, devenir supérieur à l'unité.

Volume minimal pour des tables de X valeurs choisies parmi N.

Dans le cas qui nous intéresse,nous savons à priori que la table de valeurs à représenter est de longueur fixe et que ses X éléments sont choisis arbitrairement parmi des N valeurs possibles. Nous ferons de plus l'hypothèse que chacun des éléments de l'ensemble des N valeurs possibles a la même possibilité de figurer dans la table. Cette hypothèse se justifie pleinement si l'on considère que la quantité de mémoire nécessaire pour représenter un état parmi n états possibles passe par un maximum lorsque tous les états sont équiprobables.

Compte tenu de ces hypothèses, on peut calculer aisément le nombre de bits nécessaires pour représenter une quelconque des N valeurs possibles : il faut un minimum de lb N bits. On en a conclu un peu rapidement qu'il fallait un minimum de X lb N bits pour représenter une table de X valeurs choisies arbitrairement parmi N.

Le mérite revient à SIMON (27) d'avoir le premier remarqué que cette formule qui correspondait à une structure de représentation séquentielle ne pouvait s'appliquer aux tables non triées pour lesquelles l'ordre des éléments n'est pas significatif. En se basant sur l'idée qu'il fallait des X lb N bits l'information qu'apportait l'ordre de rangement des éléments, il a été amené à proposer la valeur $X \text{ lb} \frac{N}{X}$ pour le volume minimal théoriquement nécessaire pour représenter un ensemble de X valeurs choisies parmi N.

Nous avons pu personnellement (23) apporter une justification théorique à cette formule et préciser son domaine de validité. Ce faisant nous avons pu établir le volume minimal moyen théoriquement nécessaire pour représenter un ensemble de X valeurs parmi N.

La même méthode nous permettra d'établir ici les volumes minimaux moyens correspondant à chacune des quatre structures recensées dans notre typologie des structures de données propres aux tables de valeurs.

Considérons d'abord le cas d'un ensemble de X valeurs distinctes choisies parmi N. L'ensemble des états à considérer est l'ensemble de tous les ensembles qu'on peut former en choisissant X valeurs distinctes parmi N, c'est à dire l'ensemble des combinaisons de N objets pris X à X. Comme toutes les valeurs possibles ont la même probabilité d'être choisies, tous ces états sont équiprobables et le volume minimal moyen V_M nécessaire pour représenter l'ensemble des X valeurs distinctes choisies parmi N valeurs possibles vaudra

$$V_M = \text{Lb}\, C_N^X = \text{Lb}\, \frac{N!}{(N-X)!\,X!} \text{ bits.}$$

S'il s'agit d'un ensemble de X valeurs quelconques, l'ensemble des états à considérer sera celui des combinaisons avec répétitions de N objets pris X à X. Tous ces états étant eux aussi équiprobables, le volume minimal moyen V_M nécessaire pour représenter l'ensemble des X valeurs choisies parmi N valeurs possibles vaudra

$$V_M = \text{Lb}\, {}^R C_N^X = \text{Lb}\, C_{N+X-1}^X = \text{Lb}\, \frac{(N+X-1)!}{(N-1)!\,X!} \text{ bits.}$$

Pour les ensembles ordonnés de X valeurs distinctes, l'ensemble des états à considérer sera celui des arrangements de N objets pris X à X. Le volume minimal moyen V_M nécessaire à la représentation d'un ensemble ordonné de N valeurs distinctes choisies parmi N valeurs possibles vaudra donc

$$V_M = \text{Lb}\, A_N^X = \text{Lb}\, \frac{N!}{(N-X)!} \text{ bits.}$$

Enfin, dans le cas d'un ensemble ordonné de X valeurs quelconques, l'ensemble des états à considérer sera celui des arrangements avec répétitions de N objets pris X à X et le volume minimal moyen V_M nécessaire à la représentation d'un ensemble ordonné de X valeurs quelconques choisies parmi N valeurs possibles vaudra

$$V_M = lb\ {}^{R}A_N^X = X\ lb\ N \text{ bits.}$$

Nous retrouvons, dans ce dernier cas, la valeur de X lb N correspondant à une structure de représentation de longueur fixe, chacune des X valeurs de la table étant représentée par lb N bits. Dans les autres cas, nous obtenons des expressions peu maniables. Elles peuvent toutefois se ramener à des formes plus simples chaque fois qu'il nous sera possible d'utiliser l'approximation de STIRLING ($\ln a! \simeq a \ln a - a$ pour a grand devant l'unité).

En particulier, nous avons

- pour le volume minimal moyen d'un ensemble de valeurs distinctes :

$$V_M \simeq X\ lb \frac{N}{X} + (N-X)\ lb \frac{N}{N-X} \text{ bits}$$

pour autant que N, X et N-X soient grands devant l'unité,

- pour le volume minimal moyen d'un ensemble de valeurs quelconques

$$V_M \simeq X\ lb \frac{N+X-1}{X} + (N-1)\ lb \frac{N+X-1}{N-1} \text{ bits}$$
$$\simeq X\ lb \frac{N+X}{X} + N\ lb \frac{N+X}{X} \text{ bits}.$$

pour autant que (N+X-1), (N-1) et X soient grands devant l'unité,

- pour le volume minimal moyen d'un ensemble ordonné de valeurs distinctes

$$V_M \simeq X\ lb\ (N-X) + N\ lb \frac{N}{N-X} - X \text{ bits}$$
$$\simeq X\ lb\ (N-X) + N\ lb \frac{N}{N-X} \text{ bits}.$$

pour autant que N et (N-X) soient grands devant l'unité.

On remarquera que ces formules sont susceptibles de simplifications, dont certaines sont immédiates. En particulier, il est possible de retrouver la formule proposée par SIMON.

Volume minimal pour des ensembles de textes d'un langage.

Nous nous intéresserons ici au cas des tables dont les éléments sont des textes, à la limite des mots, d'un langage connu : c'est le cas, par exemple des tables de messages.

Nous supposerons que les seules informations que nous possédons sur ces textes concernent le langage dont elles font partie, ce qui implique notamment que les probabilités d'occurrence des éléments de la table soient mutuellement indépendants.

Tout naturellement, ces textes seront représentés en mémoire par des chaînes de caractères. En l'absence de toute optimisation, nous utiliserons le jeu standard de caractère de la machine sur laquelle nous travaillons, ce qui implique selon les cas six ou huit bits de mémoire pour représenter chaque caractère.

En réalité, cette représentation est manifestement redondante : certains caractères du jeu n'apparaîtront jamais, les autres apparaitront avec des probabilités différentes dépendant du contexte.

La représentation en mémoire de textes anglais exige un alphabet minimum de 27 caractères, à savoir les 26 lettres et un caractère blanc. Si tous ces caractères avaient la même probabilité d'occurrence, il faudrait donc 4,75 bits pour représenter chaque caractère.

SHANNON a montré que l'on pouvait réduire cette valeur d'environ 50 % en tenant compte de l'information mutuelle existant entre lettres voisines dans des groupes allant jusqu'à huit lettres(12)

En pratique, on ne s'intéressera pas à la possibilité de mettre en évidence l'information mutuelle existant entre les lettres voisines et on se contentera d'une optimisation locale de la représentation de chaque caractère.

Compte tenu de ces hypothèses, nous considérerons comme optimal le code de longueur minimale que nous pourrons construire des probabilités d'occurrence P1, ..., Pn des n caractères de l'alphabet utilisé pour représenter l'alphabet considéré : ce code sera obtenu par l'algorithme de HUFFMAN (19). Dans le cadre d'une représentation binaire, sa longueur moyenne sera égale, à une fraction de bit près, au volume minimal théorique. Le volume minimal moyen nécessaire pour représenter un ensemble de textes comportent m caractères vaudra donc

$$-m \sum_{1}^{n} P_i \, lb \, P_i \text{ bits.}$$

En pratique, on n'utilisera que fort peu ce volume minimal moyen et on calculera directement le taux d'occupation de la mémoire en comparant le nombre moyen de bits, nécessaires pour coder un caractère dans la représentation dont on évaluera les performances avec la longueur moyenne du code obtenu par l'algorithme de HUFFMAN.

3.3. LES CRITERES RELATIFS AUX TEMPS D'ACCES.

Quoique notre étude concerne l'optimisation du volume mémoire, il est cependant nécessaire de nous étendre quelque peu sur les critères de performances des structures de représentation relatifs aux temps d'accès.

Dans de nombreuses applications pratiques, les temps d'accès aux informations constituent en effet un facteur critique. C'est le cas, par exemple, pour la table des symboles d'un assembleur. Aussi, lorsque nous nous trouverons face à un problème pra-

tique d'optimisation du volume mémoire, il nous sera toujours nécessaire de tenir compte de toutes les contraintes de temps d'accès à l'application considérée.

Les grandeurs mesurées :

Il n'y a pas un critère unique de temps d'accès car il n'y a pas une grandeur unique à mesurer. L'utilisateur dispose en effet de plus d'un mode d'accès aux données lui permettant de les consulter et de les modifier ; selon les applications, ce sera l'un ou l'autre de modes d'accès qui sera privilégié. Nous aurons donc autant de critères de temps d'accès que nous devrons définir d'opérations élémentaires dans notre langage de manipulation de données. Parfois même, nous serons amenés à considérer deux temps d'accès pour une même opération celui correspondant à une réponse fructueuse et celui correspondant à un abandon suite à une tentative infructueuse.

Toutefois, la grande majorité des applications, pour lesquelles les contraintes de temps d'accès jouent un rôle important, privilégient le mode d'accès aléatoire. Dans ces cas, nous pourrons nous limiter à ne considérer que les temps d'accès correspondants. Avec GUIHO (15), nous distinguerons

- le temps moyen de recherche en cas de recherche fructueuse
- le temps moyen de recherche en cas de recherche infructueuse
- le temps moyen d'insertion d'une valeur
- le temps moyen nécessaire à l'enlèvement d'une valeur.

Le problème du choix de l'unité de mesure.

Le problème du choix de l'unité de mesure exprimant le mieux les performances d'une structure de représentation relativement aux temps d'accès aux informations, n'a pas reçu jusqu'ici de solution complètement satisfaisante.

La seule mesure de temps d'accès qui ait une signification concrète aux yeux des utilisateurs, c'est la mesure du temps consacré par le système de gestion de données à l'exécution de l'accès considéré. Si le système d'exploitation de la machine hôte est un système de monoprogrammation, cette quantité peut être aussi définie comme l'intervalle de temps s'écoulant entre le début de la prise en charge par le système de gestion de données de la requête spécifiant l'accès considéré et son acquittement.

Malheureusement, cette mesure offre le gros inconvénient de dépendre dans une très grande mesure de l'environnement matériel et logiciel de la représentation dont nous nous proposons de mesurer les performances : le temps consacré par un système de gestion de données à l'exécution d'un accès dépend à la fois de la complexité des opérations à effectuer pour réaliser l'accès et des vitesses d'exécution de ces actions par le matériel supportant le système de gestion de données.

L'évaluation des performances d'une structure de représentation d'une manière indépendante de son environnement, nécessiterait donc de disposer d'une mesure de complexité des opérations à effectuer sur cette structure pour réaliser l'accès considéré. La définition de cette mesure soulèverait toutefois de nombreux problèmes : notamment ceux de la définition des opérations élémentaires à considérer et de la détermination des coëfficients de pondération à leur affecter. Il faudrait de plus s'assurer que la mesure de complexité que nous aurions définie, ne privilégie aucun matériel.

Aussi, procéderons-nous d'une autre manière. On peut en effet décomposer le temps consacré par le système de gestion de données à l'exécution d'un accès en deux parties : l'une nécessaire à la consultation de la représentation par le système, l'autre nécessaire aux opérations de préparation de ces consultations. Dans de nombreux cas, le temps pris par ces opérations sera négligeable en première approximation devant le temps de consultation de la représentation.

Ce sera notamment le cas chaque fois que l'organe de mémoire affecté à la représentation sera une mémoire secondaire ou une mémoire de masse.

Nous disposons alors d'une méthode simple de définir une mesure des temps d'accès, indépendante de l'environnement logiciel et matériel de la représentation, ce sera le nombre d'accès à l'organe de mémoire contenant la représentation.

C'est cette mesure que l'on retrouve à la fois dans les critères énumérés par GUIHO (15) et dans le reste de la littérature. Il importe toutefois de garder toujours à l'esprit le fait que le nombre d'accès à la mémoire que nous venons de définir n'est qu'une mesure approximative des performances de la représentation considérée, mesure qui peut ne plus être adéquate si l'hypothèse de départ n'était plus vérifiée.

Le problème du choix des étalons de performances :

Nous prendrons comme étalons de performances pour nos critères de temps d'accès les plus petits nombres d'accès mémoire effectivement réalisables avec la meilleure structure de représentation pour l'accès considéré.
Chaque fois qu'il s'agira d'un accès séquentiel (sauf d'une insertion) ou d'une énumération des éléments de la table, la meilleure structure de représentation sera la structure séquentielle pure et simple.
Chaque fois qu'il s'agira d'un accès aléatoire, la meilleure structure de représentation sera celle d'une rangée de N bits, la valeur du R eme bit de la rangée indiquant si le R eme élément de l'ensemble des N valeurs possibles est ou n'est pas présent dans la table.
On remarquera qu'avec cette structure de représentation, toutes ces opérations de recherche, d'insertion ou d'enlèvement d'une des N valeurs possibles, ne requièrent jamais qu'un seul accès possible.

Pour cette raison nous définirons les coëfficients de recherche, d'insertion, ou d'enlèvement d'une représentation en accès aléatoire comme le nombre moyen d'accès à la mémoire, nécessaires pour réaliser chacune de ces opérations. Nous ne ferons d'ailleurs

que reprendre les définitions classiques de la littérature, en leur donnant toutefois une justification formelle.

CHAPITRE QUATRE

REPRESENTATIONS MINIMALES DE TABLES DE X VALEURS CHOISIES ARBITRAIREMENT PARMI N VALEURS POSSIBLES.

4.1. CONSIDERATIONS GENERALES.

Dans le chapitre précédent, nous avons calculé les volumes minimaux moyens nécessaires à la représentation d'une table de X valeurs choisies arbitrairement parmi N valeurs possibles, dans les quatre cas correspondant aux quatre structures de données que nous avions recensées dans notre typologie.

La méthode que nous avions utilisée pour calculer ces volumes minimaux était basée sur l'énumération de toutes les configurations possibles pour chacune des quatre structures de données. Elle définissait donc implicitement un algorithme de construction des représentations de volume minimal : étant donné une énumération quelconque de toutes les configurations possibles pour une structure de données, il est possible de représenter n'importe laquelle de ces configurations et donc n'importe quelle table de valeurs compatible avec le structure de la table par son indice dans l'énumération des configurations.

Comme nous avions supposé que toutes les valeurs de l'ensemble des valeurs possibles avaient la même probabilité de figurer dans la table, toutes les configurations compatibles avec la structure de la table seront équiprobables et nous obtiendrons des représentations de longueur fixe.

Malheureusement, il n'existe pas d'algorithme permettant de passer directement de l'indice de la table aux valeurs de ses éléments et vice-versa, ce qui enlève tout intérêt pratique à la

méthode. Elle nécessiterait en effet que le système de gestion garde en mémoire l'énumération complète de toutes les configurations possibles, à moins qu'il ne la reconstitue lors de chaque accès.

Nous avons donc été amenés à rechercher d'autres structures de représentation se caractérisant par un taux d'occupation de la mémoire aussi voisin que possible de l'unité mais aussi une plus grande facilité de mise en oeuvre.

En particulier, nous exigerons de ces structures de représentation qu'elles ne nécessitent du système de gestion de données que la connaissance de l'ensemble des valeurs possibles et qu'elles se prêtent à des procédures de codage et de décodage s'exécutant séquentiellement en ne mettant en oeuvre qu'un nombre limité de bits à chaque étape.

La première de ces conditions porte sur le nombre d'informations que le système aura à garder en mémoire : il est évident qu'il devra au moins connaître l'ensemble de toutes les valeurs possibles. Dans les cas pratiques, on s'arrangera pour que la description de cet ensemble puisse se faire de manière concise (par exemple : toutes les chaînes de moins de m caractères engendrées par l'alphabet A,tous les nombres entiers entre -2000 et +2000). La seconde condition nous assure qu'il sera effectivement possible d'écrire des procédures de codage et de décodage compatibles avec les possibilités arithmétiques des matériels actuels.

Compte tenu de ces deux conditions, nous nous sommes orientés vers des algorithmes encodant successivement les X valeurs de la table. Afin d'aboutir à des représentations de volume moyen minimal, nous utiliserons à chaque étape du processus toutes les informations que nous pourrons retirer de la connaissance des valeurs déjà encodées. Ce sont ces algorithmes que nous présenterons ici, les deux premiers faisant l'objet par ailleurs d'une publication (24).

4.2. NOTATIONS ET HYPOTHESES DE DEPART.

Nous appellerons

N l'ensemble des N valeurs possibles pour les éléments de la table

X l'ensemble, muni éventuellement d'un ordre total, des X éléments de la table.

Si les X éléments de la table correspondent à des valeurs distinctes, X est nécessairement un sous-ensemble de N.

Nous supposerons qu'il sera toujours possible de définir un ordonnancement de N et d'affecter à chaque élément de N un indice i de 1 à N correspondant à son rang au sein de cet ordonnancement.

Nous maintiendrons l'hypothèse que tous les éléments de l'ensemble des N valeurs possibles ont la même probabilité de figurer dans la table : comme dans le passé, il en résultera que toutes les configurations de la table compatibles avec sa structure de données, auront la même probabilité d'occurrence.

4.3. APPLICATION AU CAS D'UN ENSEMBLE DE VALEURS DISTINCTES.

Puisqu'il s'agit d'un ensemble non muni d'un ordre total il nous sera possible de ranger les X éléments de la table dans l'ordre croissant de leurs indices

$$V_{\alpha_1}, V_{\alpha_2}, \ldots, V_{\alpha_i}, \ldots, V_{\alpha_N}.$$

Puisque ces éléments correspondent à des valeurs distinctes on a nécessairement

$$1 \leq \alpha_1 \leq N-X+1,$$

$$\ldots$$

$$\alpha_{i-1}+1 \leq \alpha_i \leq N-X+i,$$

$$\ldots$$

$$\alpha_{X-1}+1 \leq \alpha_X \leq N.$$

Pour représenter X, il nous suffira donc de représenter les X quantités $\alpha'_1 = \alpha_1$,

$$\alpha'_2 = \alpha_2 - \alpha_1 ,$$

$$\cdots$$

$$\alpha'_i = \alpha_i - \alpha_{i-1} ,$$

$$\cdots$$

$$\alpha'_X = \alpha_X - \alpha_{X-1} .$$

qui ne sont pas autre chose que les différences entre deux indices consécutifs.

Afin d'obtenir une représentation de longueur minimale, nous utiliserons des codes optimaux pour les X valeurs $\alpha'_1, \cdots, \alpha'_X$. A cette fin, nous allons considérer ces valeurs et les X α_i comme des variables aléatoires notées respectivement $\underline{\alpha}'_1, \underline{\alpha}'_2, \cdots, \underline{\alpha}'_X$ et $\underline{\alpha}_1, \underline{\alpha}_2, \cdots, \underline{\alpha}_X$ et définies sur l'ensemble de toutes les configurations possibles pour un ensemble de X valeurs choisies parmi N. Nous calculerons ensuite pour toutes les valeurs possibles de i, j, ..., m, n, ..., v, w les probabilités

$$Pr\left[\underline{\alpha}'_1 = i\right]_{N,X} \qquad (1 \leq i \leq N-X+1)$$

$$Pr\left[\underline{\alpha}'_2 = j \mid \underline{\alpha}_1 = i\right]_{N,X} \qquad (1 \leq i \leq N-X+1 \text{ et } 1 \leq j \leq N-i-X+2)$$

$$\cdots$$

$$Pr\left[\underline{\alpha}'_i = n \mid \underline{\alpha}_{i-1} = m\right]_{N,X} \qquad (i-1 \leq m \leq N-X+i-1 \text{ et } 1 \leq n \leq N-m-X+i)$$

$$\cdots$$

$$Pr\left[\underline{\alpha}'_X = w \mid \underline{\alpha}_{X-1} = v\right]_{N,X} \qquad (X-1 \leq v \leq N-1 \text{ et } 1 \leq w \leq N-v)$$

$Pr\left[\underline{\alpha}'_1 = i\right]_{N,X}$ étant la probabilité que le premier élément de X dans l'ordre des indices croissants soit l'indice i,

$Pr\left[\underline{\alpha}'_2 = j \mid \underline{\alpha}_1 = i\right]_{N,X}$ étant la probabilité que le second élément de X dans l'ordre des indices croissants soit d'indice $(i+j)$ si le premier est d'indice i,

$Pr\left[\underline{\alpha}'_i = n \mid \underline{\alpha}_{i-1} = m\right]_{N,X}$ étant la probabilité que le i ème élément de X dans l'ordre des indices croissants soit d'indice $(m+n)$ alors que le $(i-1)$ élément est d'indice m,

$Pr\left[\underline{\alpha}'_X = w \mid \underline{\alpha}_{X-1} = v\right]_{N,X}$ étant la probabilité que le dernier élément de X dans l'ordre des indices croissants soit d'indice $(v+w)$ alors que l'avant-dernier est d'indice v.

A partir de ces probabilités, il nous sera possible de construite des représentations optimales des X valeurs présentes dans la table en utilisant l'algorithme de HUFFMAN (19).

Détermination des probabilités $Pr\left[\underline{\alpha}'_1 = i\right]_{N,X}$

Nous savons qu'il y a C_N^X configurations possibles pour un ensemble de X valeurs choisies parmi N. Comme toutes ces configurations sont équiprobables, chacune d'entre elles aura la probabilité $\frac{1}{C_N^X}$.

Si le premier élément de l'ensemble dans l'ordre croissant des indices est d'indice i, les indices des X-1 autres éléments de l'ensemble seront nécessairement compris entre i + 1 et N, ce qui laisse N-i valeurs possibles.

Lorsque nous fixons l'indice du premier élément à la valeur i, nous réduisons le nombre de configurations possibles de C_N^X à C_{N-i}^{X-1}.

Comme toutes ces configurations sont équiprobables, nous avons pour tout i compris entre 1 et $N - X + 1$

$$Pr\left[\underline{\alpha}'_1 = i\right]_{N,X} = \frac{C_{N-i}^{X-1}}{C_N^X}$$

que nous pouvons aussi écrire

$$Pr\left[\underline{\alpha}'_1 = i\right]_{N,X} = \frac{X\,(N-X)!\,(N-i)!}{(N-X-i+1)!\;\;N!}$$

$$= \frac{X\,(N-X)\ldots(N-X-i+2)}{N\,(N-1)\ldots(N-i-1)}$$

Remarquons que nous avons

$$Pr\left[\underline{\alpha}'_1 = 1\right]_{N,X} = \frac{X}{N}$$

et

$$Pr\left[\underline{\alpha}'_1 = i+1\right]_{N,X} = \frac{N-X-i+1}{N-i} Pr\left[\underline{\alpha}'_1 = i\right]_{N,X} .$$

$Pr\left[\underline{\alpha}'_1 = i\right]_{N,X}$ est donc strictement décroissante par rapport à i pour toutes les valeurs de X supérieures ou égales à 2.

Détermination des probabilités $Pr\left[\underline{\alpha}'_i = n \mid \underline{\alpha}_i = m\right]_{N,X}$

Si le (i - 1) eme élément de l'ensemble des valeurs à représenter rangées dans l'ordre de leurs indices croissants est d'indice m, il reste N-m valeurs possibles pour les X-i+1 éléments restants.

La probabilité que le i eme élément de la table soit d'indice m+n alors que le i-1 eme élément est d'indice m est donc la même que la probabilité que le premier élément d'un ensemble de X-i+1 valeurs distinctes choisies parmi N-m valeurs possibles soit d'indice n :

$$Pr\left[\underline{\alpha}'_i = n \mid \underline{\alpha}'_{i-1} = m\right]_{N,X} = Pr\left[\underline{\alpha}_1 = n\right]_{N-m,X-i+1} .$$

Algorithme

(i) Ranger les éléments de la table dans leur ordre d'apparition dans une énumération conventionnelle de l'ensemble des X valeurs possibles.

Soient $\alpha_1, \alpha_2, \ldots \alpha_X$ les valeurs des indices correspondant dans cette énumération aux éléments de la table, calculer les X différences

$$\alpha'_1 = \alpha_1, \quad \alpha'_2 = \alpha_2 - \alpha_1, \quad \ldots, \quad \alpha'_i = \alpha_i - \alpha_{i-1}, \quad \ldots, \quad \alpha'_X = \alpha_X - \alpha_{X-1} .$$

(ii) Calculer les probabilités $Pr\left[\underline{\alpha}'_1 = i\right]_{N,X}$ pour toutes les valeurs de i comprises entre 1 et N-X+1 et construire le code préfixe optimal pour la première différence en appliquant l'algorithme de HUFFMAN.

Ecrire la représentation correspondant à la valeur α'_1.

(iii) Faire $q := 2$, $\alpha := \alpha'_1$, $\nu := N - \alpha'_1$, $\xi := X - 1$.

(iv) Calculer les probabilités

$$Pr\left[\underline{\alpha}'_q = j \mid \underline{\alpha}_{q-1} = \alpha\right]_{N,X} = Pr\left[\underline{\alpha}'_1 = j\right]_{\nu,\xi}$$

pour toutes les valeurs de $\underline{\alpha}'_q$ comprises entre 1 et $\nu - \xi + 1$ et construire le code préfixe optimal pour la q ème différence en appliquant l'algorithme de HUFFMAN.

Ecrire la représentation correspondant à la valeur α'_q à la suite des symboles déjà écrits.

(v) Si $q = X$ l'algorithme est terminé et la suite des symboles écrits est la représentation de l'ensemble des valeurs.

Si $q < X$

faire $q := q + 1$, $\alpha := \alpha + \alpha'_{q-1}$, $\nu := \nu - \alpha'_{q-1}$, $\xi := \xi - 1$.

et aller en (iv)

Efficacité de l'algorithme

Les différences que nous pourrions observer entre la longueur des représentations engendrées par l'algorithme et le volume minimal moyen théorique sont dues au fait que nous encodons séparément les X quantités $\alpha'_1, \alpha'_2, \ldots, \alpha'_X$.

Il y a en effet un effet cumulatif des légères différences que nous observons, pour chaque α'_i, entre le volume minimum théorique et la longueur moyenne du code optimal préfixe engendré par l'algorithme de HUFFMAN : rappelons que (25) l'algorithme d'HUFFMAN engendre des codes avec une longueur moyenne toujours comprise entre la longueur théorique minimum et cette longueur plus lb D bits, D étant le nombre de symboles de l'alphabet utilisé (ici D = 2).

Si nous négligeons ces différences,

le nombre de bits nécessaires pour représenter la valeur $\alpha'_1 = i$ serait (12) (26)

$$-lb\ Pr\left[\underline{\alpha}'_1 = i\right]_{N,X} = lb\ C_N^X - lb\ C_{N-i}^{X-1} \text{ bits}$$

et celui nécessaire pour représenter la valeur $\alpha_i = n$ sachant que $\alpha_{i-1} = m$ serait

$$-lb\ Pr\left[\underline{\alpha}'_i = n \mid \underline{\alpha}_{i-1} = m\right]_{N,X} = lb\ C_{N-m}^{X-i+1} - lb\ C_{N-m-n}^{X-i} \text{ bits.}$$

Il devient alors possible de calculer le volume occupé par la représentation d'un ensemble de X valeurs distinctes choisies parmi N valeurs possibles.

$lb\ C_N^X - lb\ C_{N-\alpha_1}^{X-1} + lb\ C_{N-\alpha_1}^{X-1} - lb\ C_{N-\alpha_2}^{X-2} + \ldots + lb\ C_{N-M-n}^{X-i}$

c'est à dire lb C_N^X bits.

Le volume moyen occupé par les représentations engendrées par notre algorithme sera donc comprise entre lb C_N^X et lb C_N^X + X bits.

4.4. EXEMPLE :

Soit N = $\{ A_1, A_2, A_3, A_4, A_5 \}$ et X = 3

Proposons nous de représenter l'ensemble des valeurs $\{ A_1, A_5, A_3 \}$, une des $C_{10}^3 = 10$ configurations possibles pour un ensemble de 3 valeurs choisies parmi 5 valeurs possibles.

Nous allons d'abord ranger les éléments de la table dans l'ordre de leurs indices ascendants : A_1, A_3, A_5

Nous avons dans nos notations

$\alpha_1 = 1$, $\alpha_2 = 3$, $\alpha_3 = 5$

et

$\alpha'_1 = 1$, $\alpha'_2 = 3-1 = 2$, $\alpha'_3 = 5-3 = 2$.

Ce sont ces trois quantités que nous allons encoder successivement.

Pour α'_1, il nous faudra d'abord calculer les probabilités

$Pr[\underline{\alpha}'_1 = 1]_{5,3} = C_4^2 / C_5^3 = \frac{6}{10}$

$Pr[\underline{\alpha}'_1 = 2]_{5,3} = C_3^2 / C_5^3 = \frac{3}{10}$

$Pr[\underline{\alpha}'_1 = 3]_{5,3} = C_2^2 / C_5^3 = \frac{1}{10}$

Le code préfixe optimal sera donc

"0" pour $\alpha'_1 = 1$,

"10" pour $\alpha'_1 = 2$,

"11" pour $\alpha'_1 = 3$.

Pour α'_2 nous avons les probabilités

$$Pr\left[\underline{\alpha}'_2 = 1 \mid \underline{\alpha}_1 = 1\right]_{5,3} = Pr\left[\underline{\alpha}'_1 = 1\right]_{4,2} = \frac{3}{6},$$
$$Pr\left[\underline{\alpha}'_2 = 2 \mid \underline{\alpha}_1 = 1\right]_{5,3} = Pr\left[\underline{\alpha}'_1 = 2\right]_{4,2} = \frac{2}{6},$$
$$Pr\left[\underline{\alpha}'_3 = 3 \mid \underline{\alpha}_1 = 1\right]_{5,3} = Pr\left[\underline{\alpha}'_1 = 3\right]_{4,2} = \frac{1}{6},$$

et le code préfixe optimal

"0" pour $\alpha'_2 = 1$ sachant que $\alpha_1 = 1$,

"10" pour $\alpha'_2 = 2$ sachant que $\alpha_1 = 1$,

"11" pour $\alpha'_2 = 3$ sachant que $\alpha_1 = 1$.

On a de même pour α'_3 les probabilités

$$Pr\left[\underline{\alpha}'_3 = 1 \mid \alpha_2 = 3\right]_{5,3} = Pr\left[\underline{\alpha}'_1 = 1\right]_{2,1} = \frac{1}{2}$$
$$Pr\left[\underline{\alpha}'_3 = 2 \mid \alpha_2 = 3\right]_{5,3} = Pr\left[\underline{\alpha}'_1 = 2\right]_{2,1} = \frac{1}{2}$$

et le code préfixe optimal

"0" pour $\alpha'_3 = 1$ sachant que $\alpha_2 = 3$

"1" pour $\alpha'_3 = 2$ sachant que $\alpha_2 = 3$

La représentation de l'ensemble des valeurs A_1, A_5, A_3 sera donc "0101".

Il est possible de calculer de la même manière les représentations de tous les ensembles de 3 valeurs distinctes choisies parmi 5 valeurs possibles (voir table 1). La longueur moyenne des 10 représentations est de 3,5 bits est à mettre en rapport avec le minimum théorique ($lb\ C_5^3 = 3{,}32$ bits). Nous avons donc un taux d'occupation de la mémoire $\alpha = 0{,}92$.

TABLE 1 Représentations de tous les ensembles de 3 valeurs distinctes choisies parmi 5 valeurs possibles.

α_1	α_2	α_3	α'_1	α'_2	α'_3	Représentation
1	2	3	1	1	1	000
1	2	4	1	1	2	0010
1	2	5	1	1	3	0011
1	3	4	1	2	1	0100
1	3	5	1	2	2	0101
1	4	5	1	3	1	011
2	3	4	2	1	1	1000
2	3	5	2	1	2	1001
2	4	5	2	2	1	101
3	4	5	3	1	1	11

4.5. APPLICATION AU CAS D'UN ENSEMBLE DE VALEURS QUELCONQUES.

L'algorithme qui s'applique aux ensembles de valeurs quelconques est en tout point semblable à celui que nous venons de présenter pour les ensembles de valeurs distinctes, ce qui nous permettra d'alléger sa présentation.

Soient V_{α_1} , V_{α_2} , ... , V_{α_i} , ..., V_{α_N} les X éléments de la table rangés dans l'ordre de leurs indices croissants.

Nous avons nécessairement pour ces indices

$$1 \leq \alpha_1 \leq N ,$$
$$\dots$$
$$\alpha_{X-1} \leq \alpha_X \leq N .$$

Pour représenter X, il nous suffira donc de représenter les X quantités

$$\alpha_1^* = \alpha_1 ,$$
$$\alpha_2^* = \alpha_2 - (\alpha_1 - 1) ,$$
$$\dots$$
$$\alpha_i^* = \alpha_i - (\alpha_{i-1} - 1) ,$$
$$\dots$$
$$\alpha_X^* = \alpha_X - (\alpha_{X-1} - 1) .$$

Pour obtenir un codage optimal de ces X valeurs, il nous faudra considérer les variables aléatoires $\underline{\alpha}_i$ et $\underline{\alpha}_i^*$ et calculer pour toutes les valeurs possibles de i, j, ..., m, n, ..., v, w les probabilités

$$Pr\left[\underline{\alpha}_1^* = i\right]_{N,X} \qquad (1 \leq i \leq N) ,$$
$$Pr\left[\underline{\alpha}_2^* = j \mid \underline{\alpha}_1 = i\right]_{N,X} \qquad (1 \leq i \leq N \text{ et } 1 \leq j \leq N-i+1) ,$$
$$\dots$$
$$Pr\left[\underline{\alpha}_i^* = n \mid \underline{\alpha}_{i-1} = m\right]_{N,X} \quad (1 \leq m \leq N \text{ et } 1 \leq n \leq N-m+1) ,$$
$$\dots$$
$$Pr\left[\underline{\alpha}_X^* = w \mid \underline{\alpha}_{X-1} = v\right]_{N,X} \quad (1 \leq v \leq N \text{ et } 1 \leq w \leq N-v+1) .$$

Pour $Pr\left[\underline{\alpha}^{*}_{1} = i\right]_{N,X}$ c'est-à-dire la probabilité que le premier élément de X dans l'ordre des indices croissants soit d'indice i, nous procéderons comme suit : Il y a ${}^{R}C^{X}_{N}$ configurations possibles pour un ensemble de X valeurs quelconques choisies parmi N. Si nous fixons l'indice du premier élément à la valeur i, nous laissons N-i+1 valeurs possibles pour les X-1 éléments restants, ce qui fait ${}^{R}C^{X-1}_{N-i+1}$ configurations possibles.

Nous aurons donc, pour tout i compris entre 1 et N

$$Pr\left[\underline{\alpha}^{*}_{1} = i\right]_{N,X} = \frac{{}^{R}C^{X-1}_{N-i+1}}{{}^{R}C^{X}_{N}}$$

que nous pourrons aussi écrire

$$Pr\left[\underline{\alpha}^{*}_{1} = i\right]_{N,X} = \frac{C^{X-1}_{N-i+X-1}}{C^{X-1}_{N+X-1}} = \frac{X\,(N-i+X-1)!\,(N-1)!}{(N-i)!\,(N+X-1)!}$$

Pour les autres probabilités, il nous suffira d'observer que la probabilité que la i eme valeur de X rangée dans l'ordre des indices croissants sont d'indice n + m alors que la i-1 eme valeur est d'indice n est la même que la probabilité que le premier élément d'un ensemble de X-i+1 valeurs quelconques choisies parmi N-m+1 valeurs possibles soit d'indice n.

$$Pr\left[\underline{\alpha}^{*}_{i} = n \mid \alpha_{i-1} = m\right]_{N,X} = Pr\left[\underline{\alpha}^{*}_{i} = n\right]_{N-m+1,\,X-i+1}$$

L'algorithme sera en tout point semblable à celui correspondant aux ensembles de valeurs distinctes.

Algorithme.

(i) Ranger les éléments de la table dans leur ordre d'apparition dans une énumération conventionnelle de l'ensemble des X valeurs possibles.

Soient $\alpha_1, \alpha_2, \ldots, \alpha_X$ les valeurs des indices correspondant dans cette énumération aux éléments de la table, calculer les X quantités

$$\alpha^{*}_{1} = \alpha_1,\ \alpha^{*}_{2} = \alpha_2 - \alpha_1 + 1,\ \ldots,\ \alpha^{*}_{i} = \alpha_i - \alpha_{i-1} + 1,\ \ldots, \alpha^{*}_{X} = \alpha_X - \alpha_{X-1} + 1$$

(ii) Calculer les probabilités $Pr\left[\underline{\alpha}_1^* = i\right]_{N,X}$ pour toutes les valeurs de i comprises entre 1 et N et construire le code préfixe optimal pour α_1^* en appliquant l'algorithme de HUFFMAN.
Ecrire la représentation correspondante à la valeur α_1^*.

(iii) Faire $q := 2$, $\alpha := \alpha_1^*$, $\nu := N - \alpha_1^* + 1$, $\xi := X - 1$.

(iv) Calculer les probabilités

$$Pr\left[\underline{\alpha}_q^* = j \mid \underline{\alpha}_{q-1} = \alpha\right]_{N,X} = Pr\left[\underline{\alpha}_1^* = j\right]_{\nu,\xi}$$

pour toutes les valeurs de j comprises entre 1 et ν et construire le code préfixe optimal pour α_q^* en appliquant l'algorithme de HUFFMAN.
Ecrire la représentation correspondant à la valeur α_q^* à la suite des symboles déjà écrits.

(v) Si $q = N$, l'algorithme est terminé et la suite des symboles écrits est la représentation de l'ensemble des X valeurs.
Si $q < X$,
faire $q := q+1$, $\alpha := \alpha + \alpha_{q-1}^* - 1$, $\nu := \nu - \alpha_{q-1}^*$, $\xi := \xi - 1$
et aller en (iv).

4.6. APPLICATION AUX ENSEMBLES ORDONNES DE VALEURS.

Les algorithmes qui s'appliquent aux ensembles ordonnés de valeurs n'offrent pas le même intérêt que les précédents.

En effet, lorsque l'ordre des éléments du sein de la table est fixé et qu'il n'est pas possible donc de les réarranger dans un ordre conventionnel, la connaissance des m premières valeurs de la table n'apporte pratiquement pas de renseignements sur les X-m valeurs rsstantes.

C'est pourquoi nous présenteront très brièvement les deux cas des ensembles ordonnés de valeurs distinctes et des ensembles ordonnés de valeurs quelconques.

Dans le cas d'un ensemble ordonné de valeurs distinctes, la seule information mutuelle existant entre les éléments de l'ensemble provient du fait que deux éléments de la table ne peuvent avoir la même valeur. Si nous connaissons donc le premier élément de la table, nous saurons que les éléments suivants ne disposeront plus que de N-1 valeurs possibles. Au fur et à mesure que nous encodons les valeurs de la table, nous observerons donc une réduction progressive de la quantité de mémoire nécessaire pour encoder les valeurs suivantes. Ce gain de mémoire est toutefois très modeste (moin d'un bit à chaque étape).

Algorithme :

Soient $\alpha_1, \alpha_2, \ldots, \alpha_X$ la suite des indices des éléments de la table dans une énumération conventionnelle de N.

(i) Faire $q := 1$, $\nu := N$.

(ii) Encoder α_q sur $lb\ \nu$ bits .

(iii) Si q = X, l'algorithme est terminé .

Si $q < X$

faire $\alpha_j = \begin{cases} \alpha_j & \text{si } \alpha_j < \alpha_q \\ \alpha_j - 1 & \text{si } \alpha_j > \alpha_q \end{cases}$

pour tous les j compris entre $q+1$ et X .

Faire $q := q+1$, $\nu := \nu - 1$

et aller en (ii).

Dans le cas d'un ensemble ordonné de valeurs quelconques, la connaissance d'un ou plusieurs éléments de la table n'apporte aucune renseignement sur les autres éléments de la table. La représentation minimale sera obtenue en représentant successivement sur lb N bits les X valeurs de la table. L'algorithme de codage est trivial.

4.7. CONCLUSIONS.

Des trois algorithmes que nous venons de présenter, seuls les deux premiers méritent qu'on s'y attarde. S'ils permettent en effet une gestion quasi-optimale de l'espace mémoire, ils restent difficiles à mettre en oeuvre en raison de la longueur des calculs qu'implique chaque codage et chaque décodage. Il n'y a rien d'étonnant à cela dans la mesure où ces algorithmes tiennent compte de toutes les informations mutuelles existant entre les éléments de la table et découlant de la structure des données de la table.

Nous verrons dans les chapitres suivants que les algorithmes pratiques d'optimisation du volume mémoire rechercheront avant tout à tirer parti des informations mutuelles existant entre les éléments de la table du fait des similitudes existant entre eux. Toutefois, il est des cas où il est possible de s'inspirer des algorithmes que nous venons de décrire pour concevoir des structures de représentations qui, elles, ne seront plus quasi-optimales mais seront beaucoup plus faciles à mettre en oeuvre. Si, par exemple, nous avons à représenter un ensemble de 1.000 valeurs comprises entre 1 et 50.000, il peut être intéressant de ranger ces 1.000 valeurs dans l'ordre croissant et de représenter les différences entre deux éléments consécutifs au lieu des valeurs de ces éléments.

CHAPITRE CINQ

LES TECHNIQUES D'OPTIMISATION LOCALE DU VOLUME MEMOIRE OCCUPE.

5.1. PRESENTATION GENERALE DES TECHNIQUES D'OPTIMISATION LOCALE.

Les techniques d'optimisation du volume mémoire occupé qui seront présentées dans ce chapitre présentent toutes un point commun : Elles sont applicables aux tables de valeurs dont les éléments sont des chaînes de caractères et elles s'attachent à trouver des représentations optimales pour ces chaînes de caractères sans remettre en question les structures globales des représentations. C'est pourquoi, nous parlerons de techniques d'optimisation locale par opposition aux techniques d'optimisation globale qui, elles, sont susceptibles de modifier l'ordonnance générale des structures de représentation auxquelles elles s'appliquent.

L'intérêt majeur des techniques d'optimisation locale du volume mémoire occupé provient de leur souplesse et de leur facilité de mise en oeuvre. Elles permettent en effet de réduire le volume occupé par une table de valeurs, sans devoir modifier l'architecture de la structure de représentation et donc sans affecter, le cas échéant, les temps d'accès aux informations. En particulier ces techniques sont compatibles avec l'usage du code haché et se prêtent fort bien à la représentation de tables dont le contenu varie avec le temps.

En contrepartie, ces techniques ne s'avèreront pas capables d'exploiter les renseignements que nous pourrions avoir sur la structure de la table à représenter et sur son contenu autrement que d'une manière statistique. Si la table à représenter est une table de messages d'erreurs, nous pouvons nous attendre à rencontrer beaucoup de mots et d'expressions communs à plusieurs

messages. Comme dans ce cas, le contenu de la table varie fort peu avec le temps, les techniques d'optimisation globale du volume occupé seront plus indiquées.

D'autre part, il ne faut pas oublier que les techniques d'optimisation locale conduiront souvent à des représentations de longueur variable pour les différentes valeurs de la table, ce qui compliquera la tâche du système de gestion et pourra affecter les temps d'accès aux informations.

Toutes les techniques que nous présenterons dans ce chapitre, sauf la dernière, seront toutes des techniques de recodage des chaînes de caractères. Nous présenterons à la fois des techniques fort empiriques et des méthodes beaucoup plus raffinées mais d'un usage beaucoup moins fréquent.

5.2. LES TECHNIQUES DE RECODAGE DES CHAINES DE CARACTERES.

Nous nous trouvons au départ en présence d'une table de X valeurs représentées sous la forme d'un ensemble de chaînes de caractères.

Le plus souvent ces chaînes seront de longueur fixe et le jeu de caractères utilisé sera le jeu de caractères standard de la machine qui nécessite généralement 6 ou 8 bits par caractères.

Si nous notons l la longueur supposée fixe de toutes les chaînes et c le nombre de bits nécessaires pour représenter un caractère, le volume mémoire occupé par la table de valeurs vaudra X.l.c bits.

Pour réduire cette quantité, nous pouvons agir à la fois sur l et sur c, c'est-à-dire sur la longueur des chaînes et sur le nombre de bits nécessaires pour représenter chaque caractère.

Rares sont, en effet, les cas où ces deux facteurs ne sont pas susceptibles de réduction. Tout d'abord, la partie significative des chaînes est presque toujours un texte de longueur variable, le restant de la chaîne étant occupé par des caractères de remplis-

sage. De plus, les 6 ou 8 bits utilisés pour la représentation permettent un maximum de 64 ou de 256 caractères, ce qui, dans le dernier cas surtout, est presque toujours surabondant.

Une des premières techniques possibles pour réduire le volume occupé est donc le passage du format fixe au format variable pour la représentation des chaînes de caractères.

Si $\bar{l}$ est la longueur moyenne de la partie significative des chaînes à représenter, le pourcentage de gain réalisé vaudra en première approximation

$$\frac{l - \bar{l}}{l} \cdot 100\ \% \ .$$

Il faudra toutefois tenir compte du fait que nous avons maintenant à représenter les longueurs de chaque chaîne soit au moyen d'un délimiteur, c'est-à-dire d'un caractère spécial servant à délimiter les chaînes, soit en faisant précéder chaque chaîne de l'indication de sa longueur.

Dans le premier cas, nous aurons à tenir compte du coût d'insertion du délimiteur dans le jeu de caractères (ce coût sera nul tant que le jeu de caractères sera surabondant) et de la place occupée par ce délimiteur.

Dans le second cas, nous aurons à tenir compte du nombre de bits nécessaires pour représenter la longueur des chaînes. Pour ne pas compliquer la structure de représentation, nous allons supposer que nous réserverons un caractère à cet usage, ce qui nous permettra de représenter des longueurs de chaînes allant jusqu'à 64 voir 256 caractères si nous continuons d'utiliser le jeu standard de la machine.

Compte tenu de ces hypothèses, le pourcentage de gain effectivement réalisé vaudra dans les deux cas

$$\frac{l - (\bar{l} + 1)}{l} \cdot 100\% \ .$$

Si notre jeu de caractères n'est pas surabondant, le pourcentage de gain mémoire vaudra

$$\frac{l.lbc - (l+1).lb(c+1)}{l.lbc} . 100\%$$

dans le premier cas et

$$\frac{l.lbc - (lbl + l.lbc)}{l.lbc} . 100\%$$

dans le second.

D'autres techniques de compression des caractères de remplissage sont possibles : HAHN (16) en cite quelques unes. L'éditeur de textes WYLBUR, par exemple, découpe les chaînes de caractères en segments, chaque segment pouvant décrire jusqu'à 15 caractères blancs suivis de 15 caractères non blancs. Chaque chaîne est ensuite rangée sous la forme d'une zone donnant le nombre de segments utilisés et d'une zone contenant les segments eux-mêmes.

Une autre technique de compression de textes est possible : le recodage des caractères sur une longueur moindre que les 6 ou 8 bits que requiert le jeu de caractères standard de la machine.

Dans de nombreux cas, en effet, les chaînes de caractères que nous aurons à représenter ne nécessiteront pas des jeux de caractères aussi importants. Si le jeu minimum de caractères que nécessite la représentation des éléments de la table comporte n symboles et si nous ne possédons aucune information sur leurs probabilités d'occurrence, la quantité de mémoire nécessaire pour représenter un caractère sera de lb n bits.

En général, ce nombre n'est pas entier. Comme il n'est pas possible de manipuler des fractions de bits, deux solutions s'offriront à nous : la première sera d'arrondir la valeur au nombre entier immédiatement supérieur, la seconde d'utiliser un codage polynomial. Le principe en est fort simple : Supposons que nous ayons à représenter la chaîne de caractères XYZV qui sont représentés par les valeurs, X, Y, Z, V, dans un alphabet de C caractères .

Nous les représenterons par la valeur du polynôme

$$X.C^3 + Y.C^2 + Z.C + V$$

que nous pourrons d'ailleurs calculer d'une manière itérative.

$$((X.C + Y).C + Z).C + V .$$

Pour donner une idée des gains de mémoire possibles qu'il nous suffise de dire que la représentation d'un texte purement alphabétique, c'est-à-dire sans chiffres ni caractères spéciaux, ne nécessite qu'un alphabet de 27 symboles, ce qui représente 4,75 bits par caractère. Il est donc théoriquement possible d'obtenir, dans ce cas, des gains de mémoire allant jusqu'à 40,6 % pour une représentation initiale sur 8 bits.

Plusieurs systèmes comprtent d'ailleurs des représentations spéciales pour certaines classes de chaînes de caractères. Parmi elles, les nombres décimaux occupent une place spéciale : nous ne reviendrons pas sur la possibilité de représenter les chiffres décimaux en format condensé sur 4 bits au lieu de 8, introduite par IBM dans sa série 360. CHEN et HO (2) ont d'ailleurs montré qu'il était possible d'améliorer l'efficience de la représentation des nombres décimaux tout en conservant un format fixe.

Ils se sont intéressés à la possibilité de ne plus représenter les chiffres individuellement mais par groupes de 2, 3, ..., n chiffres. En particulier, il est possible de représenter 2 chiffres avec 7 bits et 3 chiffres avec 10 bits. Le nombre minimum de bits nécessaire pour représenter un chiffre décimal étant de lb 10 = 3,32 bits, nous pouvons en déduire les taux d'occupation mémoire correspondant à ces deux modes de représentation. Ils sont respectivement de 0,947 et de 0,9965 contre 0,830 pour la représentation classique sur 4 bits.

La représentation des nombres décimaux par groupes de 3 bits est donc pratiquement optimale. CHEN et HO se sont donc attachés à définir un algorithme de codage n'apportant qu'un minimum de modifications par rapport au codage sur 4 bits et relativement facile à réaliser par un module cablé : dans plus de 50 % des cas, il n'opère qu'une troncature du premier bit.

Dans le même ordre d'idée, citons le système RADIX commun aux PDP-11 et aux DEC-10 (6). RADIX comprend à la fois un jeu réduit de 39 caractères et les procédures de transcodage d'une représentation polynomiale à raison de 3 caractères par mot de 16 bits à la représentation standard du système qui est le code ASCII.

Un troisième exemple de techniques de compression de chaînes de caractères nous est donné par le système SESAM (7). Les techniques de compression de SESAM utilisent à la fois la suppression des caractères de remplissage (blancs et zéros) et la conversion des nombres décimaux en binaire ; le résultat de la conversion dépend à la fois du contenu de la zone à comprimer et des attributs qui lui ont été assignés lors de sa déclaration (numérique ou alphanumérique, cadrée à gauche ou à droite).

Toutes les techniques de compression que nous venons de voir ont un point commun : elles ne tiennent pas compte des renseignements que nous pourrions avoir sur la fréquence des différents caractères au sein des éléments de la table. Or, dans de nombreux cas, il est possible d'obtenir ces fréquences d'une façon relativement aisée : c'est notamment le cas lorsque les éléments de la table sont des textes d'un langage connu pour lequel ces statistiques existent. Sinon, il est toujours possible de les calculer à partir d'un échantillon des éléments de la table.

Nous présenterons deux techniques de réduction du volume mémoire basées sur ces fréquences : l'une est l'algorithme de HUFFMAN (19), l'autre une méthode nouvelle due à HAHN (16).

5.3. L'ALGORITHME DE HUFFMAN.

L'intérêt majeur de l'algorithme de HUFFMAN provient du fait que c'est l'algorithme donnant les codes préfixes de longueur moyenne minimale pour un ensemble probabilisé de messages quel que soit le nombre de symboles D utilisés pour le codage ((19) voir aussi (1)).

On trouvera une description de l'algorithme dans (12) (19) et (26) ; la présentation que nous en ferons ici se voudrait la synthèse de ces trois exposés.

Pour simplifier les choses, nous allons nous limiter au cas du codage binaire.

Soit un ensemble de n messages. Sans perdre aucune généralité, il est possible de numéroter les n messages dans l'ordre décroissant de leurs probabilités

$$p(1) \geqslant p(2) \geqslant \dots \geqslant p(n-1) \geqslant p(n)$$

Si $l(i)$ désigne la longueur du mot du code correspondant au i eme message, on a pour tout code optimal

$$l(1) \leqslant l(2) \geqslant \dots \geqslant l(n-1) \geqslant l(n)$$

Pour tout code préfixe optimal on a nécessairement

$$l(n-1) = l(n)$$

(Dans un code préfixe, aucun mot du code ne peut être réutilisé comme séquence initiale d'un autre mot : les $l(n-1)$ premiers bits des mots du code correspondant au (n - 1) eme et au n eme message suffisent donc à les distinguer).

Pour obtenir le code préfixe de longueur minimale nous allons donc procéder en regroupant les deux messages les moins probables, qui seront nécessairement représentés par des mots de longueur

égale, et en réitérant le processus. Nous construirons ainsi une arborescence à partir de laquelle nous pourrons dériver l'ensemble des codes préfixes optimaux codant l'ensemble des messages.

Algorithme :

(i) Ranger les n messages à représenter dans l'ordre de leurs probabilités décroissantes.
Débuter la construction de l'arborescence en construisant l'ensemble de ses n feuilles, chaque feuille correspondant à un message possible.

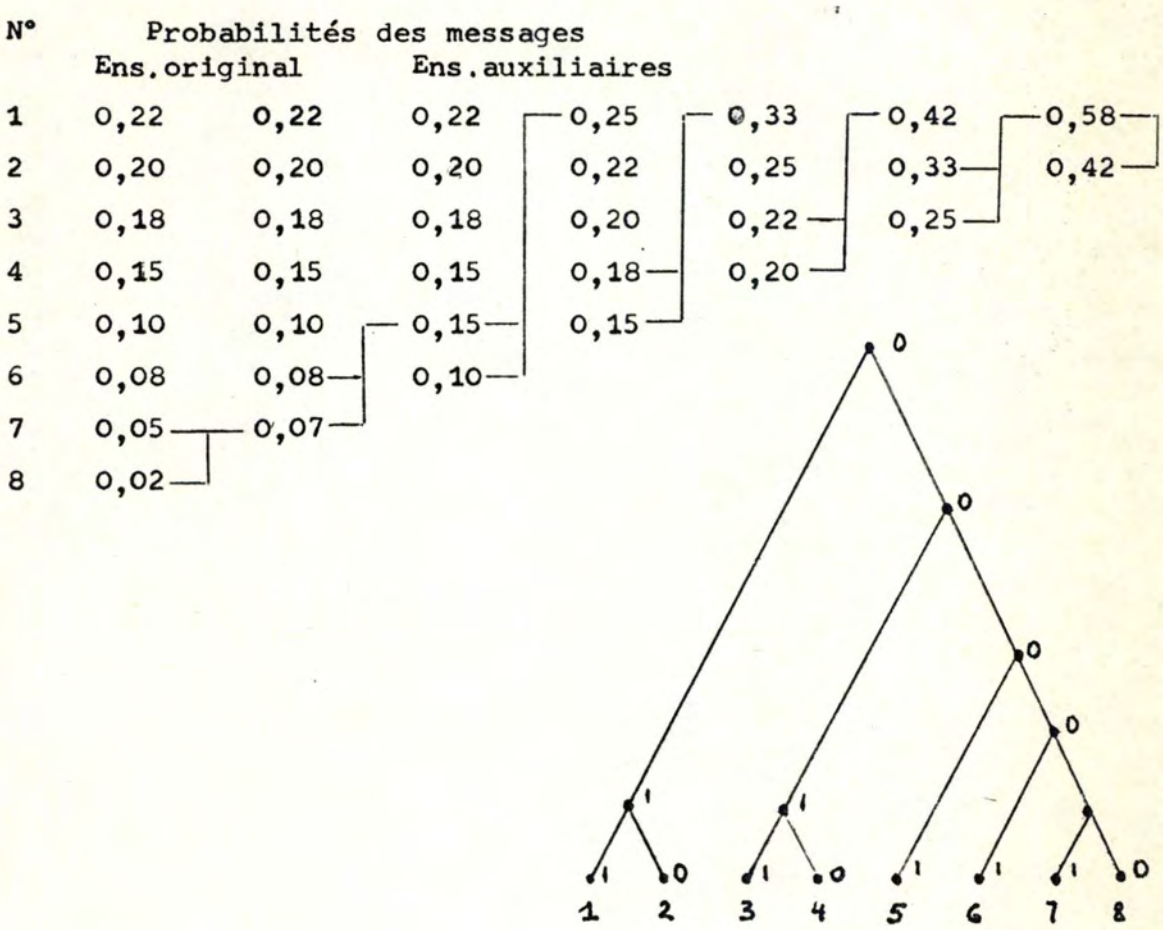
(ii) Grouper les deux messages les moins probables en un seul.
Construire le noeud de l'arborescence correspondant au message composite ainsi formé et les arcs le joignant aux deux sommets correspondant aux deux messages les moins probables.

(iii) Former une nouvelle liste où les deux messages les moins probables seront remplacés par un seul.
Si cette liste ne comporte qu'un message, aller en (iv) sinon, ranger les éléments de cette liste dans l'ordre des probabilités décroissantes et aller en (ii).

(iv) En partant de la racine du graphe, marquer respectivement "0" et "1" chacune des deux sommets adjacents. Réitérer le processus à partir de chaque sommet marqué, jusqu'à ce qu'il ne reste aucun sommet non marqué.

(v) Pour lire le mot du code correspondant à un message donné, on parcourra l'arborescence, dans le sens indiqué par les arcs, depuis la racine jusqu'à la feuille correspondant au message.

EXEMPLE



On peut lire sur l'arborescence les mots du code correspondant aux différents messages : par exemple le mot "001" qui correspond au message numéro 5.

5.4. LA METHODE DE HAHN.

La méthode de HAHN (16) est une méthode de compression de chaînes de caractères basée à la fois sur la reconnaissance des caractères de remplissage et un recodage des autres caractères tenant compte de leur fréquence d'occurrence.

Le point le plus intéressant de la méthode est la façon dont elle envisage ce recodage : généralement, les algorithmes de codage ou de recodage qui tiennent compte de la fréquence des caractères conduisent à des codes de longueur variable. C'est le cas notamment de l'algorithme de HUFFMAN. De tels codes exigent malheureusement un décodage bit à bit et sont, de ce fait, mal adaptés aux possibilités arithmétiques et logiques des ordinateurs.

HAHN propose d'obvier à cet inconvénient en représentant l'ensemble des c caractères possibles par un ou deux caractères d'un alphabet auxiliaire de B caractères. Comme il est nécessaire de prévoir un caractère spécial pour signifier que le caractère suivant n'a pas sa signification habituelle, on pourra représenter jusqu'à 2B-1 caractères distincts. Il sera donc possible de représenter les B-1 caractères les plus probables de notre alphabet initial sur 1b B bits tandis que les autres caractères seront eux codés sur 2 1b B bits.

Cette technique n'est d'ailleurs pas originale et était notamment utilisée dans le code BAUDOT des téléscripteurs pour représenter plus de 32 symboles avec un code à 5 moments. Ici, cependant apparaît le véritable point original de la méthode : comme le code que nous venons d'obtenir a les mêmes caractéristiques qu'un code de longueur constante, il devient possible d'utiliser un codage polynomial et de représenter une suite de N caractères de l'alphabet auxiliaire $a_1, a_2, \ldots, a_N$ par la valeur correspondante du polynôme

$$a_1 B^{N-1} + a_2 B^{N-2} + \ldots + a_N \quad .$$

HAHN propose de représenter ce nombre entier classiquement sur un seul mot de mémoire : pour toute valeur du facteur de groupe N il existera donc une limite supérieure au nombre de symboles B de l'alphabet auxiliaire. Cette limite supérieure correspondra d'ailleurs à la valeur optimale de B pour le facteur de groupage considéré.

Comme chaque caractère de l'alphabet auxiliaire peut être représenté par un entier compris en 0 et B-1, la plus grande valeur atteinte par un polynôme de degré N sera $B^N - 1$. Si L est la valeur du plus grand encier pouvant être représenté par un seul mot de mémoire, la valeur optimale de B correspondant à un facteur de groupage N sera donc la plus grande valeur entière de B solution de l'inéquation

$$B^N - 1 \leq L .$$

Pour chaque valeur possible de N, nous obiendrons donc une valeur optimale de B qui tiendra compte des caractéristiques de la machine qui sera utilisée pour l'application considérée. HAHN a étudié, dans le cas d'un IBM 360, les différentes valeurs B correspondant à des facteurs de groupage allant de 5 à 8. Les résultats de ses mesures statistiques sur la compression de programmes FORTRAN et Assembleur et de textes anglais semblent indiquer que le facteur de groupage optimal serait de 7 caractères par mot, ce qui correspond à un alphabet auxiliaire de 21 symboles.

Il est toutefois difficile de juger des performances de cette technique de codage dans la mesure où les seuls résultats complets communiqués par HAHN concernent les perfomances globales de la méthode et tiennent donc compte du gain de mémoire, fort important, réalisé par l'algorithme de compactage des caractères de remplissage qui compte les blancs initiaux et supprime les blancs finaux.

Contentons-nous de dire que cette méthode appliquée à un programme FORTRAN de plus de 31.000 caractères a conduit à une représentation occupant environ 2,01 bits par caractère tandis qu'appliquée à un texte anglais de plus de 27.000 symboles, elle a conduit à des valeurs de 3,38 et de 3,17 bits par caractère, selon que l'on tienne compte ou non de la différence entre majuscules et minuscules.

5.5. LA METHODE DES EMPREINTES.

Due à LOUIS-GAVET (22), cette méthode se caractérise par son approche toute particulière des problèmes que pose la concentration des tables de valeurs occupant un volume important, lorsqu'elles ne sont destinées qu'à être consultées en accès aléatoire.

Si la table de valeurs que nous avons à représenter est destinée à n'être jamais consultée qu'en accès aléatoire, toute structure de représentation permettant au système de gestion de données de répondre à la question de savoir si une chaîne donnée fait partie de la table sera une représentation acceptable de la table de valeurs. En particulier, il n'est pas nécessaire que cette représentation permette de reconstituer l'ensemble des éléments présents dans la table.

Dans de nombreuses applications, l'ensemble des valeurs effectivement susceptibles de figurer dans la table est d'un tout autre ordre de grandeur que l'ensemble des chaînes de caractères sur lequel les valeurs sont codées.

Ce sera notamment le cas chaque fois que toutes les valeurs présentes dans la table présenteront entre elles d'importantes analogies formelles ou de fortes corrélations entre la fréquence des caractères et leur contexte (position au sein de la chaîne, valeurs des caractères voisins). Dans ces cas, il serait donc possible d'utiliser les techniques de compression de volume ou d'envisager une optimisation locale.

Toutes ces techniques définissent des transformations réversibles de la structure de représentation originale qu'il est toujours possible de reconstituer à partir de la nouvelle structure de représentation. L'idée de base de la méthode des empreintes sera de considérer des transformations biunivoques mais non réversibles.

Soient N l'ensemble de toutes les chaînes de caractères considérées,

S l'ensemble de toutes les chaînes de caractères codant des valeurs effectivement susceptibles de figurer dans la table,

E un ensemble de chaînes de caractères ν que nous appellerons ensemble des empreintes,

X l'ensemble des X valeurs à représenter.

Pour bâtir notre représentation, nous allons construire une application A de N dans E faisant correspondre à toute chaîne σ de N une empreinte $\nu = f(\sigma)$ telle que

$$\text{card}\{S \cap f^{-1}(\nu)\} \leq 1 \quad \forall\ \nu \in E .$$

L'ensemble des X chaînes de E qui seront les empreintes des X valeurs par l'application constituera la structure de représentation recherchée.

On peut imaginer bien des modes de définition de A et, partant, bien des techniques de construction d'empreintes.

Dans le cadre de l'étude de la concentration d'un fichier reprenant les caractéristiques principales des livres possédés par une bibliothèque publique, LOUIS-GAVET a étudié une technique de construction d'empreinte basée sur l'extraction des caractères les plus significatifs des chaînes de caractères de l'ensemble S. Ces caractères les plus significatifs seront ceux qui seront les moins corrélés entre eux et qui présenteront les distributions de valeurs les plus uniformes possibles. On sera, par exemple, amené à ne considérer que des caractères situés à une certaine distance, 4 caractères au moins, de manière à ne prendre que des caractères indépendants.

Il est malheureusement regrettable que nous ne disposions pas des résultats nous permettant de juger de la praticabilité de la méthode : il n'en demeure pas moins que l'approche suivie par LOUIS-GAVET méritait de retenir notre attention ne fut-ce qu'en raison de son originalité.

CHAPITRE SIX

LES TECHNIQUES D'OPTIMISATION GLOBALE DU VOLUME MEMOIRE OCCUPE

6.1. OPTIMISATION LOCALE ET OPTIMISATION GLOBALE.

Nous avons vu, dans le chapitre précédent, quelques techniques d'optimisation du volume mémoire qui s'appliquaient à des tables compsées de chaînes de caractères. Toutes ces techniques conservaient le mode habituel de représentation des tables de valeurs, élément par élément. L'optimisation portait donc sur les représentations individuelles des chaînes de caractères ; c'est pourquoi nous avons parlé d'optimisation locale.

Les meilleurs résultats avaient été obtenus par des techniques tenant compte des fréquences d'apparition des différents caractères au sein des chaînes : algorithme de HUFFMAN, méthode de HAHN, méthode des empreintes.

Pour améliorer encore les taux d'occupation de la mémoire, il faudrait tenir compte des corrélations existant entre caractères voisins : nous serions ainsi amenés à définir des codes dépendant du contexte du caractère codé ou à coder les textes par paquets de plusieurs caractères. SHANNON avait d'ailleurs étudié, dès 1949, la possibilité d'encoder des textes anglais en considérant un contexte allant jusqu'à huit caractères (12).

Nous obtiendrons de cette manière des codes de plus en plus performants et donc des longueurs moyennes des chaînes tendant vers la valeur minimale théorique mais au prix d'une croissance démesurée de la complexité des algorithmes de codage et du volume occupé par ceux-ci.

Aussi pouvons-nous nous demander s'il ne serait pas possible d'arriver aux mêmes résultats par d'autres méthodes et à

et à un coût moindre. Ce que nous cherchons, à travers nos techniques d'optimisation c'est à exprimer dans notre structure de représentation le plus grand nombre de propriétés communes aux éléments de la table afin de minimiser les volumes nécessaires à la représentation de ces éléments.

Les techniques de codage, que nous venons d'envisager exprimaient ces propriétés en termes de fréquences d'apparition des caractères, de corrélations, de probabilités conditionnelles.

D'autres solutions sont possibles tenant mieux compte de la spécificité des propriétés effectivement présentes dans les tables : l'une de ces solutions est d'ailleurs d'emploi classique pour la représentation des tables de messages d'erreur des compilateurs "compile and go". C'est par elle que nous commencerons notre étude des techniques d'optimisation globale.

Un compilateur "compile and go" est un compilateur engendrant du code machine destiné à être exécuté tel quel sans faire appel à l'éditeur de liens ni au chargeur. Dans les deux cas qui nous intéressent (PUFFT (25), PL/C (28)) ce processus a lieu entièrement en mémoire centrale : il est donc nécessaire d'être particulièrement économe de l'espace mémoire, ce qui explique l'emploi de structures de représentation conçues spécialement en vue de l'optimisation du volume mémoire occupé. PUFFT (PURDUE UNIVERSITY FAST FORTRAN TRANSLATOR) utilise, en effet, pour sa table des messages d'erreurs une structure de représentation à trois niveaux

- chaque message peut comporter jusqu'à 3 phrases choisies dans une bibliothèque pouvant comporter un maximum de 511 phrases,
- chacune de ces phrases peut comporter à son tour jusqu'à 5 mots anglais choisis dans une table de 127 mots,
- chacun de ces mots peut comporter jusqu'à 12 caractères.

Il est également possible d'insérer dynamiquement des valeurs dans les messages mais leur emplacement devra avoir été spécifié dans la description du message.

PL/C utilise lui une structure à la fois plus simple et plus raffinée : elle ne comporte que deux niveaux, celui des messages et celui des phrases communes à plusieurs messages, mais la procédure de construction de la structure de représentation est partiellement automatisée.

Si la construction de l'ensemble des phrases communes à plusieurs messages est encore exécutées manuellement, le remplacement de toutes les occurrences de chaque phrase, dans les messages ou les autres phrases, par une référence à la phrase ainsi mise en évidence, s'effectue lui automatiquement et de manière optimale grâce à un algorithme de programmation dynamique.

Le gain de mémoire réalisé grâce à l'extraction des phrases communes sera de 27 % pour la table des messages d'erreur du compilateur PL/C. Ce gain n'est pas beaucoup plus important que celui qu'on eut pu attendre d'un codage polynomial ; par contre, le processus de génération des messages à partir de leur représentation est beaucoup plus rapide.

6.2 NOTION DE REPRESENTATION GLOBALE.

Si nous regardons d'un peu plus près les structures de représentation que nous venons de décrire, nous verrons qu'elles correspondent, en fait, à des représentations de grammaires non-contextuelles.

Chaque fois que l'un de ces deux compilateurs aura besoin de sortir un message, il devra faire appel à un algorithme qui engendrera le message en interprêtant les règles de production de la grammaire.

D'une manière un peu plus formelle nous dirons que ces structures de représentation sont équivalentes à des grammaires

$$G = (V_N, V_T, \Phi, S)$$

avec

$$V_N = \{ S, M_1, \ldots, M_n, \ldots \}$$

$$V_T = \{ \ldots \}$$

$$\Phi = S \rightarrow M_1 \mid \ldots \mid M_n$$

$$\ldots \rightarrow \ldots$$

Pour engendrer le i-ème message de la table, il suffira donc de chercher la dérivation de M_i ne comportant pas d'éléments de V_N (Remarquons que les tables des messages d'erreurs sont des ensembles ordonnés de valeurs distinctes : la méthode d'accès utilisée ici est donc définie sur la structure de liste linéaire de la table).

Comme on le voit, cette structure de représentation n'a plus grand chose de commun avec le mode habituel de représentation des tables. Au lieu de représenter directement les éléments de la table des messages, on représente en effet un algorithme permettant d'engendrer ces éléments. C'est pourquoi nous parlerons d'optimisation globale et de représentations globales.

Nous appellerons donc représentation globale d'une table de valeurs toute représentation d'un algorithme générateur construisant les éléments de la table, en respectant l'ordre des éléments de la table chaque fois qu'il sera significatif.

Ces représentations ne se prêtent pas, bien sûr, à la représentation de tables de valeurs dont le contenu est susceptible de changer fréquemment; lors de chaque mise à jour, il faudrait redéfinir un nouvel algorithme. Elles permettent cependant tous les modes de consultation qu'ils soient définis sur la structure de liste linéaire de la table ou en accès aléatoire.

Dès que la table de valeurs représentée atteint une certaine taille, la consultation de la table en accès aléatoire devient cependant extrêmement lourde : il faudrait, en effet, lors de chaque consultation, engendrer en moyenne

- $\frac{X}{2}$ éléments si la valeur spécifiée fait bien partie de la table
- X éléments sinon.

Aussi, faudra-t-il disposer d'un moyen plus direct de déterminer si une valeur quelconque fait ou non partie de l'ensemble des valeurs engendrées par l'algorithme. En général, il sera possible de construire un algorithme de consultation de la représentation globale permettant de répondre à cette question sans avoir à engendrer effectiement l'ensemble des valeurs.

Ce sera notamment le cas pour les structures arborescentes que nous présenterons dans la prochain chapitre et pour toutes les structures de représentations équivalentes aux grammaires régulières. Une grammaire régulière définit, en effet, à la fois un algorithme générateur appliquant les règles de production et un automate fini reconnaissant le langage engendré. On pourrait améliorer encore la structure de représentation en représentant non plus la grammaire régulière mais la matrice des transitions de l'automate fini qui lui est associé.

Par contre, dans le cas des ensembles non ordonnés de valeurs distinctes, on pourrait envisager des structures de représentation qui seraient elles des représentations du prédicat d'appartenance à l'ensemble. Deux cas peuvent se présenter selon que ces représentations permettent ou non d'énumérer les éléments de l'ensemble représenté.

Dans le premier cas, les représentations de prédicats d'appartenance se laissent aussi interprêter comme des représentations d'algorithme générateurs. Nous nous trouvons donc en présence de représentations globales ne différant des autres représentations que par leur formalisme.

Dans le cas contraire, nous nous trouvons en présence de représentations nous permettant de décider si une valeur quelconque fait ou non partie de l'ensemble des valeurs représentées mais non d'en énumérer les éléments. C'est pourquoi, ces représentations ne sont pas à proprement parler des représentations globales mais plutôt des représentations d'algorithmes de reconnaissance.

6.3.LA CONSTRUCTION DE REPRESENTATIONS GLOBALES.

Si les représentations globales présentent de grands avantages sur le plan théorique, ils posent cependant un grand nombre de problèmes tant sur le plan du choix des structures de représentation que celui des algorithmes de construction des représentations.

Premier dans l'ordre logique, le problème du choix des structures de représentation l'est aussi par son importance. S'il est, en effet, facile d'obtenir des gains de mémoire impressionnants avec des ensembles de données taillés sur mesure, il est beaucoup plus difficile de concevoir des structures de représentations globales adaptées aux structures de données réelles.

Dans la pratique, nous ne rencontrerons que fort peu de tables de valeurs bien structurées, se prêtant à une représentation concise et élégante. A ce titre, nous pouvons dire que les tables de messages d'erreurs que nous avions étudiées correspondaient à un cas particulièrement favorable en raison de la présence d'une structure linguistique commune avec de nombreuses répétitions de mots et de groupes de mots. Si la table de valeurs à représenter se trouve être une table de mots réservés ou une table de noms propres, il sera nettement plus difficile d'exploiter l'information mutuelle existant entre les éléments de la table.

Nous ne rencontrerons donc, pour ainsi dire jamais de tables de valeurs se prêtant à une représentation purement synthétique : toutes les structures de représentation que nous pourrons envisager impliqueront nécessairement la consultation de tables, c'est-à-dire de structures de représentation séquentielles où des valeurs seront rangées les unes à la suite des autres. Le gain de mémoire proviendra du fait que ces tables occuperont un volume moindre que celui qu'eut nécessité la représentation séquentielle classique de tous les éléments de la table.

Reprenons l'exemple des tables de messages d'erreurs et les solutions adoptées par PUFFT et PL/C.

Dans PUFFT, la structure de représentation de la table comporte la bibliothèque des phrases (un mot machine de 36 bits par phrase) et la table des mots (jusqu'à 12 caractères par mot). Il n'y a donc pas de représentation des messages en tant que tels et la séquence des phrases constituant le message est spécifiée dans la séquence d'appel. Comme on le voit, cette structure de représenta-

tion est extrêmement compacte : selon un de ses auteurs (25), ces deux tables et le code nécessaire pour sélectionner les messages et les transmettre à la routine de sortie occupe un peu plus de 500 mots de mémoire centrale, ce qui fait 18.000 bits compte tenu de la taille des mots de l'IBM 7094.

Dans PL/C, la structure de représentation adoptée pour la représentation de la table comprend deux tables l'une contenant les formes abréviées des messages, l'autre l'ensemble des phrases communes éventuellement abréviées elles aussi. L'ensemble de ces deux tables occupe 8.194 bytes alors que la représentation initiale des messages en exigeait 11.221.

Une fois que nous aurons fixé notre choix sur une classe de structure de représentations, se posera le problème de la construction de la représentation elle-même.

Le problème est loin d'être trivial dans la mesure où nous souhaitions obtenir une représentation optimale, c'est-à-dire plus concrètement et dans le cas qui nous intéresse, celle qui occupera le plus petit volume de toutes les représentations de sa classe.

La première solution qui vient à l'esprit est de procéder nous-même à la construction de la représentation de volume minimal et d'écrire ensuite les procédures de consultation correspondantes. Cette méthode est celle qui fut utilisée dans PUFFT pour construire la table des messages d'erreur. Si elle a l'avantage de laisser ouvertes toutes les possibilités d'optimisation, elle offre cependant l'inconvénient majeur d'être lente, pénible et coûteuse comme le sont toutes les procédures manuelles.

Aussi, a-t-on cherché, à automatiser tout au moins partiellement cette procédure : La méthode choisie pour la construction de la table des messages de PL/C supposait une sélection manuelle de l'ensemble des phrases communes à plusieurs messages mais comportait une procédure automatique d'extraction optimale de ces phrases. Ici

la procédure de consultation a pu être écrite une fois pour toute : elle ne dépend, en effet, que de la structure de représentation adoptée et non du résultat de l'optimisation.

Cette approche est sans doute satisfaisante dans le cas d'un compilateur, c'est-à-dire dans le cas d'un programme destiné à être exécuté des milliers de fois. Pour des applications plus courantes, il est exclu de faire appel à des procédures qui resteraient, ne fut-ce que partiellement manuelles.

C'est pourquoi nous nous sommes tournés vers une troisième solution : l'automatisation intégrale de la procédure de construction de la représentation globale.

Nous nous donnerons, au départ, un ensemble de représentations possibles caractérisées par une structure commune, ensemble que nous supposerons assez vaste et comportant suffisamment de possibilités d'exploiter l'information mutuelle existant entre les éléments des tables à représenter pour permettre une bonne représentation d'un grand nombre de tables. Nous appellerons cet ensemble classe de représentations.

La problème qui se posera sera alors de trouver un algorithme permettant de construire, pour toute table de valeurs possédant au moins une représentation dans la classe considérée, la représentation de longueur minimale membre de cette classe.

Comme les tables que nous considèrerons seront toujours de dimensions finies, la seule condition pour que la classe de représentations considérée comporte au moins une représentation de n'importe quelle table de valeurs représentables par des chaînes de caractères sur un alphabet fini, est qu'elle comprenne l'ensemble de toutes les représentations énumératives des ensembles de chaînes de caractères construites sur cet alphabet.

Remarquons que pour peu que les conventions de codage adoptées pour les représentations membres d'une même classe soient un tant soit peu cohérentes, il devient possible de normaliser au niveau de la

classe de représentations les procédures d'interprétation des représentations en tant qu'algorithmes générateurs et en tant que prédicats d'appartenance.

Tel que nous venons de le définir, le problème de la construction de représentations globales de longueur minimale pour des tables de valeurs n'est qu'un cas particulier du problème plus général de l'inférence de programmes (10).

D'autre part, le problème assez voisin du nôtre de l'inférence de grammaires pour des langages a donné lieu à de nombreuses publications (4) (5) (9) (11)(16) (18).

C'est pourquoi, avant d'aborder l'étude d'une réalisation pratique d'un algorithme de construction de représentations globales de longueur minimale, nous allons brièvement discuter l'apport des techniques d'inférence de grammaires à la solution du problème qui nous occupe.

6.4.L'APPORT DES TECHNIQUES D'INFERENCE.

Le problème général auquel s'attachent les techniques d'inférence est celui de la recherche du meilleur modèle explicatif pour une collection de données expérimentales.

En tant que telles, les techniques d'inférence débordent largement du cadre de l'informatique, domaine dans lesquelles elles auront été surtout utilisées en inférence grammaticale. L'inférence grammaticale se propose, en effet, de rechercher la meilleure grammaire pour un langage dont on ne connait que des échantillons finis. FELDMAN et SHIELD ont toutefois montré dans (10) que l'inférence de grammaires n'était qu'un cas particulier de l'inférence de programmes, ce qui leur a permis de généraliser à l'inférence de programmes un certain nombre de résultats théoriques présentés dans (11).

Ce qui frappe le plus le néophyte en inférence grammaticale, c'est le contraste entre le caractère limité des résultats pratiques obtenus et l'ampleur des développements théoriques portant sur la définition des critères de complexité ou les critères de convergence du processus pour les langages infinis.

Les algorithmes d'inférence de grammaires présentés dans la littérature (9) (4) (5) (16) sont encore fort élémentaires et n'ont jamais été appliqués que sur des exemples "of low complexity" comme le reconnaissent CRESPI-REGHEZZI et MELKANOFF.

Aussi ce seront surtout les résultats théoriques qui retiendront notre attention.

Le travail de définition des critères de complexité qui a été fait en inférence grammaticale et plus spécialement la définition des critères de complexité intrinsèque et extrinsèque caractérisent l'un la taille de la grammaire et l'autre la longueur moyenne des dérivations nécessaires pour engendrer les éléments du langage n'est pas sans rappeler notre discussion des critères relatifs au volume occupé et de temps d'accès aux informations. En particulier l'approche choisie pour définir la complexité extrinsèque d'une grammaire pourrait se révéler des plus utiles pour la définition de meilleurs critères de temps d'accès que les nombres d'accès moyens à la mémoire (9) (11).

Les résultats concernant l'inférence de grammaires pour des langages infinis nous permettent eux d'envisager le problème de l'inférence de représentations globales pour des ensembles de valeurs pouvant comporter un nombre infini d'éléments.

Soit en effet une classe $\mathcal{C}$ de représentations d'ensembles de valeurs non nécessairement finis.

Si toutes les représentations de cette classe sont "calculables", c'est-à-dire qu'elles définissent des algorithmes générateurs calculables, on peut montrer qu'il existe une machine $M_{\mathcal{C}}$ identifiant la "meilleure" représentation $R(X)$ de tout ensemble de valeurs possédant au moins une représentation R faisant partie de $\mathcal{C}$ à partir de n'importe quelle suite d'apprentissage pour autant qu'elle comporte <u>à la fois</u> des éléments de l'ensemble X et des éléments n'en faisant pas partie.

(la démonstration est strictement la même que celle donnée par FELDMAN pour les grammaires dans (11).)

6.5 RECHERCHE D'UNE METHODE PRATIQUE.

Si ces résultats présentent un grand intérêt théorique, leur portée pratique est faible car les algorithmes utilisés pour démontrer les résultats obtenus sont tous posés sur l'énumération de toutes les représentations de la classe rangées par ordre de complexité croissante.

Comme, d'autre part, les algorithmes utilisés en inférence de grammaires ne nous semblaient pas assez performants pour s'appliquer à la construction de représentations globales de longueur minimale, nous avons préféré établir un algorithme d'inférence de représentations basé sur la construction d'une représentation arborescente se substituant pas à pas à la représentation énumérative.

Cette structure de représentation ne présentera comme une hiérarchie d'opérateurs de réduction ou de prétraitement (14) : elle offre le grand avantage de remplacer la recherche de la représentation optimale d'un ensemble de valeurs au sein d'une classe de représentations par la recherche de la succession d'opérateurs optimaux construisant l'arborescence.

La présentation détaillée de l'algorithme et des structures arborescentes qu'il engendre, fera l'objet du chapitre suivant.

CHAPITRE SEPT

UN ALGORITHME D'INFERENCE DE REPRESENTATIONS ARBORESCENTES

7.1. PRESENTATION DE L'ALGORITHME.

L'algorithme que nous présentons ici s'applique aux ensembles de données représentables par des chaînes de caractères et conduit à des représentations arborescentes minimisant le volume total occupé.

Sa principale caractéristique réside dans le mode de construction de l'arborescence : à chaque étape de l'algorithme, on remplace une feuille de l'arborescence par un noeud supportant plusieurs feuilles.

Chaque feuille de l'arborescence représente un ensemble de chaîne et chaque remplacement correspond à une partition de l'ensemble représenté par la feuille considérée : on classifie ses éléments en autant de sous-ensembles qu'il y a de feuilles. Chaque sous-ensemble est défini par un prédicat apportant une information permettant de réduire le volume nécessaire à la représentation d'une chaîne du sous-ensemble : présence ou absence de certains caractères, longueur maximum, ...

Nous dirons alors que l'ensemble des prédicats définissant la partition constitue un opérateur de réduction et c'est cet opérateur qui occupera le noeud remplaçant la feuille traitée tandis que les nouvelles feuilles représentent les sous-ensembles issus de la classification.

A chaque étape de l'algorithme, il va falloir construire l'opérateur de réduction rassemblant les prédicats assurant la meilleure réduction.

Nous conviendrons pour des raisons pratiques de limiter notre choix à un ensemble fini d'opérateurs préselectionnés pour leur généralité et leur efficacité. On pourrait envisager de construite automatique cet ensemble d' opérateurs. Quoique ce problème présente un grand intérêt, nous ne l'avons pas abordé et nous sommes limités à deux types d'opérateurs :

Le premier type comprend les opérateurs testant la longueur effective des chaînes, caractère de remplissage non compris et les classifiant en sous-ensembles de longueur inférieur à L1, supérieur à L1 mais inférieur à L2, ..., supérieur à L2.

Le second comprend lui les opérateurs testant le nième caractère de chaque chaîne et classant ensuite les chaînes en sous-ensembles correspondant aux valeurs trouvées.

Tous ces opérateurs constituent ainsi des ensembles complets de prédicats disjonctifs et l'application d'un opérateur à un ensemble de chaînes réalise une partition de l'ensemble de départ (14).

Au sein de chacun des sous-ensembles ainsi créés, il est possible d'effectuer une réduction de la longueur des chaînes soit que toutes les chaînes soient de longueur inférieure à la longueur initialement prévue, soit qu'elles contiennent toutes le même caractère en n-ime position.

Par exemple, si on applique à un ensemble quelconque un opérateur testant le deuxième caractère de chaque chaîne et définissant les quatre prédicats "A", "E", "I" et"tout autre caractère" on réalise une partition de l'ensemble de départ en quatre classes. La première contient toutes les chaînes dont le deuxième caractère est un A, la seconde toutes celles dont le deuxième caractère est un E, la troisième toutes celles dont le deuxième caractère est un I, la quatrième toutes les autres chaînes.

Pour les trois premières classes, le deuxième caractère est redondant et peut être supprimé ; pour la quatrième classe, le nombre de caractères pouvant se présenter en deuxième position se trouve réduit de trois et il est donc possible de réduire le nombre moyen de bits nécessaire pour représenter ce caractère.

. La détermination de l'opérateur réalisant la meilleure réduction s'effectuera en calculant, pour tous les opérateurs de l'ensemble de départ, le gain de mémoire réalisable c'est-à-dire la différence entre le volume occupé par l'ancienne représentation et le volume occupé par l'opérateur et les sous-ensembles qu'il définit.

La valeur du gain réalisable dépend donc des conventions de représentation utilisées pour représenter ensembles de chaînes et opérateurs ainsi que des réductions de longueur effectivement réalisées au sein des sous-ensembles créés. On pourrait toutefois échapper à toutes ces contraintes en définissant un gain vrai tenant compte des néguentropies au lieu des volumes occupés.

Si on obtient un ou plusieurs opérateurs réalisant un gain positif, on choisira celui correspondant au gain maximal.

S'il n'existe aucun opérateur conduisant à un gain positif, on dira que l'ensemble de départ est irréductible et l'on arrêtera le processus d'itération. Toutes les feuilles de l'arborescence finale représenteront donc des sous-ensembles irréductibles.

On est ainsi arrivé à une représentation qu'il n'est plus possible d'améliorer sans faire appel à d'autres techniques. Cette représentation peut être interprétée comme un algorithme de reconnaissance : il suffit de parcourir l'arborescence depuis la racine, en passant par les arcs correspondant aux prédicats vérifiés par la chaîne à tester et d'effectuer en même temps les réductions correspondantes. Il arrivera un moment où l'on atteindra une feuille et il suffira alors de comparer la forme réduite de la chaîne aux valeurs contenues dans le sous-ensemble représenté par la feuille.

Pour que la représentation permette effectivement d'énumérer les chaînes de l'ensemble représenté, il faut et il suffit que toutes les réductions effectuées soient réversibles. Il est alors possible de reconstituer la chaîne originelle à partir de la représentation réduite et des prédicats vérifiés.

Remarquons que nous avons ainsi construit quelque chose ressemblant d'assez prêt à des règles de réécriture quoique la structure arborescente donnée à la représentation masque un peu cet aspect.

7.2 FORMALISATION DE L'ALGORITHME.

Soit $S = \{ s_1, \ldots, s_n \}$ un alphabet fini. Soit S* le monoïde engendré par S et S' une restriction de S* aux chaînes de longueur finie.

Soit $P = \{ p_1, \cdots, p_n \}$ un ensemble fini de prédicats. A chaque $p_i \in P$ on associera une fonction biunivoque

$$f_i : S' \to S'$$

telle que pour toute chaîne $c \in S'$, la longueur de $f_i(c)$ soit inférieure ou égale à la longueur de c. On appellera ces f_i fonctions de réduction.

Soit $E \subset S'$ un ensemble de chaînes tel que $\mathrm{card} E < \infty$ et soit m la longueur maximum des chaînes de E.

Etant donné E, il existe un ensemble $\mathcal{O}_E = \{ O_1, O_2, \ldots \}$ d'opérateurs, avec $\mathrm{card}\, \mathcal{O}_E < \infty$ où chaque O_i est défini de la façon suivante

$$O_i = \{ (P_{i1}, E_{i1}), \cdots, (P_{i l_i}, E_{i l_i}) \} \quad 1 \leq l_i \leq k$$

et

$$P_{ij}(c) = \begin{cases} = \text{vrai} & \forall\, c \in E_{ij} \\ = \text{faux} & \forall\, c \notin E_{ij} \end{cases}$$

L'ensemble $\{ P_{i1}, \ldots, P_{i l_i} \}$ est un ensemble complet de prédicats c'est-à-dire que pour tout $c \in E$ il existe un et un seul P_{ij} tel que $P_{ij}(c)$ soit vrai et l'ensemble des sous-ensembles E_{ij} $\{ E_{i1}, \ldots, E_{i l_i} \}$ forme lui-même une partition de E c'est-à-dire que $E = E_{i1} \cap \ldots \cap E_{i l_i}$ et $\forall\, m \neq n \quad E_{im} \cap E_{in} = \phi$

Soit G une fonction $G : \mathcal{P}(S') \times \mathcal{O} \to \mathbb{Z}$ qu'on appellera fonction gain et qui sera définie pour tout $E \in \mathcal{P}(S')$ et pour tout $O_i \in \mathcal{O}$ comme

$$G(E, O_i) = Vol(E) - \left[Vol(O_i) - \sum_{j=1}^{l_i} Vol\left(f_{ij}(E_{ij}) \right) \right]$$

où Vol (D) détermine le volume de mémoire nécessaire pour représenter l'ensemble D.

L'algorithme opère de la façon suivante.

On aura comme données un ensembles de chaînes E et son alphabet A.

(1) A partir de l'ensemble des prédicats P on construit l'ensemble des opérateurs $\mathcal{O}_E$.

(2) Pour chaque $O_i \in \mathcal{O}_E$ on calculera la fonction gain $G(E, O_i)$ et on choisit l'opérateur O_i correspondant au gain positif maximal.

(3) Si $G(E, O_i) \leq 0 \quad \forall O_i \in \mathcal{O}_E$ on dira que l'ensemble est irréductible et on ne modifie pas cet ensemble. Autrement on applique à chaque E_{ij} de l'ensemble $\{E_{i1}, \ldots, E_{il_i}\}$ défini par l'opérateur O_i la fonction de réduction f_{ij} correspondant au prédicat P_{ij}.

On obtient ainsi un nouvel ensemble $\{E'_{i1}, \ldots, E'_{il_i}\}$ où chaque $E'_{ij} = f_{ij}(E_{ij})$ et où $\sum_{\sigma} Vol(E'_{ij}) \leq \sum_{\sigma} Vol(E_{ij})$ et l'on range à la place de E la liste de prédicats $(P_{i1}, \ldots, P_{il_i})$ en associant à chaque prédicat P_{ij} l'ensemble E'_{ij} correspondant.

(4) Pour chaque E'_{ij} de $\{E'_{i1}, \ldots, E_{il_i}\}$ on réitère l'algorithme en posant $E = E'_{ij}$. L'algorithme s'arrêtera lorsqu'il ne sera plus possible d'effectuer aucune réduction.

7.3 DESCRIPTION DU PROGRAMME.

A partir de l'algorithme d'inférence de représentations arborescentes, nous avons écrit un programme d'inférence effectuant automatiquement la représentation arborescente.

Le but poursuivi était de tester les performance de l'algorithme sur de petits fichiers comportant des chaînes de longueur moyenne. Compte tenu de la nature même de la représentation, les résultats ne peuvent être que meilleurs pour de gros fichiers.

Par rapport aux autres programmes d'inférence existants (4) (9), notre programme offre l'intérêt de manipuler des ensembles de données réalistes : listes de noms propres, de mots réservés d'un langage, d'expressions arthmétiques. Nous reviendrons sur cette question lors de la discussion des résultats.

Le programme comporte certaines restrictions par rapport à l'algorithme. Nous n'avons considéré que deux types d'opérateurs : celui testant la longueur effective des chaînes et celui testant la valeur de n-ième caractère de chaque chaîne. Nous nous sommes limités aux opérateurs ne comportant pas plus de quatre prédicats.

Ecrit en FORTRAN le programme comporte un segment directeur (LISTER) qui appelle successivement les modules correspondant aux différentes fonctions (LIRE, FORMER qui appelle lui-même CHOIX, OUPH).

Avant de reprendre les différentes fonctions exercées par ces modules, nous allons d'abord décrire l'implantation en mémoire des représentations arborescentes.

Implantation en mémoire des représentations arborescentes.

Les représentations arborescentes sont toutes implantées dans une seule et même table appelée TABLE. Chaque cellule de la table peut contenir tour à tour :

soit un caractère,
soit une longueur de chaîne ou une position dans la chaîne,
soit un nombre de chaînes,
soit un type d'opérateur,
soit un pointeur vers une adresse de la table.

Les arborescences sont gérées dynamiquement et c'est le contexte qui définit le type d'information contenue dans une cellule donnée. Par exemple, la première cellule de TABLE contiendra toujours le type d'opérateur affecté à la racine de l'arborescence.

Deux structures coexistent au sein de la table : les listes et les opérateurs. Les listes correspondant aux feuilles de l'arborescence, les opérateurs aux noeuds.

Une liste contient la représentation énumérative d'un ensemble de chaînes ; elle comprend un en-tête obéissant à un format fixe et un corps essentiellement variable.

L'en-tête comporte trois cellules :
la première est toujours à zéro et annonce la structure,
la deuxième contient la longueur maximale des chaînes (LMAX),
la troisième contient le nombre de chaînes de la liste (IX).

0	LMAX	IX

Fig. 1 = En-tête de liste.

Le corps de la liste comprend LMAXxIX cellules contenant les représentations des chaînes sous format fixe

0	1	26	A	B	C	...	Y	Z

Fig. 2 : Liste représentant les vingt-six lettres de l'alphabet.

Un opérateur contient la représentation d'un ensemble complet de prédicats. A chaque prédicat correspond un arc de l'arborescence et donc une sortie.

Selon le nombre de prédicats l'opérateur comportera donc de six à dix cellules.

La première cellule spécifie le nombre de sorties (NSORT) c'est-à-dire le nombre de prédicats.

La deuxième détermine le type de l'opérateur : si son contenu est nul, l'opérateur teste la longueur, sinon il s'agit d'un opérateur testant la valeur d'un caractère et la cellule indique alors la position à tester.

Les autres cellules spécifient les prédicats (valeur du caractère ou longueur maximale de la chaîne) et les sorties associées

4	2	A	•	E	•	I	•	@	•

Fig. 3 : Opérateur testant le second caractère de chaque chaîne les quatre sorties correspondent aux valeurs A, E, I et à toutes les autres lettres.

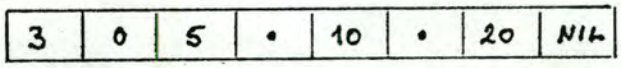


Fig. 4 : Opérateur testant la longueur des chaînes d'un ensemble déclaré de longueur maximale 20 et ne comportant pas de chaînes de longueur supérieure à 10 ; les trois sorties correspondent à $l \leq 5$, $5 \leq l \leq 10$ et $10 \leq l \leq 20$ et la dernière sortie pointe vers une liste vide.

S'il s'agit d'un opérateur testant la valeur d'un caractère, le prédicat sera représenté par le caractère sélectionné ou par un caractère spécial (@ p.ex.) pour spécifier toutes les lettres non encore sélectionnées.

S'il s'agit d'un opérateur testant la longueur des chaînes, on représentera toujours la longueur maximale des chaînes sélectionnées. Comme la sélection s'effectue toujours de gauche à droite, il n'y a pas d'ambiguité.

Lorsqu'il n'y a pas de sorties correspondant au prédicat, le pointeur est mis à zéro (NIL).

Description des modules :

LISTER :

LISTER initialise la table puis appelle successivement LIRE et FORMER. Il imprime ensuite la représentation optimale et appelle OUPH.

LIRE :

LIRE lit les données et effectue le passage de la représentation externe à la représentation interne : il remplace notamment les blancs finaux par un caractère spécial dit caractère de remplissage.

Les chaînes sont immédiatement rangées dans la table et forment une liste c'est-à-dire que TABLE (1) est à zéro, TABLE (2) contient la longueur maximale des chaînes, ...

LIRE calcule enfin le nombre de bits nécessaires pour représenter l'ensemble de chaînes par la formule

NBITS = LMAX x IX x lbM

où M est le nombre de symboles de l'alphabet, blanc compris,
LMAX la longueur maximale des chaînes,
et IX le nombre de chaînes.

FORMER :

FORMER construit l'arborescence d'opérateurs conduisant à la représentation optimale en faisant appel à CHOIX lors de chaque itération pour déterminer l'opérateur optimal.

Pour parcourir l'ensemble des feuilles à traiter, FORMER se sert de deux piles d'adresses utilisées en bascule : LIST (FROM) contient les adresses des listes à traiter et LIST (TO) les adresses des listes créées à chaque itération.

Au départ LIST (TO) est vide et LIST (FROM) ne contient que l'adresse de la première cellule. FORMER va donc accéder ainsi à l'en-tête de la liste contenant les éléments de l'ensemble à traiter et placer ses chaînes dans une matrice commune à FORMER et à CHOIX avant d'appeler ce module.

Au sortir de CHOIX, deux cas peuvent se présenter :

Si aucun opérateur ne réalise un gain de mémoire, la liste est dite irréductible ; on ne modifie pas sa représentation et on ote son adresse de la pile LIST (FROM).

Si CHOIX a pu sélectionner un opérateur réalisant un gain de mémoire, on effectuera le changement de représentation sur place.

L'opérateur transmis par CHOIX a la même structure que les opérateurs de l'arborescence mais les pointeurs sont remplacés par les nombres des chaînes vérifiant chaque prédicat.

Connaissent les fonctions de réduction associées à chaque prédicat, il est donc possible de calculer le volume occupé par les listes correspondant aux différentes sorties. On va ensuite remplacer l'en-tête de la liste traitée par l'opérateur sélectionné, calculer les adresses de début des listes correspondant à chaque sortie et construire leur en-tête.

Il ne restera plus qu'à chercher le prédicat vérifié par chaque chaîne de la matrice commune à FORMER et à CHOIX, à effectuer la réduction correspondante et à ranger la chaîne réduite dans la liste vers laquelle pointe le prédicat.

Si le calcul des adresses de début de liste est correct, la nouvelle structure aura une longueur inférieure à l'ancienne représentation et cette différence représente le gain réalisé.

L'adresse de la liste traitée est maintenant l'adresse d'un opérateur ; on va l'ôter de la pile LIST (FROM) et ranger dans LIST (TO) les adresses des listes créées.

On traitera ainsi successivement toutes les listes dont l'adresse figure dans LIST (FROM). Si la pile est vide, on permutera LIST (FROM) et LIST (TO). Si les deux piles sont toutes deux vides, c'est qu'on est arrivé à la représentation optimale.

CHOIX :

CHOIX construit d'abord une matrice M (I , J) (position, caractère) et range dans chaque M (I, J) le nombre de chaînes de l'ensemble qui ont le caractère J en n-ième position. En même temps, il construit un vecteur V (K) de longueur égale à la longueur maximale des chaînes et rang dans chaque V (K) le nombre de chaînes de longueur inférieure ou égale à K.

Pour chaque position i, on recherche dans M les r caractères qui se répètent le plus souvent et on calcule le gain réalisable par l'opérateur sélectionnant ces r caractères. Cet opérateur comporte r + 1 sortie, les r premières correspondant aux r lettres et la dernière à toutes les autres. Chaque fois qu'une chaîne contient une des r lettres en position i, une réduction de un caractère est possible. Le gain s'obtient donc en soustrayant le volume occupé par l'opérateur du nombre de chaînes contenant un de ces caractères en i-ème position.

Si $\mathcal{R} = \{J_1, \ldots, J_r\}$ désigne les r lettres les plus fréquentes et V_0 le volume occupé par l'opérateur, le gain vaudra alors

$$G = \sum_{J \in \mathcal{R}} M(I,J) - V_0 .$$

Parmi toutes les lignes de la matrice M, on choisit celle pour laquelle le gain est maximum. Si ce maximum est inférieur ou égal à zéro, on ne sélectionne aucun opérateur, sinon, on sélectionne l'opérateur correspondant à ce maximum.

On effectue un calcul analogue pour déterminer le gain réalisable par le meilleur opérateur testant la longueur. Pour toutes les suites de r éléments de K, on calcule le volume occupé par l'opérateur et les réductions correspondant à chaque sortie de l'opérateur sélectionnant successivement les éléments de longueur inférieure à $J_1, \ldots, J_r, K$. Il vient ainsi pour le gain réalisable

$$\begin{aligned} G = {} & (K-J_1)\, V(J_1) + (K-J_2)\,(V(J_2) - V(J_1)) \\ & + \ldots + (K-J_r)(V(J_r) - V(J_{r-1})) . \end{aligned}$$

On obient ainsi l'opérateur du type longueur correspondant au gain maximal. Si ce gain est négatif ou nul, on conserve l'opérateur précédemment sélectionné ; sinon, on compare les gains réalisables et on choisit le meilleur.

Si aucun opérateur n'a pu être sélectionné, on dira que l'ensemble de départ est irréductible car quel que soit l'opérateur appliqué la nouvelle représentation occuperait plus de place que l'ancienne.

OUPH :

OUPH effectue le passage de la représentation construite par FORMER à une représentation plus compacte et moins redondante.

A cet effet, on supprimera quelques éléments manifestement redondants : les pointeurs, les sorties correspondant à des classes vides et la deuxième cellule des opérateurs de type longueur (ce qui conduit à modifier alors le contenu de la première cellule).

A chaque type de cellule, on affecte un entier, appelé NCODE, qui correspond au nombre de valeurs possibles et on effectue ensuite un codage polynomial.

Ce codage remplace les valeurs contenues dans c cellules, soit $V_1, V_2, \ldots, V_c$, par la valeur unique

$$V_1 + NCODE_1 (V_2 + NCODE_2 (\ldots (V_c) \ldots)$$

Comme cette valeur devient rapidement supérieure à la capacité d'une cellule de la machine, OUPH comporte un module appelé STORER qui tronçonne automatiquement la valeur obtenue.

A partir du module OUPH, il nous est possible de calculer une valeur du gain ne dépendant que fort peu des conventions de représentation adoptées dans TABLE.

7.4. DISCUSSION DES RESULTATS.

Le programme a été testé sur trois ensembles de données qui ont permis de mettre en évidence des gains de mémoire bien supérieurs à 50 voire 60 % (cfr le tableau).

Ces trois ensembles se présentent sous la forme de suite de chaînes de caractères justifiés à gauche. La longueur maximale des chaînes au sein de chaque ensemble fait partie des données et les blancs terminaux sont reconnus comme des caractères de remplissage.

Le premier ensemble est aussi le plus petit et comprend les noms de 89 académiciens membres de l'Académie Française et de l'Académie des Inscriptions et Belles-Lettres.

Le deuxième contient les 278 mots réservés du COBOL CDC 3600.

Les deux premiers ensembles présentent une forte dispersion de la longueur des chaînes. Au début de l'arborescence, ce seront donc des opérateurs restant la longueur des chaînes qui seront sélectionnés. Par après, lorsqu'on aura construit des classes de longueur homogène, les opérateurs testant un caractère feront leur

TABLEAU COMPARATIF DES GAINS REALISES

Ensemble	Académique	COBOL	Expr. arithmétiques
Nbre de chaînes	89	278	256
Longueur maximale	22	21	10
Nbre de symboles de l'alphabet (blanc compris)	28	48	9
Nbre total de caractères	1.958	5.838	2.560
Nbre de bits nécessaires pour une représentation énumérative	9.412	32.605	8.115
Nbre initial de cellules	1.961	5.841	2.563
Nbre final de cellules	790	2.094	870
Gain en cellules	1.171	3.747	2.093
%	59 %	64 %	66 %
Nbre de bits nécessaires pour coder la repres. optimale	3.762	11.552	2.528
Gain en bits			
%	60 %	64,5 %	68,8 %

apparition pour peu que ces classes soient d'une taille suffisante, ce qui est le cas pour le deuxième ensemble mais pas pour le premier.

Tout au contraire, l'ensemble d'expressions arithmétiques présente lui une remarquable homogénéité de la longueur des chaînes et l'arborescence optimale ne comporte que fort peu d'opérateurs testant la longueur.

La taille des classes irréductibles dépend bien sûr de la nature des chaînes présentes et des opérateurs sélectionnés : elle ne dépasse jamais 35 chaînes mais est le plus souvent de l'ordre d'une dizaine de chaînes.

L'algorithme calcule les gains en cellules mais il nous est aussi possible de les calculer en nombre de bits en utilisant le codage polymonial produit par le module OUPH. Les gains exprimés en nombre de bits seront souvent proportionnellement plus élevés compte tenu de la suppression de certains éléments redondants de l'arborescence.

On remarquera que la représentation arborescente accélère le processus de recherche d'un élément au sein de l'ensemble. En effet il faut en moyenne $N/2$ accès pour trouver un élément parmi N et dans le cas d'une représentation arborescente , il suffit d'explorer un seul chemin du graphe et d'examiner ensuite les éléments d'une seule classe terminale.

Convenons d'affecter le niveau zéro à la racine de l'arborescence. Alors, si h_i est le niveau de la classe terminale i et C_i le nombre d'éléments de cette classe

$\bar{n} = \frac{\sum n_i c_i}{\sum c_i}$ est le nombre moyen de noeuds à parcourir

$\frac{\bar{c}}{2} = \frac{1}{2} \frac{\sum c_i^2}{\sum c_i}$ est le nombre moyen d'éléments à examiner.

Le nombre moyen d'accès sera donc $\bar{n} + \frac{\bar{c}}{2}$ au lieu de $N/2$.

Pour le premier ensemble N = 89 et pour sa représentation optimale $\bar{n} = 3,93$ et $\bar{c} = 13,92$. Le nombre moyen d'accès se trouve donc réduit de 44,5 à 10,9.

On peut extrapoler ce résultat en supposant que toutes les classes terminales aient la même taille c, qu'elles occupent toutes un même niveau et que tous les opérateurs aient le même nombre de sorties ns.

Dans ce cas, il y a $\frac{N}{c}$ classes terminales et pour accéder à une de ces classes, il faut parcourir $\log_{ns} \frac{N}{c}$ opérateurs. Le nombre moyen d'accès est alors

$$\frac{c}{2} + \log_{ns} \frac{N}{c}$$

Comme c ne dépend pas de N, on voit que le nombre moyen d'accès est de l'ordre de $\log N$ et non de l'ordre de N comme dans le cas d'une représentation énumérative.

En pratique, l'arborescence optimale est loin d'être aussi symétrique que nous l'avions supposée et le nombre moyen d'accès sera donc plus élevé.

Les résultats obtenus tant en temps d'accès qu'en gain mémoire restent encore améliorables. Tout d'abord, en appliquant l'algorithme à des ensembles plus importants que les trois exemples.

Si le nombre d'éléments de l'ensemble augmente, l'application du même opérateur conduit à des gains plus importants en valeur absolue mais aussi en pourcentage compte tenu du fait que le volume occupé par l'opérateur reste lui constant. On a vu, en outre, que l'algorithme de réduction s'arrêtait lorsque les classes à réduire devenaient trop petites. Si la taille de l'ensemble augmente, les

tailles de toutes les classes de l'arborescence augmentent elles aussi et il devient possible de pousser plus loin les réductions, ce qui permettra de nouveaux gains.

D'autres améliorations des performances de l'algorithme sont possible. On pourrait, par exemple, améliorer le gain en limitant moins strictement le nombre de prédicats constituant un opérateur ce qui éviterait, dans certains cas, des cascades d'opérateurs du même type.

On pourrait aussi introduire de nouveaux types d'opérateurs testant les valeurs de sous-chaînes et non plus de caractères isolés, remplaçant les digrammes les plus fréquents par de nouveaux symboles, ... Il faudrait alors établir des heuristiques de choix de l'opérateur optimal adaptées au grand nombre d'opérateurs possibles, toute procédure énumérative devenant trop longue. Il deviendrait en outre assez difficile de sélectionner les prédicats constituant l'ensemble de départ à partir duquel tous les opérateurs sont construits sans y mettre une certaine part d'arbitraire.

C O N C L U S I O N

Au cours de notre étude, nous avons été successivement amené à poser le problème de l'optimisation du volume mémoire occupé par la représentation de tables de valeurs puis à examiner un certain nombre de techniques de réduction du volume mémoire occupé.

Si ces techniques étaient fort différentes tant par leurs approches du problème que par leurs degrés de raffinement, presque toutes concernaient les tables de valeurs dont les éléments s'expriment sous la forme de chaînes de caractères.

Aussi,est-il tout naturel qu'arrivé à la conclusion de notre étude, nous nous posions quelques question sur leurs mérites comparés et que nous tachions ensuite de les classifier selon la qualité de leurs performances.

Il s'agit là cependant d'une opération bien délicate, et cela à plus d'un titre. Tout d'abord, n'oublions pas que les performances d'une technique d'optimisation sont toujours relatives aux données auxquelles elle s'applique. On ne peut comparer deux techniques que relativement à un même type de données.

Ensuite, les techniques que nous avons étudiées ne sont pas toutes du même degré de complexité. Nous avons vu que pour arriver à la représentation de longueur minimale, il fallait être capable de prendre en charge toutes les informations que nous possédions sur le contenu de la table : en général, les algorithmes les plus performants seront aussi les plus compliqués et les plus lents.

C'est pourquoi, nous pensons qu'il serait vain de vouloir définir, une fois pour toutes, "la"meilleure technique de réduction du volume mémoire. Au contraire, il faut considérer que nous sommes en présence d'un certain nombre de techniques plus ou moins performantes et plus ou moins faciles à mettre en oeuvre, ayant chacunes leurs domaines d'application.

Ainsi, la préférence donnée à une de ces techniques plutôt qu'à une autre dans une application pratique, dépendra donc à la fois des caractéristiques de l'application considérée et d'une rapide analyse coût-efficacité.

Nous disposons en effet d'un certain nombre de techniques fort simples, telles la compression des caractères de remplissage ou le codage polynomial, qui permettent déjà d'importants gains de mémoire. Ainsi dans le cas de textes purement alphabétiques, le passage de la représentation sur huit bits au code polynomial permet un gain de 40,6 % alors que le passage à un code HUFFMAN ne produirait qu'un gain supplémentaire de 7,9 %. (La comparaison devient moins favorable au code polynomial lorsque la taille de l'alphabet augmente).

Ces techniques et les améliorations qui en découlent, telles la méthode de HAHN, souffrent toutrefois d'un défaut, la lenteur des opérations de codage et de décodage nécessitant autant de multiplications ou de divisions qu'il y a de caractères présents dans la chaîne.

Aussi, pour des ensembles de valeurs très souvent consultées mais fort peu susceptibles d'être modifiées, ces techniques se feront concurrencer par les techniques d'optimisation globale qui sont très lourdes à mettre en oeuvre mais peuvent parfois conduire à des représentations fort élégantes (cfr. PUFFT et PL/C).

Voilà pour ce qui est des méthodes existantes. Notre revue des techniques d'optimisation serait toutefois incomplète si nous n'envisageons pas les améliorations qui seraient susceptibles d'être apportées à ces méthodes.

Dans le domaine des techniques d'optimisation locale, il serait intéressant de développer des codes mieux adaptés à l'architecture des machines, tels des codes utilisant comme base les quatre bits du format décimal condensé. Une évaluation préliminaire basée sur (12) nous permet d'envisager des longueurs moyennes de l'ordre

de 4,45 bits par caractère pour des textes anglais entièrement alphabétiques

C'est cependant dans le domaine des techniques d'optimisation globale que les plus grands progrès sont encore à faire. En particulier, il serait très intéressant de disposer de procédures de sélection des séquences de caractères les plus fréquentes dans des ensembles de chaînes de caractères. Ces procédures permettraient d'étendre considérablement le champ d'application des techniques d'optimisation globale : le système fonctionnerait en fait comme un système de recherche automatique d'abréviations et permettrait de généraliser l'emploi des techniques d'optimisation globale à tous les sensembles de textes possédant une structure de phrase avec des répétitions des mêmes séquences de caractères.

D'autres extensions des techniques d'optimisation globale sont encore possibles ; elles supposeraient cependant que nous disposions, à côté des grammaires formelles, d'autres outils nous permettant de décrire les structures des ensembles de chaînes de caractères.

B I B L I O G R A P H I E

(1) BRIAN-CONNELL J.
A HUFFMAN - SHANNON - FANO code.
Proc. IEEE, vol. 61, n° 7 (July 1973) pp. 1046-1047

(2) CHEN T.C., HO I.C.
Storage Efficient Représentation of Decimal Data
Comm. ACM, vol. 18, n° 1 (Jan 1975) pp. 49-52.

(3) CODASYL DATA BASE TASK GROUP REPORT.
ACM HQ, New York - April 1971.

(4) CRESPI - REGHIZZI S.
An Effective Model for Grammar Inference.
Information Processing 71, North Holland (1972) pp. 524-529.

(5) CRESPI - REGHIZZI S., MELKANOFF M.A., LICHTEN L.A.
The Use of Grammatical Inference for Designing Programming Language.
Comm. ACM, vol. 16, n° 2 (February 1973) pp. 83-90.

(6) DECSYSTEM 10 Assembly Language Handbook.
Third Edition.
DEC, Maynard 1973.

(7) DE NEVE J.
SESAM VERSION 8
SIEMENS, doc. interne 19 janvier 1973.

(8) FAURE R., KAUFMANN A., DENIS-PAPIN M.
Mathématiques Nouvelles Vol. 1.
Aide-Mémoire Dunod, Dunod 1970.

(9) FELDMAN J.A., GIPS J., HORNING J.J. - REDER S.
Grammatical Complexity and Inference.
CS 125, Computer Science Dpt, Stanford U. 1969.

(10) FELDMAN J.A., SHELDS R.C.
Total Complexity and the Inference of Best Programs
AIM-159, Stanfond Artificial Intelligence Projet, Stanford U. 1972

(11) FELDMAN J.A.
Some Decidability Results on Grammatical Inference and Complexity.
Information and Control, 20 (1972) pp. 244-262.

(12) GARNER H.L.
Information Theory and Codes
in KLERER M., KORN G.A. ed.
Digital Computer User's Handbook
Mc Graw Hill, New York 1967.

(13) GOULD I.H.
IFIP Guide to concepts and Terms in Data Processing
North Holland, Amsterdam. 1971.

(14) GUIHO G.
Gain théorique de mémoire obtenu à l'aide d'un algorithme de prétraitement.
C.R. Acad. Sci. Paris, t. 276 (3 janvier 1973) série A, pp. 65-68.

(15) GUIHO G.
Organisations de Mémoire.
Influence d'une structure et Etude d'Optimisation.
Thèse d'Etat, Institut de Programmation, U. de Paris VI. 1973.

(16) HAHN B.
A New Technique for Compression and Storage of Data
Comm. ACM, vol. 17, n° 8 (Aug. 1974) pp. 434-436.

(17) HORNING J.J.
A Procedure for Grammatical Inference.
Information Processing 71, North Holland (1972) pp. 519-523.

(18) HORNING J.J.
A Study of Grammatical Inference.
CS-139, Computer Science Dpt, Stanford U. 1969.

(19) HUFFMAN D.A.,
A Method for the Construction of Minimum - Redundancy Codes
Proc. IRE, vol. 40 (Sept. 1952) pp. 1098-1111.

(20) INTEGRATED DATA STORE.
(Ref. 00 11 115 A).
Bull General Electric, s.d.

(21) KNUTH D.E.
The Art of Computer Programming.
Vol. III : Sorting and Searching.
Addison - Wesley 1973.

(22) LOUIS-GAVET G.
Etude mathématique pour la concentration de fichiers occupant un volume important.
1ère partie: RIRO , 5 (1971), R-3, pp. 101-111
2ème partie: RAIRO, 6 (1972), R-1, pp. 71-80.

(23) PARIS J.F.
Le problème de la représentation des tables de valeurs : Apport des Techniques d'inférences.
Résumé introductif à un séminaire.
Institut d'Informatique, Facultés U. de Namur, Avril 1973.

(24) PARIS J.F.
An algorithm leading to minimal representations for unordered tables of values.
A paraître dans Revue Belge de Statistiques d'Informatique et de Recherche Opérationnelle.

(25) ROSEN S., SPURGEON R.A., DONNELLY J.K.
PUFFT - The University Fast Fontron Translator in ROSEN s. ed. Programming Systems and Languages.
Mc Graw Hill, New York 1967.

(26) SIMON J.C.
Traitement du Signal discret - La représentation discrète
Institut de Programmation, U. de Paris VI. s.d.

(27) SIMON J.C.
Représentation et Traitement de certaines structures de données.
Institut de Programmation, U. de Paris VI. 1973.

(28) WAGNER R.A.
Common Phrases and Minimum - Space Text Storage.
Comm. ACM, vol. 16, n° 3 (March 1973) pp. 148-152.

TABLE DES MATIERES

CHAPITRE VII : Un algorithme d'inférence de représentations arborescentes.