Ecole Doctorale Informatique et Information
pour la Societé

DEA Extraction de Connaissances
à partir des Données

Mémoire rédigé dans le cadre du DEA ECD

Utilisation des Index Bitmap pour la Fouille de Données

Stage réalisé au sein du Laboratoire ERIC
(Equipe de Recherche
en Ingénierie des Connaissances)
Université Lyon 2
5 Avenue Pierre Mendès-France
69676 BRON Cedex - France

http://eric.univ-lyon2.fr

Cécile FAVRE
DEA ECD
cecile.favre@etu.univ-lyon2.fr

Encadrée par :
Fadila BENTAYEB
Laboratoire ERIC - Université Lyon 2
bentayeb@eric.univ-lyon2.fr

22 Septembre 2003

Remerciements

Ce stage de DEA a été réalisé au laboratoire ERIC (Equipe de Recherche en Ingénierie des Connaissances), et plus précisément au sein de l'équipe BDD (Base de Données Décisionnelles). Je tiens à remercier toutes les personnes qui ont contribué de près ou de loin à la réussite de ce travail.
Je tiens tout d'abord à remercier particulièrement Fadila BENTAYEB pour son encadrement, son suivi, sa disponibilité sans limite et ses précieux conseils tout au long de ce stage.
Merci aussi à Jean Hugues CHAUCHAT pour ses explications et Stéphane LALLICH pour ses conseils.
Je remercie également Omar BOUSSAID.
Merci à Cédric UDREA pour ses précieuses précisions.
Merci, de plus, à l'ensemble de l'équipe Base de Données Décisionnelles du laboratoire ERIC de Lyon 2, et plus globalemant à l'ensemble des membres du laboratoire ERIC.
Mes remerciements s'adressent aussi à Amandine DUFFOUX pour ses suggestions pertinentes et pleines de bon sens, et à Rahee GHURBHURN pour ses compétences linguistiques et ses conseils avisés.
J'adresse en outre tous mes remerciements aux rapporteurs.
Mes remerciements vont également aux membres du jury, messieurs H. Briand, Y. Kodratoff et D.A Zighed.

Résumé

Les bases de données qui font l'objet de l'extraction de connaissances sont de plus en plus volumineuses. Les algorithmes classiques de fouille de données peuvent alors être poussés à leurs limites. Intégrer les méthodes de fouille de données au cœur des Systèmes de Gestion de Bases de Données (SGBD) est une solution qui peut repousser ces limites. Il s'agit de faire de la fouille de données en utilisant uniquement les concepts bases de données qui sont fournis par le SGBD (table, vue, procédure stockée, index, ...). Dans ce mémoire, nous nous concentrons sur le concept d'index bitmap, technique d'optimisation d'accès utilisée dans les SGBD. D'une part, nous étudions l'utilisation d'une variante de ceux-ci pour aider à l'extraction d'itemsets fréquents, phase préalable à l'extraction des règles d'association. D'autre part, nous étudions une approche nouvelle qui permet la construction d'arbres de décision, au cœur du SGBD, en utilisant les index bitmap. Nous nous penchons plus particulièrement sur l'élaboration d'un algorithme permettant de construire un arbre de décision par la méthode ID3 ("Induction Decision Tree") avec les index bitmap sur des grandes bases de données, dans des temps de traitement acceptables. Les arbres de décision sont construits par des partitions successives, dont les sous-populations sont caractérisées par leur effectif. En considérant que la population d'apprentissage est constituée de l'ensemble des index bitmap portant sur chacun des attributs de la base d'apprentissage, l'utilisation des index bitmap est justifiée. En effet, les index bitmap sont munis de propriétés très intéressantes, que ce soit au niveau de leur stockage, de leurs possibilités en matière de comptage, ou encore des opérations "bit à bit". Nous utilisons donc l'opérateur logique 'AND' pour combiner les index bitmap. Nous effectuons ensuite les comptages qui permettent de trouver les effectifs des différentes sous-populations. Une implémentation sous le SGBD Oracle en PL/SQL permet de valider notre approche.
Mots-clés : Fouille de Données, Intégration, Bases de Données, Index Bitmap, Itemset Fréquent, Règle d'Association, ID3.

Abstract

Due to the increasing mass of data to be analysed, traditional data mining algorithms have reached their limit in dealing with large databases. One interesting way to overcome this limit, without having to use API or SQL extensions, is to integrate data mining methods to the DBMS. We want to use the tools provided with the DBMS such as tables, views, stored procedures, indexes etc... for data mining purposes. In this work we will consider the use of bitmap indexes as a data mining tool. Firstly we will study a variant of bitmap indexes as a tool for subset searching, which is necessary to extract frequent itemsets. The frequent itemsets found are then used for association rules extraction. Secondly we will propose an approach to construct a decision tree, within the DBMS, using bitmap indexes. Indeed bitmap indexes have many useful properties such as the count operations and bite-wise operations. The objective is to use the AND operator with the bitmap indexes so as to be able to count the number of records satisfaying the defined population. The implementation is done under Oracle 8i after having definied an algorithm for the ID3 method.
Key words : Data Mining, Integration, Databases, Bitmap Indexes, Frequent Itemset, Association Rules, ID3.

Table des matières

Chapitre 1

Introduction

De nos jours, avec l'avènement d'Internet qui met à disposition une multitude de données, le problème se placerait davantage au niveau de l'interprétation de ces données qu'au niveau de leur récolte. Cette interprétation sous-entend une extraction d'informations. Nous n'avons pas a priori une bonne connaissance des données que nous cherchons à étudier, et la fouille de données doit donc permettre d'extraire des informations pertinentes.
De part l'augmentation des capacités de stockage et l'accroissement de la masse de données devant être exploitées, il s'agit d'extraire de l'information sur des bases de plus en plus volumineuses.
Il faut donc trouver des solutions appropriées pour répondre à cet objectif. En effet, les algorithmes traditionnels de fouille de données s'appliquent sur des tableaux de types attributs/valeurs [ZR00]. Ces tableaux sont extraits des bases de données aux moyens d'API ("Application Programming Interface"). Il y a donc un problème de limitation de la taille des bases de données pouvant être traitées, du fait même de la limitation de la taille de la mémoire centrale dans laquelle s'effectue l'exécution des algorithmes.
Pour faire de la fouille de données sur de grandes bases, une des voies de recherche consiste à intégrer les méthodes de fouille de données au sein des Systèmes de Gestion de Bases de Données (SGBD) [Cha98]. Cette piste de recherche est conjointement liée à l'avènement des entrepôts de données et de l'analyse en ligne (OLAP) plus particulièrement [Cod93]. En effet, OLAP est une première étape en matière d'intégration de processus d'analyse au sein même des SGBD. D'autre part, d'autres travaux de recherche ont concerné l'intégration du processus d'extraction de règles d'association dans les SGBD [MPC96, STA98].
Par ailleurs, nous pourrions penser que certains éditeurs de SGBD semblent aller dans le sens d'une intégration des méthodes de fouille de données dans les SGBD [IBM01, Ora01]. Cependant, cette intégration prend la forme d'une boîte noire dans laquelle la fouille de données est faite aux moyens d'API et d'extensions du langage SQL.
L'idée d'intégrer au sein des SGBD des processus de fouille de données est à l'origine du projet " Concepts Bases de Données pour la Fouille de Données " de l'équipe BDD du laboratoire ERIC. La démarche consiste à utiliser uniquement les concepts bases de données pour implémenter différentes méthodes de fouille de données dans un SGBD. La première piste explorée dans ce sens-là est l'implémentation de la méthode ID3 (Induction Decision Tree) au sein du

SGBD Oracle, en utilisant les vues relationnelles [BD02]. Cette méthode, qui a fait ses preuves du point de vue des résultats, puisque ceux-ci sont similaires aux résultats obtenus par des méthodes classiques, présente néanmoins un temps de traitement très long.
Rappelons que l'objectif poursuivi est d'offrir la possibilité de traiter de grandes masses de données de manière fiable, peu coûteuse, dans un temps raisonnable. Il s'agit, dès lors, d'optimiser le temps d'exécution des traitements. Un premier travail a été réalisé dans ce sens, en introduisant une phase de préparation des données. Le point clé réside dans la construction d'une table de contingence qui réduit considérablement la taille de la base d'apprentissage [UBDB03].
Dans ce mémoire, nous nous intéressons à un autre type d'optimisation. En effet, si nous parlons d'optimisation, dans un contexte purement bases de données, le concept d'indexation des données paraît inévitable. Dans [MZ98], les index bitmap ont été adaptés pour aider à la recherche d'itemsets fréquents, dans le cadre de l'extraction de règles d'association. Il s'agit de tirer pleinement profit des propriétés des index bitmap et entre autres, d'utiliser la possibilité d'appliquer les opérateurs 'AND' et 'OR' sur les bits de l'index.

Notre objectif, dans ce mémoire, est ainsi d'étudier l'utilisation des index bitmap pour faire de la fouille de données, en se plaçant au cœur des SGBD. Nous nous sommes intéressées à une variante des index bitmap qui aide à la recherche d'itemsets fréquents. Nous avons également élaboré une nouvelle approche qui consiste à utiliser les index bitmap pour la construction d'arbres de décision. Nous nous sommes ensuite penchées plus particulièrement sur la méthode ID3 [Qui86], avec une adaptation et une implémentation de l'algorithme. Nous avons choisi la méthode ID3 parce qu'il s'agit d'une méthode simple à implémenter et notre but est d'intégrer celle-ci au sein d'un SGBD, en se servant des index bitmap. L'utilisation des index bitmap est motivée par le fait que les effectifs des différentes sous-populations d'un arbre de décision peuvent être obtenus assez facilement, grâce aux index bitmap, en effectuant des opérations logiques avec 'AND' et des comptages. Il s'agit en effet de combiner les index bitmap des différents attributs, qu'ils soient prédictifs ou à prédire. Le fait de travailler avec les index et non pas sur les données brutes permet de réduire le temps de construction de l'arbre puisqu'il n'y a pas d'accès à la base. Il s'avère également qu'en utilisant les index bitmap, la fouille de données peut s'appliquer sur de grandes bases de données avec des temps de traitement acceptables puisque la méthode est intégrée au SGBD.

Le rapport est organisé de la manière suivante. Dans le chapitre 2, nous présentons les différentes approches de fouille de données sur des bases volumineuses. Nous introduisons ensuite les index bitmap dans le chapitre 3. Puis, nous nous penchons, dans le chapitre 4, sur une variante de ces index pouvant intervenir dans le processus d'extraction de règles d'association. Nous poursuivons avec le chapitre 5, dans lequel nous présentons notre approche sur l'utilisation des index bitmap pour construire des arbres de décision, nous proposons un algorithme permettant d'intégrer la méthode ID3 en utilisant ces index et donnons les caractéristiques de l'implémentation. Pour terminer, nous concluons et indiquons quelques perspectives dans le chapitre 6.

Chapitre 2

État de l'art

Il s'agit, dans cet état de l'art, de dresser un bilan sur les possibilités de faire de la fouille de données sur des grandes bases de données. Nous verrons tout d'abord comment, dans un cadre classique, l'échantillonnage peut-être nécessaire et intéressant. Nous aborderons ensuite ce qui est fait par les éditeurs de SGBD. Nous verrons enfin quels concepts bases de données ont pu être choisis pour faire de la fouille de données, tout d'abord dans l'extraction de règles d'association, puis dans la construction d'arbres de décision.

2.1 Applications classiques des méthodes de fouille de données

Les algorithmes de fouille de données fonctionnent sur des tableaux attributs/valeurs [ZR00]. Ils ne peuvent donc pas s'appliquer directement sur une base de données. Une extraction des données de la base vers de tels tableaux est alors nécessaire. Elle se fait au moyen d'API ("Application Programming Interface"). Les bases étant de plus en plus volumineuses, les algorithmes se heurtent au problème de l'insuffisance de la mémoire centrale dans laquelle les données sont traitées. Certaines bases de données faisant l'objet de la fouille de données sont tellement volumineuses que, même des algorithmes considérés comme rapides trouvent leurs limites. Une des solutions qui est envisagée est l'échantillonnage. Cette idée a été approfondie dans [Cha02]. Il est démontré qu'un apprentissage sur un bon échantillon de la population fournit des résultats quasiment aussi bons qu'un apprentissage sur l'ensemble de celle-ci.

Le choix de la méthode d'échantillonnage dépend de deux critères : le coût de la mise en œuvre de celui-ci et la précision des résultats obtenus. Le coût d'échantillonnage correspond au temps de calcul pour le tirage de l'échantillon d'une part, et d'autre part au temps consacré à l'extraction des connaissances sur cet échantillon. Dans le cas des arbres de décision, méthodes que nous allons aborder dans ce mémoire, le temps d'apprentissage peut diminuer fortement, sans dégrader de manière très significative la qualité de l'apprentissage. Pour construire un arbre de décision sur une base très volumineuse, la meilleure stratégie est de procéder à un ré-échantillonnage sur chaque nœud de l'arbre. Mais, en définitive, il n'est pas possible d'obtenir des meilleurs résultats en échantillonnant, qu'en conservant l'ensemble des données dont nous disposons.

Il est donc préférable de pouvoir apprendre sur la totalité de la population et pour cela, il faudrait avoir des outils capables de traiter de grandes bases de données. L'une des solutions consiste alors à intégrer les méthodes de fouille de données au sein des SGBD.

2.2 Outils intégrés des éditeurs de SGBD

Certains éditeurs de logiciels ont intégré la possibilité de faire de la fouille de données au sein de leur SGBD. Mais, il s'agit en fait d'une boîte noire pour l'utilisateur. Oracle a muni Oracle 9i d'une option "Data Mining" [Ora01]. Le tout fonctionne dans le SGBD grâce à une API, basée sur le langage Java. Deux algorithmes ont été implémentés. Il s'agit, dans le cadre de l'apprentissage supervisé, du Bayésien Naïf, pour faire de la reconnaissance de classes, et, dans le cadre de l'apprentissage non supervisé, de l'extraction de règles d'association. Quant à Microsoft, SQL Server 2000 Analysis Services [STY01] comporte deux modules : Microsoft Decision Trees et Microsoft Clustering. Le fonctionnement se fait au moyen d'API. Citons aussi IBM, pour lequel "DB2 Intelligent Miner Scoring" étend les possibilités d'une base de données et permet aux utilisateurs de déployer des analyses de fouille de données, en temps réel [IBM01]. Pour cela, des extensions de DB2 et d'Oracle sont utilisées. Les modèles générés sont stockés dans des bases de données relationnelles comme des objets XML. Différentes méthodes de fouille de données ont été implémentées, telles que les arbres de décision et les réseaux de neurones.

Ainsi, les éditeurs ont effectivement intégré des méthodes de fouille de données dans leur SGBD. Cependant, cette intégration prend la forme d'une boîte noire, dans laquelle ont été introduites des méthodes existantes, ce qui ne résout pas le problème de limitation des volumes de données pouvant être traités. En effet, ils n'utilisent pas les concepts bases de données fournis par leur SGBD. Néanmoins, si ce choix d'utilisation de ces concepts n'a pas été fait, il n'en demeure pas moins impossible. C'est ce que nous présentons dans ce qui suit.

2.3 Utilisation des index bitmap pour les règles d'association

Dans le cadre de l'utilisation de concepts bases de données pour faire de la fouille de données, nous nous sommes intéressées à l'utilisation des index bitmap pour l'extraction de règles d'association [MZ98]. Le problème de l'extraction de règles d'association a été formulé pour la première fois dans [AIS93]. Ce processus d'extraction peut se décomposer en deux étapes : (1) la recherche des itemsets fréquents et (2) à partir de ceux-ci, la construction des règles d'association. Chacune des deux étapes possède ses propres algorithmes de résolution. De nombreuses publications portent sur la première étape étant donné la complexité exponentielle du problème [Pas00]. En effet, si l'on considère un ensemble d'items de taille m, le nombre possible d'itemsets fréquents est 2^m. La génération des règles d'association à partir des itemsets fréquents, quant à elle, ne nécessite pas le balayage de la base de données. De plus, les temps de calcul pour cette génération sont faibles en comparaison avec le temps consacré à la recherche des itemsets fréquents. Le problème de l'optimisation du temps de réponse pour

l'extraction de règles d'association peut être réduit alors à l'optimisation de la phase de recherche des itemsets fréquents.

L'adaptation des index bitmap aux règles d'association proposée dans [MZ98] est une approche qui permet d'optimiser précisément cet aspect là. C'est une technique d'indexation, qui est une variante des index bitmap. Elle facilite la recherche des itemsets fréquents, étant donné qu'elle permet de retrouver, de manière efficace, les transactions qui contiennent un itemset donné, en utilisant les bitmaps, qui sont des tableaux de bits, et en y appliquant dessus des opérateurs tels que la conjonction 'AND'.

2.4 Intégration des méthodes de fouille de données dans les SGBD

Si nous parlons d'intégration des méthodes de fouille de données au cœur des SGBD, cela sous-entend l'utilisation de concepts bases de données pour faire de la fouille de données ; c'est à dire, ne se servir que des outils dont le SGBD est muni : tables, vues, procédures stockées, index ... Des travaux innovants en la matière ont permis l'implémentation de méthodes d'arbres d'induction telles que ID3, C4.5 et CART, en utilisant le concept de "vue relationnelle". L'idée a été développée pour la première fois dans [BD02], dans le cadre de la méthode ID3. Les arbres de décision, produits par ID3 par exemple, sont des outils d'apprentissage qui produisent des règles du type "si-alors" [ZR00]. Ils utilisent en entrée un ensemble d'objets (n-uplets) décrits par des variables (attributs). Chaque objet appartient à une classe, les classes étant mutuellement exclusives. Pour construire un arbre de décision, il est nécessaire de disposer d'une population d'apprentissage (table ou vue) constituée d'objets dont la classe est connue. Le processus d'apprentissage consiste ensuite à déterminer la classe d'un objet quelconque d'après la valeur de ses variables.
Les méthodes de construction d'arbre de décision segmentent la population d'apprentissage afin d'obtenir des groupes, au sein desquels l'effectif d'une classe est maximisé. Cette segmentation est ensuite réappliquée de façon récursive sur les partitions obtenues. La recherche de la meilleure partition lors de la segmentation d'un nœud revient à rechercher la variable la plus discriminante pour les classes. C'est ainsi que l'arbre (ou plus généralement le graphe) est constitué. Les règles de décision sont finalement obtenues en suivant les chemins partant de la racine de l'arbre (la population entière) jusqu'à ses feuilles (populations au sein desquelles une classe représente la majorité des objets). Dans l'approche [BD02], l'analogie entre un arbre de décision, comportant un ensemble de noeuds, et un arbre, composé de vues relationnelles, est développée. Ainsi, la racine d'un arbre de décision est représentée par une vue relationnelle qui correspond à la population d'apprentissage de départ. Comme chaque nœud de l'arbre de décision représente une sous-population de son nœud parent, à chaque nœud est associée une vue construite à partir de sa vue parente. Ces vues sont ensuite utilisées pour dénombrer les effectifs de chaque classe dans le nœud. Ces comptages servent finalement à déterminer le critère de partitionnement des noeuds en sous-partitions ou à conclure qu'un nœud est une feuille. Ils sont obtenus par des requêtes SQL utilisant la fonction COUNT() et la condition GROUP BY.

Un algorithme a donc été élaboré pour intégrer la méthode ID3 au cœur du SGBD. Il été implémenté sous forme de procédure stockée PL/SQL qui utilise uniquement le langage SQL et les vues relationnelles. Chaque nœud de l'arbre est donc matérialisé par une vue relationnelle. La structure de cet arbre est contenue dans une table résultat et celle-ci peut ainsi être retrouvée en exécutant une requête hiérarchique sur la table résultat. Cette méthode permet de traiter des bases sans limitation de taille. Cependant, elle présente des temps d'exécution élevés induits par la création des différentes vues relationnelles.

Un premier travail d'optimisation a été réalisé pour améliorer les temps d'exécution. En effet, dans [UBDB03], l'idée d'introduire une phase de préparation des données a été développée. Lors de cette phase de préparation, une table de contingence est créée. La table de contingence est une table des populations agrégées, c'est à dire une table contenant les comptages pré-calculés sur toutes les combinaisons possibles des modalités de toutes les variables. L'apprentissage se fait sur la table de contingence plutôt que sur la base elle-même, ce qui réduit de manière considérable la taille de la base d'apprentissage, donc le temps d'exécution. Celui-ci est également réduit parce que toutes les vues de l'arbre ne sont pas créées.

Chapitre 3

Index bitmap

3.1 Définition et algorithme

L'index bitmap constitue un type d'indexation qui est particulièrement intéressant et performant dans le cadre des requêtes de sélection. L'index bitmap est codé sur des bits, d'où son faible coût en terme d'espace occupé.
Plutôt que d'avoir un index sur un attribut qui regroupe les différentes valeurs prises par l'attribut tel que le "Value List Index", l'index bitmap considère toutes les valeurs possibles pour un attribut donné, que la valeur soit présente ou non dans la table. Pour chacune de ces valeurs, nous stockons un tableau de bits, appelé bitmap, qui contient autant de bits que de tuples présents dans la table. Pour le bitmap d'une valeur donnée d'un attribut, il y aura un codage d'absence/présence, 0 ou 1, à la position du bit correspondant à un tuple.
De cette façon, ce type d'indexation convient pour les attributs qui ont un faible nombre de modalités. Cependant, sa mise à jour étant assez coûteuse, cette indexation est alors moins performante pour des données souvent modifiées.

Nous présentons ici l'algorithme de construction d'un index bitmap :
Soit une table T = $\{t_1, .., t_n\}$ où t_j est un tuple de la table T, j = 1 à n
Soit A un attribut de la table T, noté T.A, et soit $\{a_1, .., a_m\}$ le domaine de A.
Un index bitmap sur T.A, noté B^A, est un ensemble de vecteurs bitmap $\{B_1, .., B_m\}$, tel que :

$\forall\ B_i$ (i = 1, .., m), t_j (j = 1, .., n)

Si $t_j.A = a_i$ *Alors*
  $B_i[j] = 1$
Sinon
  $B_i[j] = 0$
Fin Si

Où $B_i[j]$ est le jème bit du vecteur bitmap B_i.

3.2 Exemple

Considérons la base d'apprentissage du Titanic dans laquelle 2201 individus sont décrits par trois attributs prédictifs, en l'occurence les variables *Classe*, *Age*

et *Sexe* et une variable à prédire *Survivant*. Cette base d'apprentissage permet de déterminer si un individu aurait survécu ou non au naufrage du Titanic en fonction de sa classe dans le bateau, de son âge et de son sexe.

Voici donc les précisions concernant les quatre attributs nommés ci-dessus :

- *Classe* correspond à la classe de l'individu au sein du bateau.
  - La valeur 'Equipage' correspond à un membre de l'équipage,
  - La valeur '1ère' correspond à un individu voyageant en 1^{re} classe,
  - La valeur '2ème' correspond à un individu voyageant en 2^{me} classe,
  - La valeur '3ème' correspond à un individu voyageant en 3^{me} classe.
- *Age* est l'âge de l'individu et présente deux modalités 'Enfant' et 'Adulte'.
- *Sexe* correspond au sexe de l'individu et présente deux modalités 'Femme' et 'Homme'.
- *Survivant* exprime le fait d'avoir survécu ou non et est traduit par deux valeurs : 'Oui' qui correspond au fait d'avoir survécu et 'Non' qui est le fait que l'individu soit décédé lors du naufrage.

Par souci de clarté, nous ne considérons ici qu'un échantillon de cette base de données ; il comporte 12 tuples. Les données de l'échantillon sont les suivantes :

Classe	Age	Sexe	Survivant
1ère	Adulte	Femme	Oui
3ème	Adulte	Homme	Oui
2ème	Enfant	Homme	Oui
3ème	Adulte	Homme	Oui
1ère	Adulte	Femme	Oui
2ème	Adulte	Homme	Non
1ère	Adulte	Homme	Oui
Equipage	Adulte	Femme	Non
Equipage	Adulte	Femme	Oui
2ème	Adulte	Homme	Non
3ème	Adulte	Homme	Non
Equipage	Adulte	Homme	Non

Pour construire un index bitmap sur l'attribut *Classe*, par exemple, il faut au préalable considérer chaque modalité possible pour cet attribut, soient les valeurs 'Equipage', '1ère', '2ème' et '3ème'. Il y a donc un bitmap pour chacune de ces valeurs, en l'occurrence, quatre bitmaps. La longueur de chaque bitmap correspond au nombre de tuples de la base donc, dans notre exemple, chaque bitmap sera représenté par un tableau de 12 bits puisque la base contient 12 tuples. L'index bitmap construit sur l'attribut *Classe* comporte donc 4 bitmaps, chacun d'une longueur de 12 bits. En ligne, nous avons les valeurs distinctes que prend l'attribut sur lequel l'index bitmap est construit : 'Equipage', '1ère', '2ème' et '3ème'. En colonne, nous avons les douze tuples (le premier tuple étant représenté à droite du tableau). Un '1' à l'intersection d'un numéro de tuple et d'une valeur de l'attribut *Classe* signifie que cet individu fait partie de cette classe du bateau, un '0' indique le contraire.

Voici donc l'index bitmap de l'attribut *Classe* pour l'échantillon de la base

de données Titanic considéré :

N° tuple	12	11	10	9	8	7	6	5	4	3	2	1
Equipage	1	0	0	1	1	0	0	0	0	0	0	0
1ère classe	0	0	0	0	0	1	0	1	0	0	0	1
2ème classe	0	0	1	0	0	0	1	0	0	1	0	0
3ème classe	0	1	0	0	0	0	0	0	1	0	1	0

Nous pouvons lire l'index de la façon suivante :

- En ligne : les individus 8, 9 et 12 ont la valeur 'Equipage' pour l'attribut Classe (1ère ligne) donc ce sont des membres de l'équipage.
- En colonne : l'individu numéro 1 a la valeur '1ère' pour l'attribut Classe (dernière colonne) donc c'est un individu qui voyage en 1ère classe.

3.3 Différentes utilisations des index bitmap

Les index bitmap permettent une optimisation des performances, que ce soit dans les bases de données ou dans les entrepôts de données [VG99]. En effet, dans les deux cas, l'optimisation se fait non seulement au niveau des requêtes d'interrogation des données [CI98, Wu99], mais aussi au niveau de l'amélioration de l'exécution des jointures [OG95]. L'utilisation des index bitmap dans les bases de données s'est tout naturellement généralisée dans le cadre des entrepôts de données. Dans les bases de données de production, les requêtes de mise à jour (insertion, suppression et modification) sont nombreuses et nécessitent une mise à jour des index coûteuse. Les entrepôts de données ont, quant à eux, la particularité d'être plus "stables" et répondent donc bien à l'utilisation d'index bitmap. De plus, la volonté de faire de l'OLAP dans les entrepôts de données est une justification supplémentaire. Il est vrai que l'objectif de l'OLAP, comme son nom l'indique, est de fournir des réponses de manière quasi-instantanée. Les requêtes sont, dans ce cadre-là, essentiellement des requêtes de sélection. De part la volumétrie des entrepôts, il s'agit d'optimiser l'exécution de ces requêtes. Cela passe entre autres par l'utilisation d'index [OQ97]. Etant donné la modélisation des entrepôts de donnés en schéma en étoile, ces derniers nécessitent de nombreuses jointures pour répondre aux requêtes. L'opération de jointure étant très coûteuse, l'utilisation des index bitmap de jointure peut être un bon outil d'optimisation [OG95]. Nous allons tout d'abord nous pencher sur l'optimisation des jointures. Nous aborderons ensuite l'optimisation de la recherche sur les données, que nous examinerons en détail, puisque nous y ferons référence pour expliciter notre travail de recherche.

3.3.1 Optimisation de l'exécution des jointures

Un index de jointure est une structure d'indexation qui porte sur de multiples tables et qui permet d'augmenter les performances des jointures de ces tables. Il permet de connaître par avance les tuples qui pourront faire l'objet de la jointure. Les index bitmap de jointure, "Bitmap Join Index", augmentent la performance pour une certaine classe de requêtes de jointure, autrement dit, ils sont utilisés si la cardinalité est faible. L'index de jointure est construit en traduisant les restrictions à la valeur de la colonne de la table de dimension.

Considérons, par exemple, deux tables CLIENT et VENTE.

Table CLIENT (10 tuples)

Client_ID	Sexe	Ville
C101	F	Annonay
C102	F	Annonay
C103	M	Tournon
C104	M	Peaugres
C105	F	Davézieux
C106	F	Tournon
C107	M	Peaugres
C108	F	St Désirat
C109	M	Peaugres
C110	F	Chanas

Table VENTE (11 tuples)

Produit_ID	Client_ID	Total_Vente
P10	C105	100
P11	C102	100
P15	C105	500
P10	C107	10
P10	C106	100
P10	C101	900
P11	C105	100
P10	C109	20
P11	C109	100
P10	C102	400
P13	C105	100

Un index de jointure sur le champ *Sexe* pour la table VENTE peut être construit en utilisant le champ *Sexe* de la table CLIENT et la clé étrangère *Client_ID*, dans la table VENTE. Ainsi, l'index peut être construit alors que la table VENTE ne contient pas le champ *Sexe*. L'index bitmap de jointure est créé en considérant les 11 tuples de la table VENTE. Les bitmaps auront alors une longueur de 11 bits. Le codage présence/absence s'effectue en récupérant la valeur du champ *Sexe* dans la table CLIENT, en utilisant le champ *Client_ID* comme clé de jointure. L'index obtenu est le suivant :

N° tuple	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
F	1	1	0	0	1	1	1	0	1	1	1
M	0	0	1	1	0	0	0	1	0	0	0

Ainsi, l'index bitmap de jointure est construit en faisant une jointure grâce aux clés étrangères et peut porter sur n'importe quel champ. Les index bitmap sont par ailleurs une structure intéressante pour optimiser les recherches sur les données.

3.3.2 Recherche sur les données

Les index bitmap permettent de répondre à certaines requêtes, sans nécessiter un retour sur les données sources, optimisant ainsi les temps de réponse. Cela est possible en appliquant des opérateurs sur les bits, c'est le cas pour les requêtes nécessitant des comptages. Par exemple, pour une requête où l'on cherche à connaître le nombre de tuples pour lesquels un attribut a telle valeur, il suffit de compter en ligne le nombre de '1' (présence) pour la ligne correspondant à la modalité recherchée. La requête peut se complexifier un peu dans le cas où il y a des conditions sur plusieurs attributs. Auquel cas, avant d'effectuer un comptage, il faut transcrire cette multiple condition en utilisant l'opérateur 'AND', pour combiner les différents bitmaps concernés. Nous allons donc détailler l'opérateur 'AND' qui permet de répondre à ce type de requêtes, sans qu'il n'y ait de retour sur les données brutes.

Nous considérons de nouveau l'échantillon de la base de données Titanic, en cherchant à savoir si la classe de l'individu influence le fait qu'il soit survivant ou non. Pour cela, il s'agit de construire au préalable les index sur les attributs

Classe et *Survivant*.

Index bitmap de l'attribut *Classe* :

N° tuple	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Equipage	1	0	0	1	1	0	0	0	0	0	0	0
1ère Classe	0	0	0	0	0	1	0	1	0	0	0	1
2ème Classe	0	0	1	0	0	0	1	0	0	1	0	0
3ème Classe	0	1	0	0	0	0	0	0	1	0	1	0

Index bitmap de l'attribut *Survivant* :

N° tuple	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Non	1	1	1	0	1	0	1	0	0	0	0	0
Oui	0	0	0	1	0	1	0	1	1	1	1	1

Les survivants correspondent aux individus dont l'attribut *Survivant* a pour valeur 'Oui'. Autrement dit, il s'agit des individus qui présentent un codage de présence dans le bitmap représentant la valeur 'Oui' de l'index de *Survivant*.

Supposons par exemple que la requête doive répondre à la question suivante : " Combien y a-t-il de survivants qui appartiennent à la 3^{eme} classe? "
Il suffit alors de considérer les bitmaps *Classe*='3ème' et *Survivant*='Oui'. Nous utilisons l'opérateur 'AND' pour construire le bitmap résultat. Pour un tuple donné, s'il y a un '1' dans les deux bits, nous mettons un '1' dans le bit de résultat et un '0' sinon. Nous récupérons dans le bitmap résultat les tuples pour lesquels il y a un '1', ce qui correspond aux survivants de la 3ème classe.

N° tuple	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Survivant	0	0	0	1	0	1	0	1	1	1	1	1
3ème Classe	0	1	0	0	0	0	0	0	1	0	1	0
AND	0	0	0	0	0	0	0	0	1	0	1	0

Ainsi, il y a deux individus de la 3ème classe qui ont survécu, en l'occurrence, les individus 2 et 4.

Il est aussi possible d'utiliser l'opérateur 'OR' pour caractériser des recherches sur des domaines de valeurs, mais nous ne détaillons pas celui-ci, étant donné que nous ne nous en servons pas dans la suite de notre travail.
Ainsi, les index bitmap contiennent un grand nombre d'informations, permettant ainsi de répondre à des questions sans consulter les données brutes. De cette façon, le temps de réponse peut être diminué. L'adaptation de ceux-ci a ouvert d'autres possibilités, dans le cadre de l'extraction de règles d'association entre autres, en optimisant la recherche de sous-ensembles de données. C'est le point que nous abordons maintenant.

Chapitre 4

Les index bitmap et l'extraction de règles d'association

4.1 Définitions et présentation du problème

La formalisation du problème de l'extraction de règles d'association a été proposée pour la première fois dans [AIS93] et a été étendue dans [AS94]. Nous pouvons adopter les notations et définitions suivantes :

Soit I=$\{i_1, i_2, .., i_m\}$ un ensemble de m items.
Soit D un ensemble de transactions, où chaque transaction T est un sous-ensemble $T \subseteq I$ d'items.
Une transaction T contient un ensemble d'items, autrement dit, un itemset X, si $X \subseteq T$.
Une règle d'association est une implication de la forme $X \Rightarrow Y$ où :

- $X \subset I$
- $Y \subset I$
- $X \cap Y = \oslash$

Cette règle a une confiance c dans l'ensemble de transactions D si c% des transactions dans D qui contiennent X contiennent aussi Y. Elle a un support s dans l'ensemble de transactions D si s% des transactions de D contiennent $X \bigcup Y$.

Dans ce qui suit, c'est le support d'un itemset qui nous intéresse. Il s'agit du nombre de transactions contenant l'itemset donné.
Un itemset dont le support est supérieur ou égal à un seuil minimal donné par l'utilisateur est dit "itemset fréquent".

Le processus d'extraction de règles d'association peut se décomposer en deux étapes [AIS93] :

1. rechercher des itemsets fréquents,
2. à partir des itemsets fréquents, construire les règles d'association dont la confiance est supérieure à une confiance minimale donnée.

Rappelons que pour optimiser le processus d'extraction de règles d'association, nous devons travailler sur l'optimisation de la recherche des itemsets fréquents. Pour déterminer quels sont les itemsets fréquents, il faut, pour chaque itemset, compter au préalable le nombre de transactions qui le contiennent. Le problème se situe alors dans le fait de rechercher les transactions qui contiennent un itemset donné. Cela nous ramène à une recherche de sous-ensembles de données. Dans le cadre de l'extraction de règles d'association, c'est un problème qui intervient souvent dans des bases de données volumineuses. L'indexation proposée dans [MZ98] répond à ce problème. Prenons l'exemple de la grande distribution, qui était la première idée du développement des règles d'association dans [AIS93]. Nous pouvons transcrire la table composée de couples (transactions/items) comme une liste d'items avec un marquage présence/absence selon les transactions. La notion de 'bitmap' semble, dans ce cas, particulièrement appropriée.

Considérons la table de transactions 'TABLE_DONNEES' suivante, qui représente le contenu de trois *transactions* identifiées par un entier positif non null, avec différents *items* dont l'identifiant est compris dans l'ensemble [0,18].

transaction	item
1	0
1	7
1	12
1	13
2	2
2	4
3	10
3	17
3	20

Sur la table 'TABLE_DONNEES', nous pouvons considérer deux index bitmap possibles :

sur les transactions :

1	1	1	1	1	0	0	0	0	0
2	0	0	0	0	1	1	0	0	0
3	0	0	0	0	0	0	1	1	1

sur les items :

0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0
10	0	0	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	1	0	0
13	0	0	0	0	0	1	0	0	0
14	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0
17	0	1	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0
20	1	0	0	0	0	0	0	0	0

Dans ces deux index, nous n'avons plus de vision sur l'appartenance d'un item à une transaction. Cette indexation permet de faire une référence à un tuple. Ici, nous pouvons retrouver les tuples faisant référence soit à une transaction, soit à un item. Dans le cadre de la recherche d'itemsets fréquents, ce qui nous intéresse réellement, c'est la relation étroite qu'il y a entre deux champs, en l'occurrence, entre les champs 'item' et 'transaction', et non pas la relation entre un champ et la position du tuple dans la base. Une transaction possède, en général, plusieurs items. Et c'est effectivement ce qui nous intéresse : pouvoir retrouver les transactions qui contiennent un ensemble donné d'items. Pour cela, l'index proposé perd la référence au tuple mais permet de représenter chaque transaction et les différents items qui la composent, cela, en utilisant les bits.

4.2 Une variante de l'Index Bitmap : le "Simple Group Bitmap Index"

Le "Simple Group Bitmap Index" s'inspire de l'index bitmap. En effet, il utilise des tableaux de bits et le principe de codage présence/absence. Cet index comporte un ensemble de clés. Chaque clé, qui est un tableau de bits, représente une transaction. Il y a donc autant de clés que de transactions. La longueur de chacune des clés correspond au nombre d'items possible. Nous avons dans chaque clé, un codage de présence/absence pour chaque bit, traduisant le fait que cette transaction possède ou non l'item en question. Autrement dit, il y a un '1' à la kième position de la clé j si l'item k est présent dans la transaction j ; comme pour les index bitmap, la lecture se fait de gauche à droite. Les clés sont stockées

dans une table d'index avec les identifiants des transactions qu'elles référencent.

Pour l'exemple que nous avons présenté, nous construisons, pour chacune des trois transactions représentées dans la base, une clé comprenant 18 bits, étant donné que les identifiants d'items vont de '0' à '17'. La première transaction comporte les items '0', '7', '12' et '13'. A ces positions, il y aura donc un '1' pour marquer la présence de ces produits pour la clé de la transaction '1'. Cette transaction sera repérée par ce numéro dans l'index. Nous procédons de même pour les autres transactions. Voici donc l'index obtenu, en suivant les règles que nous venons d'expliciter :

clé bitmap																		transaction
0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	2
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3

Cette indexation permet de retrouver les transactions qui comportent un ensemble d'items donné. Pour cela, il faut calculer la clé correspondant à l'itemset recherché. Elle a la même longueur que celles composant l'index. Ensuite, la clé construite est comparée avec chacune des clés de l'index, en utilisant l'opération 'AND' sur les bits. La procédure de vérification consiste à tester si pour chaque bit où il y a un '1' dans la clé de recherche, le bit correspondant est aussi à '1'.

Prenons un exemple de recherche d'itemset qui comprend les éléments 15 et 17. Une clé de recherche, d'une longueur de 18 bits, est calculée. Celle-ci contient des '1' aux positions 15 et 17, en lisant de droite à gauche, comme suit:

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

Nous faisons agir l'opérateur 'AND' entre la clé de recherche que nous appelons 'rech' et les différentes clés présentes dans la table des index, pour obtenir la clé résultat appelée 'res':

clé bitmap																		group
0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	rech
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**res**

clé bitmap																		group
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	2
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	rech
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**res**

clé bitmap																		group
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	rech
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**res**

Les transactions que nous recherchons correspondent aux transactions pour lesquelles la clé 'res' est identique à la clé 'rech' de recherche. C'est donc la transaction 3 qui contient l'itemset {15, 17}.

Néanmoins, la longueur des clés pose problème. Les clés sont de longueur N, où N est le nombre de tous les éléments possibles. En pratique, N peut être de

l'ordre de milliers de bits. La structure du "Hash Group Bitmap Index", qui est également présenté dans l'article [MZ98], pallie à cette limite. La longueur des clés est n<N, parce que le hashage est utilisé.

4.3 Discussion

Dans [MZ98], les auteurs ont donc développé une variante des index bitmap appelée "Simple Group Bitmap Index", qu'ils ont adaptée ensuite pour qu'elle puisse être utilisée sur de très grandes bases de données, en utilisant le haschage. Cependant, l'article présentant ces adaptations de l'index bitmap ne mentionne pas l'implémentation de ce "Simple Group Bitmap Index". Un mail aux auteurs concernant celle-ci étant demeuré sans réponse, nous avons effectué une implémentation du "Simple Group Bitmap Index" au cœur du SGBD Oracle 8i, en PL/SQL. Nous avons programmé la création de l'index ainsi que la recherche d'itemsets au moyen de procédures contenues dans un package. Cet index est donc intéressant dans l'extraction de règles d'association, pour la phase de recherche des itemsets fréquents, puisque la méthode proposée permet de retrouver les transactions comportant un ensemble donné d'items. Cette indexation permet une optimisation de cette recherche car un langage comme SQL n'est pas forcément adapté pour ce type de recherche. Par exemple, si nous souhaitons obtenir le support de l'itemset {0, 7, 12, 13} dans la table 'TABLE_DONNEES' présentée précédemment, il faut déterminer quelles sont les transactions qui comportent cet itemset, afin de les dénombrer. Voici quelles seraient les requêtes SQL possibles pour obtenir le résultat :

1. SELECT transaction
   FROM TABLE_DONNEES A, TABLE_DONNEES B, TABLE_DONNEES C, TABLE_DONNEES D,
   WHERE A.transaction = B.transaction
   AND B.transaction = C.transaction
   AND C.transaction = D.transaction
   AND A.item = 0
   AND B.item = 7
   AND C.item = 12
   AND D.item = 13;
2. SELECT transaction from TABLE_DONNEES
   WHERE item (0, 7, 12, 13)
   GROUP BY transaction
   HAVING COUNT(*)=4;

Dans (1.), des copies de la table et des jointures sont utilisées. Le nombre de copies correspond à la cardinalité de l'itemset. Quelle que soit la requête (1. ou 2.), elle est lancée pour chaque itemset. Lors de la recherche des itemsets fréquents, la taille des itemsets candidats varie. Il faut donc adapter au fur et à mesure les requêtes, en fonction des cardinalités des itemsets traités. Ainsi, le langage SQL n'est pas très adapté à la recherche de sous-ensembles. L'index que nous venons de présenter est donc utile et présente l'avantage de pouvoir être intégré au cœur du SGBD.

Chapitre 5

Les Index bitmap et ID3

Après avoir présenté comment les index bitmap peuvent être utilisés dans le processus d'extraction de règles d'association, nous étudions, dans cette partie, comment se servir des index bitmap pour appliquer des méthodes de fouille de données, basées sur la construction d'arbres de décision. Nous nous intéressons en particulier à l'intégration de la méthode ID3 dans un SGBD en utilisant les index bitmap. Cette étude soulève donc deux aspects importants. Le premier aspect concerne l'adaptation de l'algorithme de la méthode ID3 pour pouvoir s'appuyer sur les index, et non pas sur la base d'apprentissage. Le deuxième aspect concerne l'intégration de ID3 dans le SGBD Oracle en utilisant les outils de celui-ci.

5.1 ID3 : le principe

ID3 est un algorithme qui se situe dans le contexte de l'apprentissage supervisé. Le principe de construction de l'arbre, décrit dans [ZR00], est le suivant. Nous partons de l'ensemble de la population d'apprentissage qui constitue la racine de l'arbre. Nous disposons de p variables, qui sont des descripteurs de cette population. Chaque individu de cette population appartient à une classe que nous connaissons. C'est cette classe que nous cherchons à prédire. En effet, il s'agit de construire un arbre de décision et d'obtenir ainsi des règles, qui permettront de définir la classe d'appartenance de nouveaux individus selon leurs caractéristiques. Cela revient à la construction d'une succession de partitions de plus en plus fines, à partir de la population d'apprentissage de départ. La partition est opérée en fonction de la répartition de la classe suivant les différentes variables du nœud père, autres que la classe. Pour déterminer la meilleure variable à sélectionner pour segmenter la population, le critère de "gain d'incertitude", appelé aussi "gain informationnel", doit être maximisé ; cela caractérise le pouvoir discriminant de la variable utilisée pour la segmentation, qui doit être maximal. La population du nœud père est alors répartie dans les noeuds fils suivant leur valeur pour la variable choisie pour la segmentation. La particularité de l'arbre construit est que les fils d'un nœud sont obtenus à partir d'une partition totale et exclusive de ce nœud. Le processus est ensuite répété pour chaque fils ainsi obtenu. Il est stoppé quand les feuilles de l'arbre ainsi générées contiennent une population de la même classe. Mais il est aussi possible

de définir un arrêt prématuré du processus, si une nouvelle partition engendrait l'apparition de sommets où les effectifs étaient trop faibles pour être significatifs.

Dans l'algorithme ID3, le pouvoir discriminant d'une variable pour la segmentation d'un nœud est exprimé par une variation d'entropie. L'entropie h_s d'un nœud s_k (plus précisément, son entropie de Shannon) est :

$$h_s(s_k) = -\sum \frac{n_{ik}}{n_k} \log_2 \frac{n_{ik}}{n_k} \tag{5.1}$$

où n_k est l'effectif de s_k et n_{ik} le nombre d'objets de s_k qui appartiennent à la classe c_i. L'information portée par une partition S_K de K noeuds est alors la moyenne pondérée des entropies :

$$E(S_K) = \sum \frac{n_k}{n_j} h_s(s_k) \tag{5.2}$$

où n_j est l'effectif du nœud s_j qui est segmenté. Finalement, le gain informationnel associé à S_K est :

$$G(S_K) = h_s(s_j) - E(S_K) \tag{5.3}$$

Comme $G(S_K)$ est toujours positif ou nul, le processus de construction d'arbre de décision revient à une heuristique de maximisation de $G(S_K)$ à chaque itération et à la sélection de la variable correspondante pour segmenter un nœud donné. L'algorithme s'arrête lorsque $G(S_K)$ devient inférieur à un seuil (gain minimum) défini par l'utilisateur.

5.2 Utilisation des index bitmap

La nouvelle approche que nous proposons pour la génération d'arbres de décision est basée sur l'utilisation des index bitmap. Cette utilisation est motivée par plusieurs raisons. Outre leur faible coût de stockage, les index bitmap se prêtent bien à l'exécution de certaines opérations. Deux de ces opérations nous intéressent particulièrement. D'une part, la possibilité de faire des comptages sur un bitmap permet de calculer simplement et facilement des effectifs. D'autre part, il est possible d'effectuer, entre différents bitmaps, des opérations logiques qui se font "bit à bit", telles que le 'AND'. Cela permet de poser des conditions sur une population qui est caractérisée par différents bitmaps. La combinaison de ces deux types d'opération permet de répondre facilement à un certain type de requêtes sans accéder aux données brutes. Cette combinaison prend alors tout son sens dans le cadre de notre recherche sur la méthode ID3. En effet, les effectifs des différentes sous-populations d'un arbre de décision peuvent être obtenus par de simples comptages sur des bitmaps. Ces derniers sont le résultat de l'opérateur 'AND' entre deux ou plusieurs bitmaps qui fournissent des caractéristiques sur la population. Il s'agit effectivement de combiner les différents index bitmap basés sur les différents attributs, qu'ils soient prédictifs ou à prédire. L'ensemble des index constitue donc notre population d'apprentissage. L'avantage de travailler avec les index est d'éviter un accès aux données brutes. Une fois les index bitmap construits, le travail s'effectue directement sur ceux-ci, ce qui permet de réduire les temps induits par les traitements. Notre objectif est donc d'élaborer un algorithme qui utilise les index bitmap en vue de

la construction des arbres de décision. Pour valider notre approche, nous avons choisi d'implémenter la méthode ID3, pour sa simplicité, au cœur d'un SGBD, en utilisant les index bitmap.

Dans les arbres de décision, et plus particulièrement dans la méthode ID3, à chaque nœud, il s'agit de segmenter la population en choisissant un attribut discriminant, dont les différentes valeurs vont permettre de générer de nouvelles sous-populations. Excluons pour l'instant le problème de l'entropie et donc le problème du choix de l'attribut discriminant. Nous considérons que nous disposons non seulement de l'ordre des attributs prédictifs à considérer, mais aussi des informations concernant le fait de segmenter un nœud ou pas en fonction d'un gain informationnel minimal. Notre but est donc de retrouver les effectifs correspondant aux modalités de la classe à prédire, pour chacun des noeuds de l'arbre.

Pour mieux comprendre notre raisonnement, nous nous appuyons sur l'exemple de la population de la base du Titanic, dont les données se présentent ainsi :

Classe	Age	Sexe	Survivant
1ère	Adulte	Femme	Oui
3ème	Adulte	Homme	Oui
2ème	Enfant	Homme	Oui
3ème	Adulte	Homme	Oui
1ère	Adulte	Femme	Oui
2ème	Adulte	Homme	Non
1ère	Adulte	Homme	Oui
Equipage	Adulte	Femme	Non
Equipage	Adulte	Femme	Oui
2ème	Adulte	Homme	Non
3ème	Adulte	Homme	Non
Equipage	Adulte	Homme	Non
...	...	...	...

Au préalable, nous construisons les index bitmap de chacun des attributs (qu'il soit prédictifs ou à prédire), en l'occurence ici 4 index comme suit :

	N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Classe	Equipage	..	1	0	0	1	1	0	0	0	0	0	0	0
	1ère	..	0	0	0	0	0	1	0	1	0	0	0	1
	2ème	..	0	0	1	0	0	0	1	0	0	1	0	0
	3ème	..	0	1	0	0	0	0	0	0	1	0	1	0

	N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Age	Enfant	..	0	0	0	0	0	0	0	0	0	1	0	0
	Adulte	..	1	1	1	1	1	1	1	1	1	0	1	1

	N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Sexe	Femme	..	0	0	0	1	1	0	0	1	0	0	0	1
	Homme	..	1	1	1	0	0	1	1	0	1	1	1	0

	N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Survivant	Non	..	1	1	1	0	1	0	1	0	0	0	0	0
	Oui	..	0	0	0	1	0	1	0	1	1	1	1	1

Notre population d'apprentissage est donc constituée des différents index suivants: l'index de la classe à prédire, SURVIVANT et les index des attributs prédictifs, SEXE, CLASSE et AGE. La première étape consiste à créer le nœud racine. Il est caractérisé par les différents effectifs des sous-populations qui sont définies selon les modalités de la classe à prédire, sans tenir compte des valeurs des différents attributs prédictifs.

Dans notre exemple, l'attribut SURVIVANT, qui est la classe à prédire dans la base Titanic, possède deux modalités: 'Oui' et 'Non'. Pour le nœud racine, nous cherchons donc à obtenir l'effectif de la population pour laquelle SURVIVANT='Oui' d'une part et celui pour laquelle SURVIVANT='Non' d'autre part. Pour cela, nous avons besoin uniquement de l'index concernant l'attribut SURVIVANT. Nous comptons alors le nombre de '1' dans chacun des bitmaps de cet index. Nous obtenons alors le nœud racine suivant:

SURVIVANT	
Oui	711
Non	1490

FIG. 5.1 – *Noeud Racine de l'Arbre de Titanic*

Supposons que l'attribut prédictif qui segmente le nœud racine est l'attribut SEXE. Il s'avère que cet attribut possède deux modalités: 'Femme' et 'Homme'. L'index bitmap qui porte sur SEXE est donc composé de deux bitmaps. Les noeuds fils de la racine de l'arbre sont donc caractérisés par les règles SEXE='Homme' d'une part, SEXE='Femme' d'autre part. Les bitmaps correspondant à ces populations sont ceux de l'index en question. Etudions le nœud issu de la règle SEXE='Homme', le principe est le même pour le nœud associé à la règle SEXE='Femme'. Deux effectifs lui sont rattachés: celui des survivants et celui des non-survivants. Pour obtenir ces deux effectifs, nous utilisons l'opérateur logique 'AND' entre le bitmap SEXE='Homme' et les bitmaps SURVIVANT='Oui' d'une part, SURVIVANT='Non' d'autre part, de la manière suivante:

N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
SEXE='Homme'	..	1	1	1	0	0	1	1	0	1	1	1	0
SURVIVANT='Oui'	..	0	0	0	1	0	1	0	1	1	1	1	1
AND	..	0	0	0	0	0	1	0	0	1	1	1	0

N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
SEXE='Homme'	..	1	1	1	0	0	1	1	0	1	1	1	0
SURVIVANT='Non'	..	1	1	1	0	1	0	1	0	0	0	0	0
AND	..	1	1	1	0	0	0	1	0	0	0	0	0

De manière générale, pour un nœud donné, à l'exception du nœud racine, il s'agit au préalable de trouver la sous-population correspondant aux caractéristiques de ce nœud, c'est à dire la sous-population qui répond aux règles de la branche du nœud. Cela dépend donc des valeurs des attributs qui ont déjà été choisis précédemment pour segmenter la population. Cette recherche s'effectue en utilisant successivement l'opérateur 'AND' entre les bitmaps des différents

index. Il s'agit ensuite de refaire une opération 'AND' avec l'index de la classe à prédire et de faire un comptage des '1' pour déterminer les différents effectifs. Néanmoins, comme nous venons de le constater, lors de la segmentation du nœud racine, le bitmap représentant les caractéristiques des sous-populations générées correspondent exactement aux bitmaps fournis par l'index de l'attribut prédictif. Nous avons donc uniquement l'opération avec les bitmaps de la classe à prédire. Nous obtenons donc les noeuds suivants :

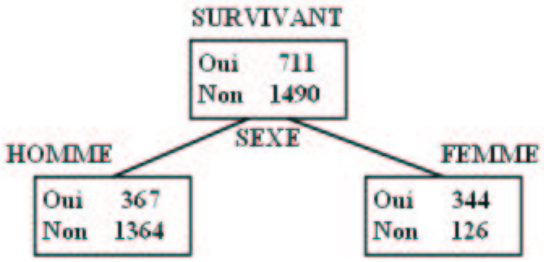

FIG. 5.2 – *Segmentation par la variable SEXE*

Considérons que l'attribut prédictif suivant est CLASSE. CLASSE est un attribut qui possède les quatre modalités 'Equipage', '1ère', '2ème' et '3ème', qui donnent lieu à la création de quatre noeuds fils. Chacun de ses noeuds est caractérisé par une règle qui a été précisée par rapport à la règle de leur père, SEXE='Homme', par exemple. Prenons le cas de la modalité '1ère', le processus est identique pour les autres valeurs de l'attribut CLASSE. Nous devons repartir du bitmap caractérisant le nœud père qui ne tient pas compte des valeurs de la classe à prédire SURVIVANT (bitmap résultat d'un 'AND' précédent). Nous devons y ajouter la règle CLASSE='1ère', en introduisant l'opération 'AND' entre le bitmap du père et le bitmap correspondant à cette règle, comme suit :

N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
SEXE='Homme'	..	1	1	1	0	0	1	1	0	1	1	1	0
CLASSE='1ère'	..	0	0	0	0	0	1	0	1	0	0	0	1
AND	..	0	0	0	0	0	1	0	0	0	0	0	0

Le bitmap résultat obtenu correspond donc à la règle "SEXE='Homme' AND CLASSE='1ère'". Pour obtenir les deux effectifs selon la classe à prédire SURVIVANT, nous effectuons une opération 'AND' entre le bitmap résultat et chacun des bitmaps de l'index de l'attribut SURVIVANT :

N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
SEXE='Homme' AND CLASSE='1ère'	..	0	0	0	0	0	1	0	0	0	0	0	0
SURVIVANT='Oui'	..	0	0	0	1	0	1	0	1	1	1	1	1
AND	..	0	0	0	0	0	1	0	0	0	0	0	0

N° tuple	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
SEXE='Homme' AND CLASSE='1ère'	..	0	0	0	0	0	1	0	0	0	0	0	0
SURVIVANT='Non'	..	1	1	1	0	1	0	1	0	0	0	0	0
AND	..	0	0	0	0	0	0	0	0	0	0	0	0

Une fois les deux bitmaps résultats obtenus, il s'agit de faire un comptage du nombre de '1' pour obtenir les deux effectifs. Nous obtenons l'arborescence suivante :

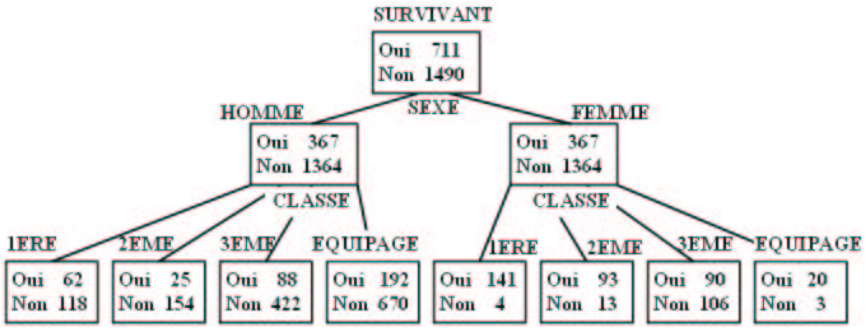

FIG. 5.3 – *Segmentation par la variable CLASSE*

En procédant de la même façon et en considérant que l'attribut prédictif suivant est AGE, nous obtenons l'arbre de décision suivant :

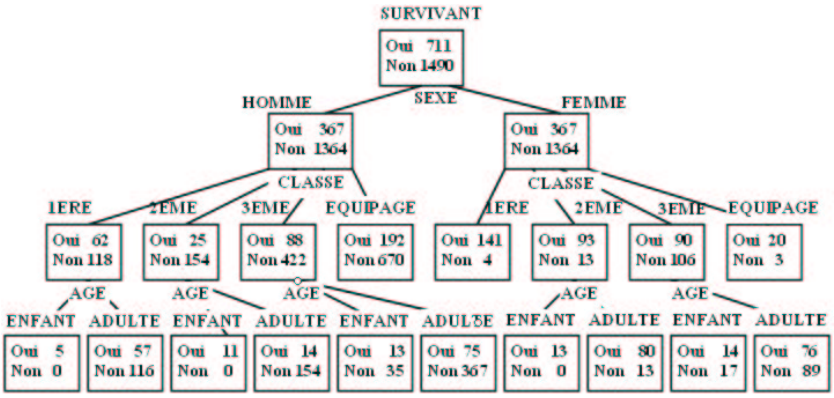

FIG. 5.4 – *Segmentation par la variable AGE*

La dernière segmentation, sur la variable AGE, n'est pas faite sur tous les noeuds car dans certains cas, elle n'est pas significative, dans la mesure où les effectifs correspondant ne sont pas suffisants.

5.3 Algorithme

Différents objets sont nécessaires pour la construction de l'algorithme (pseudo-code).

Tout d'abord, nous utilisons une pile de nœuds qui contient les données suivantes :

- num : le numéro du nœud.
- nview : le nom de la vue relationnelle associée.
- rule : la règle explicite menant à la création du nœud à partir de son père, par exemple AGE_CHILD='1'. Pour la racine, cette règle est égale à ' '.
- argnotuse : la liste de toutes les règles, reliées par des 'AND', ayant mené à la création du nœud père. Pour la racine, argnotuse vaut ' '.
- entrop : l'entropie du nœud.
- pop : la population du nœud.

Nous utilisons également un certain nombre de piles :

- sauve_nom : une pile de noms des noeuds temporairement retenus.
- sauve_entropie : la pile de l'entropie respective des noeuds temporairement retenus.
- sauv_population : la pile de la population respective des noeuds temporairement retenus.
- pile_entropie : la pile de l'entropie calculée sur un nœud, pour chaque valeur d'un champ prédictif donné.
- popstr : la pile de la population respective sur un nœud pour chaque valeur d'un champ prédictif donné.

Voici l'algorithme que nous proposons, afin de pouvoir utiliser les index bitmap dans le cadre de la méthode ID3 :

Pour i = 1 à nombre d'attributs de la population d'apprentissage[1] Faire
  création index bitmap $B^{attribut_i}$;
Fin Pour

Créer la table résultat;

Pour i = 1 à nombre de modalités de l'attribut à prédire Faire
  pop := nombre de présences pour la modalité i[2];
  $pop\_totale := pop\_totale + pop$;
Fin Pour

$entropie := 0$;

Pour i = 1 à nombre de modalités de l'attribut à prédire Faire
  pop := nombre de présences pour la modalité i;
  $entropie := entropie - (\frac{pop}{pop\_totale}) * LOG_2(\frac{pop}{pop\_totale})$;

1. y compris attribut à prédire
2. count()

Fin Pour

Insérer dans table résultat nœud racine et informations;

Empiler dans pile stack le nœud racine;

Tant que la pile stack n'est pas vide Faire

Dépiler de pile stack le nœud courant;
Initialiser max_gain à 0 et max_var à ' ';
Pour chaque attribut prédictif *var* candidat pour la segmentation[3] du nœud courant Faire
Empiler dans pile_entropie 0;
gain := entropie du nœud courant ;
Initialiser j à 0 ;
Pour chaque valeur de l'attribut candidat *val* courant Faire
bitmap_carac := bitmap_nœud_courant AND bitmap_val;
Pour i = 1 à nombre de modalités de l'attribut à prédire Faire
$bitmap\_res$:= bitmap_carac AND $bitmap\_attr\_predire_i$;
$pop\_courante$:= comptage présence dans bitmap_res;
$pile\_entropie(i) := pile\_entropie(i) - (pop\_courante) * LOG_2(pop\_courante)$;

Si on change de valeur pour l'attribut candidat Alors
$Gain := Gain - (\frac{1}{pop\_du\_noeud\_courant}) * (entropytemp(i) + (j * LOG_2(j)))$;
$entropytemp(i) := (\frac{entropytemp(i)}{j}) + LOG_2(j)$;
i := i+1;
empiler 0 dans pile_entropie;
empiler j dans popstr;
j:= 0;
Fin Si (on change de valeur pour l'attribut candidat);
j := j + pop_courante;
Fin Pour (chaque valeur de l'attribut candidat)
Si Gain>max_gain et Gain>gain minimum requis Alors
Vider les piles sauve_nom, sauve_entropie et sauve_population;
max_gain := Gain;
max_var:= var;
i:=1;
j:=0;
Tant que j<population du nœud courant faire
j:=j+popstr(i);
k:= numéro de la vue (par incrémentation);
empiler le nom de la vue dans sauve_nom;
empiler l'entropie du fils dans sauve_entropie;
empiler popstr(i) dans sauve_population;
i:=i+1;
Fin Tant que (j<population du nœud courant);
Fin Si (Gain>max_gain et Gain>gain minimum requis);
Vider les piles pile_entropie et popstr;

3. pas encore utilisé pour la segmentation

Fin pour; (chaque valeur de l'attribut candidat *val* courant)
Si le gain du meilleur champ prédictif > au gain minimum Alors
i:=1;
Empiler dans stack les fils du nœud courant par ce champ prédictif;
Argnotuse des fils:=leur règle de création + argnotuse du nœud courant;
Actualiser la table de résultat en insérant les fils;
Fin Si (le gain du meilleur champ prédictif > au gain minimum);
Vider les piles sauve_nom, sauve_entropie et sauve_population;
Fin tant que; (pile stack n'est pas vide)

5.4 Implémentation

Pour valider notre approche, nous avons effectué une implémentation. Celle-ci ne se fait pas de manière classique mais au cœur d'un SGBD. Dès lors nous ne sommes plus limitées par la taille de la mémoire centrale, comme c'est le cas lors de l'exécution des algorithmes classiques de fouille de données. Nous pouvons travailler sur de grandes bases de données, en ayant pour seule limite la taille du disque utilisé.

L'implémentation a été effectuée en PL/SQL, compatible sous Oracle 8i et Oracle 9i, sous la forme d'une procédure stockée nommée "id3", au sein d'un package de procédures nommé "DECISION_TREE"[4]. Ce package comporte également la procédure qui permet de générer automatiquement la simulation des index portant sur chacun des attributs de la table, attributs prédictifs et attribut à prédire. En effet, cette implémentation n'est qu'une simulation puisque nous ne sommes pas parvenues à accéder aux index créés par Oracle lorsque nous effectuons une commande telle que:
"CREATE BITMAP INDEX nom_index ON nomtable(nomattribut)"

Nous avons donc utilisé uniquement les outils fournis par Oracle: package, procédures, tables, curseurs, piles, ... Les résultats de la méthode ID3 sont stockées dans une table qui permet, via une requête hiérarchique, de retrouver la structure de l'arbre, comme cela était déjà proposé dans [BD02]. La structure de l'arbre permet de connaître les règles de décision qui ont été établies en parcourant tous les chemins qui relient la racine aux différentes feuilles de l'arbre.

Nous obtenons des résultats conformes à ceux obtenus par Sipina, logiciel de fouille de données [ZR96], qui fournit l'arbre suivant:

4. http://bdd.univ-lyon2.fr

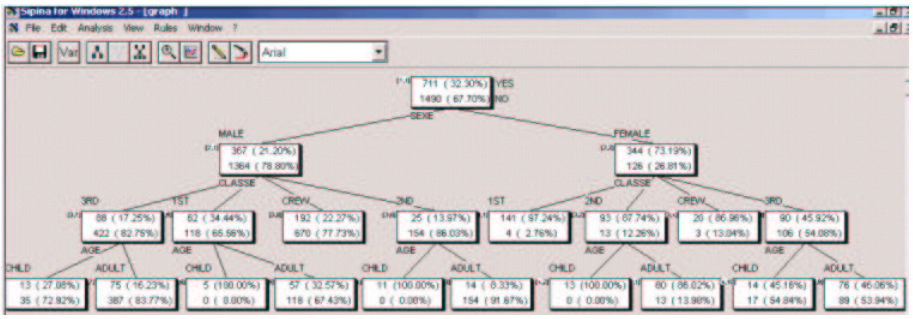

FIG. 5.5 – *Résultat de la méthode ID3 sur les données du Titanic, fourni par Sipina*

5.5 Discussion

Dans [MZ98], une variante des index bitmap, le "Simple Group Bitmap Index", a été développée pour optimiser l'extraction d'itemsets fréquents. Cet index permet de retrouver facilement les transactions contenant un ensemble donné d'items, compte-tenu du fait qu'il traduit, dans sa construction, l'appartenance d'un item à une transaction. Cependant, dans le cadre des arbres de décision, cette variante ne convient pas car elle perd toute référence au tuple, et cette référence est nécessaire dans la construction des arbres. En effet, pour la construction d'arbres de décision, nous regroupons au fur et à mesure les individus en sous-populations, en accumulant successivement de l'information sur eux, en fonction des différents index. Pour cela, il est nécessaire de garder la notion d'identification de l'individu, la notion de tuple, qui est commune à tous les index. Ainsi, c'est l'index bitmap traditionnel qui est utilisé.

Celui-ci a des propriétés intéressantes qui permettent de répondre rapidement à un certain type de requête sans accéder à la base. Ce type de requête correspond aux besoins de la création d'un arbre de décision. En effet, dans le cadre de la méthode ID3, entre autres, nous sommes amenées à faire des partitions de plus en plus fines. Ainsi, plus nous descendons dans l'arbre, plus les sous-populations sont caractérisées par des conditions restrictives quant aux valeurs des attributs les décrivant. Il s'agit de caractériser ces sous-populations en accumulant de l'information sur celles-ci, ce qui se traduit par la combinaison successive d'index en utilisant l'opérateur logique 'AND'. Les effectifs sont alors calculés facilement en effectuant de simples comptages sur les bitmaps résultant des opérations 'AND'. Ainsi, l'arbre de décision peut donc être construit sans qu'il n'y ait d'accès aux données brutes, en effectuant diverses opérations sur les index bitmap, ce qui diminue considérablement les temps de traitement.

Notre population d'apprentissage est donc constituée de l'ensemble des index bitmap. Etant donné leur faible coût de stockage, en raison de leur codage sur des bits, la population d'apprentissage s'en retrouve réduite, ce qui améliore également les temps d'apprentissage.

Dans le cadre de l'intégration de la méthode ID3 au cœur du SGBD, le problème de la continuité des attributs rencontrés dans l'application classique de l'algorithme persiste. Nous travaillons sur une population d'apprentissage qui est constituée de l'ensemble des index bitmap. Le problème de continuité

se pose donc en amont de l'apprentissage, lors de la construction des index. Il s'agirait dans ce cas là de mettre en place une stratégie de pré-traitement de la base, en l'occurence une discrétisation, avant la construction des index. De plus, le fait que les attributs continus aient de fortes cardinalités va à l'encontre des conditions optimales d'utilisation des index bitmap. Cependant, si les index bitmap conviennent plutôt pour des attributs de faible cardinalité, ce n'est pas un réel problème dans le cadre de la méthode ID3. En effet, pour obtenir un arbre dont les règles engendrent un modèle prédictif robuste, les effectifs des sous-populations doivent être significatifs. Ainsi, un trop grand nombre de modalités pour les attributs descripteurs de la population peut nuire à l'apprentissage. Auquel cas, il est nécessaire d'effectuer des regroupements afin d'éviter l'apparition de nombreux sommets enfants avec très peu d'individus ou vides [ZR00]. De plus, un nombre de modalités trop important pour certains attributs peut introduire un biais dans l'apprentissage. Effectivement, si le nombre de modalités est trop hétérogène entre les différents attributs, les attributs ayant de nombreuses modalités ont un pouvoir discriminant supérieur, ce qui introduit un biais dans l'apprentissage. C'est pour cette raison, d'ailleurs, que le critère "gain-ratio" a été introduit dans la méthode C4.5 [Qui93].

Chapitre 6

Conclusion et perspectives

Dans ce stage de DEA, nous nous sommes intéressées à la possibilité d'utiliser les index bitmap pour faire de la fouille de données sur des grandes bases de données, en ayant comme objectif d'optimiser les temps de traitement. Nous avons en particulier étudié l'utilisation des index bitmap dans le processus d'extraction de règles d'association. Il s'agit d'une variante des index bitmap qui permet d'effectuer la recherche de sous-ensembles de façon optimisée, pour aider à la découverte d'itemsets fréquents. Nous nous sommes basées sur un algorithme existant et avons, par conséquent, implémenté cette variante et la méthode de recherche, en les intégrant dans un package, au cœur du SGBD Oracle. Par ailleurs, nous avons proposé une nouvelle approche concernant les méthodes de fouille de données basées sur les arbres de décision. Il s'agit d'intégrer ces méthodes au cœur des SGBD en utilisant les index bitmap. Nous avons proposé un algorithme qui valide cette approche dans le cadre de la méthode ID3, qui est donc basé sur l'utilisation des index bitmap pour intégrer la méthode ID3 au cœur d'un SGBD. Nous avons donc implémenté cet algorithme en PL/SQL, compatible sous Oracle 8i et Oracle 9i. Nous avons montré qu'avec les index bitmap nous évitions l'accès aux données qui ralentissaient les temps de traitement. Notre approche, basée sur l'utilisation des index bitmap, peut même être transposée au cadre classique de l'application des méthodes de fouille de données, qui opèrent en dehors de la base de données. En effet, il suffit pour cela de rapatrier les index bitmap du SGBD à la mémoire et d'effectuer les calculs nécessaires à la construction de l'arbre avec un langage hôte.

Différentes perspectives sont envisagées. En premier lieu, il serait intéressant de pouvoir développer une véritable implémentation et pas seulement une simulation, pour pouvoir tester les performances de la méthode. Ces tests peuvent regrouper plusieurs aspects : l'application de notre méthode à des bases de données réelles, la comparaison de notre approche avec la méthode ID3 intégrée qui utilise les vues relationnelles d'une part et les méthodes classiques d'autre part. Nous envisageons également d'étendre l'utilisation des index bitmap pour d'autres méthodes, telles que C4.5 et CART.
De plus, si nous nous re-penchons sur l'aspect de l'optimisation de l'exécution des jointures en utilisant les "Bitmap Join Index", la piste de la fouille de données multi-relationnelle pourrait être explorée. Au lieu de créer la vue correspondant à la base d'apprentissage qui nous intéresse, il suffit de construire les

index bitmap de jointure, qui sont une variante des index bitmap, à partir des tables participant à la création de la vue. Ensuite, notre algorithme s'appliquera àux index bitmap de jointure.
Nous avons montré dans ce travail qu'il est possible d'implémenter des méthodes de fouille de données au moyen de concepts bases de données tels que les vues relationnelles [BD02] et les index bitmap. D'autres concepts bases de données pourraient avoir aussi un intérêt. Nous pensons en particulier au concept de trigger. En effet, même si le concept d'index va à l'encontre de l'idée de faire des mises à jour, notons que l'avantage d'avoir une méthode intégrée dans le SGBD doit nous permettre de travailler sur la notion d'"apprentissage incrémental". En effet, nous avons précisé dans la section 2.1 qu'il est préférable d'apprendre sur l'ensemble d'une population, plutôt que sur un échantillon de celle-ci. Même si ce n'est pas le cas dans l'exemple du Titanic, une population peut évoluer, et son apprentissage également. Ainsi, le concept de trigger, outil qui permet de déclencher un traitement lorsqu'il y a une mise à jour dans une base de données, semble intéressant. Pour cela, il faudrait étudier la possibilité de faire les modifications nécessaires au niveau de l'apprentissage en fonction des changements effectués dans la base de données.

Bibliographie

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 1994.

[BD02] Fadila Bentayeb and Jérôme Darmont. Decision tree modeling with relational views. In *XIIIth International Symposium on Methodologies for Intelligent Systems (ISMIS 2002), Lyon, France*, volume 2366 of *LNAI*, pages 423–431, Heidelberg, Germany, June 2002. Springer Verlag.

[Cha98] Surajit Chaudhuri. Data mining and database systems: Where is the intersection? *Data Engineering Bulletin*, 21(1):4–8, 1998.

[Cha02] Jean-Hugues Chauchat. *Echantillonnage, validation et généralisation en extraction des connaissances à partir des données*. Mémoire d'habilitation à diriger des recherches, Université Lumière Lyon 2 - France, Janvier 2002.

[CI98] Chee-Yong Chan and Yannis E. Ioannidis. Bitmap index design and evaluation. pages 355–366. ACM SIGMOD International Conference on Management of Data, 1998.

[Cod93] E F Codd. Providing olap (on-line analytical processing) to user-analysts: An it mandate. Technical report, E.F. Codd and Associates, 1993.

[IBM01] IBM. Db2 intelligent miner scoring. http://www-3.ibm.com/software/data/iminer/scoring, 2001.

[MPC96] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. A new sql-like operator for mining association rules. In *22th International Conference on Very Large Data Bases (VLDB 96), Mumbai, India*, pages 122–133. Morgan Kaufmann, 1996.

[MZ98] Tadeusz Morzy and Maciej Zakrzewicz. Group bitmap index: A structure for association rules retrieval. pages 284 – 288. International Conference on Knowledge Discovery in Databases and Data Mining: KDD'98, 1998.

[OG95] Patrick E. O'Neil and Goetz Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Record*, 24(3):8–11, 1995.

[OQ97] P. O'Neil and D. Quass. Improved query performance with variant indexes. pages 38–49. ACM SIGMOD International Conference on Management of Data, May 1997.

[Ora01] Oracle. Oracle 9i data mining. White paper, June 2001.

[Pas00] Nicolas Pasquier. *Data Mining : Algorithmes d'Extraction et de Réduction des Règles d'Association dans les Bases de Données.* PhD thesis, Université Clermont-Ferrand 2, Janvier 2000.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui93] J.R. Quinlan. *C4.5 : Programs for Machine Learning.* Morgan Kaufmann, 1993.

[STA98] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating mining with relational database systems: Alternatives and implications. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 98), Seattle, USA*, pages 343–354. ACM Press, 1998.

[STY01] Sanjay Soni, Zhaohui Tang, and Jim Yang. Performance study microsoft data mining algorithms. Technical report, Microsoft Corp., 2001.

[UBDB03] Cédric Udréa, Fadila Bentayeb, Jérôme Darmont, and Omar Boussaid. Préparation des données pour une intégration efficace de méthodes de fouille de données dans les sgbd. Technical report, Laboratoire ERIC - Université Lyon 2 - France, 2003.

[VG99] Sirirut Vanichayobon and Le Gruenwald. Indexing techniques for data warehouses'queries. Technical report, The University of Oklahoma, School of Computer Science, July 1999.

[Wu99] Ming-Chuan Wu. Query optimization for selections using bitmaps. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 227–238. ACM Press, 1999.

[ZR96] D A Zighed and R Rakotomalala. Sipina-w(c) for windows: User's guide. Technical report, ERIC laboratory, University of Lyon 2, France, 1996.

[ZR00] D.A. Zighed and R. Rakotomalala. *Graphes d'induction. Apprentissage et Data Mining.* Hermes Science Publication, 2000.