Intégration efficace des arbres de décision dans les SGBD : utilisation des index bitmap

Cécile Favre, Fadila Bentayeb

Laboratoire ERIC, Lyon 2
5 Avenue Pierre Mendès France
69676 Bron CEDEX

{cfavre,bentayeb}@eric.univ-lyon2.fr,

Résumé. Nous présentons dans cet article une nouvelle approche de fouille qui permet d'appliquer des algorithmes de construction d'arbres de décision en répondant à deux objectifs : (1) traiter des bases volumineuses, (2) en des temps de traitement acceptables. Le premier objectif est atteint en intégrant ces algorithmes au cœur des SGBD, en utilisant uniquement les outils fournis par ces derniers. Toutefois, les temps de traitement demeurent longs, en raison des nombreuses lectures de la base. Nous montrons que, grâce aux index bitmap, nous réduisons à la fois la taille de la base d'apprentissage et les temps de traitements. Pour valider notre approche, nous avons implémenté la méthode ID3 sous forme d'une procédure stockée dans le SGBD Oracle.

Mots clés : Index bitmap, bases de données, fouille de données, arbres de décision, performance, complexité.

1 Introduction

L'application efficace de méthodes de fouille sur des bases de données volumineuses devient un enjeu de recherche de plus en plus important. Les algorithmes traditionnels de fouille de données s'appliquent sur des tableaux attributs/valeurs (Zighed et Rakotomalala 2000). La volumétrie des bases étant croissante, les algorithmes classiques se heurtent au problème de la limitation de la taille de la mémoire centrale dans laquelle les données sont traitées. La "scalabilité" (capacité de maintenir des performances malgré un accroissement du volume de données), peut alors être assurée en optimisant soit les algorithmes (Agrawal et al. 1996, Gehrke et al. 1998), soit l'accès aux données (Ramesh et al. 2001, Dunkel et Soparkar 1999). Une autre issue au problème consiste à réduire la volumétrie des données à traiter. Pour cela, une phase de prétraitement est généralement appliquée sur les données : l'échantillonnage (Ttoivonen 1996, Chauchat et Rakotomalala 2000) ou la sélection d'attributs (Lia et Motoda 1998).

Récemment, une nouvelle approche de fouille de données est apparue pour pallier au problème de limitation de la taille de la mémoire. Il s'agit d'intégrer les méthodes de fouille de données au cœur des Systèmes de Gestion de Bases de Données (SGBD) (Chaudhuri 1998). Ainsi, le volume des données traitées n'est plus limité par la taille de la mémoire. Cette piste de recherche est conjointement liée à l'avènement des entrepôts de données et de l'analyse en ligne (OLAP) plus particulièrement (Codd 1993).

En effet, OLAP constitue une première étape en matière d'intégration de processus d'analyse au sein des SGBD. Plusieurs travaux ont traité de l'intégration de méthodes de fouille de données. Ils portent sur des extensions du langage SQL, pour la création de nouveaux opérateurs (Meo et al. 1996, Sarawagi et al. 1998, Geist et Sattler 2002) et le développement de nouveaux langages (Han et al. 1996, Imielinski et Virmani 1999, Wang et al. 2003). Dans le même temps, quelques éditeurs de logiciels ont intégré certaines méthodes de fouille de données au sein de leur SGBD, grâce à des extensions du langage SQL et à l'utilisation d'API ("Application Programming Interface") : DB2 (IBM 2001), Oracle (Oracle 2001), SqlServer (Soni et al.2000). Contrairement aux solutions citées ci-dessus, de récents travaux concernant l'intégration des méthodes de type "arbre de décision" dans les SGBD sont proposés, en utilisant uniquement les outils offerts par ces derniers (tables, vues, ...). Pour cela, il est nécessaire d'adapter les algorithmes de fouille de données à l'environnement des SGBD. Dans (Bentayeb et Darmont 2002), les auteurs utilisent les vues relationnelles pour modéliser les arbres de décision. Avec cette approche, il est possible de traiter de grandes bases de données sans limitation de taille. Toutefois, cette approche présente des temps de traitement élevés étant données les nombreuses lectures de la base. Dans (Bentayeb et al. 2004, Udréa et al. 2004), il est proposé une amélioration de cette approche en introduisant une phase de préparation des données au cœur du SGBD, en remplaçant la table d'apprentissage initiale par la table de contingence correspondante. Les algorithmes sont alors adaptés pour s'appliquer à cette table de contingence plutôt qu'à la base initiale. L'intérêt de la table de contingence, qui contient les effectifs pré-agrégés des populations, est que sa taille est généralement inférieure à celle de la base initiale.

Dans cet article, nous proposons une nouvelle approche de fouille de données qui consiste à réduire à la fois la taille de la base d'apprentissage et les temps d'accès à la base. Pour cela, les SGBD offrent un outil performant : les index. Il s'agit d'une approche originale qui utilise les index bitmap, pour construire des arbres de décision. Le principe de notre approche se base sur les constats suivants. Lors de la construction d'un arbre de décision, on a besoin de calculer les effectifs des populations successives, au moyen d'opérations de comptage et d'opérations logiques. Or, ce sont ces mêmes opérations qui constituent les propriétés des index bitmap. Notre idée consiste donc à indexer la base d'apprentissage à la fois sur ses attributs prédictifs et sur son attribut à prédire. En effet, l'originalité de notre travail réside dans la représentation de la base d'apprentissage initiale par l'ensemble de ses index bitmap. D'autre part, nous associons à chaque noeud de l'arbre des bitmaps caractérisant ses populations correspondantes.

Les index bitmap présentent différents avantages. Tout d'abord, ils occupent un faible espace de stockage grâce à leur codage binaire. D'autre part, ils possèdent une propriété très intéressante qui consiste à répondre à certains types de requêtes sans retourner aux données sources, optimisant ainsi les temps de réponse. Cela est possible grâce aux opérations de comptage et aux opérateurs logiques tels que *AND*, *OR*, etc. Nous montrons, dans cet article, que ce type de requêtes, basées sur les index bitmap, est exactement le type de requêtes nécessaire et suffisant à la construction des arbres de décision.

Pour valider notre approche, nous avons implémenté la méthode *ID3* (Induction Decision Tree (Quinlan 1986)) au sein du SGBD Oracle, en utilisant les index bitmap,

qui sont exploités au travers de requêtes SQL. De plus, nous montrons que l'utilisation des index bitmap permet de traiter de grandes bases de données sans limitation de taille, contrairement aux méthodes classiques opérant en mémoire, et ceci, avec des temps de traitement acceptables. Nous avons également réalisé une étude de complexité pour appuyer nos résultats expérimentaux.

Cet article est organisé de la façon suivante. Dans la section 2, nous introduisons le concept d'index bitmap. Dans la section 3, nous développons notre méthode de construction d'arbres de décision en utilisant les index bitmap et montrons l'application de notre approche sur un exemple. Dans la section 4, nous présentons l'implémentation de la méthode *ID3*, les résultats obtenus, ainsi que l'étude de complexité qui valident notre approche. Pour terminer, nous concluons et indiquons quelques perspectives de recherche dans la section 5.

2 Index bitmap

2.1 Définition et exemple

Un index bitmap est une structure de données définie dans un SGBD, utilisée pour optimiser l'accès aux données dans les bases de données. C'est un type d'indexation qui est particulièrement intéressant et performant dans le cadre des requêtes de sélection. L'index bitmap d'un attribut est codé sur des bits, d'où son faible coût en terme d'espace occupé. Toutes les valeurs possibles de l'attribut sont considérées, que la valeur soit présente ou non dans la table. A chacune de ces valeurs correspond un tableau de bits, appelé bitmap, qui contient autant de bits que de n-uplets présents dans la table. Ainsi, ce type d'index est très efficace lorsque les attributs ont un faible nombre de valeurs distinctes. Chaque bit représente donc la valeur d'un attribut, pour un n-uplet donné. Pour chacun des bits, il y a un codage de présence/absence (1/0), ce qui traduit le fait qu'un n-uplet présente ou non la valeur caractérisée par le bitmap.

Pour illustrer nos propos, nous nous référerons tout au long de cet article à la table Titanic (TAB. 1). Cette table est une base d'apprentissage fréquemment utilisée pour l'application des algorithmes de fouille de données. Nous l'utilisons ici à titre d'exemple comme une base de données relationnelle (composée d'une seule table). Il s'agit d'une table qui comporte 2201 n-uplets (individus) décrits par quatre attributs *Classe*, *Age*, *Sexe* et *Survivant*. Ne sont représentés ici que les huit premiers n-uplets de la table. Ainsi, l'index bitmap construit sur l'attribut *Survivant* de cette table se présente de la façon suivante (TAB. 1).

2.2 Propriétés

Les index bitmap possèdent une propriété très intéressante qui consiste à répondre à certains types de requêtes sans retourner aux données elles-mêmes, optimisant ainsi les temps de réponse. Cela est possible grâce aux opérations de comptage ($COUNT$) et aux opérateurs logiques (*AND*, *OR*, etc) qui agissent "bit à bit" sur les bitmaps.

Considérons, par exemple, la question : "Combien y'a-t-il eu d'hommes qui ont survécu au naufrage du navire ?". Celle-ci peut être formulée par la requête SQL sui-

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
...	...	...	...

ID n-uplet	...	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
"*Non*"	...	1	0	1	0	0	0	0	0
"*Oui*"	...	0	1	0	1	1	1	1	1

TAB. 1 – Table Titanic et index bitmap construit sur l'attribut Survivant.

vante : 'SELECT COUNT(*) FROM Titanic WHERE *Survivant* = "*Oui*" AND *Sexe* = "*Homme*" '. La réponse à cette requête est donnée en se basant uniquement sur les index bitmap des attributs *Sexe* et *Survivant*, sans parcourir la table Titanic.

Cela consiste à effectuer une opération AND entre le bitmap associé à la valeur "*Oui*" pour l'attribut *Survivant* et le bitmap associé à la valeur "*Homme*" pour l'attribut *Sexe* (TAB. 2). Un comptage des "*1*" dans le bitmap résultat permet de déterminer l'effectif.

ID n-uplet	...	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Survivant="Oui"	...	0	1	0	1	1	1	1	1
Sexe="Homme"	...	0	1	1	0	1	1	1	0
AND	...	0	**1**	0	0	**1**	**1**	**1**	0

TAB. 2 – Bitmap(Survivant="Oui") AND Bitmap(Sexe="Homme").

3 Notre approche

3.1 Arbres de décision

Les arbres de décision sont des méthodes de fouille de données, qui relèvent de l'apprentissage supervisé. Le principe de construction d'un arbre de décision est décrit dans (Zighed et Rakotomalala 2000) de la façon suivante. Nous partons de l'ensemble de la population d'apprentissage initiale qui constitue le nœud racine de l'arbre. Nous disposons de p variables (attributs prédictifs), qui sont des descripteurs de cette population. Chaque individu (n-uplet) de cette population appartient à une des classes que nous connaissons préalablement. Ces classes correspondent aux différentes valeurs de l'attribut à prédire. L'objectif est de construire un arbre de décision et d'obtenir ainsi des règles, qui permettront de définir la classe d'appartenance de nouveaux individus en fonction de leurs caractéristiques. Cela revient à segmenter 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 chercher l'attribut descriptif le plus discriminant pour les classes. Par exemple, pour l'algorithme *ID3*, le pouvoir discriminant est exprimé par une variation d'entropie (en-

tropie de Shannon), que l'on appelle gain informationnel. L'attribut prédictif choisi pour la segmentation doit donc présenter un gain informationnel maximal. A chaque segmentation, les individus de la population d'un nœud père sont alors répartis dans les différents nœuds fils suivant leur valeur pour l'attribut de segmentation sélectionné. C'est ainsi que l'arbre de décision est progressivement constitué.

3.2 Arbres de décision fondés sur les index bitmap

Notre approche permet de construire des arbres de décision en exploitant uniquement les outils offerts par le SGBD, tout en répondant à un double objectif. D'une part, appliquer les algorithmes de construction d'arbres de décision sur de grandes bases de données, sans limitation de taille. Et d'autre part, obtenir des temps de traitement acceptables.

Notre premier objectif est atteint en intégrant les algorithmes de construction d'arbres de décision dans les SGBD. Et le deuxième objectif est atteint en utilisant les index bitmap. Une fois que la base d'apprentissage est indexée sur les attributs prédictifs et à prédire, l'arbre de décision est construit directement à partir de ceux-ci, ce qui évite l'accès aux données et optimise ainsi les temps de traitement.

Pour illustrer notre approche, nous avons sélectionné la méthode ID3 pour sa simplicité. Néanmoins, notre approche peut être appliquée à d'autres méthodes d'arbre de décision, telles que CART et C4.5, moyennant quelques modifications.

Dans les méthodes de type "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 populations. Dans le cas de la méthode *ID3*, l'attribut discriminant sélectionné est celui pour lequel le gain informationnel est maximal.

Chaque nœud de l'arbre est caractérisé par une distribution des effectifs de l'attribut à prédire. Notre idée consiste à utiliser les propriétés des index bitmap qui nous permettent de trouver facilement cette distribution. En effet, les effectifs à calculer pour chaque nœud peuvent être obtenus par des requêtes SQL opérant sur les index bitmap décrivant ce nœud, plutôt que sur les données elles-mêmes. Ces effectifs permettent de déterminer au préalable l'entropie de Shannon, donc le gain informationnel. Une fois l'attribut de segmentation sélectionné, la partition est construite.

3.2.1 Construction de la base d'apprentissage

Etant donnée une base d'apprentissage initiale, nous construisons les index bitmap sur tous les attributs (attributs prédictifs et attribut à prédire) de celle-ci. L'ensemble de ces index bitmap constitue alors la nouvelle base d'apprentissage.

Considérons à nouveau l'exemple de la table Titanic (TAB. 1). C'est une base d'apprentissage qui permet de déterminer si un individu aurait survécu ou non au naufrage du Titanic, en fonction de sa classe dans le navire (équipage, 1ère, 2ème ou 3ème), de son âge (enfant ou adulte) et de son sexe (homme ou femme). L'attribut à prédire est donc *Survivant* et les trois attributs prédictifs sont : *Age*, *Sexe* et *Classe*. Ainsi, dans cet exemple, la base d'apprentissage considérée est représentée dans le tableau TAB. 3. Cela revient à considérer la représentation disjonctive complète de la

population. Il s'agit en effet de coder de manière binaire chacune des valeurs distinctes de l'ensemble des attributs. Le fait de représenter la base d'apprentissage par l'ensemble de ses index bitmap permet de réduire à la fois la taille de la base d'apprentissage et les temps de traitement. Nous montrons, dans ce qui suit, comment les seules informations contenues dans les index bitmap permettent de construire les arbres de décision.

	ID n-uplet	...	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Classe	"*Equipage*"	...	1	0	0	0	0	0	0	0
	"*1ère*"	...	0	1	0	1	0	0	0	1
	"*2ème*"	...	0	0	1	0	0	1	0	0
	"*3ème*"	...	0	0	0	0	1	0	1	0
Age	"*Enfant*"	...	0	0	0	0	0	1	0	0
	"*Adulte*"	...	1	1	1	1	1	0	1	1
Sexe	"*Femme*"	...	1	0	0	1	0	0	0	1
	"*Homme*"	...	0	1	1	0	1	1	1	0
Survivant	"*Non*"	...	1	0	1	0	0	0	0	0
	"*Oui*"	...	0	1	0	1	1	1	1	1

TAB. 3 – Base d'apprentissage : ensemble des index bitmap de la base Titanic.

3.2.2 Construction de l'arbre de décision

Construction du nœud racine. Le nœud racine est caractérisé par les effectifs de l'attribut à prédire, sans tenir compte des valeurs des différents attributs prédictifs. Pour construire le nœud racine et, par conséquent, pour obtenir les différents effectifs de l'attribut à prédire, il faut effectuer un simple comptage des "*1*" sur chacun des bitmaps de l'index de l'attribut à prédire. Ainsi, dans le cas de la base Titanic (3), le nœud racine est déterminé selon la figure FIG. 1.

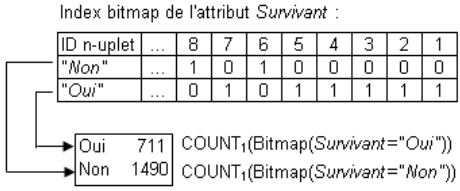

FIG. 1 – Construction du nœud racine.

Construction d'un nœud quelconque. La détermination de la distribution des effectifs de l'attribut à prédire pour un nœud quelconque se fait en trois étapes.

1. *Caractériser la population du nœud courant par un unique bitmap*
   Chacun des nœuds descendant directement de la racine est caractérisé par le bitmap correspondant à la valeur de l'attribut de segmentation qui génère ce nœud courant. Pour les autres nœuds, ce bitmap est le résultat d'une ou de plusieurs opérations *AND* qui traduisent la règle de construction du nœud, en

considérant les valeurs prises par les attributs de segmentation successifs, de la racine jusqu'au nœud courant.

2. *Caractériser la population du nœud courant pour chaque valeur de l'attribut à prédire par un bitmap résultat*
   Appliquer l'opérateur *AND* entre le bitmap caractéristique de la population du nœud courant et chacun des bitmaps de l'index de l'attribut à prédire.
3. *Déterminer les effectifs de l'attribut à prédire pour le nœud courant*
   Cela revient à faire un comptage des "*1*" dans chacun des bitmaps résultats. Ces bitmaps résultats représentent les différentes populations du nœud courant, en tenant compte de la valeur de l'attribut à prédire.

Avant la création de chaque nouvelle partition, un attribut doit être sélectionné pour effectuer la segmentation. Il correspond à celui qui dispose d'un gain informationnel maximal. Le gain informationnel correspondant à une variation d'entropie, il peut être directement calculé à partir des index bitmap. En effet, le calcul d'entropie nécessite de déterminer différents effectifs et donc d'effectuer des comptages. Comme nous avons pu le montrer précédemment, ces comptages peuvent être faits de manière efficace en utilisant les index bitmap. Ainsi, la partition optimale est donc construite en utilisant les index bitmap, non seulement lors du choix de l'attribut de segmentation à travers les calculs d'entropie mais également pour générer la partition lors de la construction des différents nœuds.

Dans notre exemple, le calcul du gain informationnel indique que l'attribut à considérer pour la segmentation du nœud racine est *Sexe*. Suivant la méthode proposée, il s'agit alors de caractériser les populations par des bitmaps, pour ensuite calculer les effectifs correspondants (TAB. 4).

ID n-uplet	...	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Sexe="Homme"	...	0	1	1	0	1	1	1	0
Survivant="Oui"	...	0	1	0	1	1	1	1	1
AND	...	0	1	0	0	1	1	1	0

ID n-uplet	...	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Sexe="Homme"	...	0	1	1	0	1	1	1	0
Survivant="Non"	...	1	0	1	0	0	0	0	0
AND	...	0	0	1	0	0	0	0	0

TAB. 4 – Bitmaps caractérisant les hommes ayant et n'ayant pas survécu.

En agissant de même pour la valeur *Femme* de l'attribut *Sexe*, on obtient alors la segmentation présentée dans la figure FIG. 2. On procède alors de la même façon pour les autres nœuds, jusqu'à obtenir l'arbre de décision final.

4 Validation

4.1 Implémentation

Pour valider notre approche, nous avons implémenté la méthode *ID3* en PL/SQL, compatible sous Oracle 9i, sans aucune extension du langage SQL. Cette implémentation

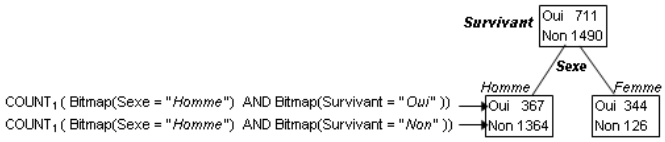

FIG. 2 – Arbre de décision obtenu après segmentation selon l'attribut Sexe.

prend la forme d'une procédure stockée nommée "ID3_Bitmap" au sein d'un package de procédures appelé "BITMAP_DECISION_TREE"[1]. Cette procédure permet d'une part de créer l'ensemble des index bitmap, et d'autre part, de construire l'arbre de décision.

Pour construire l'arbre de décision, les effectifs des différentes populations sont obtenus grâce à des requêtes SQL utilisant la fonction d'agrégat *COUNT( )*. Ces requêtes sont exécutées sans accéder aux données sources puisqu'elles sont appliquées directement sur les index bitmap. En effet, même si l'optimiseur du SGBD Oracle utilise généralement les index B-arbres pour réduire le temps de réponse lors de l'exécution de requêtes, il est possible de forcer l'utilisation des index bitmap construits.

La procédure "ID3_Bitmap" construit un arbre de décision dont la structure est stockée dans une table de résultats. Cette table contient pour chacun des nœuds créés, son identifiant, ses effectifs, ainsi que son nœud parent. L'exécution d'une requête hiérarchique sur cette table de résultats permet de retrouver la structure de l'arbre et d'en déduire les règles de type "si-alors" en parcourant tous les chemins qui relient les feuilles à la racine.

4.2 Résultats expérimentaux

Afin de comparer les performances de notre approche vis-à-vis aussi bien des méthodes de fouille de données classiques opérant en mémoire que de la méthode intégrée utilisant les vues relationnelles (Bentayeb et Darmont 2002) (approche n'utilisant pas d'index), nous avons effectué des tests sur la base Titanic (TAB. 1). Pour que les temps de traitement et les volumes de données soient significatifs, nous avons augmenté la taille de la base en dupliquant ses n-uplets. Nous avons comparé les temps de traitement de notre méthode intégrée qui utilise les index bitmap ("ID3_Bitmap") avec ceux obtenus par l'implémentation de ID3 sous le logiciel Sipina ("ID3_Sipina") d'une part, et ceux de la méthode intégrée utilisant les vues relationnelles ("ID3_Vues") d'autre part. Ces tests sont effectués sur un ordinateur PC disposant de 128 Mo de mémoire vive.

Comme nous pouvons l'observer (FIG. 3), les résultats obtenus montrent l'efficacité de la méthode "ID3_Bitmap" par rapport à "ID3_Vues", puisque le temps de traitement est en moyenne réduit de 40%. Par ailleurs, les temps pour "ID3_Bitmap" augmentent de façon linéaire avec la taille de la base, ce qui n'est pas le cas pour "ID3_Sipina". Au delà d'un certain volume de données (50 Mo dans la configuration matérielle utilisée pour les tests), Sipina est dans l'incapacité de construire l'arbre de décision demandé,

[1] http ://bdd.univ-lyon2.fr/download/decision_tree.zip

contrairement à "ID3_Bitmap". Concernant les résultats obtenus par "ID3_Sipina", nous n'avons pas introduit le temps de chargement de la base, qui peut être particulièrement long selon la taille de celle-ci. Quant aux index, nous avons constaté que le temps de création de ces derniers s'élève à 10% du temps de traitement global, mais une fois créés, seul un temps de maintenance serait à prendre en compte. Cependant, cette maintenance est indépendante du traitement, contrairement au temps de chargement de la base qui est récurrent pour les méthodes opérant en mémoire et qui fait partie intégrante du traitement.

Lorsque les données "tiennent" en mémoire, Sipina est plus performant. Ceci étant, notre approche, fondée sur les index bitmap, trouve un grand intérêt lorsque le volume de données est très important, en présentant des temps de traitement acceptables.

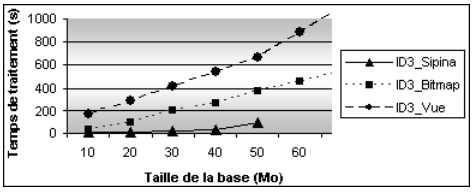

FIG. 3 – Temps de traitement en fonction de la taille de la base.

4.3 Etude de complexité

L'étude de complexité suivante nous permet de confirmer les résultats expérimentaux obtenus. Cette étude est menée d'un point de vue théorique. Nous raisonnons dans le "pire des cas", en l'occurence, nous supposons que les index ne tiennent pas en mémoire.

Soient N le nombre de n-uplets de la base initiale, K le nombre d'attributs, L la longueur moyenne en bits de chaque attribut. Soit A le nombre moyen de valeurs par attribut.

Nous nous intéressons d'abord à la taille des bases d'apprentissage. Etant données les notations adoptées, la taille de la base initiale est de $N * L * K$ bits. Pour notre approche, l'étape préalable à la construction de l'arbre est de créer la population d'apprentissage constituée de l'ensemble des index bitmaps construits sur chacun des attributs. K index bitmaps sont donc créés avec en moyenne A bitmaps par index. Chaque bitmap a une taille de N bits. La taille de l'ensemble des index bitmaps est donc de $N * A * K$ bits. En terme de taille de base d'apprentissage, et donc de temps de chargement, l'approche par index bitmaps est plus avantageuse dès lors que $A < L$, ce qui correspond à une majorité des cas.

Nous nous intéressons à présent au temps passé à la lecture des données. Nous considérons qu'un bit est lu en une unité de temps.

Le nombre total de nœuds au niveau de profondeur i peut être approximé par A^{i-1}, puisque A est le nombre moyen de valeurs des attributs. En effet, l'hypothèse posée est que l'arbre de décision est équilibré et complet.

Le nombre d'attributs prédictifs restant à considérer pour le niveau i est de $(K - i)$. La base d'apprentissage doit être lue une fois par attribut restant dans les deux approches, soit $(K - i)$ fois.

Dans la cadre de l'approche "classique", la taille de la base d'apprentissage à lire est approximée par $N * L * K$. Ainsi, au niveau i, le temps de lecture s'exprime de la façon suivante (en unités de temps) : $(K - i) * N * L * K * A^{i-1}$. On obtient alors pour la construction de l'ensemble de l'arbre un temps de : $\sum_{i=1}^{K}(K - i) * N * L * K * A^{i-1}$

Dans le cadre de l'approche utilisant les bitmaps, la taille d'un index à lire est approximée par $N * A$ bits. Au niveau de profondeur i de l'arbre, pour un attribut prédictif donné, le nombre d'index à lire pour pouvoir déterminer le niveau suivant est $(i + 1)$. Ainsi, pour le niveau i, le temps de lecture dans l'approche par index bitmap, s'exprime de la façon suivante : $(i + 1)(K - i)N.A^i$; ce qui donne pour la construction de l'ensemble de l'arbre : $\sum_{i=1}^{K}(i + 1)(K - i)N * A^i$

Pour évaluer le gain, on construit le ratio :

$$R = \frac{temps\ approche\ classique}{temps\ approche\ index\ bitmap} = \frac{\frac{KL}{A}\sum_{i=1}^{K}(K - i) * A^i}{\sum_{i=1}^{K}(K - i)(i + 1) * A^i} \quad (1)$$

Après développement, on a :

$$R = \frac{\frac{KL}{A}\sum_{i=1}^{K}(K - i) * A^i}{\sum_{i=1}^{K}(K - i) * A^i + \sum_{i=1}^{K} i(K - i) * A^i} \quad (2)$$

D'où :

$$R^{-1} = \frac{A}{KL}(1 + \frac{\sum_{i=1}^{K} i(K - i) * A^i}{\sum_{i=1}^{K}(K - i) * A^i}) = \frac{A}{KL}(1 + G) \quad (3)$$

En considérant les polynômes de plus haut degré, G est de complexité K. R^{-1} est donc de complexité $\frac{A}{L}$. En effet $R^{-1} = \frac{A}{KL}(1 + K) = \frac{A}{L}(1 + \frac{1}{K})$ et $\frac{1}{K}$ est négligeable. Notre approche par index bitmap est intéressante dans le cas où le ratio R^{-1} est inférieur à 1, donc si $A < L$. Ce qui correspond à une majorité des cas.

5 Conclusion et perspectives

Dans cet article, nous avons proposé une nouvelle approche d'intégration de méthodes de fouille de données dans les SGBD. Un apport majeur de notre approche est qu'elle offre une solution générale au problème de la fouille dans les grandes bases de données. Elle permet d'une part de traiter des bases d'apprentissage sans limitation de taille, et d'autre part d'obtenir des temps de traitement acceptables. Notre approche permet de construire des arbres de décision au travers de requêtes SQL en utilisant les index bitmap, sans extension du langage SQL, contrairement aux approches intégrées de fouille de données proposées dans la littérature. La base d'apprentissage initiale est remplacée par l'ensemble de ses index bitmap. Les effectifs des différentes populations de l'arbre de décision sont obtenus facilement par application des opérations logiques et de comptage sur les bitmaps. Ainsi, il n'est pas nécessaire d'accéder aux données de la base. Pour valider notre approche, nous avons implémenté la méthode *ID3* sous forme

d'une procédure stockée en PL/SQL, compatible sous Oracle 9i. Les tests que nous avons réalisés montrent que les temps de traitement de notre approche augmentent de façon linéaire avec la taille de la base. Mais contrairement aux méthodes de fouille opérant en mémoire, nous ne sommes pas limités par la taille des bases à traiter. En outre, notre méthode présente des temps de traitement acceptables lorsque le volume de données augmente sensiblement. Nous avons également réalisé une étude de complexité qui permet de déterminer sous quelles conditions notre approche est pertinente.

Ce travail ouvre de nombreuses perspectives de recherche. Nous projetons de tester notre approche sur des bases de données réelles et de mesurer l'impact, en terme de performance, de paramètres tels que le nombre d'attributs et leur cardinalité. Par ailleurs, notre approche peut être facilement adaptée à d'autres méthodes de type "arbres de décision". Par exemple, il suffit d'utiliser l'opérateur logique *OR* dans le cadre de méthodes utilisant le regroupement de modalités .

Références

Agrawal R., Mannila H., Srikant R. et al. (1996), Fast discovery of association rules, Advances in Kowledge Discovery and Data Mining, pp 307-328, 1996.

Bentayeb F., Darmont J. et Udréa C. (2004), Efficient integration of data mining techniques in DBMSs, 8th International Database Engineering and Applications Symposium, 2004.

Bentayeb F. et Darmont J. (2002), Decision tree modeling with relational views, XIIIth International Symposium on Methodologies for Intelligent Systems, 2002.

Chauchat J.H. et Rakotomalala R. (2000), A new sampling strategy for building decision trees from large databases, 7th Conference of the International Federation of Classification Societies, 2000.

Chaudhuri S. (1998), Data mining and database systems : Where is the intersection ?, Data Engineering Bulletin, 21(1), pp 4-8, 1998.

Codd E.F. (1993), Providing OLAP (On-Line Analytical Processing) to useranalysts : An it mandate, Technical report, 1993.

Dunkel B. et Soparkar N. (1999), Data organization and access for efficient data mining, ICDE, pp 522-529, 1999.

Gehrke J., Ramakrishnan R. et Ganti V. (1998), Rainforest - a framework for fast decision tree construction of large datasets, 24th International Conference on Very Large Data Bases, 1998.

Geist I. et Sattler K.U. (2002), Towards data mining operators in database systems : Algebra and implementation, 2nd International Workshop on Databases, Documents, and Information Fusion - Information Integration and Mining in Databases and on the Web, 2002.

Han J., Fu Y., Wang W., Koperski K., et Zaiane O. (1996), DMQL : A data mining query language for relational databases, SIGMOD, Workshop on Research Issues in Data Mining and Knowledge Discovery, 1996.

IBM (2001), Db2 intelligent miner scoring, http ://www-306.ibm.com/software/data/iminer/scoring/, 2001.

Imielinski T. et Virmani A. (1999), Msql : A query language for database mining, DataMining and Knowledge Discovery : An International Journal, 3, pp 373-408, 1999.

Lia H. et Motoda H. (1998), Feature selection for knowledge discovery and data mining, Kluwer Academic Publishers, 1998.

Meo R., Psaila G. et Ceri S. (1996), A new SQL-like operator for mining association rules, VLDB Journal, pp 122-133, 1996.

Oracle (2001), Oracle 9i data mining, White paper, 2001.

Quinlan J.R. (1986), Induction of decision trees. Machine Learning, 1, pp 81-106, 1986.

Ramesh G., Maniatty W.A. et Zaki M. (2002), Indexing and data access methods for database mining, ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2002.

Sarawagi S., Thomas S. et Agrawal R. (1998), Integrating mining with relational database systems : Alternatives and implications, ACM SIGMOD International Conference on Management of Data, 1998.

Soni S., Tang Z. et Yang J. (2001), Performance study microsoft data mining algorithms, Technical report, Microsoft Corp., 2001.

Toivonen H. (1996), Sampling large databases for association rules, International Conference on Very Large Data Bases, 1996.

Udréa C., Bentayeb F., Darmont J. et Boussaid O. (2004), Intégration efficace de méthodes de fouille de données dans les SGBD, 4èmes Journées Francophones d'Extraction et de Gestion des Connaissances, 2004.

Wang H., Zaniolo C. et Luo C.R. (2003), Atlas : a small but complete SQL extension for data mining and data streams, Proceedings of the 29th VLDB Conference, 2003.

Zighed D.A. et Rakotomalala R. (2000), Graphes d'induction. Apprentissage et Data Mining, Hermes Science Publication, 2000.

Summary

We propose in this paper an original method to apply data mining algorithms, namely decision tree-based methods, taking into account not only the size of processed databases but the processing time too. Our key idea consists in constructing a decision tree, within the DBMS, using bitmap indices. Indeed bitmap indices have many useful properties such as the count and bite-wise operations that we show are sufficient to build decision trees. In addition, by using bitmap indices, the DBMS does not need to access raw data. This implies clear improvements in terms of processing time.