# Bitmap Index-Based Decision Trees

Cécile Favre and Fadila Bentayeb

ERIC - Université Lumière Lyon 2,
Bâtiment L, 5 avenue Pierre Mendès-France
69676 BRON Cedex – FRANCE
{cfavre, bentayeb}@eric.univ-lyon2.fr

**Abstract.** In this paper we propose an original approach to apply data mining algorithms, namely decision tree-based methods, taking into account not only the size of processed databases but also the processing time. The 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 bit-wise operations. We will show that these operations are efficient to build decision trees. In addition, by using bitmap indices, we don't need to access raw data. This implies clear improvements in terms of processing time.

**Keywords:** Bitmap indices, databases, data mining, decision trees, performance.

## 1 Introduction

Mining large databases efficiently has been the subject of many research studies during the last years. However, in practice, the long processing time required by data mining algorithms remains a critical issue. Indeed, traditional data mining algorithms operate on main memory, which limits the size of the processed databases. There are three approaches to overcome this limit. The first way consists in preprocessing data by using feature selection [11] or sampling [4, 18] techniques. The second way is to increase the scalability, by optimising data accesses [6, 15] or algorithms [1, 7]. The third way consists in integrating data mining methods within DataBases Management Systems (DBMSs) [5]. Many integrated approaches have been proposed. They usually use SQL extensions for developing new operators [8, 12, 16] or new languages [9, 10, 19]. This trend has been confirmed with the integration of data mining tools in commercial solutions [13, 17], but these are real "black boxes" requiring also the use of Application Programming Interfaces (APIs).

In opposition to these different solutions, recent works have proposed to integrate decision tree-based methods within DBMSs, using only the tools offered by these latter. The analogy is made between building successive partitions, representing different populations, and successive relational views modeling the decision tree [2]. Building views tend to increase processing time due to multiple accesses. This can be improved by using a contingency table as proposed in [3].

M.-S. Hacid et al. (Eds.): ISMIS 2005, LNAI 3488, pp. 65–73, 2005.

Instead of applying data mining algorithms to the training set, we apply them to the contingency table which is much smaller. In this paper, we propose to improve processing time by reducing not only the size of the training set, but also the data accesses. Indeed, our approach is to use bitmap indices to represent the training set and apply data mining algorithms on these indices to build our decision tree. The population frequencies of a given node in the decision tree are obtained by applying counting and low-cost boolean operations on selected bitmap indices. To run these operations, the DBMS does not need to access raw data. Thus, in opposition to the integrated approaches proposed in the literature, our approach presents three advantages: (1) no extension of the SQL language is needed; (2) no programming through an API is required and (3) no access data sources is necessary. To validate our approach, we implemented the ID3 (Induction Decision Tree) method [14] as a PL/SQL stored procedure, named *ID3_Bitmap*. To prove that our implementation of ID3 works properly, we successfully compared the output of our procedure with the output of an existing and validated data mining in-memory software, Sipina [20]. Moreover, we showed that with bitmap indices we did not have a size limit when dealing with large databases as compared to in-memory software, while obtaining linear processing times.

The remainder of this paper is organized as follows. First, we present the principle of bitmap indices in Section 2. In Section 3, we present our bitmap-based approach for decision tree-based methods. Section 4 details our implementation of ID3 method, presents the experimental results and the complexity study. We finally conclude this paper and provide future research directions in Section 5.

## 2 Bitmap Indices

A bitmap index is a data structure used to efficiently access large databases. Generally, the purpose of an index is to provide pointers to rows in a table containing given key values. In a common index, this is achieved by storing a list of rowids for each key corresponding to the rows with that key value. In a bitmap index, records in a table are assumed to be numbered sequentially from 1. For each key value, a bitmap (array of bits) is used instead of a list of rowids. Each bit in the bitmap corresponds to an existing rowid. If the bit is set to "*1*", it means that the row with the corresponding rowid contains the key value ; otherwise the bit is set to "*0*". A mapping function converts the bit position to an actual rowid, so the bitmap index provides the same functionality as a regular index even though it uses a different representation internally.

### 2.1 Example

To illustrate how bitmap indices work, we use the Titanic database as an example. Titanic is a table containing 2201 tuples described by four attributes *Class*, *Age*, *Gender* and *Survivor* (Table 1). A bitmap index built on the *Survivor* attribute is presented in Table 2.

**Table 1.** Titanic Database

| Class | Age | Gender | Survivor |
|---|---|---|---|
| 1st | Adult | Female | Yes |
| 3rd | Adult | Male | Yes |
| 2nd | Child | Male | Yes |
| 3rd | Adult | Male | Yes |
| 1st | Adult | Female | Yes |
| 2nd | Adult | Male | No |
| 1st | Adult | Male | Yes |
| Crew | Adult | Female | No |
| Crew | Adult | Female | Yes |
| 2nd | Adult | Male | No |
| 3rd | Adult | Male | No |
| Crew | Adult | Male | No |
| ... | ... | ... | ... |

**Table 2.** *Survivor* Bitmap Index

| | Rowid | .. | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Survivor | No | .. | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | Yes | .. | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |

## 2.2 Properties

Bitmap indices are designed for efficient queries on multiple keys. Hence, queries are answered using bit-wise operations such as intersection (AND), and union (OR). Each operation takes two bitmaps of the same size and the operator is applied on corresponding bits. In the resulting bitmap, every "1" bit marks the desired tuple. Thus counting the number of tuples is even faster. For some select queries "SELECT COUNT()...WHERE ... AND ...", those logical operations could provide answers without returning to data sources. In addition to standard operations, the SQL engine can use bitmap indices to efficiently perform special set-based operations using combinations of multiple indices. For example, to find the total number of *"male survivors"*, we can simply perform the logical AND operation between bitmaps representing *Survivor*=" *Yes*" and *Gender*=" *Male*", then count the number of "*1*" (Table 3).

**Table 3.** Bitmap(`Survivor`=*"Yes"*) **AND** Bitmap(`Gender`=*"Male"*)

| Rowid | .. | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Survivor*=" *Yes*" | .. | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| *Gender*=" *Male*" | .. | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| **AND** | .. | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | **1** | **1** | **1** | 0 |

## 3 Bitmap-Based Decision Trees Methods

Induction trees are among the most popular supervised data mining methods proposed in the literature. They may be viewed as a succession of smaller and smaller partitions of an initial training set. They take as input a set of objects (tuples, in the relational databases vocabulary) described by a collection of predictive attributes. Each object belongs to one of a set of mutually exclusive classes (attribute to be predicted). Induction tree construction methods apply successive criteria on the training population to obtain these partitions, wherein the size of one class is maximized. In the ID3 algorithm, the discriminating power of an attribute for segmenting a node of the decision tree is expressed by a variation of entropy and more precisely, its entropy of Shannon.

### 3.1 Principle

In this paper, we propose to integrate decision tree-based methods within DBMSs. To achieve this goal, we only use existing structures, namely, bitmap indices, that we exploit through SQL queries. Bitmap indices improve select queries performance by applying count and bit-wise logical operations such as AND. This type of queries coincides exactly to those we need to build a decision tree and more precisely to define the nodes' frequencies. Indeed, as we have shown in Table 3, to find the total number of the *"male survivors"*, SQL engine performs logical AND and COUNT operations onto bitmap indices and gets the result. In the case of a decision tree-based method, this query may correspond to a segmentation step for obtaining the population frequency of the class *Survivor*="*Yes*" in the node *Gender*="*Male*".

To show that our approach is efficient and relevant, we introduced the use of bitmap indices into the ID3 method. This induces changes in the computation of the information gain for each predictive attribute. Thus, in our approach, for an initial training set, we create its associated set of bitmap indices for predictive attributes and the attribute to be predicted. Then, the ID3 algorithm is applied onto the set of bitmap indices rather than the whole training set. For the root node of the decision tree, the population frequencies are obtained by simply counting the total number of "*1*" in the bitmaps built on the attribute to be predicted. For each other node in the decision tree, we generate a new set of bitmaps, each one corresponding to the class in the node. The bitmap characterizing each class in the current node is obtained by applying the AND operation between the bitmap associated to the current node and the bitmaps corresponding to the successive nodes from the root to the current node. To get the population frequency of each class in this node, we count the total number of "*1*" in the resulting bitmap. Since the information gain is based on population frequencies, it is also computed with bitmap indices.

### 3.2 Running Example

To illustrate our approach, let us take the Titanic database as an example (Table 1). The aim is to predict which classes of the Titanic passengers are

more likely to survive the wreck. Those passengers are described by different attributes which are: *Class*={*1st*; *2nd*; *3rd*; *Crew*}; *Age*={*Adult*; *Child*}; *Gender*={*Female*; *Male*} and *Survivor*={*No*; *Yes*}.

For each predictive attribute (*Gender*, *Class* and *Age*) and the attribute to be predicted (*Survivor*), we create its corresponding bitmap index (Table 4). Thus, our learning population is precisely composed of these four different bitmap indices. Hence, we apply the decision tree building method directly on this set of bitmap indices.

**Table 4.** Bitmap Indices for Titanic's Database

| | Rowid | .. | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | Crew | .. | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1st | .. | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| | 2nd | .. | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| | 3rd | .. | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| Age | Child | .. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | Adult | .. | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| Gender | Female | .. | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | Male | .. | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| Survivor | No | .. | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | Yes | .. | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |

To obtain the root node of the decision tree, we have to determine the population frequency of each class. In our running example, these frequencies are obtained by counting the number of "*1*" in the bitmaps associated to *Survivor*= "*Yes*" and *Survivor*= "*No*" respectively (Fig. 1).

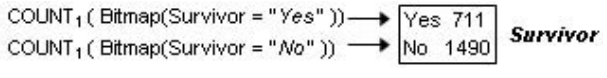


**Fig. 1.** Root node

Then, the variation of entropy indicates that the segmentation attribute is *Gender*. The population of the root node is then divided into two sub-nodes corresponding to the rules *Gender*= "*Male*" and *Gender*= "*Female*" respectively. Each sub-node is composed of two sub-populations, those that survived and those that did not. To obtain the population frequencies of the node *Gender*= "*Male*" , we apply the logical operation AND firstly between the *Gender*= "*Male*" bitmap and the *Survivor*= "*Yes*" bitmap and secondly between the *Gender*= "*Male*" bitmap and the *Survivor*= "*No*" bitmap. Then we count the total number of "*1*" in the corresponding *And_bitmap* (Table 5). The same process is performed for the node *Gender*="*Female*" (Fig. 2). This process is repeated for all the other predictive attributes to obtain the final decision tree.

**Table 5.** AND-bitmaps for the node *Gender*= "*Male*"

| Rowid | .. | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Survivor*="*Yes*" | .. | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| *Gender*="*Male*" | .. | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| **AND** | .. | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **1** | **1** | **1** | **0** |
| *Survivor*="*No*" | .. | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| *Gender*="*Male*" | .. | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| **AND** | .. | **1** | **1** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** |

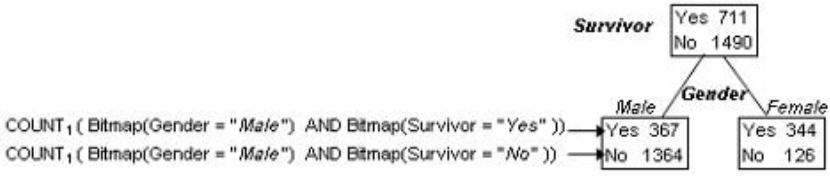


**Fig. 2.** Nodes of the first level of the decision tree

## 4 Validation

### 4.1 Implementation

We have implemented the ID3 method using bitmap indices as a PL/SQL stored procedure, namely *ID3_Bitmap*, under Oracle 9i, which is part of a broader package named *decision_tree* that is available on-line[1]. This stored procedure allows us to create the necessary bitmap indices for a given training set and then build the decision tree. Since Oracle uses by default B-Tree indices, we forced it to use bitmap indices. The nodes of the decision tree are built by using an SQL query that is based on the AND operation applied on its own bitmaps and its parent bitmaps. Then, the obtained *And_bitmaps* are used to count the population frequency of each class in the node with simple COUNT queries. These counts are used to determine the criterion that helps either partitioning the current node into a set of disjoint sub-partitions based on the values of a specific attribute or concluding that the node is a leaf, i.e., a terminal node. In the same way, to compute the information gain for a predictive attribute, our implementation uses bitmap indices rather than the whole training set.

### 4.2 Experimental Results

In order to validate our bitmap-based approach and compare its performances to classical in-memory data mining approaches, we tested it on the Titanic training set (Table 1). To have a significant database size, we duplicated the records of Titanic database. Moreover, these tests have been carried out in the same environment: a PC with 128 MB of RAM and the Personal Oracle DBMS version

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

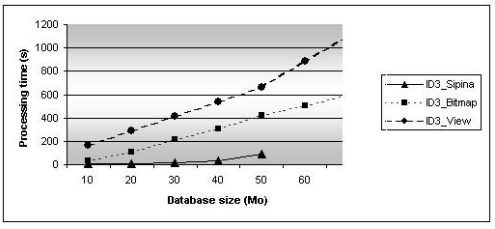


**Fig. 3.** Processing time with respect to database size

9i (Fig. 3). First, we compared the processing times of $ID3\_Bitmap$ to those of a classical in-memory implementation of ID3 available in the Sipina data mining software [20]. We clearly observe that $ID3\_Bitmap$ allowed us mining large databases without size limit comparing with $ID3\_Sipina$, while obtaining linear processing times with respect to the database size. For databases over 50 Mo, with the hardware configuration used for the tests, Sipina is unable to build a decision tree as compared to our method. Secondly, we compared the processing times of our $ID3\_Bitmap$ with the view-based approach $ID3\_View$ [2]. The results we obtain clearly underline the gain (40%) induced by $ID3\_Bitmap$, that has a much lower processing time than $ID3\_View$ on an average.

### 4.3 Complexity Study

Our objective is to confirm, from a theoretical point of view, the results obtained by our approach. For this study we place ourselves in the worst case, i.e. the indices are too large to be loaded in memory.

Let N be the total number of tuples in the training set, K the number of attributes, L the average length, in bits, of each attribute and A the average number of values of each attribute.

First we evaluate the size of training sets. The size of the initial training set is $N * L * K$ bits. For our bitmap-based approach, this initial training set is replaced by the set of bitmap indices. Thus K bitmap indices are created with an average number of A bitmaps for each index. Each bitmap has a size of N bits. In this case, the size of the training set is $N * A * K$ bits. As regards to the size of the training set and thus the loading time, our approach is preferable if $A < L$, which corresponds to a majority of cases.

In terms of time spent to data reading, we consider that a bit is read in one time unit.

The total number of nodes on the $i^{th}$ depth level can be approximated by $A^{i-1}$. Indeed we suppose that the obtained decision tree is complete and balanced. To reach level $i+1$ from an unspecified level $i$ of the tree, each training set must be read as many times as there are predictive attributes remaining at this level, i.e. $(K-i)$.

In the classical approach, as the size of the training set is $N * L * K$, reading time for level $i$ (in time units) is $(K-i) * N * L * K * A^{i-1}$. Hence, to build the

whole decision tree, in the classical approach, the reading time is : $\sum_{i=1}^{K}(K - i) * N * L * K * A^{i-1}$.

In our bitmap index-based approach, the index size is approximated by $N * A$ bits. To reach level $i + 1$ from an unspecified level $i$ of the tree, for a given predictive attribute, the number of index to read is $i + 1$. Thus, at level i, the reading time is : $(i + 1)(K - i)N * A^i$. Hence, to build the whole decision tree with our bitmap index-based approach, the reading time is : $\sum_{i=1}^{K}(i + 1)(K - i)N * A^i$.

To evaluate the gain in time, we build the following ratio:

$$R = \frac{time\ with\ classical\ approach}{time\ with\ bitmap\ index-based\ approach} = \frac{\frac{KL}{A}\sum_{i=1}^{K}(K-i)*A^i}{\sum_{i=1}^{K}(K-i)(i+1)*A^i}.$$

After computing we obtain : $R = \frac{\frac{KL}{A}\sum_{i=1}^{K}(K-i)*A^i}{\sum_{i=1}^{K}(K-i)*A^i+\sum_{i=1}^{K}i(K-i)*A^i}$

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

As we consider the polynomials of higher degree, G is of complexity K. Thus $R^{-1}$ is of complexity $\frac{A}{L}$. Indeed $R^{-1} = \frac{A}{KL}(1 + K) = \frac{A}{L}(1 + \frac{1}{K})$ and $\frac{1}{K}$ is insignificant. Our approach is interesting if the ratio $R^{-1}$ is lower than one, that means if $A < L$, which corresponds to a majority of the cases.

## 5 Conclusion and Perspectives

These last years, an important research effort has been made to apply efficiently data mining methods on large databases. Traditional data mining algorithms operate on main memory, which limits the size of the processed databases. Recently, a new approach has emerged. It consists in integrating data mining methods within DBMSs using only the tools offered by these latter. Following the integrated approach, we proposed in this paper an original method for applying decision trees-based algorithms on large databases. This method uses bitmap indices which have many useful properties such as the count and the bit-wise operations that can be used through SQL queries. Our method has two major advantages: it is not limited by the size of the main memory and it improves processing times because there is no need to access data sources since our method uses bitmap indices rather than the whole training set. To validate our approach, we implemented the ID3 method, under the form of a PL/SQL stored procedure into the Oracle DBMS. We showed that our bitmap-based implementation allowed us mining bigger databases as compared to the in-memory implementations while obtaining linear processing times with respect to the database size.

There are different perspectives opened by this study. We plan to implement other data mining methods using bitmap indices. More precisely, extending our approach to deal with the OR operator in the case for example of the CART method allowing grouping attribute values into classes. Moreover, we intend

to compare the performances of these implementations to their equivalent in-memory software on real-life databases.

## References


1. R. Agrawal, H. Mannila, R. Srikant, and et al. Fast discovery of association rules. In *Advances in Kowledge Discovery and Data Mining*, pages 307–328, 1996.
2. F. Bentayeb and J. Darmont. Decision tree modeling with relational views. In *XIIIth International Symposium on Methodologies for Intelligent Systems (ISMIS 02), France*, 2002.
3. F. Bentayeb, J. Darmont, and C. Udréa. Efficient integration of data mining techniques in dbmss. In *8th International Database Engineering and Applications Symposium (IDEAS 04), Portugal*, 2004.
4. J. H. Chauchat and R. Rakotomalala. A new sampling strategy for building decision trees from large databases. In *7th Conference of the International Federation of Classification Societies (IFCS 00), Belgium*, 2000.
5. S. Chaudhuri. Data mining and database systems: Where is the intersection? *Data Engineering Bulletin*, 21(1):4–8, 1998.
6. B. Dunkel and N. Soparkar. Data organization and access for efficient data mining. In *ICDE*, pages 522–529, 1999.
7. J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest - a framework for fast decision tree construction of large datasets. In *24th International Conference on Very Large Data Bases (VLDB 98), USA*, 1998.
8. I. Geist and K. U. Sattler. Towards data mining operators in database systems: Algebra and implementation. 2nd International Workshop on Databases, Documents, and Information Fusion (DBFusion 2002).
9. J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational databases. In *SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96)*, Canada, 1996.
10. T. Imielinski and A. Virmani. Msql: A query language for database mining. *DataMining and Knowledge Discovery : An International Journal*, 3:373–408, 1999.
11. H. Liu and H. Motoda. *Feature Selection for knowledge discovery and data mining.* Kluwer Academic Publishers, 1998.
12. R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *The VLDB Journal*, pages 122–133, 1996.
13. Oracle. Oracle 9i data mining. White paper, June 2001.
14. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
15. G. Ramesh, W. Maniatty, and M. Zaki. Indexing and data access methods for database mining, 2001.
16. S. Sarawagi, S. Thomas, and R. Agrawal. Integrating mining with relational database systems: Alternatives and implications. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 98), USA*, 1998.
17. S. Soni, Z. Tang, and J. Yang. Performance study microsoft data mining algorithms. Technical report, Microsoft Corp., 2001.
18. H. Toivonen. Sampling large databases for association rules. In *In Proc. 1996 Int. Conf. Very Large Data Bases*, 1996.
19. H. Wang, C. Zaniolo, and C. R. Luo. Atlas : a small but complete sql extension for data mining and data streams. In *29th VLDB Conference*. Germany, 2003.
20. D. A. Zighed and R. Rakotomalala. Sipina-w(c) for windows: User's guide. Technical report, ERIC laboratory, University of Lyon 2, France, 1996.