
ELSEVIER

Data & Knowledge Engineering 37 (2001) 47–64

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

TBAR: An efficient method for association rule mining in relational databases

Fernando Berzal *, Juan-Carlos Cubero, Nicolás Marín, José-María Serrano

Dpto. de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Avenida de Andalucia 38, 18071 Granada, Spain

Received 12 March 2000; received in revised form 18 October 2000; accepted 29 November 2000

Abstract

In this paper, we propose a new algorithm for efficient association rule mining, which we apply in order to discover interesting patterns in relational databases. Our algorithm, which is called Tree-Based Association Rule mining (TBAR), redefines the notion of item and employs an effective tree data structure. It can also use techniques such as Direct Hashing and Pruning (DHP). Experiments with real-life datasets show that TBAR outperforms Apriori, a well-known and widely used algorithm.

Keywords: Data mining; Association rules; Relational databases

1. Introduction

KDD, which stands for Knowledge Discovery in Databases, has been defined as the non-trivial extraction of potentially useful information from a large volume of data where the information is implicit (although previously unknown).

Data Mining is a generic term which covers research results, techniques and tools used to extract useful information from large databases. Data Mining algorithms are included in the KDD process, which also involves data preprocessing and result interpretation. It should be noted that KDD is more general than Data Mining. If KDD were reduced to Data Mining, it would be no more than data dredging and not Knowledge Discovery.

Association rules are one of the most popular Data Mining techniques. They are particularly useful for discovering relationships among data in huge databases.

Association rule mining has been traditionally applied to databases of sales transactions (referred to as basket data). A transaction T is a set of items and contains a set of items I if $I \subseteq T$. Such a set of items is called k-itemset, where k is the number of items in the set.

* Corresponding author. Tel.: +34-58-244-078; fax: +34-58-243-317.

E-mail addresses: fberzal@decsai.ugr.es (F. Berzal), jc.cubero@decsai.ugr.es (J.-C. Cubero), nicm@decsai.ugr.es (N. Marín), jmserrano@decsai.ugr.es (J.-M. Serrano).

0169-023X/01/$ - see front matter
PII: S0169-023X(00)00055-0

An association rule is an implication $X \Rightarrow Y$, where X and Y are itemsets with no items in common. The intuitive meaning of such a rule is that the transactions (or tuples) that contain X also tend to contain Y. The rule $X \Rightarrow Y$ holds with confidence c if $c\%$ of transactions (or tuples) that contain X also contain Y. The rule $X \Rightarrow Y$ has support s if $s\%$ of the transactions (or tuples) in the database contain $X \cup Y$.

Given a database, the problem of mining association rules is to generate all the association rules that have support and confidence greater than the user-specified minimum thresholds *MinSupport* and *MinConfidence*, respectively. The association rule mining problem is usually broken down into two subproblems:

- Finding all the frequent itemsets (whose support is greater than the user-specified minimum support threshold), also called covering or large itemsets in the literature. This problem can be solved by constructing a candidate set of potentially frequent itemsets and identifying the frequent itemsets within this candidate set. The size of the considered itemsets is progressively increased until no more frequent itemsets can be found.
- Generating the association rules derived from the frequent itemsets. If $X \cup Y$ and X are frequent itemsets, the rule $X \Rightarrow Y$ holds if the ratio of $support(X \cup Y)$ to $support(X)$ is, at least, as large as the minimum confidence threshold. It should be noted that the rule will have minimum support because $X \cup Y$ is frequent.

Several algorithms have been proposed to solve the problem of association rule mining, from AIS [3] and SETM [16] to Apriori [6] and all its variants [9,18]. Here we will discuss Apriori, a landmark in association rule mining, and Direct Hashing and Pruning (DHP) [18] an algorithm developed from Apriori. The Offline Candidate Determination (OCD) algorithm [17] is similar to the Apriori algorithm but less efficient. See [14] for a comparison of selected algorithms, and [2] for a more extensive survey of the field. A more rigorous treatment of the algorithmic aspects of association rule mining has recently been published [15]. More references can be found in the aforementioned papers.

In this paper, we will place special emphasis on the application of association rule mining to relational databases, since a large number of relational systems are now in the market and most distributed database systems are also relational. This makes relational databases a good target for Data Mining.

The notion of item must be redefined in a relational database. Henceforth an item will be a pair $a : v$, where a is an attribute (a column in a relational table) and v is the value of a. A tuple t contains an item $a : v$ if its column a has the value v. A tuple t contains an itemset I if it contains all the items in itemset I.

A fundamental property of the itemsets derived from a relational table is that they cannot contain more than one item per table column. All items in any given itemset must belong to different table columns. In other words, if $a_1 : v_1$ and $a_2 : v_2$ belong to an itemset I, with $v_1 \neq v_2$, then $a_1 \neq a_2$. This is just a consequence of the First Normal Form (1NF): a relation is in 1NF if all its attribute domains contain only atomic values. The property above will allow us to prune the candidate set during itemset generation and justifies our distinction between items in transactional databases and items in relational databases.

The Data Mining algorithms discussed in this paper are independent of the type of database which the KDD process is applied to. These algorithms could be applied to a retail organization's large database of sales transactions or to a typical relational database (to a simple table or to the result of a relational expression) with minimal modifications.

The rest of the paper is organized as follows. In Section 2, we give a brief description of Apriori and DHP, two well-known algorithms widely used in association rule mining. We then propose our algorithm, which we have called TBAR, in Section 3. In Section 4, we discuss the performance results obtained; and, finally, we present our conclusions and suggestions in Section 5.

2. Background

The most time-consuming part of the process is to discover frequent itemsets while the generation of association rules given the frequent itemsets is straightforward enough. We will therefore focus our attention on the discovery of frequent itemsets.

Let L_k denote the set of all frequent k-itemsets, where L stands for large, and let C_k be the set of candidate k-itemsets (i.e. potentially frequent k-itemsets).

2.1. Apriori algorithm

The Apriori algorithm [6] makes multiple passes over the data to find frequent itemsets. In the kth pass the algorithm finds all frequent k-itemsets. Each pass consists of two phases: the candidate generation phase to obtain C_k from L_{k-1}, and the support counting phase to find the frequent itemsets L_k. The algorithm terminates once L_k, or C_k, is empty. It is assumed that the items in an itemset are lexicographically ordered. A hash-tree data structure is used to store and manage C_k efficiently.

It should be noted, however, that the way Apriori finds all the frequent itemsets is not the only possible approach to solving this problem. In [13], for example, there is no need to build a candidate set of potentially relevant itemsets.

2.1.1. Candidate generation

In the candidate generation phase, the set of all $(k-1)$-itemsets found in the $(k-1)$th pass is used to generate the candidate itemsets C_k. C_k is necessarily a superset of L_k, the set of all frequent k-itemsets. Since all subsets of a frequent itemset are also frequent, C_k can be obtained from L_{k-1} as follows:

- *Join step*: A superset of C_k is created by joining L_{k-1} with itself, which can be done effectively when items and itemsets are lexicographically ordered. Candidate k-itemsets are then obtained by the natural join $L_{k-1} \bowtie L_{k-1}$ on the first $k-2$ items of L_{k-1}.
- *Prune step*: All itemsets $c \in C_k$ with some $(k-1)$-subsets not in L_{k-1} are deleted. After the join step, we know that two of the $(k-1)$-subsets of the itemsets in C_k are already frequent. The remaining $k-2$ subsets must be checked by using additional joins.

2.1.2. Support counting

The database is scanned in the support counting phase. For each transaction, the candidates in C_k also contained in the transaction are determined and their support count is increased by one. At the end of the scan, the support counts are examined to determine which candidates are frequent. A hash-tree data structure is used to count the occurrences of every itemset efficiently.

2.1.3. Mining association rules in relational databases with Apriori

Minor modifications are needed to adapt Apriori to the problem of mining association rules in relational databases. Using the fundamental property of the itemsets in relational databases (i.e.

they cannot contain more than one item per table column), the join step in the candidate generation phase is adapted to reflect this and to prune the candidate set by not taking into account itemsets which are not in 1NF.

2.2. Direct hashing and pruning

The DHP algorithm [18] was devised from Apriori. Like this algorithm, DHP uses a hash-tree data structure to store itemsets. In addition, this algorithm employs a hash table for $(k+1)$-itemsets when counting k-itemsets. Each bucket of the hash table contains the sum of the number of occurrences for all existing $(k+1)$-itemsets with the same hash value. When creating C_{k+1}, given a $(k+1)$-itemset, if the count stored in the bucket corresponding to that itemset is less than the minimum support threshold (*MinSupport*) then the itemset is not included in C_{k+1}, since it cannot be frequent.

The use of such a hash table during the first iterations allows smaller candidate sets to be generated. It is important to remember that determining the frequent itemsets from a huge amount of candidate itemsets is the most time-consuming part of the process.

For later iterations, Apriori may be used to reduce the overhead caused by using the additional hash table. In any case, the improvements in performance obtained with this technique depend heavily on the nature of the available dataset. These will be remarkable in typical databases of sales transactions but could be null in a relational table with data collected to solve a classification problem.

Moreover, DHP reduces the size of the database at each iteration. If a transaction contains one frequent $(k+1)$-itemset, it must contain at least $k+1$ frequent k-itemsets. Those transactions that cannot contain $(k+1)$-itemsets can be discarded, thus reducing the size of the database in future iterations. It should be noted that this transaction trimming might not be applicable if there is not enough space available to store the temporary databases used in each iteration.

3. $\overline{T}$ (TBAR): Tree-based association rule mining

TBAR, which stands for Tree-Based Association Rule mining, employs a different data structure to represent the sets of candidate and frequent itemsets in order to reduce the computing resources needed by the association rule mining algorithm (both CPU time and storage requirements). TBAR stores all the itemsets found in a tree data structure which is similar to a set-enumeration tree. We will first briefly describe the TBAR algorithm in Section 3.1 and we will then examine the tree data structure in greater depth in Section 3.2.

3.1. TBAR overview

The TBAR algorithm follows the same philosophy as most association rule mining algorithms: first it finds frequent itemsets, and then it generates association rules with the previously found itemsets. The introduction of a new data structure to manage the itemsets, which we have called the itemset tree, is of prime importance.

3.1.1. Relevant itemsets

The first step in our data mining algorithm is to discover potentially interesting patterns (also called frequent, covering or large itemsets in the literature).

Within the standard association rule mining framework, we must find all the itemsets whose support is at least *MinSupport*, although other interest measures could be used. The large itemset method has in fact been heavily criticized [1,2].

It is because of these criticisms that we refer to such potentially interesting itemsets as 'relevant itemsets' instead of 'frequent itemsets'. We will use the following algorithm to find all the relevant itemsets:

```
set.Init(MinSupport);
itemsets=set.Relevants(1);
k=2;
while (k ⩽ columns && itemsets ⩾ k) {
      itemsets=set.Candidates(k);
      if (itemsets>0)
        itemsets=set.Relevants(k);
      k++;
}
```

where `set` denotes the itemset tree data structure. The method `Init` creates and initializes the data structure. `Relevants(k)` generates L_k while `Candidates(k)` creates C_k from L_{k-1}. Both methods return the number of itemsets in the last generated set. All these methods will be described more thoroughly in Section 3.2.

It should be noted that, in a relational database, the maximum number of items in an itemset is equal to the number of columns in the relational table (or the number of columns in the result of a relational expression).

Obviously, you cannot expect to obtain a relevant k-itemset from an empty candidate set since C_k must be a superset of L_k.

Furthermore, you cannot find a relevant k-itemset if you do not have at least k relevant $(k-1)$-itemsets because every subset of a relevant itemset is also a relevant itemset. This property is exploited by Apriori [6] and OCD [17] to reduce the size of C_k during the candidate generation phase.

3.1.2. Association rules

Once all the relevant itemsets have been found, the association rules derived from them can be obtained by the proper traversal of the itemset tree. The user-specified threshold *MinConfidence* is employed to obtain all the association rules with enough confidence to be considered potentially interesting. Of course, the format of the association rules or the itemsets involved at this stage can also be restricted to obtain only some of the association rules which can be derived from the itemsets in the itemset tree (see [7,22] for related information).

The association rule generation given the relevant itemset tree is performed by the `Rules` method of the TBAR data structure, which will be described in the following section.

3.2. TBAR data structure

A unique tree is used to store all the itemsets found during the Data Mining process. This compact representation provides substantial storage savings when compared to the Apriori algorithm.

We will use an example to illustrate the data structure employed by TBAR. Suppose that, after some preprocessing, you have obtained the simple dataset shown in Table 1 and that *MinSupport* is set at 40% (i.e. 2 tuples out of 5). From the previous dataset, the relevant itemsets presented in Table 2 can be obtained.

These sets of itemsets can be represented by a set-enumeration tree, and a compact representation of such a tree is shown in Fig. 1. This kind of tree requires less memory space and is more suitable for our algorithm purposes.

It should be noted that the items are lexicographically ordered, as in Apriori. The number of k-itemsets contained in the itemset tree equals the number of items located at level $L[k]$. The corresponding k-itemsets can be retrieved by concatenating the items found in the paths from the root of the tree to each item located at level $L[k]$. The support of any k-itemset is stored in the tree accompanying the kth item of the itemset.

A hash table can be used within a node for optimization purposes when the number of items in a node is greater than a given threshold. Such a hash table can be indexed by the pair $a : v$. This hashing technique allows efficient access to the itemsets stored in the itemset tree and is analogous to the one used in a hash tree by the Apriori algorithm.

The candidate generation phase of the algorithm is quite simple. In order to generate C_{k+1}, you must create a child node for each item $a : v$ at level $L[k]$, and put in it all the items which appear at

Table 1
A simple dataset derived from a real-life database (the information is encoded)

A	B	C
0	0	0
0	0	1
0	1	1
1	1	1
1	1	1

Table 2
Relevant itemsets derived from dataset in Table 1

Set $L[k]$	Itemsets {items} [support]	Cardinality $\#L[k]$
$L[1]$	$\{A:0\}[3]$ $\{A:1\}[2]$ $\{B:0\}[2]$ $\{B:1\}[3]$ $\{C:1\}[4]$	5
$L[2]$	$\{A:0,B:0\}[2]$ $\{A:0,C:1\}[2]$ $\{A:1,B:1\}[2]$ $\{A:1,C:1\}[2]$ $\{B:1,C:1\}[3]$	5
$L[3]$	$\{A:1,B:1,C:1\}$ [2]	1

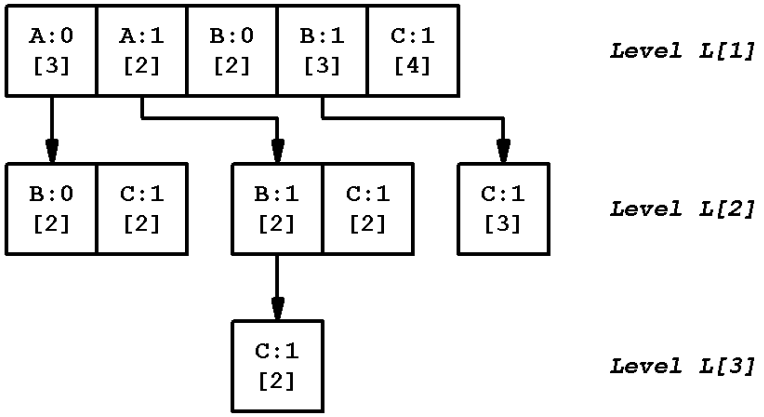


Fig. 1. Itemset tree corresponding to the itemsets shown in Table 2.

the right of $a : v$ at level $L[k]$. The items with the same attribute as $a : v$ can be discarded when working with relational databases (recall 1NF). After that, the support of each candidate $(k + 1)$-itemset in the tree is counted and all the non-relevant itemsets are pruned, thus leaving L_{k+1} at level $L[k + 1]$ of the tree.

In the following paragraphs we will look at the different primitives needed to work with the itemset tree Abstract Data Type (ADT) in greater detail.

3.2.1. Itemset tree initialization

The `Init` primitive creates the itemset tree data structure, sets its parameters (e.g. *MinSupport*), includes all the items found in the database (i.e. the candidate set C_1) in the tree, and collects some extra information. This information is necessary if hashing techniques are used to provide efficient access to the itemsets in the tree, and also if direct hashing and pruning is used within TBAR, since DHP is optional in our algorithm (see Fig. 2).

3.2.2. Relevant itemsets

A call to `Relevants(`k`)` is used to obtain L_k. This primitive of the itemset tree ADT obtains all the relevant k-itemsets, given that the itemset tree contains the candidate k-itemsets. It

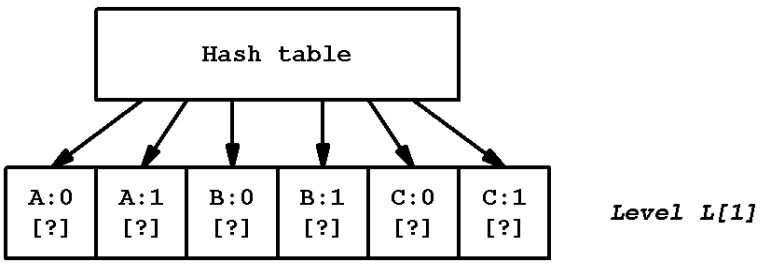


Fig. 2. Itemset tree after initialization (with a hash table in the root node).

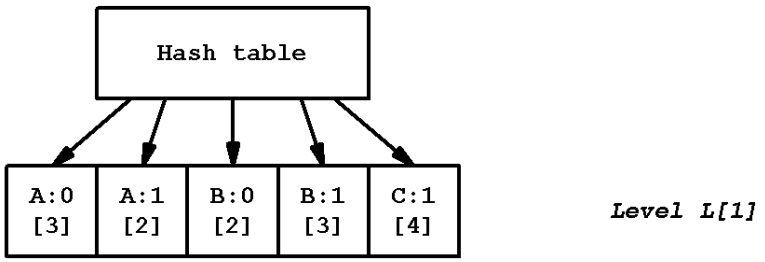


Fig. 3. Itemset tree after support counting and non-relevant itemset pruning.

performs a complete scan of the database to count candidate k-itemsets support and prunes non-relevant k-itemsets.

If a *MinSupport* threshold is used, the relevant itemsets will be those candidate k-itemsets whose support count after the database scan is higher than the user-specified threshold. Other interest measures could also be used (e.g. see [1]).

In order to count the candidate k-itemsets support efficiently during the database scan, several hashing techniques can be used. In this way, the number of comparisons needed to traverse the tree is minimized.

The final pruning step is trivial: all that remains to be done is to remove all the items corresponding to non-relevant k-itemsets from the nodes at level $L[k]$ (see Fig. 3).

3.2.3. Candidate generation

The `Candidates(k)` primitive is in charge of the candidate k-itemset generation. As stated earlier, the process consists in copying all the items to the right of a given item in the current node to a newly created child node. This copy mechanism can be restricted to only generate feasible itemsets. In a relational table, for example, an itemset cannot contain two items related to the same column. TBAR is also able to reduce the number of potentially relevant itemsets by using techniques such as DHP.

It is worthwhile mentioning that the prune step in the candidate generation phase of Apriori has been suppressed in TBAR because the savings obtained as a result of reducing the size of the candidate set do not compensate for the additional overhead. In fact, since the candidate set needs to be computed and stored anyway, and TBAR tree data structure is efficient enough to access and update the count of the itemsets it stores (regardless of how many there are), the prune step does not yield a noticeable improvement in performance.

Moreover, the prune step during candidate generation is only useful when generating C_k with $k \geqslant 3$, while the algorithm bottleneck is usually located at its first stages (C_2), where DHP is more productive. At later stages, even when the number of relevant itemsets is still very high, it is better to use DHP locally at each node of the itemset tree instead of the Apriori pruning step. It is important to remember that the higher the number of relevant itemsets, the bigger the candidate set, and therefore, the bigger the improvements achieved by DHP, which prunes its size (see Figs. 4–6).

3.2.4. Rule generation

The last primitive of the itemset tree ADT, `Rules`, returns all the association rules derived from the itemsets contained in the tree. Two tree iterators are needed for this purpose.

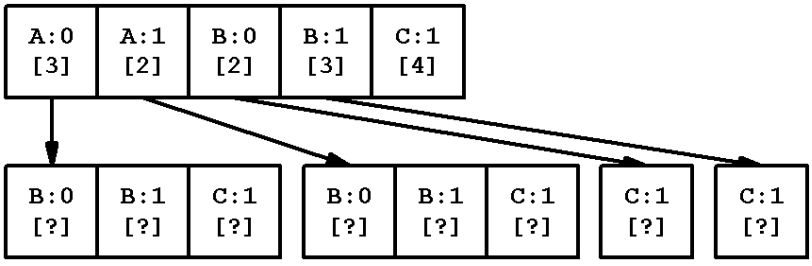


Fig. 4. Candidate 2-itemsets.

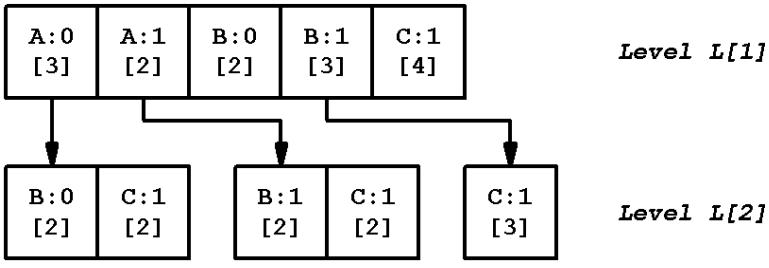


Fig. 5. Relevant 2-itemsets.

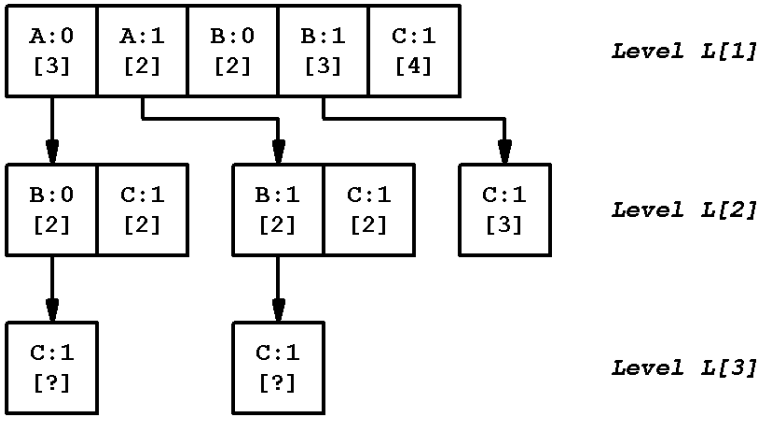


Fig. 6. Candidate 3-itemsets.

The first iterator, called `nextRuleItemset`, obtains all the relevant k-itemsets with $k \geqslant 2$ from the itemset tree (i.e. the itemsets from which association rules can be derived). The second iterator, `nextSubItemset`, gives all the proper subsets of a given itemset. The following pseudo-code corresponds to the rule generating algorithm:

```
// Apply nextRuleItemset to the tree
For each relevant k-itemset l_k in the tree with k ⩾ 2
```

```
// Generate rules derived from l_k
// using nextSubItemset iterator
For each itemset l_i ⊂ l_k
    // Choose rules with confidence above threshold
    If support(l_k) ⩾ MinConfidence * support(l_i)
        Output rule l_i ⇒ (l_k − l_i)
        with confidence = support(l_k)/support(l_i)
        and support = support(l_k)
```

The `nextRuleItemset` iterator performs a sequential scan of the set L_k. It starts from the root node of the itemset tree with the empty itemset. First, it tries to expand the current k-itemset looking for children of the actual node. If no such children exist, it looks for alternative k-itemsets in the same node of the tree. Finally, if there are no more alternatives, it backtracks to find any relevant m-itemsets ($m < k$) derived from the first $m-1$ items of the current k-itemset, with a different mth item.

The `nextSubItemset` iterator performs a similar traversal of the itemset tree to find proper subitemsets of a given itemset. It looks for extensions of the current subitemset (starting with the empty itemset) in a depth-first way, and then backtracks like `nextRuleItemset` to explore all the alternatives.

This algorithm shows excellent performance characteristics using the hashing techniques provided within the itemset tree ADT for the access and retrieval of the itemsets.

It is worth mentioning that the proposed algorithm does not generate duplicate rules, which is one of the main shortcomings of the rule generation algorithm set out in [6].

4. Experimental results

Apriori and TBAR have been implemented as stand-alone Java applications using Java DataBase Connectivity (JDBC). The Java standard Call-Level Interface (CLI) was chosen because of its portability and simplicity. Both algorithms were also coded in C++ using Borland C++ Builder and Borland Database Engine (BDE), yielding comparatively similar results.

See Sarawagi et al. [20] for a detailed discussion on the implementation alternatives for coupling Data Mining with database systems. The implementation of Apriori as a tightly coupled application is presented in [5]. Here we opted for a loosely coupled implementation because, although its expected performance is worse, it is database-independent. Enhanced portability compensates for the loss of performance. If you take advantage of Java platform independence, loosely coupled implementations of Data Mining algorithms are suitable for use with a wide range of database management systems. In fact, our implementation can be used directly with the vast majority of database vendors on the market.

Both TBAR and Apriori algorithms have been applied to several classical datasets, most of them taken from the UCI Machine Learning Database Repository at http://www.ics.uci.edu/~mlearn/MLRepository.html

- *Golf*: This is the simple dataset used by Quinlan [19] in his classic paper on ID3. It is used to build a tree classifier which decides whether or not to play golf given the weather conditions (outlook, temperature, humidity, and wind).

- *Voting records*: The VOTE dataset at the UCI Repository includes votes for each of the 435 US House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac in 1984. It also includes the party of each Congressman.
- *Soybean database*: It is also taken from the UCI Repository and it was prepared for Soybean Disease diagnosis. This dataset contains 683 instances for 19 different classes with 35 predictor attributes.
- *Mushroom database*: This dataset, also from the UCI Repository, includes descriptions of 8124 hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. The last class was combined with the poisonous one. The samples have 22 predictor attributes and 2480 missing values.
- *Census database*: This collects data extracted from the census bureau database found at http://www.census.gov/ftp/pub/DES/www/welcome.html. More details about the meaning of the particular attributes can be found at http://www.bls.census.gov/cps/cpsmain.htm. This database is usually employed to determine the income level for the person represented by each record (above or below $50K USD). It should be noted that the original CENSUS database which we used in our experiments has been replaced by the ADULT database in the UCI Repository.

The results we obtained using the above datasets are summarized in Table 3. All the experiments have been performed on three different configurations using Sun Java Development Kit (JDK) 1.2.2:

- *Configuration* 1: *Standard PC*. A 166 MHz Pentium PC with 32 MB EDO RAM running MS Windows NT 4.0 Workstation and Personal Oracle Lite 3.0.
- *Configuration* 2: *Standard PC accessing a high-performance PC server*. A 90 MHz Pentium-based PC (running Microsoft Windows NT 4.0 Workstation with 32 MB of main memory) running the Data Mining application while the database was located on a dual 333 MHz Pentium II MMX PC with 128 MB SDRAM with Oracle 8i Relational DBMS and Windows NT 4.0 Server. Both client and server were connected via TCP/IP through a low workload 10 Mbps Ethernet LAN.

Table 3
Experimental results (without using DHP)

Dataset	Description	Size (in items)	Relevant itemsets	Timings (in s) for the different configurations			
1	Golf	0.07K	104	Apriori	0.9	–	–
				TBAR	0.6	–	–
2	Voting records	7.4K	6734	Apriori	102	36	7.5
				TBAR	70	23	5.0
3	Soybean database	24.6K	70 047	Apriori	998	272	65
				TBAR	259	69	15
4	Mushroom database	186.9K	29 807	Apriori	1743	583	151
				TBAR	688	188	38
5	Census database	3.7M	101 456	Apriori	–	–	8975
				TBAR	–	–	3414

- *Configuration* 3: *High-performance PC server*. Data Mining application and Oracle 8i DBMS server running on the dual Pentium II described in the previous configuration.

It should be noted that the timings corresponding to TBAR (even without using DHP) are always better than those achieved by Apriori. All the results displayed in Table 3 were obtained without using 'Direct Hashing and Pruning' (DHP), since DHP is just an additional technique which can be used to improve the performance of any algorithm based on Apriori (and TBAR certainly is).

TBAR can also use DHP locally at each node of the itemset tree, which achieves even better results. Table 4 shows that TBAR is still much better than Apriori when DHP is employed to reduce the size of the candidate set.

TBAR outperforms Apriori even for simple datasets. When there is an increase in the number and size of the patterns discovered, the performance gap between the algorithms becomes more important.

TBAR is not only faster than Apriori, it also requires less memory (which delays swapping pages in and out). This is specially relevant when memory resources are scarce, i.e. when the data mining algorithm is running on the user PC (configurations 1 and 2). TBAR also produces less memory fragmentation, which helps the Java garbage collector to perform its work more efficiently and allows the continued execution of the mining algorithms on several datasets without the need to restart the Java Virtual Machine. This aspect is essential in web services such as servlets, for example (see Figs. 7 and 8).

One of the dominating factors for the overall running time is obviously the database scan required at each iteration. This becomes especially important in the census database and justifies the smaller speed-up obtained with our algorithm for that dataset. In any case, the relative performance of TBAR against Apriori shown in Fig. 9 illustrates the computational improvements offered by TBAR. It should be noted that, the more iterations performed, the better the relative speed-up obtained with TBAR, as shown in Figs. 10–12.

The experimental results show that TBAR, as any Apriori-based algorithm does, scales up linearly on the size of the input dataset. TBAR outperforms Apriori when there are thousands of relevant itemsets and also when their size is increased, so it is better for mining long patterns from databases.

The time required by Apriori quadruples the time spent by TBAR for the soybean database. It is remarkable that TBAR is able to find all relevant 5-itemsets in the same time required by Apriori to discover up to the 4-itemsets in that dataset.

Working on the mushroom database, as it did with the soybean database, TBAR finds all relevant 4-itemsets (of which there are thousands) in the time needed by Apriori to obtain only up

Table 4
DHP effect on running time (Apriori + DHP vs. TBAR + DHP)

Dataset	Apriori + DHP (s)	TBAR + DHP (s)	Speed-up
Voting records	9.3	3.0	$\times 3.1$
Soybean	26.9	8.0	$\times 3.4$
Mushroom	76.9	31.3	$\times 2.5$

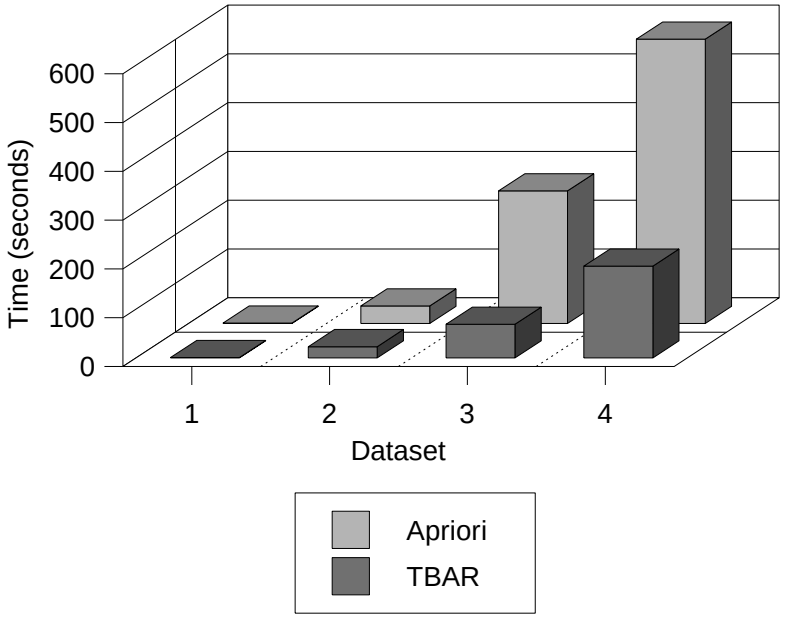


Fig. 7. Timings corresponding to configuration 2.

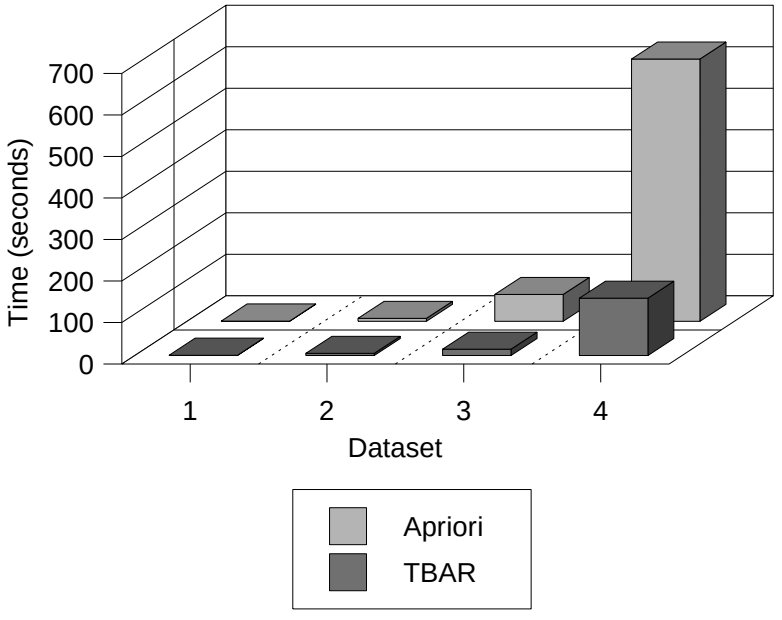


Fig. 8. Timings corresponding to configuration 3 when $L[4]$ is found for all datasets.

to the relevant 3-itemsets. Moreover, TBAR encounters all 5-itemsets before Apriori obtains the 4-itemsets (besides the fact that Apriori needs secondary storage resources while TBAR runs exclusively on main memory).

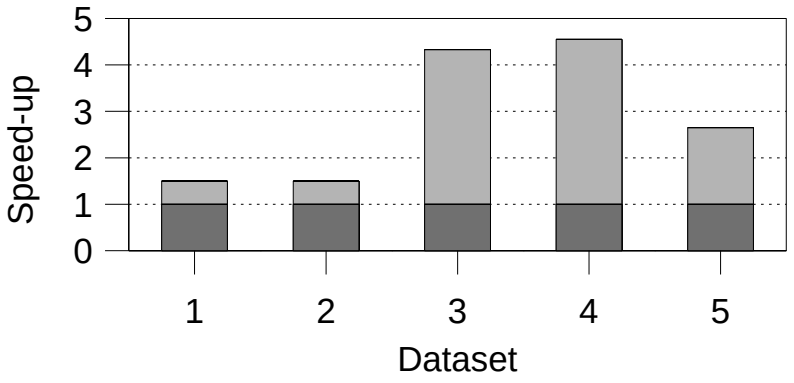


Fig. 9. Relative speed-up.

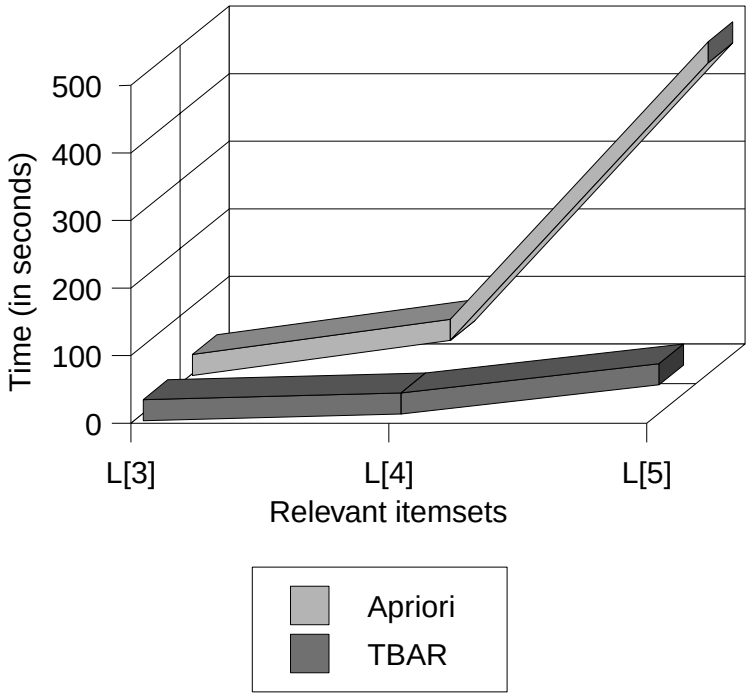


Fig. 10. Soybean database.

We have also tested our algorithm and Apriori with a larger dataset, the Census Bureau Database, and the improvement in performance achieved by TBAR is maintained. The numeric attributes in this dataset (i.e. AHRSPAY, CAPGAIN, CAPLOSS, DIVVAL, MARSUPWT, and WKSWORK) were treated as categorical attributes and their numeric values were grouped into intervals by using clustering algorithms (such as the well-known K-Means). For example, WKSWORK was considered as 0, [1,40], [41,52], and MARSUPWT was split into five intervals

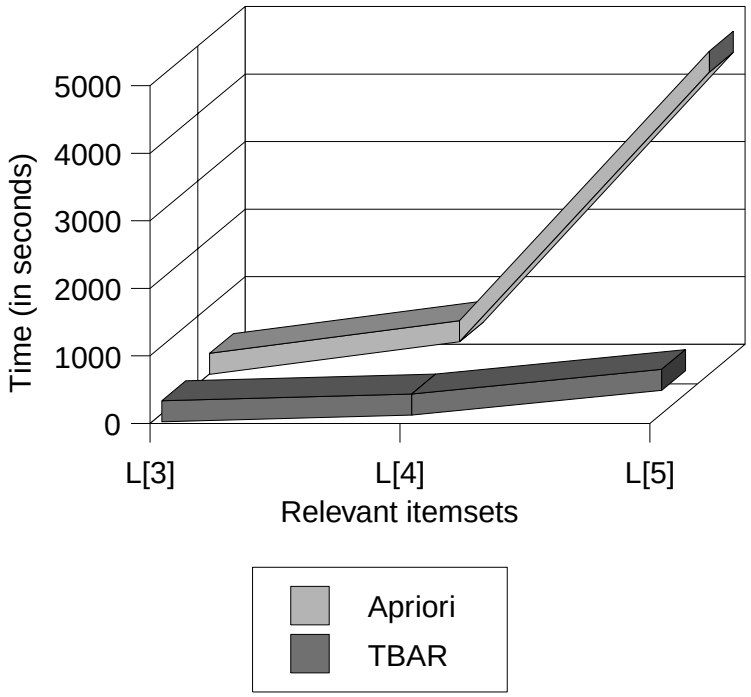


Fig. 11. Mushroom database.

using equi-depth partitioning (see [21]). Two sample association rules discovered from this dataset were:

```
if asex = 'Female' then income<50000
    with confidence = 97.4% and support = 50.6%
if asex = 'Male' then income<50000
    with confidence = 89.8% and support = 43.1%
```

5. Conclusions

We have presented and evaluated TBAR, an acronym which stands for Tree-Based Association Rule mining. TBAR is an effective algorithm for association rule mining which outperforms the Apriori algorithm. TBAR uses an optimized tree data structure and several hashing techniques (including DHP) to improve the relevant itemset generation phase of association rule mining.

TBAR assumes that if a given itemset is relevant, then all its subsets must also be relevant. Frequent itemsets satisfy this monotonicity property, although other criteria could be used. This subset requirement is essential for any algorithm based on generating a candidate set of potentially relevant itemsets. Both Apriori and TBAR follow this approach.

TBAR has been incorporated into a relational-oriented Data Mining tool currently under development. Our TBAR implementation also permits the definition of domains (which can overlap) for the attributes found in the database. In our prototype system, a domain is a set of

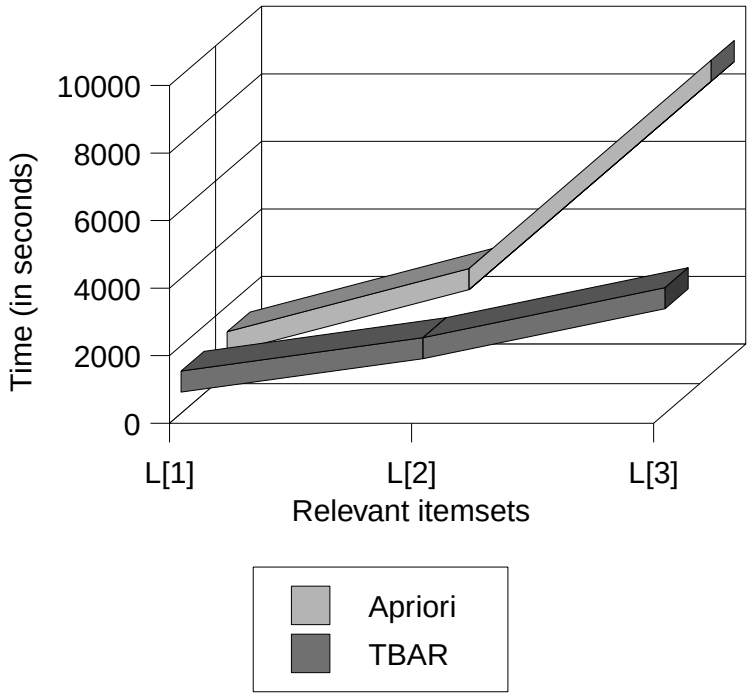


Fig. 12. Census database.

values which are considered to be equivalent. This grouping of values allows, for example, the automatic clustering of numeric attributes using intervals [21].

The TBAR algorithm has been devised to outperform Apriori in relational databases and like this well-known algorithm, it can be adapted to alternative models of association rule mining [1] and can also be used for classification problems [7].

Concurrently to our work on association rule mining, a similar data structure for mining long patterns from databases has been proposed in [8]. The algorithm described in that paper, called MaxMiner, resembles the philosophy of our algorithm. MaxMiner and TBAR were designed with different objectives and although they are similar, TBAR requires less memory and works better for association rule mining while MaxMiner is optimized to find large itemsets without finding all their subsets.

An incremental updating technique for the maintenance of the discovered association rules [10–12] can also be easily developed from TBAR.

Although a parallel version of TBAR has not yet been implemented, a parallel version of our algorithm could be derived from parallel versions of Apriori with minor modifications. Several alternative parallel versions of Apriori are explored in [4].

Acknowledgements

We wish to thank Ignacio Blanco, our group DBA, for his patience and effort in Oracle DBMS tuning. We would also like to express our gratitude towards the anonymous reviewers whose

insightful comments made this paper better for DKE readers. Most of the work described in this paper was carried out at the Department of Computer Science and Artificial Intelligence, under a research grant from the University of Granada (Spain).

References

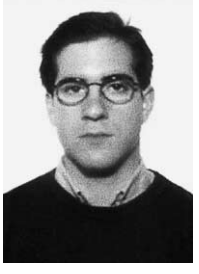
[1] C.C. Aggarwal, P.S. Yu, A new framework for itemset generation, in: Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Seattle, Washington, June 1–3, 1998, pp. 18–24.
[2] C.C. Aggarwal, P.S. Yu, Mining large itemsets for association rules, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1998.
[3] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, May 26–28, 1993, pp. 207–216.
[4] R. Agrawal, J.C. Shafer, Parallel mining of association rules, IEEE Transactions on Knowledge and Data Engineering 8 (6) (1996) 962–969.
[5] R. Agrawal, K. Shim, Developing tightly-coupled applications on IBM DB2/CS relational database system: methodology and experience, in: Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon, August 1996, pp. 287–290.
[6] R. Agrawal, R. Skirant, Fast Algorithms for Mining Association Rules, IBM Research Report RJ9839, IBM Almaden Research Center, San Jose, CA, June 1994.
[7] K. Ali, S. Manganaris, R. Skirant, Partial classification using association rules, in: KDD'97 Proceedings of the Third International Conference on Knowledge Discovery in Databases and Data Mining, August 14–17, 1997, Newport Beach, CA, USA, pp. 115–118.
[8] R.J. Bayardo Jr., Efficiently mining long patterns from databases. in: SIGMOD 1998 Proceedings ACM SIGMOD International Conference on Management of Data, June 2–4, 1998, Seattle, Washington, USA, pp. 85–93.
[9] S. Brin, R. Motwani, J.D. Ullman, Shalom Tsur, Dynamic itemset counting and implication rules for market basket data, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, May 11–15, 1997, Tucson, AZ, USA, pp. 255–264.
[10] D.W. Cheung, J. Han, V.T. Ng, C.Y.Wong, Maintenance of discovered association rules in large databases: an incremental updating technique, in: Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana, February 1996, pp. 106–114.
[11] D.W. Cheung, V.T. Ng, B.W. Tam, Maintenance of discovered knowledge: a case in multi-level association rules, in: KDD-96 Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Oregon, August 1996, pp. 307–310.
[12] R. Feldman, A. Amir, Y. Auman, A. Zilberstien, H. Hirsh, Incremental algorithms for association generation, in: Proceedings of the First Pacific-Asia Conference on Knowledge Discovery and Data Mining, in: H. Lu, et al. (Eds.), KDD: Techniques and Applications, World Scientific, Singapore, 1997, pp. 227–240.
[13] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 15–18, 2000, Dallas, TX, USA, pp. 1–12.
[14] J.L. Han, A.W. Plank, Background for association rules and cost estimate of selected mining algorithms, in: CIKM '96 Proceedings of the Fifth International Conference on Information and Knowledge Management, November 12–16, 1996, Rockville, MD, USA, pp. 73–80.
[15] J. Hipp, U. Güntzer, G. Nakhaeizadeh, Algorithms for association rule mining – a general survey and comparison, SIGKDD Explorations 2 (1) (2000) 58–64.
[16] M. Houtsma, A. Swami, Set-oriented mining for association rules, IBM Research Report RJ9567, IBM Almaden Research Center, San Jose, CA, October 1993.
[17] H. Mannila, H. Toivonen, A.I. Verkamo, Improved Methods for Finding Association Rules, Technical Report, Department of Computer Science, University of Helsinki, Helsinki, Finland, December 1993 (Revised February 1994).
[18] J.S. Park, M.S. Chen, P.S. Yu, An effective hash-based algorithm for mining association rules, in: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, CA, May 22–25, 1995, pp. 175–186.
[19] J.R. Quinlan, Induction on decision trees, Machine Learning 1 (1996) 81–106.
[20] S. Sarawagi, S. Thomas, R. Agrawal, Integrating association rule mining with relational database systems: alternatives and implications, in: SIGMOD 1998 Proceedings of the ACM SIGMOD International Conference on Management of Data, June 2–4, 1998, Seattle, Washington, USA, pp. 343–354.

[21] R. Skirant, R. Agrawal, Mining quantitative association rules in large relational tables, in: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Que., Canada, June 4–6, 1996, pp. 1–12.
[22] R. Skirant, Q. Vu, R. Agrawal, Mining association rules with item constraints, in: KDD'97 Proceedings of the Third International Conference on Knowledge Discovery in Databases and Data Mining, Newport Beach, CA, August 1997, pp. 67–73.

Fernando Berzal is a research grant holder at the Department of Computer Science and Artificial Intelligence at the University of Granada, where he is an Intelligent Systems Ph.D. student. He received his Bachelor's degree in Computer Engineering from the University of Granada in 1999 and has been awarded the Spanish Computer Studies First National Prize in 2000. His current research interests include Knowledge Discovery in Databases and Data Mining, OLAP & Data Warehousing, and Intelligent Information Systems.

Juan-Carlos Cubero received his Ph.D. in Mathematics from the University of Granada in 1994 where he is currently an Associate Professor of Computer Science. He is active in the field of databases and fuzzy technologies and has served as a program committee member and area chairman for several international conferences. He has also published several books about programming. His current research focuses on data mining and knowledge discovery, and more specifically on classification, dependence analysis, association rule discovery and fuzzy techniques.

Nicolás Marín Ruiz received his BS degree in Computing from the University of Granada in 1998, and was awarded the Best Academic Record Award on Computer Studies at his University. In 1998, he obtained a fellowship at the Artificial Intelligence Research Institute (IIIA) in Barcelona (Spain), which belongs to the Spanish Scientific Research Council (CSIC). Since 1999, he holds a fellowship from the Spanish Ministry of Education and Science at the Department of Computer Science and Artificial Intelligence of the University of Granada, where he is pursuing his Ph.D. degree. His research areas of interest include Knowledge Discovery in Databases, Data Mining, Fuzzy Object-Oriented Databases, and Fuzzy Deductive Relational Databases.



José-María Serrano Chica was born in Baeza (Jaén, Spain) in 1975. He received his Bachelor of Science Degree in Computer Science from the University of Granada (Spain), in 1998. Currently, he works as a Research Grant Holder under the FEDER project 1FD97-0255-C03-2, at the Department of Computer Science and Artificial Intelligence of the University of Granada, where he is an Intelligent Systems Ph.D. student. His research areas are Fuzzy Relational Databases, Knowledge Discovery, Data Mining, and Geographic Information Systems.