Query driven knowledge discovery in multidimensional data

Jean-François Boulicaut
INSA Lyon
LISI bât. 501
F-69621 Villeurbanne, France
jfboulic@lisi.insa-lyon.fr

Patrick Marcel
Université F. Rabelais
3 place J. Jaurès
F-41000 Blois, France
pmarcel@lisi.insa-lyon.fr

Christophe Rigotti
INSA Lyon
LISI bât. 501
F-69621 Villeurbanne, France
crigotti@lisi.insa-lyon.fr

Abstract

We study KDD (Knowledge Discovery in Databases) processes on multidimensional data from a query point of view. Focusing on association rule mining, we consider typical queries to cope with the pre-processing of multidimensional data and the post-processing of the discovered patterns as well. We use a model and a rule-based language stemming from the OLAP multidimensional representation, and demonstrate that such a language fits well for writing KDD queries on multidimensional data. Using an homogeneous data model and our language for expressing queries at every phase of the process appears as a valuable step towards a better understanding of interactivity during the whole process.

1 Introduction

Discovering knowledge from data appears as a complex iterative and interactive process containing many steps: understanding the data, preparing the data set (also called pre-processing), discovering potentially interesting patterns (mining phase), post-processing of discovered patterns and finally putting the results in use [7]. Different kinds of patterns might be used and therefore different data mining techniques are needed (e.g., association and episode rules for alarm analysis, clusters and decision trees for sales data analysis). While impressive results have been obtained the last three years for the data mining phase (e.g., the efficient discovery of frequent sets in binary data), less attention has been given to the pre-processing and post-processing phases.

This paper adresses the challenge of supporting KDD processes using a querying approach. Following Imielinski and Mannila [11], second generation data mining systems might support the discovery process by means of powerful query languages. Indeed, managing the whole KDD process is one of the crucial issues to design methodologies and integrated developement environments for relevant classes of applications. This is a long term goal for which one can not expect a comprehensive answer in the near future.

As a valuable step towards such a goal, we study how a rule-based language dedicated to multidimensional data manipulation can be used during the process of association rule mining [2]. The data mining phase is done by a system that computes frequent sets from a binary data table. It means that, given a schema $R = \{a_1, \ldots, a_n\}$ of attributes with domain $\{0, 1\}$, and a relation r over R, an algorithm that computes all the frequent sets (i.e., all the sets of attributes X such that enough rows of r have a 1 in each column of X) is available. Then, rules can be derived from frequent sets (see Section 3).

Indeed, the user-analyst has to pre-process the raw data to build the binary relevant representation r (relevancy w.r.t. the kind of properties he/she is interested in). Furthermore, he/she can post-process the extracted patterns to derive "interesting" association rules (interestingness w.r.t. objective measures like confidence [2], conviction [5] or subjective measures like templates [12]).

These interactive non trivial tasks, pre-processing and post-processing are often ad-hoc and simply done by means of standard database queries and/or programming scripts. We argue that these steps should be performed with a single model and query language so that raw data and mined patterns can be accessed uniformly, and thus giving rise to static analysis and optimization of these steps.

The idea of integrating mining and OLAP (On-Line Analytical Processing) [6] stems from Han [10]. Indeed OLAP treatments facilitate interactive data analysis by using a multidimensional data (also called cube) model. We introduce here a model and a language for expressing queries at different steps of a KDD process. The context is as follows:

- the data to be mined (called raw data) are contained in multidimensional cubes or can be put under the form of cubes,
- we are interested in association rule mining,
- the frequent sets from which association rules are derived, are discovered by using available algorithms (e.g., [3] or better algorithms that have been designed since that time).

Under these assumptions, the pre-processing step is a collection of queries on raw data to provide binary tables. The post-processing step turns to be a collection of queries on raw data and on the results of the mining step to select interesting patterns (i.e., rules) from the extracted frequent sets.

Main contributions We show that a multidimensional data model enables to handle the description of raw data, binary tables, collections of frequent item sets and association rules. We show that OLAP queries are well suited for expressing some important non trivial treatments within typical rule mining processes.

Outline In section 2, we present informally the OLAP data model and the language we use. This language is not the main contribution of the paper, since it has been already described (e.g., in [9]). Notice that it fits well in this new context without any modification. Section 3 is a brief introduction to the kind of KDD process we study. Section 4 illustrates via examples the use of our query language during association rule mining. We conclude in section 5.

2 Overview of the multidimensional data model and language

One of the first thing one has to do in a KDD process is to collect the data to be mined: the raw data. These data may be under very different forms (e.g., databases, flat files). In this paper, we assume that raw data are contained in cubes. Furthermore, we assume the reader is familiar with Datalog (see e.g., [1]).

We describe the multidimensional data model for representing cubes and the language we use for manipulating them. The model and language are fully described in the work of [9, 15, 14]. The language provides a declarative and concise way to specify the basic standard restructuring and summarizing operations used in OLAP systems. It allows to express non trivial treatments useful in the OLAP context (estimates or ad hoc aggregates) as well. One goal of this paper is to show that it can also be used to express KDD operations. In facts, each query in that language acts as a declarative specification that helps process understanding and analysis.

Data model In our data model, data are organized in *cells*. Our language is based on the point of view that a Datalog fact represents an entry (called *cell reference*) in a cube. Associations of cells contents with cells references are represented by *ground atoms* of the form:

$$N(N_1, N_2, \ldots, N_p) : \langle N_{p+1}, \ldots, N_{p+q} \rangle$$

where $N(N_1, N_2, \ldots, N_p)$ is the cell reference (i.e., the coordinate of the cell), and the tuple $\langle N_{p+1}, \ldots, N_{p+q} \rangle$ denotes the cell contents[1]. A *cube* is simply a set of ground atoms having a common cube name, in which the same reference does not appear more than once to ensure cell monovaluation, and a *multidimensional database* is a set of ground atoms in which the same reference does not appear more than once.

Throughout this paper, we call a table a 2-dimensional (matrix-like) structure, and cube a 3-dimensional structure. Examples of a cube and a table are given in figure 1 and figure 2. For example, the amount of *beer* bought by *kate* in *january* is represented by the cell $sales(kate, jan, beer) : 20$ and the age of the customer *rick* is represented by the cell $cust(rick, age) : 80$.

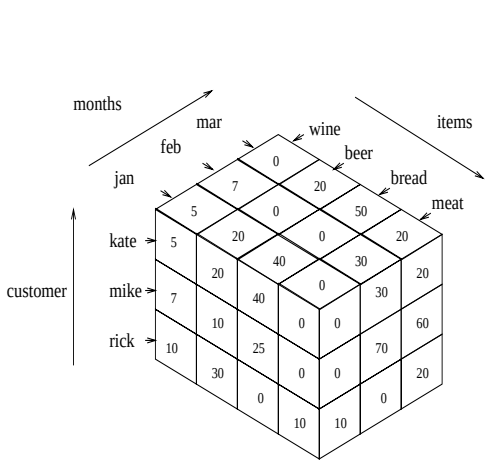

Figure 1: The cube *sales*

cust	age	income	lives in
kate	20	3000	west
mike	40	2000	east
rick	80	1000	east

Figure 2: The table *cust*

Language We use an extension of Datalog devoted to the manipulation of data organized in multidimensional cubes. Consider the rule $p(X) \longleftarrow q(X, Y), r(Y)$. The standard (Datalog) informal meaning of this rule is *if*

[1] For the sake of readability, we omit the symbols $\langle\rangle$ if the cell contents is a single constant.

q(X,Y) holds and r(Y) holds, then p(X) holds. The basic intuition of our extension is to read such a rule in the following way: *if there are two cells of references q(X,Y) and r(Y), then there is a cell of reference p(X).* We also add the handling of cell contents, and then a typical rule will be: $p(X) : \langle W \rangle \longleftarrow q(X,Y) : \langle W \rangle, r(Y) : \langle X \rangle$. This rule will be informally read: *if there exists a cell of reference q(X,Y) containing W, and there exists a cell of reference r(Y) containing X, then there exists a cell of reference p(X) containing W.*

We now give some examples of typical OLAP operations expressed by the rule-based query language. The first is the *rotate* operation that allows to reorientate the multidimensional view:

rotatedSales(M,C,I): S ⟵sales(C,M,I): S.

The second is the *nesting* operation. *Nesting* means combining the members of two dimensions so as to reduce the dimensionality of a cube. For example, months can be nested with items:

nestedSales(C,M.I): S ⟵sales(C,M,I): S.

In the table *nestedSales*, $M.I$ is a nested attribute composed of the two attributes M and I (Note that variables range only over non nested attributes).

Finally, we illustrate how a *split* operation that allows to decompose a dimension can be expressed. Assume the user want a table per months. The following query can be used to split the cube *sales*:

M(C,I): S ⟵sales(C,M,I):S.

Notice that in the OLAP framework, dimensions are often hierarchized (e.g., countries are decomposed in regions that are decomposed in cities). The model and language can also be used to handle navigation along dimension hierarchies. For the sake of space/simplicity, this feature is not illustrated here although it is of interest for selecting the level of detail at which extraction is to be performed. The reader can refer to [15] for a presentation of rule-based navigation along hierarchies and aggregates as well.

3 Modeling the KDD process

The association rule mining problem has received much attention since its introduction in [2]. Given a schema $R = \{a_1, \ldots, a_n\}$ of attributes with domain $\{0,1\}$, and a relation r over R, an *association rule* about r is an expression of the form $X \Rightarrow Y$, where $X \subseteq R$ and $Y \subseteq R$. The intuitive meaning of the rule is that if a row of the matrix r has a 1 in each column of X, then the row tends to have a 1 also in each column of Y. This semantics is captured by *support* and *confidence* values. Given $W \subseteq R$, $support(W, r)$ denotes the fraction of rows of r that have a 1 in each column of W. W is called a frequent set if its support is greater than some user-given support threshold. The frequency of $X \Rightarrow Y$ in r is defined to be $support(X \cup Y, r)$ while its confidence is $support(X \cup Y, r)/support(X, r)$. Typically, we are interested in association rules for which the frequency and the confidence are greater than given thresholds.

Frequent sets are a concise representation for binary data from which association rules are easily derived [3]. The intuition is that for every frequent set X, one can test for the interestingness (e.g., confidence is greater than the user-given threshold) of the rule $X \setminus Y \Rightarrow Y$ where $Y \subset X$. Though an exponential search space is concerned, association rules can be computed thanks to the thresholds on one hand and a safe pruning criteria that drastically reduce the search space on the other hand (the so-called `apriori` trick [3]).

The association rule mining process is modelized as follows. The result of the pre-processing step is a collection of binary tables that are the input of the mining algorithm. The binary tables have attributes as row names and attributes as column names. Examples of binary tables are depicted in figures 3 and 4. The output of the mining step is a collection of tables representing the frequent item sets mined on the corresponding *input* tables. The $frequents$ tables have attributes as column names and frequent set identifiers as row names. They are binary tables as well: a cell contains 1 if the frequent set (whose identifier is the row name) contains the attribute as column names. A column that contains the support for each frequent set is added. Examples of $frequents$ tables are depicted in figures 5 and 6.

The data model leads us to take the following conventions:

- a naming convention for binary tables: nested names are used for differenciating them.
- when constructing a binary table, every row or column attribute has the same level of nesting. Otherwise, post-processing queries like inclusion would be less easy to specify.

Association rules are derived from frequent sets and represented in tables too. The form we choose is the following (but OLAP queries can be used to restructure the *rules* tables to derive more user-friendly representations): the row name refers to the body of the rule and the column name refers to the head. If this rule holds in the raw data, then the cell contains its support and its confidence. It might also contain other interestingness measures (e.g., conviction) associated with this rule. This model emphasizes the role of frequent sets as a condensed representation from which various knowledge elicitation can be derived [13].

4 Queries in the KDD process

4.1 Pre-processing queries

The main task the user has to perform during the pre-processing step is to filter and transform the raw data so that it can be used by the mining algorithms. In this paper, we decided that the pre-processing step has to provide binary tables. Hence the pre-processing step consists in using OLAP queries to construct binary tables. Generalizing that, we can describe the pre-processing phase as a collection of queries that transform the raw data into a set of binary tables (each being an input to the mining algorithm to perform various mining tasks).

We describe how to get binary tables from the table *cust* and the cube *sales*. It illustrates how queries express typical pre-processing steps.

Binary tables In our model, a binary table can be defined as a table whose cells contain either the constant 1 or the constant 0. 2 binary table(s) are depicted in figures 3 and 4. Note that cells from the cube *sales* are of the form $sales(C, M, I) : S$ where C is a customer, M is a month, I is an item and S is the number of item I bought by C in M.

Some of the typical OLAP operations (e.g., nest, pull) are very useful to change the structure of cubes and tables, and can be used to construct the binary table(s) from raw data. In the following, we suppose that the user does not want to represent the number of items sold, then a nest operation and a selection can be used to obtain a binary table *input.cust* from the cube *sales*:

input.cust(C,M.I):1 ⟵sales(C,M,I): S, S > 0.
input.cust(C,M.I):0 ⟵sales(C,M,I): 0.

Expressing background knowledge The queries can be used to incorporate the background knowledge (knowledge not to be discovered but not explicitly represented in the data).

Suppose the user wants to describe the fact that a customer is a good customer if he/she is a regular customer and has an income over 2000. This can be done by the following query, where *regularCustomer* is a table containing previously known (or extracted) information:

cust(C,type): good ⟵cust(C,income): I,
I > 2000, regularCustomer(C).

Filtering Since the OLAP language can express classical relational operations (e.g., selection, projection, join, difference), OLAP queries can be used to discard unwanted attributes. Suppose the user is only interested in good customers. The query for constructing the binary table becomes:

	age		jan		...
input cust	lt_50	ge_50	wine	beer	...
kate	1	0	1	1	...
mike	1	0	1	0	...
⋮	⋮	⋮	⋮	⋮	⋱

Figure 3: The table *input.cust*

	kate		...
input items	jan	feb	...
wine	1	1	...
beer	1	0	...
⋮	⋮	⋮	⋱

Figure 4: The table *input.items*

input.cust(C,M.I):1 ⟵sales(C,M,I): S, S > 0,
cust(C,type):good.
input.cust(C,M.I):0 ⟵sales(C,M,I): 0,
cust(C,type):good.

Discretizing Raw data often contain continuous valued attributes (e.g., age, salary). It is common to discretize these attributes (i.e., to group them into intervals) to ensure that they reach a sufficient level of support. For example, assume the user wants the binary table to contain only two categories of age:

input.cust(C,age.lt_50):1 ⟵cust(C,age): A, A < 50.
input.cust(C,age.lt_50):0 ⟵cust(C,age): A, A ≥ 50.
input.cust(C,age.ge_50):1 ⟵cust(C,age): A, A ≥ 50.
input.cust(C,age.ge_50):0 ⟵cust(C,age): A, A < 50.

Selecting A rotate operation can be used on the binary table to select which kind of association rules are to be extracted. For example, in the table *input* of figure 3, association rules will concern each customer behavior. If the user wants to compare the behavior on each item, he/she can use the following query to get the binary table of figure 4 from that of figure 3:

input.items(I,C.M):V ⟵input.cust(C,M.I):V, M ≠ age.

In the OLAP framework, dimensions can be hierarchized and navigation through hierarchy, called roll-up and drill-down in the OLAP litterature, can be used to select the level of detail to construct the binary table used as input for the extraction.

Once binary tables are built, it is possible to run the mining algorithm. The user may specify support thresholds, or more generally interestingness measures that fit to the mining task. The result can also be modeled as tables and cubes.

frequents cust	age		jan		...	support
	lt 50	geq 50	wine	beer	...	
f_1	1	0	1	1	...	0.1
f_2	1	0	1	0	...	0.03
⋮	⋮	⋮	⋮	⋮	⋱	

Figure 5: The table *frequents cust*

frequents items	kate		...
	jan	feb	...
f_1	1	1	...
f_2	1	0	...
⋮	⋮	⋮	⋱

Figure 6: The table *frequents items*

4.2 Post-processing queries

Frequent sets can be represented in a binary table where each row name is a frequent set identifier and each column name is an attribute. The cell identified by a given column name (an attribute) and a given row name (a frequent set identifier) contains 1 if this attribute is part of this frequent set. Figure 5 (resp. 6) shows the result of the extraction for the table of figure 3 (resp. 4).

When it is clear from the context, we use the "generic" table name *frequents* instead of using *frequents.cust*, *frequents.items*, etc. The same idea applies for tables named *input* and *rules*.

Presenting the result The representation of frequent item sets introduced above may seem unfriendly, but here again OLAP queries can be used to switch to a convenient presentation. For example, a split query can be used to get a frequent set per table:

F(M,I):1 ⟵frequents(F,M.I): 1, M ≠ age.

Note that the *age* attribute can be taken apart form the other attributes since for a particular frequent item set, its value holds for every combination of the other attributes.

F(age,value): A ⟵frequents(F,age.A):1.

Useful queries The representation of the frequent sets in a binary table provides an easy way to write some typical post-processing queries. For example, inclusion of frequent sets can be simply written:

```
frequents-inclusion(F1,F2) ⟵not(non-include(F1,F2)),
                              frequent-id(F1),
                                 frequent-id(F2).
non-include(F1,F2) ⟵frequents(F2,A.B):0,
                           frequents(F1,A.B):1.
```

Where *frequent-id* range over the frequent set identifiers.

Item set complement can also be defined by means of a ternary predicate $complement(F1, F2, F3)$ which holds if $F1 = F2 \cup F3$ and $F2 \cap F3 = \emptyset$. Its definition is not given here due to the lack of space.

Support, confidence, interestingness measures Association rules and their interestingness measures can be derived from the *frequents* table by queries. Let us compute the support of the rules of the form $F2 \Rightarrow F3$ as follows.

```
rule-support(F2,F3):S ⟵frequents(F1,support):S,
                           complement(F1,F2,F3).
```

Then rule confidence can be computed by:

```
rule-conf(F2,F3):C ⟵frequents(F1,support):S1,
                       frequents(F2,support):S2,
                          complement(F1,F2,F3),
                          C = S1/S2.
```

Other interestingness measures can be specified. For example, conviction [5] can be expressed by:

```
rule-convic(F2,F3): C ⟵complement(F1,F2,F3),
                          frequents(F1,support):S1,
                          frequents(F2,support):S2,
                          frequents(F3,support):S3,
                       C = (S2 × (1 − S3))/(S2 − S1).
```

Presenting the association rules A table *rules* can be constructed from the tables *rule-conf* and *rule-support*, such that each cell of the table describes a rule:

```
rules(F1,F2): ⟨C, S⟩ ⟵rule-support(F1,F2):S,
                       rule-conf(F1,F2):C
```

rules	f_1	f_2	...
f_1	C_{11} S_{11}	C_{12} S_{12}	...
f_2	C_{21} S_{21}	C_{22} S_{22}	...
⋮	⋮	⋮	⋮

Figure 7: The table *rules*

Selection/filtering of rules can now be expressed on the table *rules*. For instance, the following query asks for the rules having confidence higher than 90 %:

?- rules(F1,F2): ⟨C, S⟩, C ≥ 0.9

The following query asks for the rules whose head contains *beer*:

?- rules(F1,F2): ⟨C, S⟩, frequents.cust(F2,_beer).

This one asks for pairs of rules such that the body of the first and the head of the second share an item:

```
?- rules(F1,F2): ⟨C1, S1⟩, rules(F3,F4): ⟨C2, S2⟩,
      frequents.cust(F1,A.B), frequents.cust(F4,A.B).
```

Links to raw data Having a unique data model enables that raw data and mined patterns are uniformly accessed. For instance, assume the user wants to get the customers violating a given rule, this can be obtained by the following rules:

violates(C,F1.F2) ⟵rules(F1,F2): ⟨_, _⟩,
input-inclusion(F1,C),
not(input-inclusion(F2,C)).
input-inclusion(F,C) ⟵frequent-id(F),
input.cust(C,_):1,
not(non-include-input(F,C)).
non-include-input(F,C) ⟵frequents.cust(F,A.B):1,
input.cust(C,A.B):0.

We can write generic queries by using variables in table names. For instance, it is straightforward to build tables $violates.X(C, F1.F2)$, $input\text{-}inclusion.X(F, C)$ and $non\text{-}include\text{-}input.X(F, C)$ where X will be bound to *cust* to focus on customers or *items* to deal with items.

Templates Templates have been introduced in [12] as conditions (1) on the size of the rules, and (2) on the occurrence of attributes in their head or body.

Assume the user wants to select the rules whose body concern customers having a high income (on the frequent sets mined on the second binary table). This condition can be expressed with the following rule:

get(F1,F2): $\langle C, S\rangle$ ⟵rules(F1,F2): $\langle C, S\rangle$,
frequents(F1,Cust._):1,
cust(Cust,income):I, I $\geq$ 2000.

Covering Rules can be compared with each other. For example, structural covering [17] consists in filtering the rules being less general than a particular one. A rule $R1$ is less general than a rule $R2$ if the confidence of $R1$ is lesser than the confidence of $R2$, and $R1$ has a shorter head and a longer body than $R2$.

covers(F1.F2,F3.F4) ⟵rules(F1,F2): $\langle C1, S1\rangle$,
rules(F3,F4): $\langle C2, S2\rangle$,
C1 $\geq$ C2,
frequent-inclusion(F1,F3),
frequent-inclusion(F4,F2).

Such a rule is quite close to the specification of the task. Obviously, it is one of the major advantage of a rule-based language with a formal declarative semantics. It is clear that evaluating such queries can be very difficult even though operational semantics have been already studied.

The data model and the query language can be used to enhance the different steps of the KDD process. Our aim is not to act on the mining algorithm. Instead we can act on the different tables or cubes used during the process: the $input.X$ binary table, the $frequents.X$ table that provides the frequent item sets and the $rules.X$ table that provides rules. Different mining phases can be performed for more or less complex post-processing phases (e.g., querying simultaneously $frequents.cust$ and $frequents.items$ tables).

5 Conclusion

This paper has demonstrated that a multidimensional data manipulation language enables to express typical pre-processing and post-processing within the association rule mining context. The uniform access to data and patterns is interesting : it means that the whole process (beside the mining phases) is a collection of queries written with the same language. [8] presents a similar approach by using OQL as a query language. Indeed, in that case, multidimensional features are not considered. Dedicated languages like for instance MINE RULE [16] also enable to select data and specify various mining tasks. However, they do not consider the multidimensional aspect of data and patterns.

Modeling KDD processes as queries give rise to optimization possibilities e.g., when complex processes have to be reused on various datasets. This research is part of a project related to the inductive database framework [13, 4]. An inductive database is a database that contains intensionally defined generalizations about the data, in addition to the usual data. The KDD process can then be described as a sequence of queries on an inductive database and mining phases are part of the evaluation process for queries that select specific inductive properties of the data. Indeed, implementing inductive databases, not only for association rules but also for other classes of patterns, will be a long term effort.

We consider that studying typical queries using languages with a clean formal semantics is one mean to understand interactivity during KDD processes. The next step of such a research consists in relaxing the assumption that condensed representation of data (e.g., frequent sets) are materialized (i.e., computed beforehand by some data mining algorithm) but instead that computing it is part of the query evaluation process.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley Publishing Company, Reading, MA, 1995.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)*, pages 207 – 216, Washington, D.C., USA, May 1993. ACM.

[3] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, Menlo Park, CA, 1996.

[4] Jean-François Boulicaut, Mika Klemettinen, and Heikki Mannila. Modeling KDD processes within the inductive database framework. In *Proc. of the 1st Int. Conf. on Data Warehousing and Knowledge Discovery (DaWak'99)*, Florence, (I), September 1999. To appear.

[5] Sergey Brin, Rejeev Motwani, and Craig Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proc. of ACM SIGMOD Conf. on Management of Data (SIGMOD'97)*, pages 265 – 276, Tucson, AZ, May 1997. ACM.

[6] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (On-Line Analytical Processing) to user-analysts: An IT mandate [on-line]. 31p. White Paper Available from <http://www.arborsoft.com/essbase>, 1993.

[7] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA, 1996.

[8] Bart Goethals, Jan Van den Bussche, and Koen Vanhoof. Decision support queries for the interpretation of data mining results. Manuscript, University of Limburg (Belgium), 1998. Available at http://www.luc.ac.be/∼vdbuss.

[9] Mohand-Said Hacid, Patrick Marcel, and Christophe Rigotti. A rule-based language for ordered multidimensional databases. In *Proc. 5th Workshop on Deductive Database and Logic Programming (DDLP'97)*, volume 317 of *GMD-Studien*, pages 69–81, Leuven (B), July 1997.

[10] Jiawei Han. OLAP mining: An integration of OLAP with data mining. In *Proc. 1997 IFIP Conference on Data Semantics (DS-7)*, pages 1–11, Leysin (CH), October 1997.

[11] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58 – 64, November 1996.

[12] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A. Inkeri Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. of the 3rd Int. Conf. on Information and Knowledge Management (CIKM'94)*, pages 401 – 407, Gaithersburg, MD, USA, November 1994. ACM.

[13] Heikki Mannila. Inductive databases and condensed representations for data mining. In *Proc. of the Int. Logic Programming Symposium (ILPS'97)*, pages 21 – 30, Port Jefferson, USA, October 13-16 1997. MIT Press.

[14] Patrick Marcel. *Manipulations de données multidimensionnelles et langages de règles*. PhD thesis, INSA de Lyon, December 1998. In french.

[15] Patrick Marcel and Christophe Rigotti. Rule-based approach to OLAP manipulations. Submitted, 1999.

[16] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. A new SQL-like operator for mining association rules. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 122 – 133, Mumbay, India, September 1996. Morgan Kaufmann.

[17] Hannu Toivonen, Mika Klemettinen, Pirjo Ronkainen, Kimmo Hätönen, and Heikki Mannila. Pruning and grouping of discovered association rules. In *Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, pages 47 – 52, Heraklion, Crete, Greece, April 1995. MLnet.