Querying Inductive Databases: A Case Study on the MINE RULE Operator

Jean-François Boulicaut*, Mika Klemettinen, and Heikki Mannila

University of Helsinki, Department of Computer Science
P.O. Box 26, FIN-00014 University of Helsinki, Finland
e-mail:{boulicau,mklemett,mannila}@cs.helsinki.fi

Abstract. Knowledge discovery in databases (KDD) is a process that can include steps like forming the data set, data transformations, discovery of patterns, searching for exceptions to a pattern, zooming on a subset of the data, and postprocessing some patterns. We describe a comprehensive framework in which all these steps can be carried out by means of queries over an *inductive database*. An inductive database is a database that in addition to data also contains intensionally defined generalizations about the data. We formalize this concept: an inductive database consists of a normal database together with a subset of patterns from a class of patterns, and an evaluation function that tells how the patterns occur in the data. Then, looking for potential query languages built on top of SQL, we consider the research on the `MINE RULE` operator by Meo, Psaila and Ceri. It is a serious step towards an implementation framework for inductive databases, though it addresses only the association rule mining problem. Perspectives are then discussed.

1 Introduction

Data mining sets new challenges to database technology and new concepts and methods are needed for general purpose query languages [9]. A possible approach is to formulate a data mining task as locating interesting sentences from a given logic that are true in the database. Then the task of the user/analyst can be viewed as querying this set, the so-called *theory* of the database. Formally, given a language $\mathcal{L}$ of sentences (or patterns), the theory of the database $\mathbf{r}$ with respect to $\mathcal{L}$ and a selection predicate q is the set $Th(\mathbf{r}, \mathcal{L}, q) = \{\theta \in \mathcal{L} \mid q(\mathbf{r}, \theta)\}$. The predicate q indicates whether a sentence of the language is interesting. This definition is quite general: asserting $q(\mathbf{r}, \theta)$ might mean that θ is a property that holds, that almost holds, or that defines (in some way) an interesting subgroup of $\mathbf{r}$. This approach has been more or less explicitely used for various data mining tasks (see [12] for a survey and [13] for a detailed study of this setting).

Discovering knowledge from data can be seen as a process containing several steps: understanding the domain, preparing the data set, discovering patterns, postprocessing of discovered patterns, and putting the results into use [6]. This is an interactive and iterative process for which many related theories have to

* On sabbatical leave from INSA Lyon (F). This work is partly supported by AFFRST, Association Franco-Finlandaise pour la Recherche Scientifique et Technique.

be computed: different selection predicates and also classes of patterns might be used. Therefore, a general-purpose query language should enable the user to select subsets of data, but also to specify and select patterns. It should also support crossing the boundary between data and patterns, e.g., when exceptions to a pattern are to be analysed or for sophisticated postprocessing methods like rule covering [17]. This has motivated the concept of *inductive databases*, i.e., databases that contain inductive generalizations about the data, in addition to the usual data [9].

The contribution of this paper concerns a formalization of this concept of inductive database and a first approach for an implementation based on SQL servers. The formalization carries a twopart basic message: (i) an inductive database consists of a normal database associated to a subset of patterns from a class of patterns, and an evaluation function that tells how the patterns occur in the data; (ii) an inductive database can be queried (in principle) just by using a straightforward extension of relational algebra. This point of view is also considered in [3] although that paper has a different focus: it considers the add-value of this framework for KDD process modeling. The search for solutions based on SQL is motivated by the industrial perspective of relational database mining. A huge amount of work has already been done to provide efficient and portable implementations of SQL, and tightly-coupled architectures between SQL servers and data mining systems are being developed. As a starting point, we study the `MINE RULE` operator proposed by Meo, Psaila and Ceri [14, 15] and consider its connections with our framework.

The paper is organized as follows. In Section 2 we define the inductive database framework. Section 3 is an overview of the `MINE RULE` operator. Section 4 is a discussion of the adequacy of this proposal as an inductive database implementation framework. Section 5 is a short conclusion.

2 Inductive Databases

Our goal is to describe a data model that makes it possible to view the whole KDD process as querying a database structured according to the model. Thus the database has to contain both data and generalizations about that data. This motivates the following definition (simplified from the one in [11]).

Schema The *schema of an inductive database* is a pair $\mathcal{R} = (\mathbf{R}, (\mathcal{Q}_\mathbf{R}, e, \mathcal{V}))$, where $\mathbf{R}$ is a database schema, $\mathcal{Q}_\mathbf{R}$ is a collection of patterns, $\mathcal{V}$ is a set of *result values*, and e is the *evaluation function* that defines how patterns occur in the data. This function maps each pair $(\mathbf{r}, \theta_i)$ to an element of $\mathcal{V}$, where $\mathbf{r}$ is a database over $\mathbf{R}$ and θ_i is a pattern from $\mathcal{Q}_\mathbf{R}$.

Instance An *instance* $(\mathbf{r}, s)$ of an inductive database over the schema $\mathcal{R}$ consists of a database $\mathbf{r}$ over the schema $\mathbf{R}$ and a subset $s \subseteq \mathcal{Q}_\mathbf{R}$.

The simple association rule mining problem has received much attention since its introduction in [1]. The concept of inductive database is quite general and is not dedicated to this class of patterns. However, for didactic reasons, we use it in our examples.

s_0	e(r_0).f	e(r_0).c
$A \Rightarrow B$	0.25	0.33
$A \Rightarrow C$	0.50	0.66
$B \Rightarrow A$	0.25	0.50
$B \Rightarrow C$	0.50	1.00
$C \Rightarrow A$	0.50	0.66
$C \Rightarrow B$	0.50	0.66
$AB \Rightarrow C$	0.25	1.00
$AC \Rightarrow B$	0.25	0.50
$BC \Rightarrow A$	0.25	0.50

s_1	e(r_1).f	e(r_1).c
$A \Rightarrow B$	0.33	0.33
$A \Rightarrow C$	0.66	0.66
$B \Rightarrow A$	0.33	1.00
$B \Rightarrow C$	0.33	1.00
$C \Rightarrow A$	0.66	1.00
$C \Rightarrow B$	0.33	0.50
$AB \Rightarrow C$	0.33	1.00
$AC \Rightarrow B$	0.33	0.50
$BC \Rightarrow A$	0.33	1.00

s_2	e(r_2).f	e(r_2).c
$B \Rightarrow C$	0.50	1.00

Instance r_0

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

Table 1. Patterns in three instances of an inductive database.

Example 1 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 $X \Rightarrow B$, where $X \subseteq R$ and $B \in R \setminus X$ [1]. Intuitively, 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 column B. This semantics is captured by *frequency* and *confidence* values. Given $W \subseteq \mathbf{R}$, $freq(W, \mathbf{r})$ denotes the fraction of rows of $\mathbf{r}$ that have a 1 in each column of W. The frequency of the rule $X \Rightarrow B$ in $\mathbf{r}$ is defined to be $freq(X \cup \{B\}, \mathbf{r})$ while its confidence is $freq(X \cup \{B\}, \mathbf{r})/freq(X, \mathbf{r})$. Typically, we are interested in association rules for which the frequency and the confidence are greater than given thresholds. However, we can define an inductive database such that $\mathcal{Q}_R$ contains all association rules, i.e., $\mathcal{Q}_R = \{X \Rightarrow B \mid X \subseteq R, B \in R \setminus X\}$. In this case, $\mathcal{V}$ is the set $[0, 1]^2$, and $e(\mathbf{r}, \theta) = (f(\mathbf{r}, \theta), c(\mathbf{r}, \theta))$, where $f(\mathbf{r}, \theta)$ and $c(\mathbf{r}, \theta)$ are the frequency and the confidence of the rule θ in the database $\mathbf{r}$. □

Queries A typical KDD process operates on both of the components of an inductive database. At each stage of manipulating the inductive database $(\mathbf{r}, s)$, the user can think that the value of $e(\mathbf{r}, \theta)$ is available for each pattern θ which is present in the set s. Obviously, if the pattern class is large, an implementation will not compute all the values of the evaluation function beforehand; rather, only those values $e(\mathbf{r}, \theta)$ that user's queries require to be computed should be computed. Mining association rules as defined in Example 1 is now considered as querying inductive database instances of schema $(R, (\mathcal{Q}_R, e, [0, 1]^2))$.

Example 2 Assume the dataset is the instance r_0 in Table 1 of the schema $R = \{A, B, C\}$. The inductive database $idb = (r_0, s_0)$ associates to r_0 the rules on the leftmost table of Table 1. We illustrate the selection on tuples (Q_1) and the selection on patterns (Q_2).

1. (Q_1) Select tuples from (r_0, s_0) for which the value for A is not 0. The result is a new instance (r_1, s_1) where the data part r_1 does not contain the tuple $(0, 1, 1)$, and the pattern part s_1 contains the rules in the middle table of Table 1, i.e., the rules of s_0 with updated frequency and confidence values.
2. (Q_2) Select rules from (r_0, s_0) that exceed the frequency and confidence thresholds 0.5 and 0.7, respectively. A new instance (r_0, s_2) is provided where s_2 contains the rules in the rightmost table of Table 1.

An important feature is that operations can be composed due to the *closure property*: an operation takes an instance of an inductive database and provides a

new instance. For instance, the query $Q_2 \circ Q_1$ if applied to (r_0, s_0) gives (r_3, s_3), where r_3 is r_1 as defined above and s_3 is reduced to the association rule $C \Rightarrow A$ with frequency 0.66 and confidence 1. □

Query Languages Using the above definition for inductive databases it is easy to formulate query languages for them. For example, we can write relational algebra queries, where in addition to the normal operations we can also refer to the patterns and the value of the evaluation function on the patterns. To refer to the values of $e(\mathbf{r}, \theta)$ for any $\theta \in s$, we can think in terms of object-oriented databases: the evaluation function e is a method that encodes the behavior of the patterns in the data.

For the association rule example, it motivates the notations $e(\mathbf{r}).f$ and $e(\mathbf{r}).c$ when values for frequency and confidence are needed. Furthermore, it is useful to consider that other properties of patterns should be available; as for instance, the values for part of them, their lengths, etc. Following an abstract data type approach, we can consider operations that provide theses properties. Hence, continuing Example 1, we use *body*, *lbody* and *head* to denote respectively the value of the left-hand side, its length and the value of the right-hand side of an association rule. More generally, specifying an inductive database requires the definition of all these properties.

We now give a few queries by using, hopefully, self-explanatory notations for the simple extension of the relational algebra that fits to our need. Selection of tuple and patterns are respectively denoted by σ and τ. As it is clear from the context, the operation is also applied on inductive database instances, e.g., we write $\sigma_C((r, s))$ to denote $(\sigma_C(r), s))$.

Example 3 We now consider association rules in the concrete and popular context of the basket analysis problem. Assume data is available in an instance of the schema $R =$(`Tid, Item, Price, Date`). `Tid` denotes the transaction identifier, `Item` the product purchased, `Price` its price and finally, `Date` the date for this transaction. By (r, s) we denote an inductive database for association rules between itemsets; s_0 denotes the intensionally defined collection of all these rules. Table 2(a) gives a dataset called r_0 in the sequel and one sample collection of patterns with their properties and answers in r_0. Notice that such a collection can typically be stored in a nested relation, e.g., an SQL3 table [10].

Consider the following process. First, the user decides to look at association rules derived from r_0, the dataset for the current month, and he/she wants to prune out all rules that have confidence under 30% or frequency under 5% or more than 7 items (phase 1 in Table 2(b)). Then, he/she decides to focus on the rules that hold for the data about the last discount day (say `Date` $= 13$) and to restrict to 5 the maximum amount of items in the rule (phase 2). Then, he/she wants to eliminate all the patterns that contain item D in their body. Finally, he/she tries to get association rules that imply expensive items (say `Price` ≥ 7). A lower threshold for frequency (say 1%) is considered for phase 4. □

Different types of KDD processes are easily described using the notion of inductive database. The key is the closure property, which makes the composition of queries possible [3].

Tid	Item	Price	Date
1	A	7	1
2	A	7	1
2	B	5	1
2	C	9	1
3	A	7	1
3	C	9	1
4	B	5	1
4	C	9	1

body	*head*	*lbody*	e(r_0).f	e(r_0).c
$\{A\}$	$\{B\}$	1	0.25	0.33
$\{A\}$	$\{C\}$	1	0.50	0.66
$\{A, B\}$	$\{C\}$	2	0.25	1
$\{A, C\}$	$\{B\}$	2	0.25	0.5

(a)

Phase	Query and conditions
1	$\tau_{F_1}((r_0, s_0)) = (r_0, s_1)$ $F_1 = e(r_0).f \geq 0.05 \wedge e(r_0).c \geq 0.3 \wedge$ $lbody \leq 6$
2	$\tau_{F_2}(\sigma_{C_1}((r_0, s_0))) = (r_1, s_2)$ $C_1 = (Date = 13)$ $F_2 = e(r_1).f \geq 0.05 \wedge e(r_1).c \geq 0.3 \wedge$ $lbody \leq 4$
3	$\tau_{F_3}((r_1, s_2)) = (r_1, s_3)$ $F_3 = D \notin body$
4	$\tau_{F_4}(\sigma_{C_2}((r_1, s_0))) = (r_2, s_4)$ $C_2 = (Price \geq 7)$ $F_4 = e(r_2).f \geq 0.01 \wedge e(r_2).c \geq 0.3 \wedge$ $lbody \leq 4 \wedge D \notin body$

(b)

Table 2. *Basket data as an inductive database (a) and a few queries (b).*

3 MINE RULE Operator

In the following, we provide an overview of the `MINE RULE` operator [14, 15] and then discuss how it can be related to our framework for inductive databases. `MINE RULE` is an SQL-like operator which captures most of the association rule mining tasks that have been formulated so far (simple or generalized association rules, association rules with item hierarchies, etc). Moreover, there are quite efficient evaluation techniques that ensure the possibility of solving these mining tasks. It is not possible here to consider all the aspects of such an operator. We introduce it by means of one typical example and refer to [14] for other examples and a complete definition of its syntax and operational semantics.

Given the dataset r_1 as defined in Table 2, phase 4 is defined by the `MINE RULE` statement in Table 3. The `MINE RULE` operator takes a relational database and produces an SQL3 table [10] in which each tuple denotes a mined rule.

Several possibilities exist to precisely define the input data. Basically, the whole potential of SQL can be used here. The input tables might themselves have been selected using the second `WHERE` clause. Rules are extracted from groups as defined by a `GROUP BY` clause (frequency is related to groups and if the clause is missing, any tuple is a group). The schema of the output table is determined by the `SELECT` clause that defines the structure of the rules (here, `BODY`, `HEAD`, `SUPPORT` and `CONFIDENCE`). Sizes of the two components of a rule can be bounded (4 and 1 in our example). The keyword `DISTINCT` specifies that duplicates are not allowed in these components.

Data is encoded such that one gets all possible couples of itemsets (extracted from the groups) for the body and the head of a rule. It is possible to express mining conditions (first `WHERE` clause) that limit the tuples involved in this en-

```
MINE RULE s1 AS
SELECT DISTINCT
      1..6 Item AS BODY,
      1..1 Item AS HEAD,
      SUPPORT, CONFIDENCE
FROM r0
GROUP BY Tid
EXTRACTING RULES WITH
      SUPPORT: 0.05,
      CONFIDENCE: 0.03
```

(Phase 1)

```
MINE RULE s2 AS
SELECT DISTINCT
      1..4 Item AS BODY,
      1..1 Item AS HEAD,
      SUPPORT, CONFIDENCE
FROM (SELECT * AS r1
      FROM r0
      WHERE Date = 13)
GROUP BY Tid
EXTRACTING RULES WITH
      SUPPORT: 0.05,
      CONFIDENCE: 0.3
```

(Phase 2)

```
SELECT * AS s3
FROM s2
WHERE D NOT IN BODY
```

(Phase 3)

```
MINE RULE s4 AS
SELECT DISTINCT
      1..4 Item AS BODY,
      1..1 Item AS HEAD,
      SUPPORT, CONFIDENCE
WHERE BODY.Item <> D
FROM (SELECT * AS r2
      FROM r1
      WHERE Price >= 7)
GROUP BY Tid
EXTRACTING RULES WITH
      SUPPORT: 0.01,
      CONFIDENCE: 0.3
```

(Phase 4)

Table 3. *Phases 1 to 4 of Table 2 using* MINE RULE.

coding. In our example, the mining condition indicates that `Item` in the body should not be D. An interesting feature is that mining conditions can be different for body and head, e.g., `BODY.price < 7 AND HEAD.price >= 7` indicates that one wants association rules with cheap products (less than 7) in the body and an expensive product in the head. It is possible to choose the types of the elements in the rules (e.g., `Price` instead of `Item`) as well as grouping attributes. This enables the specification of many different mining tasks over the same dataset. Another important feature of `MINE RULE` is the possibility to consider clusters. One can require that the body and the head of mined rules refer to clusters within the same group. Typically, it makes possible to look for association rule of items purchased at the same date if a `CLUSTER BY` clause on the `Date` attribute is used. In fact, most of the association rule mining tasks identified in the literature can be specified by means of a `MINE RULE` statement.

The architecture proposed in [15] has been designed on top of an SQL-server. It is a tightly-coupled architecture that provides a closure property: the user can materialize the selected dataset as well as the collection of association rules that hold in the dataset and satisfy its mining requirements. Data and patterns are then a collection of SQL3 tables. The phases of the simple scenario given in Table 2(b) are easily translated into `MINE RULE` queries as given in Table 3. Note that phase 3 is not achieved by means of a `MINE RULE` statement. Instead, we use a query over the materialization of s_2.

The mining algorithms that can not be expressed in terms of SQL queries are activated by the so-called `core` operator. The three main components of the architecture defined in [15] are:

- *Preprocessor.* After the interpretation of a `MINE RULE` statement, a preprocessor retrieves source data, evaluates the mining, grouping and cluster conditions and encodes the data that will appear in the rules: it produces a set of encoded tables that are stored in the database. These encoded tables are optimized in the sense that mining conditions have been already applied and that unfrequent items do not appear anymore.
- `core` *operator.* The `core` operator uses these encoded tables and performs the generation of the association rules using known algorithms, e.g., APRIORI [2]. It then provides encoded rules. Basically, from each pair of body and head, elements are extracted to form a rule that satisfy mining conditions and both frequency and confidence criteria.
- *Postprocessor.* At the end of the process, the post-processor decodes the rules and produces the relations containing the desired rules in a table that is also stored in the database.

4 Discussion

Consider an inductive database for association rules. An instance is made of a current dataset and a current collection of rules. As defined in the previous section, both can be materialized as SQL3 tables. Modifying the current dataset implies a new computation of the evaluation function for the current collection of rules. The only way to modify the current collection of rules is to apply selections or set-oriented operations on it. However, a fundamental feature of inductive databases is that these selections can refer to arbitrary rules, e.g., rules whose frequencies and confidences have not been computed beforehand. The challenge for implementing an inductive database is then to use available materializations and constraints in order to compute efficiently the desired rules.

Indeed, the architecture presented in [15] provides a basis for an implementation. Using it, it is possible to define the current dataset, and it is possible to select patterns that hold in it and that satisfy a selection condition like a frequency and/or a confidence threshold, or other mining conditions. Other typical operations (e.g., union of patterns) can be simulated by SQL3 queries over the output tables and a "degenerate" `MINE RULE` statement that would just recompute frequency and confidence values for the current collection of rules using the current dataset. Note that sophisticated postprocessing is often needed in order to reduce the number of rules and that it can also be considered as a query on the current collection of rules. For instance, [7] shows how to use OQL in order to perform "structural rule covering" and "rule covering" [17], two global filtering methods over a collection of association rules stored in a nested relation.

So, the `MINE RULE` architecture can be used to simulate the closure property we defined on inductive databases. This closure property is not cosmetic: it

enables the description of complex mining processes as sequence of queries. This view is an essential basis for an efficient query compilation when replays occur, a frequent situation according to our experience. For instance, suppose that three steps in a scenario lead to the following composition: $\tau_{F_3}(\tau_{F_2}(\tau_{F_1}((r_0, s_0)))) = (r_0, s_1)$, where $F_1 = e(r_0).f \geq 0.05 \wedge e(r_0).c \geq 0.3$, $F_2 = lbody \leq 4$, and $F_3 = A \notin \{body \cup head\}$. Assume now that one decides to replay this sequence on a new dataset. The situation is that some selections are cheap (e.g., bounds on the number of items) while some are very expensive (e.g., bounds on frequency and confidence values). So it is possible to optimize such sequences by combining or modifying the sequence of selections.

From an object-oriented viewpoint, the specific property of inductive databases is that we use some very expensive methods that can lead to database scans. A framework for query optimization has been studied for this case [8] and can serve as a basis for optimization strategies. Note also the interesting work [16] that starts a systematic study of those constraints that can be used actively in order to speed up association rule mining.

Another major issue is to consider the genericity of the approach. There are two aspects: specifying a new inductive database (or some **MINE RULE** operator for other kind of rules) and implementing it. Indeed, an inductive database can easily be defined for other kind of patterns than association rules: episode rules, integrity constraints in databases, classifiers, etc. The feasibility of implementation in some general case is an open problem. The most challenging problem comes from the pattern selection and can be formulated as follows: given a dataset and a potentially infinite collection of patterns, how can we use the selection criteria in order to optimize the generation/evaluation of the relevant patterns.

Generic mining algorithms should also be defined. Interesting ideas come from recent generalizations of APRIORI. [4] generalizes it in the context of frequent atomsets. It provides an inductive logic programming tool that mines the so-called frequent Datalog queries [5]. [18] consider query flocks that are parametrized Datalog (or SQL) queries for which filter condition is related to the number of parameters values. They propose an optimizing scheme that provides subqueries for eliminating parameter values at a cheaper price.

5 Conclusion

The concept of inductive database has been suggested in [9] as a basis for a long-term database perspective on data mining. We presented a formalization for inductive databases considering that the whole process of data mining can be viewed as a querying activity. It appears that without introducing any additional concepts, object-oriented and relational database terminology enable to carry out inductive database querying, though effort to query optimization techniques for expensive selections must be pursued. From the architectural point of view, there is clearly a challenge in identifying the border between mining systems and relational processing that should be given to SQL servers. The architecture proposed by Meo, Psaila and Ceri for the **MINE RULE** operator is a good starting point, but it is an open problem to extend it to other classes of patterns.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD'93*, pages 207 – 216. ACM, 1993.
2. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, 1996.
3. J.-F. Boulicaut, M. Klemettinen, and H. Mannila. Modeling KDD processes within the inductive database framework. Technical Report C-1998-29, Department of Computer Science, P.O. Box 26, FIN-00014 University of Helsinki, Finland, June 1998. Submitted.
4. L. Dehaspe and L. D. Raedt. Mining association rules in multiple relations. In *ILP'97*, volume 1297 of *LNAI*, pages 125–132. Springer-Verlag, 1997.
5. L. Dehaspe and H. Toivonen. Frequent query discovery: a unifying ILP approach to association rule mining. Technical Report CW-258, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, March 1998.
6. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI Press, 1996.
7. B. Goethals, J. V. den Bussche, and K. Vanhoof. Decision support queries for the interpretation of data mining results. Manuscript, University of Limburg (Belgium), available at http://www.luc.ac.be/~vdbuss, 1998.
8. J. M. Hellerstein. Optimization techniques for queries with expensive methods. *ACM Transaction on Database Systems*, 1998. To appear.
9. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58 – 64, Nov. 1996.
10. G. Lausen and G. Vossen. *Models and Languages of Object-Oriented Databases.* Addison-Wesley Publishing Company, 1997.
11. H. Mannila. Inductive databases and condensed representations for data mining. In *ILPS'97*, pages 21–30. MIT Press, 1997.
12. H. Mannila. Methods and problems in data mining. In *ICDT'97*, volume 1186 of *LNCS*, pages 41–55. Springer-Verlag, 1997.
13. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241 – 258, 1997.
14. R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *VLDB'96*, pages 122–133. Morgan Kaufmann, 1996.
15. R. Meo, G. Psaila, and S. Ceri. A tightly-coupled architecture for data mining. In *ICDE'98*, pages 316–322. IEEE Computer Society Press, 1998.
16. R. Ng, L. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *SIGMOD'98*, pages 13–24. ACM, 1998.
17. H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hätönen, and H. 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. MLnet, 1995.
18. D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rozenthal. Query flocks: A generalization of association-rule mining. In *SIGMOD'98*, pages 1–12. ACM, 1998.