Modeling KDD Processes within the Inductive Database Framework

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

* INSA de Lyon, LISI Bâtiment 501, F–69621 Villeurbanne cedex, France

† University of Helsinki, Department of Computer Science
P.O. Box 26, FIN–00014 University of Helsinki, Finland

‡ Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399, USA

Abstract. One of the most challenging problems in data manipulation in the future is to be able to efficiently handle very large databases but also multiple induced properties or generalizations in that data. Popular examples of useful properties are association rules, and inclusion and functional dependencies. Our view of a possible approach for this task is to specify and query inductive databases, which are databases that in addition to data also contain intensionally defined generalizations about the data. We formalize this concept and show how it can be used throughout the whole process of data mining due to the closure property of the framework. We show that simple query languages can be defined using normal database terminology. We demonstrate the use of this framework to model typical data mining processes. It is then possible to perform various tasks on these descriptions like, e.g., optimizing the selection of interesting properties or comparing two processes.

1 Introduction

Data mining, or knowledge discovery in databases (KDD), sets new challenges to database technology: new concepts and methods are needed for general purpose query languages [8]. 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 [12].

Discovering knowledge from data, the so-called KDD process, contains several steps: understanding the domain, preparing the data set, discovering patterns, postprocessing of discovered patterns, and putting the results into use. This is a complex interactive and iterative process for which many related theories have to be computed: different selection predicates but also different classes of patterns must be used.

For KDD, we need a query language that enables the user to select subsets of the data, but also to specify data mining tasks and select patterns from the corresponding theories. Our special interest is in the combined pattern discovery and postprocessing steps via a querying approach. For this purpose, a closure

* Email: jfboulic@lisi.insa-lyon.fr, mklemett@cs.helsinki.fi, mannila@microsoft.com.

Mukesh Mohania and A Min Tjoa (Eds.): DaWaK'99, LNCS 1676, pp. 293-302, 1999

property of the query language is desirable: the result of a KDD query should be an object of a similar type than its arguments. Furthermore, the user must also be able to cross the boundary between data and patterns, e.g., when exceptions to a pattern are to be analysed. This gives rise to the concept of *inductive databases*, i.e., databases that contain inductive 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. The inductive database concept has been suggested in [8, 11]. In this paper, we use the simple formalization we introduced in [4]. However, the topic is different. In [4], we considered the `MINE RULE` operator as a possible querying language on association rule inductive databases. Here we emphasize the genericity of the framework and its use for KDD process modeling. It leads us to propose a research agenda to design general purpose query languages for KDD applications. Our basic message is very simple: (1) 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. (2) An inductive database can be queried (in principle) just by using normal relational algebra or SQL, with the added property of being able to refer to the values of the evaluation function on the patterns. (3) Modeling KDD processes as a sequence of queries on an inductive database gives rise to chances for reasoning and optimizing these processes.

The paper is organized as follows. In Section 2 we define the inductive database framework and introduce KDD queries by means of examples. Section 3 considers the description of KDD processes and the add-value of the framework for their understanding and their optimization. Section 4 is a short conclusion with open problems concerning the research in progress.

2 Inductive Databases

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

Example 1 *If the patterns are boolean formulae about the database, $\mathcal{V}$ is* {`true`, `false`}*, and the evaluation function $e(\mathbf{r}, \theta)$ has value true iff the formula θ is true about $\mathbf{r}$. In practice, a user might select the true or the false formulas from the intensionally defined collection of all boolean formulas.* □

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 (as it is the case for boolean formulas), an implementation can 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.

A typical KDD process operates on both of the components of an inductive database. The user can select a subset of the rows or more generally select data from the database or the data warehouse. In that case, the pattern component remains the same. The user can also select subsets of the patterns, and in the answer the data component is the same as before.

The situation can be compared with deductive databases where some form of deduction is used to augment fact databases with a potentially infinite set of derived facts. However, within the inductive database framework, the intensional facts denote generalizations that have to be learned from the data. So far, the discovery of the patterns we are interested in can not be described using available deductive database mechanisms.

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 semantics of the patterns.

In the following, we first illustrate the framework on association (Section 2.1), and then we generalize the approach and point out key issues for query evaluation in general (Section 2.2).

2.1 Association Rules

The association rule mining problem has received much attention since its introduction in [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 $B \in R \setminus X$. 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 column B. This semantics is captured by *frequency* and *confidence* values. Given $W \subseteq \mathbf{R}$, $support(W, \mathbf{r})$ denotes the fraction of rows of $\mathbf{r}$ that have a 1 in each column of W. The frequency of $X \Rightarrow B$ in $\mathbf{r}$ is defined to be $support(X \cup \{B\}, \mathbf{r})$ while its confidence is $support(X \cup \{B\}, \mathbf{r})/support(X, \mathbf{r})$. Typically, we are interested in association rules for which the frequency and the confidence are greater than given thresholds. Though an exponential search space is concerned, association rules can be computed thanks to these thresholds on one hand and a safe pruning criteria that drastically reduce the search space on the other hand (the so-called `apriori` trick [2]).

However, the corresponding inductive database schema defines intensionally all the potential association rules. 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 $\mathbf{r}$. Notice that many other objective interestingness measures have been introduced for that kind of patterns (e.g., the J-measure [15] or the conviction [5]). All these measures could be taken into account by a new evaluation function.

We now describe the querying approach by using self-explanatory notations

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.

for the simple extension of the relational algebra that fits to our need [2].

Example 2 *Mining association rules is now considered as querying inductive database instances of schema* $(R, (\mathcal{Q}_R, e, [0,1]^2))$. *Let us consider the data set is the instance* r_0 *in Table 1 of the relational schema* $R = \{A, B, C\}$.

The inductive database $idb = (r_0, s_0)$ *associates to* r_0 *the association rules on the leftmost table of Table 1. Indeed, in such an example, the intensionally defined collection of all the association rules can be presented. We illustrate (1) the selection on tuples, and (2) the selection on patterns in the typical situation where the user defines some thresholds for frequency and confidence.*

1. $\sigma_{A \neq 0}(idb) = (r_1, s_1)$ *where* $r_1 = \sigma_{A \neq 0}(r_0)$ *and* s_1 *contains the association rules in the middle table of Table 1.*
2. $\tau_{e(r_0).f \geq 0.5 \wedge e(r_0).c \geq 0.7}\ (idb) = (r_2, s_2)$ *where* $r_2 = r_0$ *and* s_2 *contains the association rules from the rightmost table (on the top) of Table 1.*

To simplify the presentation, we have denoted by $e(\mathbf{r}).f$ *and* $e(\mathbf{r}).c$ *the values for frequency and confidence.* □

An important feature is that operations can be composed due to the closure property.

Example 3 *Consider that the two operations given in Example 2 are composed and applied to the instance* $idb = (r_0, s_0)$. *Now,* $\tau_{e(r_0).f \geq 0.5 \wedge e(r_0).c \geq 0.7}\ (\sigma_{A \neq 0}(idb))$ $= (r_3, s_3)$ *where* $r_3 = \sigma_{A \neq 0}(r_0)$ *and* s_3 *is reduced to the association rule* $C \Rightarrow A$ *with frequency 0.66 and confidence 1.* □

The selection of association rules given in that example is rather classical. Of course, a language to express selection criteria has to be defined. It is out of the scope of this paper to provide such a definition. However, let us just emphasize that less conventional association rule mining can also be easily specified.

Example 4 *Consider an instance* $idb = (r_0, s_0)$. *It can be interesting to look for rules that have a high confidence and whose right-hand side does not belong to a set of very frequent attributes* F: $\tau_{e(r_0).c \geq 0.9 \wedge e(r_0).rhs \notin F}(idb)) = (r_0, s_1)$. *The intuition is that rhs denotes the righ-hand side of an association rule. The*

[2] Selection of tuples and patterns are respectively denoted by σ and τ. As it is always clear from the context, the operation can also be applied on inductive database instances while formally, we should introduce new notations for them.

rules in s_1 are not all frequent (no frequency constraint) but have a rather high confidence while their right-hand sides are not very frequent. Indeed, computing unfrequent rules will be in practice untractable except if other constraints can help to reduce the search space (and are used for that during the mining process). □

The concept of exceptional data w.r.t. a pattern or a set of patterns is interesting in practice. So, in addition to the normal algebraic operations, let us introduce the so-called *apply operation*, denoted by α, that enables to cross the boundary between data and patterns by removing the tuples in the data set such that all the patterns are true in the new collection of tuples.

In the case of association rules, assume the following definition: a pattern θ is false in the tuple t if its left-hand side holds while its right-hand side does not hold; in the other cases a pattern is true. In other terms, an association rule θ is true in a tuple $t \in r$ iff $e(\{t\}, \theta).f = e(\{t\}, \theta).c = 1$. Let us define $\alpha((r, s)) = (r', s)$ where r' is the greatest subset of r such that $\forall \theta \in s$, $e(r', \theta).c = 1$. Note that $r' \setminus r$ is the collection of tuples that are exceptions w.r.t. the patterns in s.

Example 5 *Continuing Example 2, assume the instance (r_0, s_4) where s_4 contains the rule $AC \Rightarrow B$ with frequency 0.25 and confidence 0.5. Let $\alpha((r_0, s_4)) = (r_4, s_4)$. Only the tuple $\langle 1, 0, 1 \rangle$ is removed from r_0 since the rule $AC \Rightarrow B$ is true in the other ones. The pattern $AC \Rightarrow B$ remains the unique pattern (s_4 is unchanged) though its frequency and confidence in r_4 are now 0.33 and 1, respectively.* □

2.2 Generalization to Other Pattern Types

The formal definition we gave is very general. In this section, we first consider an other example of data mining task where inductive database concepts can be illustrated. We also point out crucial issues for query evaluation.

One typical KDD process we studied is the discovery of approximate inclusion and functional dependencies in a relational database. It can be useful either for debugging purposes, semantic query optimization or even reverse engineering [3]. We suppose that the reader is familiar with data dependencies in relational databases.

Example 6 *Assume $R = \{A, B, C, D\}$ and $S = \{E, F, G\}$ with the two following instances in which, among others, $S[\langle G \rangle] \subseteq R[\langle A \rangle]$ is an inclusion dependency and $AB \rightarrow C$ a functional dependency (see Table 2(a–b)).* □

Dependencies that almost hold are interesting: it is possible to define natural error measures for inclusion dependencies and functional dependencies. For instance, let us consider an error measure for an inclusion dependency $R[X] \subseteq S[Y]$ in $\mathbf{r}$ that gives the proportion of tuples that must be removed from r, the instance of R, to get a true dependency. With the same idea, let us consider an error measure for functional dependencies that gives the minimum number of rows that need to be removed from the instance r of R for a dependency $R : X \rightarrow B$ to hold.

Example 7 *Continuing Example 6, a few approximate inclusion and functional dependencies are given (see Table 2(c)).* □

A	B	C	D
1	2	4	5
2	2	2	3
3	1	1	2
4	2	2	3

(a)

E	F	G
1	2	3
2	3	4
3	2	2

(b)

Inclusion dependencies	Error
$R[\langle B\rangle] \subseteq S[\langle E\rangle]$	0
$R[\langle D\rangle] \subseteq S[\langle E\rangle]$	0.25
$S[\langle E\rangle] \subseteq R[\langle B\rangle]$	0.33
$R[\langle C, D\rangle] \subseteq S[\langle E, F\rangle]$	0.25

Functional dependencies	Error
$B \rightarrow A$	0.5
$C \rightarrow A$	0.25
$BC \rightarrow A$	0.25
$BCD \rightarrow A$	0.25

(c)

Table 2. Tables for Examples 6 and 7.

It is now possible to consider the two inductive databases that associate to a database all the inclusion dependencies and functional dependencies that can be built from its schema. Evaluation functions return the respective error measures. When the error is null, it means that the dependency holds. Indeed, here again it is not realistic to consider that querying can be carried out by means of queries over some materializations of all the dependencies that almost hold.

Example 8 *Continuing again Example 6, a user might be interested in "selecting" only inclusion dependencies between instances r and s that do not involve attribute R.A in their left-hand side and have an error measure lower than* 0.3*. One expects that a sentence like* $R[\langle C, D\rangle] \subseteq S[\langle E, F\rangle]$ *belongs to the answer. The "apply" operation can be used to get the tuples that are involved in the dependency violation. One can now search for functional dependencies in s whose left-hand sides are a right-hand side of a previously discovered inclusion dependency. For instance, we expect that a sentence like* $EF \rightarrow G$ *belongs to the answer. Evaluating this kind of query provides information about potential foreign keys between* $R = \{A, B, C, D\}$ *and* $S = \{E, F, G\}$*.* □

Query evaluation We already noticed that object-relational query languages can be used as a basis for inductive database query languages. However, non-classical optimization schemes are needed since selections of properties lead to complex data mining phases. Indeed, implementing such query languages is difficult because selections of properties are not performed over previously materialized collections. First one must know efficient algorithms to compute collection of patterns and evaluate the evaluation function on very large data sets. But the most challenging issue is the formal study of selection language properties for general classes of patterns: given a data set and a potentially infinite collection of patterns, how can we exploit an active use of a selection criteria to optimize the generation/evaluation of the relevant patterns.

Example 9 *When mining association rules that do not involve a given attribute, instead of computing all the association rules and then eliminate those which contain that attribute, one can directly eliminate that attribute during the candidate generation phase for frequent sets discovery. Notice that such a simple trick can not be used if the given attribute must be avoided in the left-hand side only.* □

The complexity of mining frequent association rules mainly consist of finding frequent sets. Provided boolean constraints over attributes, [16] show how

to optimize the generation of frequent sets using this kind of constraints during the generation/evaluation process. This approach has been considerably extended in [14]. Other interesting ideas come from the generalization of the `apriori` trick, and it can be found in different approach like [6] or [17]. [6] propose an algorithm that generalize the `apriori` trick to the context of frequent atomsets. This typical inductive logic programming tool enable to mine association rules from multiple relations. [17] consider query flocks that are parametrized Datalog queries for which a selection criteria on the result of the queries must hold. When the filter condition is related to the frequency of answers and queries are conjunctive queries augmented with arithmetic and union, they can propose an optimizing scheme. In the general framework, three important questions arise:

1. *How to evaluate a class of similar patterns faster than by looking at each of them individually?* An explicit evaluation of all the patterns of the schema against the database (and all databases resulting from it by queries) is not feasible for large data sets. Safe pruning criteria have to be found.
2. *How to evaluate patterns without looking at the whole data set?* This is an important issue to reduce dimensionality of the mining task, e.g., via sampling. In somes cases, it might be also possible not to use the data set and perform a simple selection over a previously materialized collection of patterns or more or less condensed representation [11].
3. *How to evaluate operation sequences, e.g., in replays, more efficiently?* Compiling schemes can be defined for this purpose. For instance, crucial issues are the study of pattern selection commutativity for useful classes of patterns. The formal study of selection criteria for pattern classes that are more complex than frequent sets is to be done.

A framework for object-oriented query optimization when using expensive methods [7] can also serve as a basis for optimization strategies.

3 Inductive Databases and KDD Processes

Already in the case of a unique class of patterns, real-life mining processes are complex. This is due to the dynamic nature of knowledge acquisition, where gathered knowledge often affects the search process, giving rise to new goals in addition to the original ones.

In the following, we introduce a scenario about telecommunication networks fault analysis using association rules. It is a simplified problem of knowledge discovery to support off-line network surveillance, where a network manager tries to identify and correct faults based on sent alarms. A comprehensive discussion on this application is available in [10].

Assume that the schema for the data part is R = *(alarm type, alarming element, element type, date, time, week, alarm severity, alarm text)*. We consider items as equalities between attributes and values, while rule left-hand and right-hand sides are sets of items. Notice also that we use in the selection conditions expressions that concern subcomponents of the rules. Typically, one wants to select rules with a given attribute on the left-hand side (LHS) or on its right-hand

alarm type	alarming element	element type	date	time	week	alarm severity	alarm text
1111	E1.1	ABC	980119	233605	4	1	LINK FAILURE
2222	E2	CDE	980119	233611	4	3	HIGH ERROR RATE
3333	A	EFG	980119	233627	4	1	CONNECTION NOT ESTABLISHED
4444	B2.1	GHI	980119	233628	4	2	LINK FAILURE

s_0	$e(r_0).f$	$e(r_0).c$
alarm_type=1111 ⇒ element_type=ABC	0.25	1.00
alarm_type=222 ⇒ alarming_element=E2, element_type=CDE	0.25	1.00
alarm_type=1111, element_type=ABC ⇒ alarm_text=LINK_FAILURE	0.25	1.00
alarm_type=5555 ⇒ alarm_severity=1	0.00	0.00

Table 3. Part of an inductive database consisting of data part r_0 (upper table) and rule part s_0 (lower table).

side (RHS), or give bounds to the number of occurring items. Self-explanatory notations are used for this purpose. A sample of an instance of this schema is given in Table 3.

Scenario The network manager decides to look at association rules derived from r_0, the data set for the current month. Therefore, he/she "tunes" parameters for the search by pruning out all rules that have confidence under 5% or frequency under 0.05% or more than 10 items (phase 1 in Table 4). The network manager then considers that attributes "alarm text" and "time" are not interesting, and projects them away (phase 2). The number of rules in the resulting rule set, s_2, is still quite large. The user decides to focus on the rules from week 30 and to restrict to 5 the maximum amount of items in the rule (phase 3). While browsing the collection of rules s_3, the network manager sees that a lot of rules concern the network element E. That reminds him/her of maintenance operation and he/she decides to remove all rules that contain "alarming element = E or its subcomponent" (phase 4). We omit the explanation of dealing with the taxonomy of components. The resulting set of rules seems not to show anything special. So, the network manager decides to compare the behavior of the network to the preceding similar period (week 29) and find out possible differences (phases 5–6). The network manager then picks up one rule, s_8, that looks interesting and is very strong (confidence is close to 1), and he wants to find all exceptions to this rule; i.e. rows, where the rule does not hold (phases 7–8).

Except for the last phases, the operations are quite straightforward. In the comparison operation, however, we must first replay the phases 3–4. This is because we have to remove the field "week" from the schema we used in creating rules for week 30, so that we can compare these rules with the rules from week 29. Then we create for week 29 the same query (except for the week information), take the intersection from these two rulesets, and calculate the frequencies and confidences of the rules in the intersection. The search for exceptions is performed using the *apply operation* introduced in Section 2.

This simple scenario illustrates a typical real-life data mining task. Due to the closure property, KDD processes can be described by sequences of operations, i.e., queries over relevant inductive databases. In fact, such sequences of queries are abstract and concise descriptions of data mining processes. An inter-

Phase	Operation	Query and conditions
1	*Selection*	$\tau_{F_1}((r_0, s_0)) = (r_0, s_1)$ $F_1 = e(r_0).f \geq 0.005 \wedge e(r_0).c \geq 0.05 \wedge \lvert LHS\rvert \leq 10$
2	*Projection*	$\pi_T((r_0, s_1)) = (r_1, s_2)$ $T = R \setminus \{alarm\ text, time\}$
3	*Selection*	$\tau_{F_2}(\sigma_{C_1}((r_1, s_2)))) = (r_2, s_3)$ $C_1 = (week = 30)$ and $F_2 = \lvert LHS \cup RHS\rvert \leq 5$
4	*Selection*	$\tau_{F_3}((r_2, s_3)) = (r_2, s_4)$ $F_3 = (\text{alarming element} = E*) \notin \{LHS \cup RHS\}$
5	*Replay 3–4 (week 30)*	$\tau_{F_3}(\tau_{F_2}(\pi_U(\sigma_{C_1}((r_1, s_2))))) = (r_3, s_5)$ $U = T \setminus \{week\}$, other conditions as in 3–4
6	*Replay 3–4 (week 29)*	$\tau_{F_3}(\tau_{F_2}(\pi_U(\sigma_{C_2}((r_1, s_2))))) = (r_4, s_6)$ $C_2 = (week = 29)$, other conditions as in 5
7	*Intersection*	$\cap((r_3, s_5), (\emptyset, s_6)) = (r_3, s_7)$
8	*Apply*	$\alpha((r_3, s_8)) = (r_5, s_9)$

Table 4. *Summary of the phases of the experiment.*

esting point here is that these descriptions can even be annotated by statistical information about the size of selected dataset, the size of intermediate collection of patterns etc., providing knowledge for further use of these sequences.

4 Conclusions and Future Work

We presented a framework for inductive databases considering that the whole process of data mining can be viewed as a querying activity. Our simple formalization of operations enables the definition of mining processes as sequence of queries, thanks to a closure property. The description of a non-trivial mining process using these operations has been given and even if no concrete query language or query evaluation strategy is available yet, it is a mandatory step towards general purpose query languages for KDD applications.

Query languages like M-SQL [9] or `MINE RULE` [13] are good candidates for inductive database querying though they are dedicated to boolean and association rule mining, respectively. A simple Pattern Discovery Algebra has been proposed in [18]. It supports pattern generation, pattern filtering and pattern combining operations. This algebra allows the user to specify discovery strategies, e.g., using different criteria of interestingness but at a macroscopic level; implementation issues or add-value for supporting the mining step are not considered.

We introduced, as an example, an inductive database for association rules, and gave a realistic scenario using simple operations. It appears that without introducing any additional concepts, standard database terminology enable to carry out inductive database querying and that recent contributions to query optimization techniques can be used for inductive database implementation. A significant question is whether the inductive database framework is interesting for a reasonable collection of data mining problems. We currently study KDD processes that need different 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, May 1993. ACM.
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. A KDD framework to support database audit. In *WITS'98*, volume TR 19, pages 257 – 266, December 1998. University of Jyväskylä.
4. J.-F. Boulicaut, M. Klemettinen, and H. Mannila. Querying inductive databases: A case study on the MINE RULE operator. In *PKDD'98*, volume 1510 of *LNAI*, pages 194 – 202, September 1998. Springer-Verlag.
5. S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97*, pages 255 – 264, 1997. ACM Press.
6. L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In *Proceedings 7th Int'l Workshop on Inductive Logic Programming*, volume 1297 of *LNAI*, pages 125–132. Springer-Verlag, 1997.
7. J. M. Hellerstein. Optimization techniques for queries with expensive methods. *ACM Transaction on Database Systems*, 1998. Available at http://www.cs.berkeley.edu/~jmh/miscpapers/todsxfunc.ps.
8. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58 – 64, November 1996.
9. T. Imielinski, A. Virmani, and A. Abdulghani. DataMine: Application programming interface and query language for database mining. In *KDD'96*, pages 256 – 261, August 1996. AAAI Press.
10. M. Klemettinen, H. Mannila, and H. Toivonen. Rule discovery in telecommunication alarm data. *Journal of Network and Systems Management*, 1999. To appear.
11. H. Mannila. Inductive databases and condensed representations for data mining. In *Proceedings of the International Logic Programming Symposium (ILPS'97)*, pages 21 – 30, October 1997. MIT Press.
12. H. Mannila. Methods and problems in data mining. In *ICDT'97*, volume 1186 of *LNCS*, pages 41–55. Springer-Verlag, 1997.
13. R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *VLDB'96*, pages 122 – 133, September 1996. Morgan Kaufmann.
14. R. Ng, L. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *SIGMOD'98*, pages 13 – 24, 1998. ACM Press.
15. P. Smyth and R. M. Goodman. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(4):301 – 316, August 1992.
16. R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *KDD'97*, pages 67 – 73, 1997. AAAI Press.
17. D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: A generalization of association-rule mining. In *SIGMOD'98*, pages 1 – 12, 1998. ACM Press.
18. A. Tuzhilin. A pattern discovery algebra. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Technical Report 97-07 University of British Columbia*, pages 71 – 76, 1997.