A Relational View of Pattern Discovery

Arnaud Giacometti, Patrick Marcel, and Arnaud Soulet

Université François Rabelais Tours, LI
3 place Jean Jaurès
F-41029 Blois France
forename.surname@univ-tours.fr

Abstract. The elegant integration of pattern mining techniques into database remains an open issue. In particular, no language is able to manipulate data and patterns without introducing opaque operators or loop-like statement. In this paper, we cope with this problem using relational algebra to formulate pattern mining queries. We introduce several operators based on the notion of cover allowing to express a wide range of queries like the mining of frequent patterns. Beyond modeling aspects, we show how to reason on queries for characterizing and rewriting them for optimization purpose. Thus, we algebraically reformulate the principle of the levelwise algorithm.

1 Introduction

Pattern discovery is a significant field of Knowledge Discovery in Databases (KDD). A broad spectrum of powerful techniques for producing local patterns has been developed over the two last decades [3–5]. But, it is widely agreed that the need of theoretical fusion between database and data mining still remains a crucial issue [14, 18, 23, 24]. We would force the pattern mining methods to fit in the relational model [1] which is the main database theory. Unlike most of the proposals [6, 10, 14, 16, 20, 23, 28, 33, 34], we desire to only address the pattern mining that we distinguish from the construction of global models [17] like decision trees.

Let us consider the popular task of frequent pattern mining [3] as a motivating example. Most works treat this task as a "black box" which input parameters are defined by the user [6, 7, 14, 16, 20, 28, 32, 34]. Instead of only specifying the minimal frequency threshold and the dataset, we think that the user query should fully formalize the notion of frequent patterns (e.g., it should describe how the frequency of a pattern is computed starting from the dataset). Ideally, we would like to express the frequent pattern mining query in the relational algebra in order to manipulate both the data and the patterns. As declarative aspects should be promoted on physical ones, a pattern discovery process has to be fully specified without considering algorithmic points. For this purpose, loop-like operators [10, 23, 33] are not relevant for us. Furthermore, the improvement of query performances mainly rests on physical optimizations in the field of pattern mining. Typically, the frequent pattern mining is efficiently performed

J.X. Yu, M.H. Kim, and R. Unland (Eds.): DASFAA 2011, Part I, LNCS 6587, pp. 153–167, 2011.

by an adequate implementation [3–5, 25]. Such algorithmic optimizations (even specified at a higher level [10, 23, 33]) reduce the opportunity of integrating other optimizations. We prefer to favor logical reasoning for optimizing query performances. For instance, the rewriting of the naive frequent pattern mining query should enable us to algebraically formulate the levelwise pruning [25].

The main goal of this paper is to propose an algebraic framework for pattern discovery for expressing a wide range of queries without introducing opaque operators or loop-like statements. Our framework brings two meaningful contributions: expressive modeling and logical reasoning. First, it allows a large set of queries manipulating relations which contain both data and patterns. We add to the relational algebra several specific operators, like the cover operator $\triangleleft$, to coherently and easily join such relations. We also define a new operator Δ for generating a language starting from a relation. Typically, the query $\sigma_{freq \geq f}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(\Delta(L) \triangleleft D))$ returns the patterns of language L frequent in dataset D. Second, the pattern-oriented relational algebra enables to characterize and rewrite queries in order to optimize their performance. In particular, we formalize the notions of syntactic constraint [9] and global constraint [12] by characterizing the degree of dependence between a query and a relation. Besides, we not only benefit from usual query rewriting methods stemming from the relational model, but we also algebraically reformulate the levelwise pruning.

This paper is organized in the following way. Section 2 introduces basic notions about the relational algebra and the pattern discovery. Section 3 defines the cover-like and domain operators which are at the core of our algebra. We then study the properties of downward closure and independence in Section 4. We rewrite queries satisfying such properties for optimization purpose in Section 5. Finally, Section 6 provides a related work.

2 Basic Notions

2.1 Relational Algebra

We enumerate here our notations for the relational algebra mainly inspired from [1]. Let $\mathbf{att}$ be a set of distinct literals, named *attributes*, $\mathbf{dom}(A)$ denotes the finite *domain* of the attribute $A \in \mathbf{att}$. The *relation schema* (or relation for brevity) $R[U]$ denotes a relation named by R where $U \subset \mathbf{att}$. An *instance* of R is a subset of $\mathbf{dom}(U) = \times_{A \in U} \mathbf{dom}(A)$. Given a relation $R[A_1, \ldots, A_n]$, R' renames the attributes $A_1, \ldots, A_n$ into $A'_1, \ldots, A'_n$. A *database schema* is a nonempty finite set $\mathbf{R} = \{R_1[U_1], \ldots, R_n[U_n]\}$ of relations. A *database instance* of $\mathbf{R}$ is a set $\mathbf{I} = \{I_1, \ldots, I_n\}$ such that I_i is an instance of the relation R_i. Finally, a *query* q maps a database instance to an instance of a relation. The set of attributes of this relation is denoted by $\mathbf{sch}(q)$. A query q' is *equivalent* to q, denoted by $q' \equiv q$, iff for any database instance $\mathbf{I}$, one has $q'(\mathbf{I}) = q(\mathbf{I})$.

Let I be an instance of R and J be an instance of S. The relations can be manipulated by means of set operators including Cartesian product $R \times S$ where $I \times J = \{(t, u) | t \in I \wedge u \in J\}$. If R and S are relations which have the same schema, then $R \cup S$, $R \cap S$ and $R - S$ are respectively the union, the intersection

and the difference of R and S. *Selection:* $\sigma_f(I) = \{t | t \in I \wedge f(t)\}$ selects the tuples of I satisfying the logical formula f where f is built from (i) the logical operators ($\wedge$, $\vee$ or $\neg$), (ii) the arithmetic relational operators and (iii) operands based on attributes and constants. *Extended projection:* $\pi_{A_1,\ldots,A_n}(I) = \{t[A_1,\ldots,A_n] | t \in I\}$ only preserves the attributes $A_1,\ldots,A_n$ of R. Besides, the projection also permits to extend the relation by arithmetic expressions and to (re)name expressions. For instance, $\pi_{A+B \to B', C \to C'}(R)$ creates a new instance where the first attribute named B' results from the arithmetic expression $A + B$ and the second attribute corresponds to C, renamed C'. *Grouping:* $\gamma_{A_1,\ldots,A_n,\texttt{AGG}(B)}(I) = \{(a_1,\ldots,a_n, \texttt{AGG}(\pi_B(\sigma_{A_1=a_1 \wedge \cdots \wedge A_n=a_n}(I))) \mid (a_1,\ldots,a_n) \in \pi_{A_1,\ldots,A_n}(I)\}$ groups tuples of I by attributes $A_1,\ldots,A_n$ and applies an aggregate function AGG on B.

2.2 Pattern Discovery

We provide here an overview of pattern discovery based on [25, 32] focusing on the main proposals of the field. A *language* $\mathcal{L}$ is a set of patterns: itemsets $\mathcal{L}_\mathcal{I}$ [3], sequences $\mathcal{L}_\mathcal{S}$ [4] and so on [5]. A *specialization relation* $\preceq$ of a language $\mathcal{L}$ is a partial order relation on $\mathcal{L}$ [25, 27]. Given a specialization relation $\preceq$ on $\mathcal{L}$, $l \preceq l'$ means that l is more general than l', and l' is more specific than l. For instance, the set inclusion is a specialization relation for the itemsets. Given two posets $(\mathcal{L}_1, \preceq_1)$ and $(\mathcal{L}_2, \preceq_2)$, a *cover relation* is a binary relation $\lhd \subseteq \mathcal{L}_1 \times \mathcal{L}_2$ iff when $l_1 \lhd l_2$, one has $l'_1 \lhd l_2$ (resp. $l_1 \lhd l'_2$) for any pattern $l'_1 \preceq_1 l_1$ (resp. $l_2 \preceq_2 l'_2$). The relation $l_1 \lhd l_2$ means that l_1 covers l_2, and l_2 is covered by l_1. The cover relation is useful to relate different languages together (e.g., for linking patterns to data). Note that a specialization relation on $\mathcal{L}$ is also a cover relation on $\mathcal{L}$ (e.g., the set inclusion is a cover relation for the itemsets).

The pattern can be manipulated by means of three kinds of operators non exhaustively illustrated hereafter. 1) *Pattern mining operators* produce patterns starting from a dataset: theory [25], MINERULE [26] and so on. More precisely, the *theory* denoted by $\mathcal{T}h(\mathcal{L}, q, \mathcal{D})$ returns all the patterns of a language $\mathcal{L}$ satisfying a predicate q in the dataset $\mathcal{D}$ [25]. Typically, the minimal frequency constraint selects the patterns which occur in at least f transactions [3, 4]: $freq(\varphi, D) > f$. As mentioned in introduction, we notice that the query $\mathcal{T}h(\mathcal{L}, freq(\varphi, \mathcal{D}) \geq f, \mathcal{D})$ does not make explicit how the frequency of a pattern is computed from the dataset. Other approaches find the k patterns maximizing a measure m in the dataset $\mathcal{D}$ [12, 15]. 2) *Pattern set reducing operators* compress a collection of patterns. For instance, the minimal and maximal operator denoted by $\mathcal{M}in(\mathcal{S})$ and $\mathcal{M}ax(\mathcal{S})$, return respectively the most general and specific patterns of $\mathcal{S}$ w.r.t. a specialization relation $\preceq$ [25]. The notion of negative and positive borders [25] is very similar. 3) *Pattern applying operators* cross patterns and data. For instance, the data covering operator $\theta_d(P, \mathcal{D}) = \{d \in \mathcal{D} | \exists p \in P : p \lhd d\}$ returns the data of $\mathcal{D}$ covered by at least one pattern of P [32]. Dually, the pattern covering operator $\theta_p(P, \mathcal{D})$ returns the patterns of P covering at least one element of $\mathcal{D}$ [32].

The next sections aim at stating an algebra based on the relational model to simultaneously and homogeneously handle data and patterns. In particular, all the manipulations of patterns described here will be expressed in our algebra.

3 Pattern-Oriented Relational Algebra

3.1 Pattern-Oriented Attributes

The pattern-oriented relational algebra pays attention to the attributes describing patterns, named *pattern-oriented attributes*. Indeed, several operations are specifically designed to handle such attributes which the domain corresponds to a pattern language together with a specialization relation.

Definition 1 (Pattern-oriented attributes). *The pattern-oriented attributes* **patt** *is a subset of the attributes:* $\mathbf{patt} \subseteq \mathbf{att}$ *such that for every* $A \in \mathbf{patt}$, $\mathbf{dom}(A)$ *is a poset. Let* $U \subseteq \mathbf{att}$ *be a set of attributes, the pattern-oriented attributes of* U *is denoted by* $\widetilde{U}$.

For example, Table 1 provides instances of relations D, L and P containing pattern-oriented attributes. The relations $D[trans]$ and $L[patt]$ respectively describe a transactional dataset and the corresponding language in the context of (a) itemsets and (b) sequences. The relation $P[item, type, price]$ gives the item identifier, the type and the price of products. We consider that *trans*, *patt* and *item* are pattern-oriented attributes where $\mathbf{dom}(item) = \mathcal{I}$ and $\mathbf{dom}(trans) = \mathbf{dom}(patt) = \mathcal{L}_{\mathcal{I}}$ for itemsets (or $= \mathcal{L}_{\mathcal{S}}$ for sequences). Thereafter, the proposed queries can address instances where the domain of *patt* differs from that of *trans*.

Of course, the relations can be handled with relational operators. For instance, the query $\sigma_{patt \preceq \varphi}(L)$ returns all the patterns of L being more general than the pattern φ. The formula $patt \preceq \varphi$ is allowed because $\sigma_{patt \preceq \varphi}(L) \equiv \pi_{patt}(\sigma_{patt=left \wedge right=\varphi}(L \times C))$ where the relation $C[left, right]$ extensively enumerates in its instance the tuples (l, r) such that $l \preceq r$. On the contrary, the query $\sigma_{freq(patt,D) \geq f}(L)$ is not correct for computing the frequent patterns

Table 1. Instances for pattern discovery

(a) Itemset context

D

trans
ABE
ABC
ABCD
AD

Dataset

L

patt
∅
A
B
C
D
E
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
ABC
...
ABCDE

Language of itemsets

(b) Sequence context

D

trans
(AB)(E)
(AB)(C)(A)
(AB)(C)(D)
(B)(C)(D)(B)

Sequential data

L

patt
∅
(A)
(B)
(C)
(D)
(E)
(AB)
(A)(B)
...

Language of sequences

P

item	type	price
A	*snack*	3
B	*snack*	10
C	*beer*	5
D	*soda*	8
E	*soda*	6

Product description

because the formula $freq(patt, D)$ requires a relation D and it is not allowed in a selection (see Section 2.1). Besides, we desire to make the computation of frequency explicit. The next section explains how to compute it with the relational algebra.

3.2 Cover, Semi-cover and Anti-cover Operators

We now indicate how to formulate the frequent pattern mining query (fpm query in brief) in the relational algebra which illustrates the need of the cover-like operators. Assume that $L[patt]$ and $D[trans]$ are two relations that respectively contain the language and the dataset as proposed in Table 1. The main challenge is to compute the frequency of each pattern of L. The Cartesian product of L by D gathers all the patterns of L with all the transactions of D. Of course, we only select the relevant tuples such that the pattern covers the transaction: $\sigma_{patt \lhd trans}(L \times D)$. Finally, we count for each pattern how many transactions it covers and we select the frequent ones: $\sigma_{freq \geq s}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(\sigma_{patt \lhd trans}(L \times D)))$. As the notion of cover relation plays a central role to relate pattern-oriented attributes, we introduce three operators based on this notion. The cover operator for the pattern discovery is as important as the join operator for classical data manipulations.

Cover operator. The result of a cover operation gathers all the combinations of tuples in R and S that have comparable pattern-oriented attributes.

Definition 2 (Cover operation). *The cover of a relation $R[U]$ for a relation $S[V]$ w.r.t. a cover relation*[1] $\lhd \subseteq \mathbf{dom}(\widetilde{U}) \times \mathbf{dom}(\widetilde{V})$ *is* $R \lhd S = \sigma_{\widetilde{U} \lhd \widetilde{V}}(R \times S)$, *i.e. for any instances I of R and J of S,* $I \lhd J = \{(t,u) | t \in I \wedge u \in J \wedge t[\widetilde{U}] \lhd u[\widetilde{V}]\}$.

As θ-join is a shortcut of $\sigma_f(R \times S)$, the cover operator is derived from primitive operations defined in Section 2.1. In fact, $R \lhd S$ is equivalentl to $\sigma_{\widetilde{U} \lhd \widetilde{V}}(R \times S)$ where the formula $\widetilde{U} \lhd \widetilde{V}$ can be expressed with usual relational operators as done above with $patt \preceq \varphi$. Then, as semi-cover and anti-cover defined below, the cover operator does not increase the expressive power of the relational algebra. However, such operators bring two main advantages. First, algebraic properties of cover-like operators can be formulated, in order to be used by a query optimizer (see Section 5). Second, specialized and efficient query evaluation methods for these operators could be developed.

Let us illustrate the cover operation on several examples of pattern manipulations. Given a dataset $D[trans]$ and a language $L[patt]$, the frequent patterns (with their frequency) correspond to the following query:

$$F = \sigma_{freq \geq f}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(L \lhd D))$$

This fpm query fulfills our modeling objective by explicitly and declaratively describing how the frequency is computed. Given the instances of L and D

[1] Definitions 2 to 4 consider that the binary relation $\lhd$ is a cover relation w.r.t. the specialization relations $\preceq_{\widetilde{U}}$ and $\preceq_{\widetilde{V}}$ respectively defined on $\mathbf{dom}(\widetilde{U})$ and $\mathbf{dom}(\widetilde{V})$.

Table 2. Instances containing mined patterns of instance D

(a) Itemset language

F

patt	freq
$\emptyset$	4
A	4
B	3
C	2
D	2
AB	3
AC	2
AD	2
BC	2
ABC	2

Frequent itemsets

C

patt	freq
A	4
AB	3
AD	2
ABC	2

Frequent closed itemsets

M

patt	freq
AD	2
ABC	2

Frequent maximal itemsets

(b) Sequence language

F

patt	freq
$\emptyset$	4
(A)	3
(B)	4
(C)	3
(D)	2
(AB)	3
(A)(C)	2
(B)(C)	3
(B)(D)	2
(C)(D)	2
(AB)(C)	2
(B)(C)(D)	2

Frequent sequences

provided by Table 1 and $f = 2$, it exactly returns the instance of F (see Table 2). In the fpm query, the relation $\triangleleft \subseteq \mathbf{dom}(patt) \times \mathbf{dom}(trans)$ is a cover relation w.r.t. $\preceq_{patt}$ and $\preceq_{trans}$ (e.g., the inclusion for itemsets [3] or sequences [4]).

As mentioned earlier, a specialization relation is a particular kind of cover relation. Thereby, it can be exactly used as a cover operator. For instance, starting from the frequent patterns F, the frequent closed patterns of D [5] are computed as follows: $C = \pi_{patt,freq}(\sigma_{freq>max}(\gamma_{patt,freq,\texttt{MAX}(freq')\rightarrow max}(F \prec F')))$ (we recall that F' renames the attributes *patt* and *freq* into *patt'* and *freq'*). Table 2 illustrates this query applied to a particular instance of F in the case of itemsets. Furthermore, the query $\gamma_{patt,\texttt{MAX}(freq')\rightarrow freq}(L \preceq C')$ regenerates the instance F.

Semi-cover operator. The semi-cover operator returns all the tuples of a relation covering at least one tuple of the other relation:

Definition 3 (Semi-cover operation). *The semi-cover of a relation $R[U]$ for a relation $S[V]$ w.r.t. a cover relation $\triangleleft \subseteq \mathbf{dom}(\widetilde{U}) \times \mathbf{dom}(\widetilde{V})$ is $R \triangleleft_{\ltimes} S = \pi_U(R \triangleleft S)$.*

Definition 3 implicitly means that $R \triangleright_{\ltimes} S$ returns all the tuples of R covered by at least one tuple of S. Indeed, $R \triangleright_{\ltimes} S$ has a sense because if the binary relation $\triangleleft$ is a cover relation on $\mathbf{dom}(\widetilde{U}) \times \mathbf{dom}(\widetilde{V})$ w.r.t. $\preceq_{\widetilde{U}}$ and $\preceq_{\widetilde{V}}$, then $\triangleright$ is also a cover relation on $\mathbf{dom}(\widetilde{U}) \times \mathbf{dom}(\widetilde{V})$ w.r.t. $\succeq_{\widetilde{U}}$ and $\succeq_{\widetilde{V}}$. Table 3 illustrates Definition 3 by showing semi-cover operation of L for D which is the whole set of patterns occurring at least once in D: $L \triangleleft_{\ltimes} D$. Then, $\sigma_{patt \preceq \varphi}(L \triangleleft_{\ltimes} D)$ returns the patterns being more general than φ and present in D.

Let us come back to the data and pattern covering operators [32] presented in Section 2.2. The operation $\theta_p(P, D)$ which gives the tuples of P covering at least one tuple of D, is equivalent to $P \triangleleft_{\ltimes} D$. Dually, $\theta_d(P, D) = D \triangleright_{\ltimes} P$ returns the tuples of D covered by at least one tuple of P.

Anti-cover operator. The anti-cover operator returns all the tuples of a relation not covering any tuple of the other relation:

Table 3. The semi-cover and anti-cover of L for D

$L \lhd_{\ltimes} D$
patt
$\emptyset$
A
B
C
D
E
AB
AC
AD

AE
BC
BD
BE
CD
ABC
ABD
ABE
ACD
BCD
ABCD

$L \lhd_{\neg} D$
patt
CE
DE
ACE
ADE
BCE
BDE
CDE
ABCE
ABDE

ACDE
BCDE
ABCDE

Definition 4 (Anti-cover operation). *The anti-cover of a relation $R[U]$ for a relation $S[V]$ w.r.t. a cover relation $\lhd \subseteq \mathbf{dom}(\widetilde{U}) \times \mathbf{dom}(\widetilde{V})$ is $R \lhd_{\neg} S = R - R \lhd_{\ltimes} S$.*

As for the semi-cover relation, $R \rhd_{\neg} S$ has a sense and returns all the tuples of R not covered by any tuple of S. Table 3 gives the patterns of L that do not occur in D by means of the anti-cover of L for D: $L \lhd_{\neg} D$. The anti-cover operator enables us to easily express the minimal and maximal pattern operators [25] (see Section 2.2): $\mathcal{M}in(R) = R \succ_{\neg} R$ and $\mathcal{M}ax(R) = R \prec_{\neg} R$. For instance, the frequent maximal itemsets are the frequent itemsets having no more specific frequent itemset: $M = F \prec_{\neg} F$ (see Table 2). A pattern of L is either present in D (i.e., in $L \lhd_{\ltimes} D$) or absent from D (i.e., in $L \lhd_{\neg} D$). Then, we obtain that $L = L \lhd_{\ltimes} D \cup L \lhd_{\neg} D$ (see Table 3). More generally, the semi-cover and anti-cover operator are complementary by definition (see Definitions 3 and 4): $R = R \lhd_{\ltimes} S \cup R \lhd_{\neg} S$ for any relations R and S.

3.3 Domain Operator

Let us come back to the query $\sigma_{freq \geq f}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(L \lhd D))$ that can be applied to any instance of relation L. However, in a practical pattern discovery task, the instance of L has to gather all the existing patterns of $\mathbf{dom}(patt)$ (as given by Table 1). To cope with this problem, we introduce a new operator that outputs the domain of the schema for a given relation.

Definition 5 (Domain operation). *The domain of a relation $R[U]$ is $\Delta(R)$ where for any instance I of R, $\Delta(I) = \mathbf{dom}(U)$.*

As the domain of each attribute is finite, the instance $\Delta(I)$ is finite. Assume that $I = \emptyset$ is an instance of $L[patt]$, $\Delta(I)$ returns the instance depicted by Table 1. The domain operator enables us to complete the frequent pattern mining query: $\sigma_{freq \geq f}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(\Delta(L) \lhd D))$. Other practical queries require the use of a language of patterns. For instance, negative border of R [25] can now be formulated: $\mathcal{B}d^{-}(R) = (\Delta(R) - R) \succ_{\neg} (\Delta(R) - R)$. Similarly, the downward and upward closure operators of R are respectively expressed by $\Delta(R) \preceq_{\ltimes} R$ and $\Delta(R) \succeq_{\ltimes} R$.

3.4 Scope of the Pattern-Oriented Relational Algebra

The *pattern-oriented relational algebra* which refers to the relational algebra plus the cover-like operators plus the domain operator, is strictly more expressive than the relational algebra. As aforementioned, the cover-like operators do not increase the expressive power of the relational algebra. In contrast, the domain operator cannot be expressed with relational operators because it induces domain dependent queries [1]. Let us note that [10] has already demonstrated that the frequent pattern mining query cannot be formulated in terms of the relational algebra.

From a practical point of view, the large number of query examples illustrating the previous sections (partially reported in Table 4 with q_1-q_5) highlights the generality of the pattern-oriented relational algebra. The other queries of Table 4 complete this overview by giving examples about the top-k frequent pattern mining with q_6 [15], the syntactic pattern mining q_7 [9], the utility-based pattern mining q_8 or the association rule mining q_9 [3]. Note that $\in$ is a cover relation on $\mathbf{dom}(item) \times \mathbf{dom}(patt)$ that relates one item with an itemset or a sequence. The query q_7 returns the patterns of L occurring in D and not containing a product of type 'snack'. q_8 returns the patterns of L occurring in D such that the sum of product prices is less than a threshold t.

Table 4. Examples of pattern-oriented queries and their properties

	Pattern-oriented query	DC	Dependence: Local	Dependence: Global
q_1	$\sigma_{freq \geq f}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(L \triangleleft D))$	L	L	D
q_2	$\pi_{patt,freq}(\sigma_{freq > max}(\gamma_{patt,freq,\texttt{MAX}(freq') \rightarrow max}(F \prec F')))$			F
q_3	$\sigma_{patt \prec \varphi}(L)$	L	L	
q_4	$\sigma_{patt \prec \varphi}(L \triangleleft_{\ltimes} D)$	L	L, D	
q_5	$F \prec_{\neg} F$			F
q_6	$\sigma_{rank \leq k}(\gamma_{patt,freq,\texttt{COUNT}(patt') \rightarrow rank}(\sigma_{supp \leq supp'}(F \times F')))$	F		F
q_7	$(L \triangleleft_{\ltimes} D) \ni_{\neg} \sigma_{type=snack}(P)$	L	L, D, P	
q_8	$\sigma_{total \leq t}(\gamma_{patt,\texttt{SUM}(price) \rightarrow total}(P \in (L \triangleleft_{\ltimes} D)))$	L	L, D, P	
q_9	$\pi_{patt' \rightarrow head,patt \setminus patt' \rightarrow body,freq,freq/freq' \rightarrow conf}(F' \prec F)$			F

Most of these typical queries are difficult to evaluate because the handled instances may be very large especially when the domain operator is used for generating the language. The following sections explain how to rewrite queries for optimization purpose.

4 Characterizing Pattern-Oriented Queries

In the field of pattern mining, it is well known that some properties are useful to reduce the computation time (e.g., anti-monotone constraint or pre/post-processing ability). This section aims at characterizing such properties in the pattern-oriented relational algebra. More precisely, we first study the structuration of the instance resulting from a query w.r.t. the initial instance. Then, we analyze three levels of dependency between a query and a relation.

Thereafter we assume that q is a query formulated with the pattern-oriented relational algebra and the database schema $\{R_1[U_1], \ldots, R_{n-1}[U_{n-1}], R[U]\}$. Then, this query q is often applied to the database instance $\mathbf{I} = \{I_1, \ldots, I_{n-1}, I\}$.

4.1 Downward Closed Query

Intuitively, the notion of downward closed query expresses that of anti-monotone constraints [25] in the pattern-oriented relational algebra. A query q is downward closed in R if for any instance I of $R[U]$, any tuple of I more general than at least one tuple of $\pi_U(q(\mathbf{I}))$ also belongs to $\pi_U(q(\mathbf{I}))$.

Definition 6 (Downward closed queries). *A query q is downward closed in $R[U]$ w.r.t. $\preceq$ iff $U \subseteq \mathbf{sch}(q)$ and $(R \preceq_{\ltimes} q) \equiv \pi_U(q)$.*

Definition 6 means that if a tuple t of R is more general than at least one tuple of the answer of q, t is also present in this answer. The downward closed property is very interesting for pruning an instance (more details are given in Section 5.2). The query $\sigma_{freq \geq f}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(L \triangleleft D))$ is downward closed in L w.r.t. $\preceq$. Indeed, all the generalizations of a frequent pattern are frequent (e.g., ABC is frequent and then, A, B, C, AB and so on are also frequent, see Table 1). Similarly, the top-k frequent pattern query q_6 is also downward closed in F w.r.t. $\preceq$. The column 'DC' of Table 4 indicates the relations in which the query is downward closed w.r.t. $\preceq$.

4.2 Local and Global Dependent Queries

A query is dependent on the relation R whenever its result varies with the instance of R. Whereas the query $\sigma_{patt \preceq \varphi}(L)$ is independent of D, $\sigma_{patt \preceq \varphi}(L \triangleleft_{\ltimes} D)$ depends on D because it only returns the tuples of $\sigma_{patt \preceq \varphi}(L)$ that cover at least one tuple of the instance of D. Definition 7 formalizes the notion of total independence (or independence in brief):

Definition 7 (Total independence). *A query q is totally independent of R iff for any instances I, J of R, one has $q(\{I_1, \ldots, I_{n-1}, I\}) = q(\{I_1, \ldots, I_{n-1}, J\})$.*

In other words, a query which is independent of R is equivalent to another query not involving R. Note that the queries which are totally independent of D correspond to syntactical constraints [9].

We now refine this notion of dependence by introducing the *global independence*. Both queries $\sigma_{patt \preceq \varphi}(L \triangleleft_{\ltimes} D)$ and $\sigma_{freq \geq f}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(L \triangleleft D))$ are dependent on D. But, the dependency of the second query on D is stronger than that of the first query. Indeed, the computation of the frequency for a tuple of L requires to simultaneously take into account several tuples of D.

Definition 8 (Local/global dependence). *A query q is globally independent of R iff for any instances I, J of R, one has $q(\{I_1, \ldots, I_{n-1}, I \cup J\}) = q(\{I_1, \ldots, I_{n-1}, I\}) \cup q(\{I_1, \ldots, I_{n-1}, J\})$. A query being globally independent of R but dependent on R is said to be locally dependent on R.*

Definition 8 formalizes the notion of global constraints [12] which compare several patterns together to check whether the constraint is satisfied or not. The queries (like q_2, q_5, q_6 or q_9) which are globally dependent on L or F correspond to such global constraints. Besides, the query q_1 globally depends on D and locally depends on L. It means that q_1 can be evaluated by considering separately each tuple of the instance of L. Conversely, it is impossible to consider individually each tuple of the instance of D. Thus, the higher the overall number of global dependencies, the harder the evaluation of the query. The columns 'Local' and 'Global' of Table 4 indicates the local/global dependent relations for each query. As expected, the queries q_1, q_4, q_7 and q_8 depend on D because they benefit from the dataset to select the right patterns. We also observe that the queries q_2, q_5, q_6 and q_9 globally depend on F as they postprocess the frequent patterns by comparing them.

5 Rewriting Pattern-Oriented Queries

This section examines algebraic equivalences to rewrite queries into forms that may be implemented more efficiently.

5.1 Algebraic Laws Involving Cover-Like Operators

Let us consider the query q_4: $\sigma_{patt \preceq \varphi}(L \triangleleft_{\ltimes} D)$. As the predicate $patt \preceq \varphi$ is highly selective, it is preferable to first apply it for reducing the language. Thereby, the equivalent query $\sigma_{patt \preceq \varphi}(L) \triangleleft_{\ltimes} D$ may be more efficient than $\sigma_{patt \preceq \varphi}(L \triangleleft_{\ltimes} D)$. The property below enumerates equivalences:

Property 1 (Laws involving cover-like operators). *Let $R[U]$ and $S[V]$ be two relation schemas. Let f and g be two predicates respectively on R and S. Let A and B be two sets of attributes such that $\widetilde{U} \subseteq A \subseteq U$ and $\widetilde{V} \subseteq B \subseteq V$. One has the following equivalences:*

1. $\sigma_{f \wedge g}(R \triangleleft S) \equiv \sigma_f(R) \triangleleft \sigma_g(S)$	$\pi_{A \cup B}(R \triangleleft S) \equiv \pi_A(R) \triangleleft \pi_B(S)$
2. $\sigma_f(R \triangleleft_{\ltimes} S) \equiv \sigma_f(R) \triangleleft_{\ltimes} S$	$\pi_A(R \triangleleft_{\ltimes} S) \equiv \pi_A(R) \triangleleft_{\ltimes} S$
3. $\sigma_f(R \triangleleft_{\neg} S) \equiv \sigma_f(R) \triangleleft_{\neg} S$	$\pi_A(R \triangleleft_{\neg} S) \equiv \pi_A(R) \triangleleft_{\neg} S$
4. $R \triangleleft_{\ltimes} S \equiv R \triangleleft_{\ltimes} (S \prec_{\neg} S)$	$R \triangleleft_{\neg} S \equiv R \triangleleft_{\neg} (S \prec_{\neg} S)$

Intuitively, the right hand side of each equivalence listed in Property 1 (proofs are omitted due to lack of space) may lead to optimize the query. Indeed, Lines 1 to 3 "pushes down" the selection and projection operators to reduce the size of the operands before applying a cover-like operator. This technique is successfully exploited in database with Cartesian product or join operator [1]. Besides, Line 4 benefits from the maximal tuples of S (i.e., $S \prec_{\neg} S$) as done in pattern mining [25]. If a tuple t of the instance of R covers a tuple of the instance J of

S, then t also covers a tuple of $J \prec_\neg J$. As $|J \prec_\neg J| \leq |J|$, the rewritten query $R \triangleleft_\ltimes (S \prec_\neg S)$ may be less costly than $R \triangleleft_\ltimes S$ provided $J \prec_\neg J$ is not too costly.

5.2 Algebraic Reformulation of the Levelwise Algorithm

We now take into account the downward closed and the global independence properties for reformulating queries. For instance, assume that the instance of L is now equal to $\pi_{patt}(F)$. A new computation of q_1 again returns F: $F = \sigma_{freq \geq 2}(\gamma_{patt,\texttt{COUNT}(trans) \rightarrow freq}(\pi_{patt}(F) \triangleleft D))$. Of course, this query is faster to compute than the original fpm query because the instance of F is very small compared to $\Delta(L)$. We generalize this observation:

Property 2. *Let q be a downward closed query in $R[U]$ w.r.t. $\preceq$ and globally independent of R such that $U \subseteq \mathbf{sch}(q)$, one has $q(\mathbf{I}) = q(\mathbf{J})$ for any instances $\mathbf{I} = \{I_1, \ldots, I_{n-1}, I\}$ and $\mathbf{J} = \{I_1, \ldots, I_{n-1}, J\}$ such that $\pi_U(q(\mathbf{J})) \subseteq I \subseteq J$.*

Given a downward closed and independent query q, Property 2 demonstrates that $q(\mathbf{I}) = q(\mathbf{J})$ when I is an instance of R such that $\pi_U(q(\mathbf{J})) \subseteq I \subseteq J$. As $I \subseteq J$ and then $|I| \leq |J|$, we suppose that evaluating $q(\mathbf{I})$ is less costly than evaluating $q(\mathbf{J})$ because the cost generally decreases with the cardinality of the instance. Thus, in order to reduce the cost of the evaluation of $q(\mathbf{I})$, we aim at turning I into the smallest instance of R including $q(\mathbf{J})$. Such an approach can be seen as a *pruning* of the instance of R.

Table 5. Levelwise computation of the fpm query (level 2)

L

patt
AB
AC
AD
BC
BD
CD

ABC
ABD
ACD
BCD
ABCD

$\mathcal{C} = L \succ_\neg L$

patt
AB
AC
AD
BC
BD
CD

$\mathcal{S}$

patt	supp
AB	3
AC	2
AD	2
BC	2

$L \succ_\ltimes \mathcal{S}$

patt
ABC
ABD
ACD
BCD
ABCD

$(L \succ_\ltimes \mathcal{S}) \succeq_\neg (\mathcal{C} \succeq_\neg \mathcal{S})$

patt
ABC

Table 5 illustrates how to prune the instance L for evaluating the fpm query q_1. As q_1 is globally independent of L, we first divide L into two parts: the most general tuples of L denoted by $\mathcal{C} = L \succ_\neg L$ (i.e., the candidates of the level 2 of APRIORI [3]) and others, i.e. $L \succ_\ltimes L$. We then apply the fpm query to $\mathcal{C}$ for computing $\mathcal{S}$: the frequent patterns of $\mathcal{C}$ and their frequency. Finally, we benefit from $\mathcal{S}$ for pruning $L \succ_\ltimes L$ using the downward closed property of q_1 in L w.r.t. $\preceq$ (see Definition 6). We only preserve the tuples which are more specific than at least one frequent tuple of $\mathcal{S}$: $L \succ_\ltimes \mathcal{S}$. Finally, we filter out the tuples having a non-frequent generalization: $(L \succ_\ltimes \mathcal{S}) \succeq_\neg (\mathcal{C} \succeq_\neg \mathcal{S})$. As the cardinality of this instance is smaller than $|L \succ_\ltimes L|$, we have achieved our goal.

This principle is generalized with this theorem:

Theorem 1 (Levelwise equivalence). *Let q be a downward closed query w.r.t. $\preceq$ and globally independent of R, one has the below equality for any database instance $\mathbf{I} = \{I_1, \ldots, I_{n-1}, I\}$:*

$$q(\mathbf{I}) = \underbrace{q(\{I_1, \ldots, I_{n-1}, \underbrace{I \succ_\neg I}_{\mathcal{C}=}\}) \cup q(\{I_1, \ldots, I_{n-1}, (I \succ_\ltimes \mathcal{S}) \succeq_\neg (\mathcal{C} \succeq_\neg \mathcal{S})\})}_{\mathcal{S}=}$$

Proof. Let q be a downward closed query w.r.t. $\preceq$ and globally independent of R. To alleviate the notations, $q(I)$ refers to $q(\{I_1, \ldots, I_{n-1}, I\})$ where I is any instance of R. Besides, we fix that $\mathcal{C} = I \succ_\neg I$ and $\mathcal{S} = q(I \succ_\neg I) = q(\mathcal{C})$:

$$\begin{aligned}
q(I) &= q(I \succ_\neg I \cup I \succ_\ltimes I) = q(\mathcal{C} \cup I \succ_\ltimes I) && (1)\\
&= q(\mathcal{C}) \cup q(I \succ_\ltimes I) && (2)\\
&= q(\mathcal{C}) \cup q(I \succ_\ltimes q(\mathcal{C})) = q(\mathcal{C}) \cup q(I \succ_\ltimes \mathcal{S}) && (3)\\
&= q(\mathcal{C}) \cup q((I \succ_\ltimes \mathcal{S}) \succeq_\neg (\mathcal{C} \succeq_\neg \mathcal{S})) && (4)
\end{aligned}$$

Line 1 stems from the complementary property: $R = R \triangleleft_\ltimes S \cup R \triangleleft_\neg S$. Line 2 is allowed because q is globally independent of R. Line 3-4 are due to the downward closed property in R (see Definition 6). □

Theorem 1 can be used for rewriting queries by considering two important points. Firstly, the redundant subqueries as candidate tuples $\mathcal{C} = I \succ_\neg I$ and satisfied tuples $\mathcal{S} = q(\{I_1, \ldots, I_{n-1}, I \succ_\neg I\})$ have to be evaluated only once. Secondly, the practical evaluation of q requires to recursively apply the equality proposed in Theorem 1. Indeed, the subquery $q(\{I_1, \ldots, I_{n-1}, (I \succ_\ltimes \mathcal{S}) \succeq_\neg (\mathcal{C} \succeq_\neg \mathcal{S})\})$ can also be rewritten by a query plan optimizer using the same identity. Therefore, Theorem 1 leads to algebraically reformulate the levelwise algorithm [3, 4, 25]. This algorithm repeats this equality for computing which candidate patterns satisfy the predicate and then, generating those of the next level. Other efficient pruning strategies like depth-first search techniques [5] could also be expressed in pattern-oriented relational algebra. Finally, as observed in [12, 15], we cannot apply Theorem 1 to q_6 because it globally depends on F.

6 Related Work

Inductive databases [18, 24] aims at tightly integrating databases with data mining. Our approach is less ambitious because it is "only" restricted to the pattern mining. Obviously, many proposals provide an environment merging a RDBMS with pattern mining tools: Quest [2], ConQueSt [7], DBminer [16], Sindbad [34] and many other prototypes [6]. In such a context, there are many extensions of the SQL language [31] like DMX or MINERULE [26]. There are also extended relational model [13] like 3W model [20]. However, such methods don't fuse the theoretical concepts stemming from both the relational model and the pattern

discovery. For instance, the query optimizer of DBMS is isolated from pattern mining algorithms. Indeed, most of the approaches consider a pattern mining query as the result of a "black box". Only few works [10, 23, 33] express pattern mining operators by benefiting from the relational algebra. Such approaches add a loop statement for implementing the levelwise algorithm. On the contrary, our proposal extends the relational algebra by still using a declarative approach.

Many frameworks inspired from relational and logical databases, but created from scratch, are proposed during the last decade: constraint-based pattern mining [9, 25], distance-based framework [14], rule-base [19], tuple relational calculus [28], logical database [29], pattern-base [32] and so on. Other directions are suggested in [24] like probabilistic approach or data compression. Besides, constraint programming is another promising way for expressing and mining patterns [21, 30]. Such frameworks are less convenient for handling data (which are often initially stored in relational databases). Besides, they suffer from a lack of simple and powerful languages like the relational algebra (in particular, the manipulation of patterns is frequently separated from that of data).

From a more general point of view, many works add new operators to the relational algebra in order to express more sophisticated queries. Even if such new operators don't necessary increase the expressive power of the relational algebra, most of the time they facilitate the formulation of user queries and provide specific optimizations. Typically, several operators are introduced for comparing tuples with each other, as does a specialization relation with patterns. For instance, the winnow operator is specifically dedicated to handle preferences [11]. Several operators are dedicated for selecting the best tuples by means of relational dominant queries [8] or relational top-k queries [22]. The cover-like operators are very closed to such operators. But, they enable to compare tuples based on different languages, as does a cover relation with patterns. Finally, the domain operator enables us to manipulate values not initially present in the relations. The same concept is used in [13] for generating tables containing patterns.

7 Conclusion

In this paper, we have proposed a new and general framework for pattern discovery by only adding cover-like and domain operators to the relational algebra. The pattern-oriented relational algebra interestingly inherits good properties from the relational algebra as closure or declarativity. This framework deals with any language of patterns for expressing a wide spectrum of queries including constraint-based pattern mining, condensed representations and so on. We identify crucial aspects of queries as the downward closed and independence properties. We then benefit from such properties to algebraically reformulate the levelwise algorithm. We think that our algebraisation is an important step towards the elegant integration of pattern discovery in database systems.

Further work addresses the implementation of a complete system based on the pattern-oriented relational algebra. As done in the database field, we project to implement the physical cover operators and to design a query plan optimizer

taking advantage of our proposed algebraic laws. We also study the test of local and global dependence between a query and a relation.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Agrawal, R., Mehta, M., Shafer, J.C., Srikant, R., Arning, A., Bollinger, T.: The quest data mining system. In: KDD, pp. 244–249 (1996)
3. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) VLDB, pp. 487–499. Morgan Kaufmann, San Francisco (1994)
4. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Yu, P.S., Chen, A.L.P. (eds.) ICDE, pp. 3–14. IEEE Computer Society, Los Alamitos (1995)
5. Arimura, H., Uno, T.: Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In: SDM, pp. 1087–1098. SIAM, Philadelphia (2009)
6. Blockeel, H., Calders, T., Fromont, É., Goethals, B., Prado, A., Robardet, C.: An inductive database prototype based on virtual mining views. In: KDD, pp. 1061–1064. ACM, New York (2008)
7. Bonchi, F., Giannotti, F., Lucchese, C., Orlando, S., Perego, R., Trasarti, R.: ConQueSt: a constraint-based querying system for exploratory pattern discovery. In: ICDE, p. 159. IEEE Computer Society, Los Alamitos (2006)
8. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430. IEEE Computer Society, Los Alamitos (2001)
9. Boulicaut, J.F., Jeudy, B.: Constraint-based data mining. In: Maimon, O., Rokach, L. (eds.) The Data Mining and Knowledge Discovery Handbook, pp. 399–416. Springer, Heidelberg (2005)
10. Calders, T., Lakshmanan, L.V.S., Ng, R.T., Paredaens, J.: Expressive power of an algebra for data mining. ACM Trans. Database Syst. 31(4), 1169–1214 (2006)
11. Chomicki, J.: Querying with intrinsic preferences. In: Jensen, C.S., Jeffery, K., Pokorný, J., Šaltenis, S., Hwang, J., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 34–51. Springer, Heidelberg (2002)
12. Crémilleux, B., Soulet, A.: Discovering knowledge from local patterns with global constraints. In: Gervasi, O., Murgante, B., Laganà, A., Taniar, D., Mun, Y., Gavrilova, M.L. (eds.) ICCSA 2008, Part II. LNCS, vol. 5073, pp. 1242–1257. Springer, Heidelberg (2008)
13. Diop, C.T., Giacometti, A., Laurent, D., Spyratos, N.: Composition of mining contexts for efficient extraction of association rules. In: Jensen, C.S., Jeffery, K., Pokorný, J., Šaltenis, S., Hwang, J., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 106–123. Springer, Heidelberg (2002)
14. Dzeroski, S.: Towards a general framework for data mining. In: Džeroski, S., Struyf, J. (eds.) KDID 2006. LNCS, vol. 4747, pp. 259–300. Springer, Heidelberg (2007)
15. Fu, A.W.C., Kwong, R.W., Tang, J.: Mining n-most interesting itemsets. In: Ohsuga, S., Raś, Z.W. (eds.) ISMIS 2000. LNCS (LNAI), vol. 1932, pp. 59–67. Springer, Heidelberg (2000)
16. Han, J., Fu, Y., Wang, W., Chiang, J., Gong, W., Koperski, K., Li, D., Lu, Y., Rajan, A., Stefanovic, N., Xia, B., Zaïane, O.R.: DBMiner: a system for mining knowledge in large relational databases. In: KDD, pp. 250–255 (1996)

17. Hand, D.J.: Pattern detection and discovery. In: Hand, D.J., Adams, N.M., Bolton, R.J. (eds.) Pattern Detection and Discovery. LNCS (LNAI), vol. 2447, pp. 1–12. Springer, Heidelberg (2002)
18. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Commun. ACM 39(11), 58–64 (1996)
19. Imielinski, T., Virmani, A.: MSQL: a query language for database mining. Data Min. Knowl. Discov. 3(4), 373–408 (1999)
20. Johnson, T., Lakshmanan, L.V.S., Ng, R.T.: The 3W model and algebra for unified data mining. In: Abbadi, A.E., Brodie, M.L., Chakravarthy, S., Dayal, U., Kamel, N., Schlageter, G., Whang, K.Y. (eds.) VLDB, pp. 21–32. Morgan Kaufmann, San Francisco (2000)
21. Khiari, M., Boizumault, P., Crémilleux, B.: Combining CSP and constraint-based mining for pattern discovery. In: Taniar, D., Gervasi, O., Murgante, B., Pardede, E., Apduhan, B.O. (eds.) ICCSA 2010. LNCS, vol. 6017, pp. 432–447. Springer, Heidelberg (2010)
22. Li, C., Chang, K.C.C., Ilyas, I.F., Song, S.: RankSQL: query algebra and optimization for relational top-k queries. In: Özcan, F. (ed.) SIGMOD Conference, pp. 131–142. ACM Press, New York (2005)
23. Liu, H.C., Ghose, A., Zeleznikow, J.: Towards an algebraic framework for querying inductive databases. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 306–312. Springer, Heidelberg (2010)
24. Mannila, H.: Theoretical frameworks for data mining. SIGKDD Explorations 1(2), 30–32 (2000)
25. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Min. Knowl. Discov. 1(3), 241–258 (1997)
26. Meo, R., Psaila, G., Ceri, S.: A new SQL-like operator for mining association rules. In: Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L. (eds.) VLDB, pp. 122–133. Morgan Kaufmann, San Francisco (1996)
27. Mitchell, T.M.: Generalization as search. Artif. Intell. 18(2), 203–226 (1982)
28. Nijssen, S., Raedt, L.D.: IQL: a proposal for an inductive query language. In: Džeroski, S., Struyf, J. (eds.) KDID 2006. LNCS, vol. 4747, pp. 189–207. Springer, Heidelberg (2007)
29. Raedt, L.D.: A logical database mining query language. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 78–92. Springer, Heidelberg (2000)
30. Raedt, L.D., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: KDD, pp. 204–212. ACM, New York (2008)
31. Romei, A., Turini, F.: Inductive database languages: requirements and examples. Knowledge and Information Systems 1–34 (2010), http://dx.doi.org/10.1007/s10115-009-0281-4
32. Terrovitis, M., Vassiliadis, P., Skiadopoulos, S., Bertino, E., Catania, B., Maddalena, A., Rizzi, S.: Modeling and language support for the management of pattern-bases. Data Knowl. Eng. 62(2), 368–397 (2007)
33. Wang, H., Zaniolo, C.: ATLaS: a native extension of SQL for data mining. In: Barbará, D., Kamath, C. (eds.) SDM. SIAM, Philadelphia (2003)
34. Wicker, J., Richter, L., Kessler, K., Kramer, S.: SINDBAD and SiQL: an inductive database and query language in the relational model. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part II. LNCS (LNAI), vol. 5212, pp. 690–694. Springer, Heidelberg (2008)