Mining Databases with Multiple Tables: Problems and Perspectives

A. Faye†, A. Giacometti‡, D. Laurent†‡, N. Spyratos†

† LRI - Université Paris 11 - 91405 Orsay Cedex France
‡ LI/E3I - Université de Tours - 3, place J. Jaures - 41000 Blois France

E-mail: {faye, spyratos}@lri.fr, {giaco, laurent}@univ-tours.fr

Abstract: We consider the problem of discovering patterns from a given logic that are significant (i.e. interesting and sufficiently valid) with respect to a given data set. We first define two types of patterns that extend the notions of query and clause, and we propose new measures of interest and confidence. In this framework, we establish connections between our measures, first-order logic and logics of probability of Halpern. Then, we present mining algorithms based on the generic levelwise search method proposed by Mannila, and discuss implementation issues in a relational database environment. Finally, we offer concluding remarks and suggestions for future research.

Keywords: Databases, Data Mining, Rule Discovery, Inductive Logic Programming

1 Introduction

In the last decade, the usage and size of databases and datawarehouses have grown dramatically, due to a constant decrease in the cost of both the collection and the storage of huge amounts of data. The need to develop techniques and tools with the ability to exploit these amounts of data has grown accordingly and has given rise to an exciting and rapidly evolving research field known as Data Mining and Knowledge Discovery in Databases (KDD) (Fayad et al, 1996). This new field, at the intersection of Machine Learning, Statistics and Databases, aims at discovering relevant knowledge in very large databases.

One problem that has received great attention from the KDD community is the mining of significant patterns (Faye et al, 1998; Faye et al, 1999; Dehaspe 1998). Given a data set S, a logic $\mathcal{L}$, and two measures of interest called support and confidence, mining significant patterns aims at discovering patterns of $\mathcal{L}$ that are interesting (meaning that their support is over a user given threshold called the minimum support threshold) and sufficiently valid (meaning that their confidence is over a user given threshold called the minimum confidence threshold) (Agrawal et al, 1996; Faye et al, 1998; Dehaspe, 1998).

In this paper, we propose an extension of these approaches, in the sense that the patterns considered in (Dehaspe, 1998) are particular cases of those considered here. In this framework, we define new measures of interest and confidence, and address the following issues:

- We establish important connexions between the measure of interest (which we call support), and first-order logic on the one hand, and logics of probability (Halpern, 1990) on the other hand. In our opinion, this point is very important because it allows to appreciate the reliability of the chosen measures.
- We show that, although the rules that are learned in our approach are not clauses, it is always possible to transform them into a Datalog program (through the introduction of new predicates). The importance of this point lies in the fact that, in a database environment, only Datalog rules can be managed.
- We discuss implementation issues of our approach, based on the general framework proposed in (Mannila and Toivonen, 1997). We note that the reduction of the search space in our algorithms heavily relies on the properties mentioned in the first item above. Moreover, our algorithms combine techniques from algorithms Apriori (Agrawal et al, 1996) and Warmr (Dehaspe, 1998). We also define a language bias from which possible performance improvements are designed, using by example join indices techniques (Li and Ross, 1998).

The paper is organized as follows: in Section 2, we define two types of patterns, called K-query and K-rule, that extend the notions of query and clause. Then, we present new measures of support and confidence, and study their properties on a first-order logic basis. In Section 3, we introduce the declarative language bias that is used to reduce the size of the search space, and we propose algorithms for discovering interesting K-queries, as well as significant K-rules. In this section, we also discuss implementation issues. In Section 4, we offer concluding remarks and suggestions for future research. Due to lack of space, the proofs of propositions are omitted.

2 Patterns

2.1 Basic Definitions and Notation

We assume a fixed set of predicates with given arities. If p is a predicate with arity m, then $p(t_1, ..., t_m)$ is an *atom*, where each t_i is a term, i.e. either a constant or a variable. Atoms are the basic ingredients of patterns. We also assume a fixed set of *facts,* i.e. atoms of the form $p(a_1, ..., a_m)$, where each a_i is a constant. This set, called the *data set*, is the set from which interesting patterns will be mined.

Moreover, we consider that the constants that appear in the data set come from pre-defined sets of values that we call *domains.* More precisely, we assume that, for every predicate p with arity m, each entry k of p is associated with a pre-defined set of values, denoted by $dom(k, p)$, where $k = 1 \ldots m$. The domain $dom(k, p)$ is the set of values [1] that a term appearing at the k-th position of an atom over p can have. In the remaining of the paper, we assume the data set to be *domain-relevant*, i.e. all constants appearing in the data set to be in the corresponding domains.

Let us illustrate the concepts introduced so far through an example that we shall use throughout the paper as our running example.

Running Example 1 Suppose that we look for relationships between the characteristics of customers and the categories of products that they buy. Our fixed set of predicates consists of *Cust*, *Sale* and *Prod*, with arities 3,2 and 2 respectively. Additionally, assume the following domains:

- $dom(1, Cust) = dom(1, Sale) = \{C1, C2, C3\}$; these constants are identifiers of customers.
- $dom(2, Cust) = \{Manager, Teacher, Lawyer\}$; these constants are the possible professions of customers.
- $dom(3, Cust) = [0, 100]$; this interval defines the possible values for the age of customers.
- $dom(2, Sale) = dom(1, Product) = \{P1, P2, P3, P4\}$; these constants are identifiers of products.
- $dom(2, Prod) = \{Beer, Tea, Milk\}$; these constants are the possible categories of products.

Let the data set S be the following set of facts:

$$\begin{aligned} S = \{ \quad & Cust(C1, Teacher, 30), Cust(C2, Teacher, 60), Cust(C3, Lawyer, 40) \\ & Prod(P1, Beer), Prod(P2, Beer), Prod(P3, Tea), Prod(P4, Milk) \\ & Sale(C1, P1), Sale(C1, P2), Sale(C2, P1), \\ & Sale(C2, P3), Sale(C3, P3), Sale(C3, P4)\}. \end{aligned}$$

Here, a fact such as $Cust(C1, Teacher, 30)$ means that customer $C1$ is a 30 years old teacher, facts such as $Sale(C2, P1)$ and $Sale(C2, P3)$ mean customer $C2$ buys product $P1$ and $P3$. Additionally, facts such as $Prod(P1, Beer)$ and $Prod(P2, Beer)$ mean that both products $P1$ and $P2$ are beer. We note that S is domain-relevant as every constant appearing in S is in the corresponding domain.

[1]The notion of domain, as defined here, corresponds to that of "active domain" in relational databases (Maier, 1983).

In this paper, we also consider *constraints* that are conjunctions of *elementary* constraints of the form $x_i = a_i$, $x_i = y_i$, $x_i \leq a_i$, $x_i > b_i$ where a_i, b_i are constants, and x_i, y_i are variables. Let Γ be a conjunction of elementary constraints. We recall that the *solution* of Γ, denoted by $Sol(\Gamma)$ is the set of all substitutions θ such that $\Gamma\theta$ is true. By example, if Γ is the constraint $(x = Teacher \wedge y \leq 30)$, then:

$$Sol(\Gamma) = \{\theta \mid \theta(x) = Teacher \text{ and } \theta(y) \leq 30\}$$

Moreover, we say that a constraint Γ is more *restrictive* than a constraint Γ' iff $Sol(\Gamma) \subseteq Sol(\Gamma')$.

2.2 Different types of patterns

Different types of patterns have been considered in the literature so far. In this paper, we introduce two types of patterns, called *K-query* and *K-rule*, that extend the notions of *query*, *query-extension* and *clause* proposed in (Dehaspe, 1998).

Intuitively speaking, a K-query is a quantified conjunction of atoms and elementary constraints where some of the variables are universally quantified, and the others are existentially quantified. Likewise, a K-rule is a quantified implication where some of the variables are universally quantified, and the others are existentially quantified. In the following definitions, we denote by $Var(\mathcal{P})$ the set of all variables in a formula $\mathcal{P}$:

Definition 1 - K-query. *Let $\mathcal{L}$ be a conjunction of atoms and Γ be a constraint such that $Var(\Gamma) \subseteq Var(\mathcal{L})$. We call K-query, a quantified formula of the form $(\forall \vec{K})(\exists \vec{Y})(\mathcal{L} \wedge \Gamma)$, where K is a subset of $Var(\mathcal{L})$ and $Y = Var(\mathcal{L}) \setminus K$. We denote such a formula by $(\mathcal{L} \wedge \Gamma)_K$.*

Running Example 2 In the context of our Running Example 1, the quantified formula $(\forall y)(\exists x, z)(Cust(x, y, z) \wedge (z > 30))$ is a K-query that can be written as $(Cust(x, y, z) \wedge (z > 30))_y$. The meaning of this K-query is that for every profession y, there exists a customer x who is more than 30 years old.

It is now important to note that the notion of K-query extends the notion of *query* proposed in (Dehaspe, 1998). In our framework, a query is a K-query $\mathcal{Q}_K$ without constraints and where no variable is quantified universally, i.e. $K = \emptyset$. We can also note that different K-queries can be formed from the same conjunction of atoms and constraints. Let us consider the conjunction of atoms $\mathcal{L} = Cust(x, Teacher, z) \wedge Sale(x, y) \wedge Prod(y, Beer)$. The K-query $\mathcal{L}_{x,y}$ means that every teacher buys every type of beer, whereas the K-query $\mathcal{L}_x$ means that every teacher buys beer. Finally, the K-query $\mathcal{L}_y$ means that every type of beer is bought by at least one teacher.

We also note that in a previous paper (Faye et al, 1998), we considered only K-queries of the form $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ where:

- there is no occurrence of constants in the atoms of $\mathcal{L}$,
- the constraints in Γ are only of the form $x_i = a_i$, where x_i is a variable occurring in $\mathcal{L}$ and a_i is a constant,
- K is the set of all variables in $\mathcal{L}$ and Γ, i.e. $X = Var(\mathcal{L})$.

We now define the notion of K-rules.

Definition 2 - K-rule. *Let $\mathcal{B}$ and $\mathcal{H}$ be two conjunctions of atoms, and Γ_1 and Γ_2 be two constraints such that $Var(\Gamma_1) \subseteq Var(\mathcal{B})$ and $Var(\Gamma_2) \subseteq (Var(\mathcal{B}) \cup Var(\mathcal{H}))$. We call K-rule, a quantified formula of the form $(\forall \vec{K})((\exists \vec{Y})(\mathcal{B} \wedge \Gamma_1) \rightarrow (\exists \vec{Y'})(\mathcal{B} \wedge \mathcal{H} \wedge \Gamma_1 \wedge \Gamma_2))$, where K is a subset of $Var(\mathcal{B})$, $Y = Var(\mathcal{B}) \setminus K$ and $Y' = (Var(\mathcal{B}) \cup Var(\mathcal{H})) \setminus K$. We denote such a formula by $(\mathcal{B} \wedge \Gamma_1 \rightarrow \mathcal{H} \wedge \Gamma_2)_K$.*

Running Example 3 In the context of our Running Example 1, the formula:

$$\begin{aligned}(\forall x, z)(\quad & Cust(x, Teacher, z) \wedge (z > 30) \rightarrow \\ & (\exists y)(Cust(x, Teacher, z) \wedge (z > 30) \wedge Sale(x, y) \wedge Prod(y, Beer)))\end{aligned}$$

is a K-rule that can be written as $(Cust(x, Teacher, z) \wedge (z > 30) \rightarrow Sale(x, y) \wedge Prod(y, Beer))_{x,z}$. The meaning of this K-rule is that every teacher who is more than 30 years old buys beer.

We note that the notion of K-rule extends the notion of *clause*, as well as the notion of *query-extension* proposed in (Dehaspe, 1998). Indeed, a clause is a K-rule $(\mathcal{B} \rightarrow \mathcal{H})_K$ where every variable is quantified universally, i.e. $K = Var(\mathcal{B}) \cup Var(\mathcal{H})$. On the other hand, a query-extension is a K-query $(\mathcal{B} \rightarrow \mathcal{H})_K$ where no variable is quantified universally, i.e. $K = \emptyset$.

Finally, it must be emphasized that K-rules, as well as query-extensions, are not clauses, meaning that they cannot be managed in Deductive Database environments. However, any K-rule can be transformed into a clause by introduction of new predicates. For example, let us consider the K-rule:

$$\mathcal{R}_x : (Cust(x, Teacher, z) \wedge (z > 60) \rightarrow Sale(x, y) \wedge Prod(y, Beer))_x$$

Given the new predicates *OldTeacher* and *BeerCustomer* defined by:

$$\begin{aligned}&(\forall x, z)(Cust(x, Teacher, z) \wedge (z > 60) \rightarrow OldTeacher(x)) \\ &(\forall x, y)(Sale(x, y) \wedge Prod(y, Beer) \rightarrow BeerCustomer(x))\end{aligned}$$

$\mathcal{R}_x$ can be transformed into the clause $(\forall x)(OldTeacher(x) \rightarrow BeerCustomer(x))$. Consequently, in the light of the above example, it turns out that any set of K-rules generated by the algorithms given in this paper can be translated into a Datalog program.

2.3 Support of a K-query

In the remaining of the paper, we assume that if a variable x occurs at different positions in a conjunction of atoms and constraints $\mathcal{Q}$, then all these positions are associated with the same domain, denoted by $dom(x, \mathcal{Q})$. Moreover, we denote by $\Theta(\mathcal{Q})$ the set of ground substitutions θ such that $(\forall x \in Var(\mathcal{Q}))(\theta(x) \in dom(x, \mathcal{Q}))$ and $(\forall x \notin Var(\mathcal{Q}))(\theta(x) = x)$. Finally, we asume that if a variable x occurs in two different conjunctions $\mathcal{Q}$ and $\mathcal{P}$, then $dom(x, \mathcal{Q}) = dom(x, \mathcal{P})$.

The previous condition is not a restriction, but rather a condition according to which the use of variables is "coherent" with respect to their domains. Thus, in our Running Example 1, the K-query $\mathcal{Q}_{x,y} = (\forall x, y)(Cust(x, x, y))$ is not considered, as $dom(1, Cust) \neq dom(2, Cust)$. On the other hand, the K-query $\mathcal{Q}'_{x,y} = (\forall x, y)(Cust(x, Teacher, 10) \wedge Sale(x, y) \wedge Prod(y, Beer))$ is considered, since we assume $dom(x, \mathcal{Q}') = dom(1, Cust) = dom(1, Sale)$ and $dom(y, \mathcal{Q}') = dom(2, Sale) = dom(1, Prod)$.

We can now define the notion of example of a K-query.

Definition 3 - Example of a K-query. *Let $(\mathcal{L} \wedge \Gamma)_K$ be a K-query where $\mathcal{L}$ is a conjunction of atoms and Γ is a constraint. Given a data set S, an* example of $(\mathcal{L} \wedge \Gamma)_K$ in S *is a substitution θ in $\Theta(\mathcal{L})$ such that:*

- *$\Gamma\theta$ is true, and*
- *$L\theta$ is in S, for every atom L in $\mathcal{L}$.*

The fact that θ is an example in S of the K-query $\mathcal{Q}_K$ is denoted by $S \models \mathcal{Q}\theta$.

Denoting by $\theta_{|K}$ the restriction of θ over the set of variables K (i.e. if x is a variable in K, then $\theta_{|K}(x) = \theta(x)$, otherwise $\theta(x)_{|K}(x) = x$), the support of a K-query is defined as follows:

Definition 4 - Support of a K-query. *Let $\mathcal{Q}_K$ be a K-query such that every variable in K has a finite domain. Given a data set S, the support of $\mathcal{Q}_K$ in S, denoted by $Sup(\mathcal{Q}_K, S)$, is defined by:*

$$Sup(\mathcal{Q}_K, S) = \frac{\left|\{\theta_{|K} \mid \theta \in \Theta(\mathcal{Q}) \wedge S \models \mathcal{Q}\theta\}\right|}{\left|\{\theta_{|K} \mid \theta \in \Theta(\mathcal{Q})\}\right|}$$

where $|E|$ denotes the cardinality of a set E.
A K-query is called interesting *(or* frequent*) if its support in S is over a given threshold, called the* minimum support threshold *and denoted by* minsup.

It can be noticed that the denominator of the support is equal to $\prod_{k_i \in K} |dom(k_i, \mathcal{Q})|$, which explains why the support of a K-query is not defined if the set K contains a variable k_i whose domain is not finite.

Running Example 4 In the context of our Running Example 1, let us consider the K-query $\mathcal{Q}_K = (Cust(x, Teacher, z) \wedge (z > 40) \wedge Sale(x, y) \wedge Prod(y, Beer))_x$. Here:

- the substitution $\theta_1 = \{x \to C1, y \to P1, z \to 30\}$ is not an example of $\mathcal{Q}_K$ in S as $Cust(C1, Teacher, 30)$, $Sale(C1, P1)$, $Prod(P1, Beer)$ are facts in S, but the constraint $(z > 30)$ is not satisfied. On the other hand, the substitution $\theta_2 = \{x \to C2, y \to P1, z \to 60\}$ is an example of $\mathcal{Q}_K$ in S as $Cust(C2, Teacher, 60)$, $Sale(C2, P1)$, $Prod(P1, Beer)$ are facts in S and the constaint $(z > 30)$ is satisfied. It is the only example of $\mathcal{Q}_K$ in S.
- $\prod_{k_i \in K} |dom(k_i, \mathcal{Q})| = |dom(x, \mathcal{Q})| = |dom(1, Cust)| = 3$.

Thus we obtain $Sup(\mathcal{Q}_K, S) = 1/3$.

So far, different measures of support have been proposed in the literature. Comparing our definition of support to those of (Faye et al, 1998) and (Dehaspe, 1998), we have the following:

- The measure proposed in (Faye et al, 1998), is equivalent but less general than the measure given in Definition 4. Indeed, in (Faye et al, 1998), we consider only K-queries $\mathcal{Q}_K$ where K is the set of all variables in $\mathcal{Q}$.
- In (Dehaspe, 1998), the author defines the frequency $frq(\mathcal{Q}, key, S)$ of a query $\mathcal{Q}$ with respect to a data set S, provided that a particular atom key occurs in every query. In our framework, a query $\mathcal{Q}$ that contains this particular atom key is equivalent to a K-query $\mathcal{Q}_K$ where K is the set of variables in key, and we have $frq(\mathcal{Q}, key, S) = (1/\alpha) \times Sup(\mathcal{Q}_K, S)$ where $\alpha = Sup(\{key\}_K, S)$.

We now give important properties of the support.

Proposition 1 *For every K-query $\mathcal{Q}_K$ and every data set S:*

1. $0 \leq Sup(\mathcal{Q}_K, S) \leq 1$.
2. $Sup(\mathcal{Q}_K, S) = 1$ *iff S is a model of* $(\forall \vec{K})(\exists \vec{Y})(\mathcal{Q})$ *where* $Y = Var(\mathcal{Q}) \setminus K$.

The following proposition states that as the data set increases so does the support, provided that no new constants are introduced.

Proposition 2 *Let S and T be two data sets over the same set of predicates, and with the same domains. If $S \subseteq T$, then $Sup(\mathcal{Q}_K, S) \leq Sup(\mathcal{Q}_K, T)$ for every K-query $\mathcal{Q}_K$.*

The importance of this proposition lies in the fact that it relates changes in the data set to changes in the support, and thus to changes in the interestingness of a K-query. More precisely, it implies that the insertion of new facts in the data set may generate new interesting K-queries, whereas the deletion of facts will never generate new interesting K-queries.

The following proposition will be used to reduce the size of the search space.

Proposition 3 *Let $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ and $\mathcal{Q}_{K'} = (\mathcal{L} \wedge \Gamma)_{K'}$ be two K-queries. If $K' \subseteq K$, then $Sup(\mathcal{Q}_K, S) \leq Sup(\mathcal{Q}_{K'}, S)$ for every data set S.*

As a consequence of Proposition 3, the computation of interesting K-queries $\mathcal{Q}_K$ with some K fixed by the user can use the results of a previous step based on a set of variables K' such that $K' \subset K$. Indeed, a K-query $\mathcal{Q}_K$ can not be interesting if a K-query $\mathcal{Q}_{K'}$ with $K' \subset K$ has been evaluated before as being not interesting.

It is now important to note that our Definition 4 of support complies with logical implication, as stated in Proposition 4 below. Roughly, this proposition says that if a K-query $\mathcal{P}_K$ implies a K-query $\mathcal{Q}_K$, then the support of $\mathcal{P}_K$ is smaller than the support of $\mathcal{Q}_K$.

Proposition 4 *Let $\mathcal{P}_K$ and $\mathcal{Q}_K$ be two K-queries. Given a data set S, if S is a model of the K-rule $(\mathcal{P} \rightarrow \mathcal{Q})_K$, then $Sup(\mathcal{P}_K, S) \leq Sup(\mathcal{Q}_K, S)$.*

Running Example 5 In the context of our Running Example 1, the data set S is a model of the logical implication $(\forall x, y, z)(Cust(x, y, z) \wedge (z > 50) \rightarrow Sale(x, P3))$. According to Proposition 4, it follows that:

$$Sup((Cust(x,y,z) \wedge (z > 50))_x, S) \leq Sup((Sale(x, P3))_x, S)$$

Indeed, it is easy to see that $Sup((Cust(x, y, z) \wedge (z > 50))_x, S) = 1/3$ and $Sup((Sale(x, P3))_x, S) = 2/3$.

Most of the pattern discovery algorithms use an order relation on patterns based on θ-subsumption (see (Plotkin, 1970)) to reduce the size of the search space. According to Proposition 4, we could use logical implication to compare K-queries and their supports. However, testing if a data set S is a model of the implication $(\forall \vec{X})(\mathcal{P} \rightarrow \mathcal{Q})$ cannot be efficient in any case. For that reason, we propose an other order relation based on subset relation (see Definition 5). This order relation is less general than logical implication, but allows to compare efficiently K-queries, independently from a data set S.

Definition 5 *Let $(\mathcal{L}_1 \wedge \Gamma_1)_K$ and $(\mathcal{L}_2 \wedge \Gamma_2)_K$ be two K-queries. We say that $(\mathcal{L}_1 \wedge \Gamma_1)_K$ is more general than $(\mathcal{L}_2 \wedge \Gamma_2)_K$, denoted by $(\mathcal{L}_1 \wedge \Gamma_1)_K \succeq_g (\mathcal{L}_2 \wedge \Gamma_2)_K$ if $\mathcal{L}_1 \subseteq \mathcal{L}_2$ and $Sol(\Gamma_2) \subseteq Sol(\Gamma_1)$.*

Running Example 6 In the context of our Running Example 1, we have:

$$\begin{array}{rcl} (Cust(x,y,z))_x & \succeq_g & (Cust(x,y,z) \wedge (y = Teacher))_x \\ & \succeq_g & (Cust(x,y,z) \wedge (y = Teacher) \wedge (z > 30))_x \\ & \succeq_g & (Cust(x,y,z) \wedge Sale(x,y) \wedge Prod(y,w) \wedge \\ & & (y = Teacher) \wedge (z > 30) \wedge (w = Beer))_x \end{array}$$

Now, we compare the relation $\succeq_g$ defined above with that of C-subsumption introduced in (Mizoguchi and Ohwada, 1995). To this end, we first recall that, if $(\mathcal{L}_1 \wedge \Gamma_1)_K$ and $(\mathcal{L}_2 \wedge \Gamma_2)_K$ are two K-queries, then $(\mathcal{L}_1 \wedge \Gamma_1)_K$ *C-subsumes* $(\mathcal{L}_2 \wedge \Gamma_2)_K$ if there exists a substitution θ such that $\mathcal{L}_1\theta \subseteq \mathcal{L}_2$ and $Sol(\Gamma_2) \subseteq Sol(\Gamma_1\theta)$. It turns out that the relation $\succeq_g$ is less general than that of C-subsumption. Indeed, in the context of our Running Example 1, the K-queries $(Cust(x, y, z))_x$ and $(Cust(x, Teacher, z) \wedge (z > 30))_x$ are not comparable, whereas the K-query $(Cust(x, y, z))_x$ C-subsumes the K-query $(Cust(x, Teacher, z) \wedge (z > 30))_x$. However, it can be shown that the following property holds:

Proposition 5 *Let* $(\mathcal{L}_1 \wedge \Gamma_1)_K$ *and* $(\mathcal{L}_2 \wedge \Gamma_2)_K$ *be two K-queries. If* $(\mathcal{L}_1 \wedge \Gamma_1)_K$ C-subsumes $(\mathcal{L}_2 \wedge \Gamma_2)_K$, *then there exists a K-query* $(\mathcal{L}'_2 \wedge \Gamma'_2)_K$ *such that* $Sup((\mathcal{L}'_2 \wedge \Gamma'_2)_K, S) = Sup((\mathcal{L}_2 \wedge \Gamma_2)_K, S)$ *and* $(\mathcal{L}_1 \wedge \Gamma_1)_K \succeq_g (\mathcal{L}'_2 \wedge \Gamma'_2)_K$.

Considering the previous example, it can be seen that, as the K-query $(Cust(x, y, z))_x$ C-subsumes the K-query $(Cust(x, Teacher, z) \wedge (z > 30))_x$, the K-query $(Cust(x, y, z) \wedge (y = Teacher) \wedge (z > 30))_x$ is such that $Sup((Cust(x, Teacher, z) \wedge (z > 30))_x, S) = Sup((Cust(x, y, z) \wedge (y = Teacher) \wedge (z > 30))_x, S)$ and $(Cust(x, y, z))_x \succeq_g (Cust(x, y, z) \wedge (y = Teacher) \wedge (z > 30))_x$.

It is now important to note that the relation $\succeq_g$ is a special case of logical implication. Thus, according to Proposition 4, we have the following Corollary 1 which is used in Section 3 to reduce the size of the search space.

Corollary 1 *Let $\mathcal{Q}_K$ and $\mathcal{P}_K$ be two K-queries such that $\mathcal{Q}_K \succeq_g \mathcal{P}_K$. Then, for every data set S, $Sup(\mathcal{P}_K, S) \leq Sup(\mathcal{Q}_K, S)$.*

We end this section with an important remark concerning the relationship between our definition of support and Halpern's work (Halpern, 1990), which provides semantics to first-order logics of probability. Indeed, it turns out that our definition of support can be expressed in terms of probability. Using the formalism of (Halpern, 1990) and the notation introduced previously, we have:

$$Sup(\mathcal{P}_K, S) = w_K((\exists \vec{Y})(\mathcal{P}))$$

where $\vec{Y}$ is the vector of variables in $Var(\mathcal{P}) \setminus X$, and $w_K((\exists \vec{Y})(\mathcal{P}))$ is the probability that the formula $(\exists \vec{Y})(\mathcal{P})$ is true if the variables in K are randomly chosen in their respective domains (assuming a uniform probability distribution on all domains).

2.4 Confidence of a K-rule

Apart from frequency or support, rules are traditionally selected according to a second quality criterion, called confidence. We now define the notion of confidence in the case of K-rules as follows:

Definition 6 - Confidence of a K-rule. *Let $(\mathcal{B} \rightarrow \mathcal{H})_K$ be a K-rule where $\mathcal{B}$ and $\mathcal{H}$ are sets of atoms and constraints, and let S be a data set. The confidence of $(\mathcal{B} \rightarrow \mathcal{H})_K$ in S, denoted by $Conf((\mathcal{B} \rightarrow \mathcal{H})_K, S)$, is defined by:*

$$Conf((\mathcal{B} \rightarrow \mathcal{H})_K, S) = \frac{\left|\{\theta_{|K} \mid \theta \in \Theta(\mathcal{B} \wedge \mathcal{H}) \wedge S \models (\mathcal{B} \wedge \mathcal{H})\theta\}\right|}{\left|\{\theta_{|K} \mid \theta \in \Theta(\mathcal{B}) \wedge S \models \mathcal{B}\theta\}\right|}$$

A K-rule $\mathcal{R}_K$ is sufficiently valid *if its confidence is greater than a given threshold, called the* minimum confidence threshold *and denoted by* minconf.

Definition 7 - Significant K-rule. *A K-rule $\mathcal{R}_K : (\mathcal{B} \rightarrow \mathcal{H})_K$ is* significant *if it is sufficiently valid and if the K-query $(\mathcal{B} \wedge \mathcal{H})_K$ is interesting.*

The following proposition relates confidence to first-order logic, in the sense that the confidence of a K-rule is 1 if and only if the data set S is a model of the corresponding formula. Nevertheless, it is important to note that this property holds only in the case where the so-called Closed World Assumption (CWA) of (Clark, 1978) is assumed. We simply recall here, that assuming CWA roughly means that if an atom cannot be proved to be true, then this atom is considered to be false.

Proposition 6 *Let $(\mathcal{B} \rightarrow \mathcal{H})_K$ be a K-rule and let S be a data set. Considering the closed world assumption, we have $Conf((\mathcal{B} \rightarrow \mathcal{H})_K, S) = 1$ iff the data set S is a model of $(\mathcal{B} \rightarrow \mathcal{H})_K$.*

It is important to note that, contrary to the support (see Corollary 1), the confidence does not enjoy any monotonicity property. That is, if a K-rule $\mathcal{R}_K$ is more specific than a K-rule $\mathcal{R}'_K$, then neither $Conf(\mathcal{R}_K, S) \leq Conf(\mathcal{R}'_K, S)$ nor $Conf(\mathcal{R}'_K, S) \leq Conf(\mathcal{R}_K, S)$ hold in general. The following example illustrates this point.

Example 1 Let S be the data set $S = \{p(a,b), p(a',b'), q(a)\}$, where $dom(1,p) = dom(1,q) = \{a, a'\}$ and $dom(2,p) = \{b, b'\}$. Consider now the following K-rules over S:

$$\mathcal{R}_x : (p(x,y) \rightarrow q(x))_x \text{ and } \mathcal{R}_x^1 : (p(x,b) \rightarrow q(x))_x \text{ and } \mathcal{R}_x^2 : (p(x,b') \rightarrow q(x))_x$$

It is clear that $\mathcal{R}_x^1$ and $\mathcal{R}_x^2$ are both more specific than $\mathcal{R}_x$. However, based on Definition 6, we have $Conf(\mathcal{R}_x, S) = 1/2$, $Conf(\mathcal{R}_x^1, S) = 1$ and $Conf(\mathcal{R}_x^2, S) = 0$, showing that $Conf(\mathcal{R}_x, S) \leq Conf(\mathcal{R}_x^1, S)$ and $Conf(\mathcal{R}_x, S) \geq Conf(\mathcal{R}_x^2, S)$.

3 Pattern-Discovering Algorithms

In this section, we present algorithms for interesting K-query discovery (see Section 3.2) and significant K-rule discovery (see Section 3.3). To this end, we first introduce the declarative language bias that we use to reduce the size of the search space. Then we present the algorithms and discuss possible optimizations when considering implementation issues.

3.1 A Declarative Language Bias

In Inductive Logic Programming, the notion of declarative language bias has been studied extensively (see (Dehaspe, 1998; Weber, 1998)) since it is crucial for reducing the size of the search space.

In this paper, we define a search space by means of a declarative language bias grammar $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$ where:

- $\mathcal{K}_G$ is a set of variables whose domains are finite. It means only K-queries and K-rules where K is a subset of $\mathcal{K}_G$, are to be considered,
- $\mathcal{L}_G$ is a set of atoms. The atoms in $\mathcal{L}_G$ are the basic ingredients of the K-queries and K-rules that we consider,
- Γ_G is a set of elementary constraints of the form $x_i = a_i$, $x_i \leq a_i$ or $x_i > b_i$ where a_i and b_i are constants and x_i is a variable.

Moreover, we consider only conjunctions of atoms that are *connected*, according to the following definition:

Definition 8 - Connectivity. *Let $\mathcal{L}$ be a conjunction of n atoms. We say that $\mathcal{L}$ is connected if it can be written as $\mathcal{L} = L_1 \wedge L_2 \wedge \ldots \wedge L_n$ where L_i, $i = 1 \ldots n$, are atoms such that: for every k, $1 < k \leq n$, there exists $i < k$ such that L_k has at least one variable in common with L_i.*

We now define the set of K-queries that corresponds to a given grammar $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$ as being the set of K-queries $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ such that:

- $K = Var(\mathcal{L}) \cap \mathcal{K}_G$ where K is not empty,
- $\mathcal{L} \subseteq \mathcal{L}_G$ and $\mathcal{L}$ is connected,
- $\Gamma \subseteq \Gamma_G$ and $Var(\Gamma) \subseteq Var(\mathcal{L})$.

We denote this set by $\mathcal{G}^*$. Then, the set of K-rules that are built from the grammar $\mathcal{G}$ is the set of K-rules $(\mathcal{B} \to \mathcal{H})_K$ such that $\mathcal{B}_K$ and $(\mathcal{B} \wedge \mathcal{H})_K$ are in $\mathcal{G}^*$.

Running Example 7 In the context of our Running Example 1, let $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$ be the grammar defined by:

- $\mathcal{K}_G = \{k_1\}$,
- $\mathcal{L}_G = \{Cust(k_1, x_1, x_2), Sale(k_1, k_2), Sale(k_1, k_2'), Prod(k_2, y_2), Prod(k_2', y_2')\}$,
- $\Gamma_G = \{x_1 = Teacher, x_2 > 30\} \cup \{y_2 = a, y_2' = a \mid a \in dom(2, Prod)\}$.

The following K-query and K-rule are in $\mathcal{G}^*$:

$$\mathcal{Q}_1 : (Cust(k_1, Teacher, x_2) \wedge (x_2 > 30) \wedge Sale(k_1, k_2) \wedge Prod(k_2, Beer))_{k_1}$$
$$\mathcal{R}_1 : (Sale(k_1, k_2) \wedge Sale(k_1, k_2') \wedge Prod(k_2, Milk) \to Prod(k_2', Tea))_{k_1}$$

whereas the following K-query and K-rule are not:

$$\mathcal{Q}_2 : (Cust(k_1, x_1, x_2) \wedge Prod(k_2, y2) \wedge (x_1 = Lawyer))_{k_1}$$
$$\mathcal{R}_2 : (Prod(k_2, y_2) \wedge (y_2 = Milk) \rightarrow Sale(k_1, k_2))_{k_1}$$

Indeed, the K-query $\mathcal{Q}_2$ is not connected and contains an elementary constraint $(x_1 = Lawyer)$ that is not in Γ_G. On the other hand, the K-rule $\mathcal{R}_2$ is not considered as valid, since $(Prod(k_2, y_2) \wedge (y_2 = Milk))_{k_1}$ is not a K-query.

3.2 Discovering interesting K-queries

Given a grammar $\mathcal{G}$, the purpose of this section is to give algorithms for the computation of all interesting K-queries in $\mathcal{G}^*$ over a given data set S. In order to explore the search space $\mathcal{G}^*$, a K-query $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ can be specialized either by adding a new literal L to $\mathcal{L}$, or by introducing a constraint Γ' more restrictive than Γ. To this end, we introduce two refinement operators ρ_L and ρ_C as follows:

Definition 9 - Refinement Operators. *Let $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$ be a grammar and $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ be a K-query in $\mathcal{G}^*$. Then, $\rho_L(\mathcal{Q}_K)$ is the set of specialized K-queries $(\mathcal{L} \wedge L \wedge \Gamma)_{K'}$ such that:*

- *L is an atom in $\mathcal{L}_G$ and $(\mathcal{L} \wedge L)$ is connected,*
- *$K' = Var(\mathcal{L} \wedge L) \cap \mathcal{K}_G$.*

On the other hand, $\rho_C(\mathcal{Q}_K)$ is the set of specialized K-queries $(\mathcal{L} \wedge \Gamma')_K$ such that:

- *Γ' is a subset of Γ_G more restrictive than Γ, i.e. $Sol(\Gamma') \subset Sol(\Gamma)$,*
- *there does not exist a constraint $\Gamma'' \subseteq \Gamma_G$ such that $Sol(\Gamma') \subset Sol(\Gamma'')$ and $Sol(\Gamma'') \subset Sol(\Gamma)$.*

We now emphasize that, if $\overline{\mathcal{G}^*}$ denotes the set of K-queries in $\mathcal{G}^*$ without constraints, then:

$$\overline{\mathcal{G}^*} = \rho_L^*(\emptyset) \quad \text{and} \quad \mathcal{G}^* = \bigcup_{\mathcal{L}_K \in \overline{\mathcal{G}^*}} \rho_C^*(\mathcal{L}_K)$$

where $\rho_L^*(\emptyset)$ and $\rho_C^*(\mathcal{L}_K)$ are defined by:

$$\rho_L^*(\emptyset) = \bigcup_{i=1}^{|\mathcal{L}_G|} \rho_L^i(\emptyset) \quad \text{and} \quad \rho_C^*(\mathcal{L}_K) = \bigcup_{j=1}^{|\Gamma_G|} \rho_C^j(\mathcal{L}_K)$$

Given a K-query $\mathcal{Q}_K \in \mathcal{G}^*$, we can also note that for every K-query $\mathcal{Q}'_{K'}$ in $\rho_L(\mathcal{Q}_K)$, we have $\mathcal{Q}_K \succeq_g \mathcal{Q}'_K$ and $K \subseteq K'$. On the other hand, for every K-query $\mathcal{Q}'_K$ in $\rho_C(\mathcal{Q}_K)$, we have $\mathcal{Q}_K \succeq_g \mathcal{Q}'_K$. Therefore, according to Proposition 3 and Corollary 1, we have the following corollary:

Corollary 2 *Let $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$ be a grammar. For every K-query $\mathcal{Q}_K$ in $\mathcal{G}^*$:*

- *If $\mathcal{Q}'_{K'}$ is a K-query in $\rho_L(\mathcal{Q}_K)$, then $Sup(\mathcal{Q}'_{K'}, S) \leq Sup(\mathcal{Q}_K, S)$,*
- *If $\mathcal{Q}'_K$ is a K-query in $\rho_C(\mathcal{Q}_K)$, then $Sup(\mathcal{Q}'_K, S) \leq Sup(\mathcal{Q}_K, S)$.*

The implementation of the operator ρ_L follows directly from its definition. In Figure 2, we propose an algorithm that builds $\overline{\mathcal{G}^*}$ from a given grammar $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$. On the other hand, the implementation of the operator ρ_C depends on the form of the elementary constraints in Γ_G. Let us assume that the set Γ_G contains only elementary constraints of the form $x_i = a_i$, $x_i \leq a_i$ or $x_i > b_i$, and that, given a variable x, it contains either equality constraints or inequality constraints (but not both) concerning x. Let $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ be a K-query in $\mathcal{G}^*$. Then, the set $\rho_C(\mathcal{Q}_K)$ contains all the K-queries $\mathcal{Q}'_K = (\mathcal{L} \wedge \Gamma')_K$ where:

- $\Gamma' = \Gamma \wedge (x_i = a_i)$, if $(x_i = a_i) \in \Gamma_G$ and $x_i \notin Var(\Gamma)$,
- $\Gamma' = \Gamma \wedge (x_i \leq a_i)$, if $(x_i \leq a_i) \in \Gamma_G$ and $a_i = \max\{a \mid (x_i \leq a) \in \Gamma_G \setminus \Gamma\}$,
- $\Gamma' = \Gamma \wedge (x_i > b_i)$, if $(x_i > b_i) \in \Gamma_G$ and $b_i = \min\{b \mid (x_i > b) \in \Gamma_G \setminus \Gamma\}$.

Based on the operators ρ_L and ρ_C, the algorithm MIQ (MIQ stands for Mining Interesting K-Queries) that computes all interesting K-queries is shown in Figure 1.

Most algorithms for interesting pattern discovery are based on the generic levelwise search method proposed in (Mannila and Toivonen, 1997). This method starts from the most general patterns, and builds all interesting patterns level by level. Each iteration step consists of two phases:

- the *candidate generation phase* computes the candidate patterns of level $d+1$ using the interesting patterns of level d. This generation phase is based on the property that a pattern can not be interesting if it is more specific than a pattern that was evaluated before as being not interesting.
- the *candidate evaluation phase* computes the supports of all candidate patterns at level $d+1$. One important property of this phase is that all the supports can be evaluated through a single database pass.

In our framework, the computation of all interesting K-queries is decomposed into N steps, where N is the number of atoms in $\mathcal{L}_G$. At step n, $0 < n \leq N$, algorithm MIQ builds for every K-query of cardinality $n+1$ in $\overline{\mathcal{G}^*}$, the set $\lambda_K(\mathcal{L})$ of interesting K-queries in $\rho^*_C(\mathcal{L}_K)$, i.e. $\lambda_K(\mathcal{L}) = \{\mathcal{Q}_K \in \rho^*_C(\mathcal{L}_L) \mid Sup(\mathcal{Q}_K, S) \geq minsup\}$. It can be seen that the sets $\lambda_K(\mathcal{L})$ have a semi-lattice structure with respect to the order relation $\succeq_g$, and thus, these sets can be obtained according to the generic levelwise search method of (Mannila and Toivonen, 1997). In this respect, algorithm MIQ computes every set $\lambda_K(\mathcal{L})$ level by level, using algorithm MIQ-Eval (see Figure 3) for the candidate evaluation phase, and algorithm MIQ-Gen (see Figure 4) for the candidate generation phase. More precisely:

- Algorithm MIQ-Eval computes the supports of a set $\mathcal{C}^d$ of P candidate K-queries $(\mathcal{L} \wedge \Gamma_j)_K$, $j = 1 \ldots P$. While the main loop (step 3.) iterates over all possible substitutions σ of variables in K, the inner loop (step 4.) iterates over all examples θ of $\mathcal{L}_K$ such that $\theta_{|K} = \sigma$. According to Definition 4 of support, a boolean b_j allows to increment at most once the support counter c_j, when there exists several examples θ of a K-query $(\mathcal{L} \wedge \Gamma_j)_K$ such that $\theta_{|K} = \sigma$.
- Algorithm MIQ-Gen computes a set $\mathcal{C}^{d+1}$ of candidate K-queries from a set $\mathcal{C}^d$ of interesting K-queries. For every K-query $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ in $\mathcal{C}^d$, MIQ-Gen adds to $\mathcal{C}^{d+1}$ every K-query $\mathcal{Q}'_K = (\mathcal{L} \wedge \Gamma')_K$ in $\rho_C(\mathcal{Q}_K)$. Nevertheless, MIQ-Gen reduces the number of candidates, using Corollary 2. More precisely:
  - At step 5., MIQ-Gen tests if there exists a K-query $\mathcal{P}_K$ in $\rho_C^{-1}(\mathcal{Q}'_K)$, that was evaluated before as being not interesting ($\mathcal{P}_K \notin \mathcal{C}^d$),
  - At step 6., MIQ-Gen tests if there exists a K-query $(\mathcal{L}' \wedge \Gamma')_{K'}$ such that $K' \subseteq K$, $\mathcal{L}' \subseteq \mathcal{L}$, $Sol(\Gamma) \subseteq Sol(\Gamma')$, that was evaluated before as being not interesting ($(\mathcal{L}' \wedge \Gamma')_{K'} \notin \lambda_{K'}(\mathcal{L}')$).

```
Algorithm: MIQ
 Input:   the grammar $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$
 Output:  all the sets of interesting $K$-queries $\lambda_K(\mathcal{L})$ where $\mathcal{L} \in \overline{\mathcal{G}^*}$
 Uses:    a minimum support threshold minsup and a data set $S$
 1.  Compute $\overline{\mathcal{G}^*}$ using MIQ-Init and Initialize $n = 0$
 2.  While $n < N$ where $N$ is the cardinality of $\mathcal{L}_G$ do
 3.    For each $K$-query $\mathcal{L}_K$ of cardinality $n+1$ in $\overline{\mathcal{G}^*}$ ($\mathcal{L}_K \in \overline{\mathcal{G}^*_{d+1}}$) do
 4.      // Compute the set $\lambda_K(\mathcal{L})$ of interesting $K$-queries $(\mathcal{L} \wedge \Gamma)_K$
 5.      Initialize level $d = 0$ and $\lambda_K(\mathcal{L}) = \emptyset$
 6.      Initialize the set of candidate $K$-queries $\mathcal{C}^0 = \{\mathcal{L}_K\}$
 7.      While $\mathcal{C}^d$ is not empty do
 8.        Compute $Sup((\mathcal{L} \wedge \Gamma)_K, S)$ for all $(\mathcal{L} \wedge \Gamma)_K \in \mathcal{C}^d$ using MIQ-Eval
 9.        Delete from $\mathcal{C}^d$ the $K$-queries $(\mathcal{L} \wedge \Gamma)_K$ with support below minsup
 10.       Add all $K$-queries $(\mathcal{L} \wedge \Gamma)_K \in \mathcal{C}^d$ to $\lambda_K(\mathcal{L})$
 11.       Compute the set of candidate $K$-queries $\mathcal{C}^{d+1}$ from $\mathcal{C}^d$ using MIQ-Gen
 12.       Increment $d$
 13.     End
 14.   End
 15. End
```

Figure 1: The MIQ Algorithm

```
Algorithm: MIQ-Init
 Input:    the grammar $\mathcal{G} = <\mathcal{K}_G, \mathcal{L}_G, \Gamma_G>$
 Output:   the set $\overline{\mathcal{G}^*}$ of $K$-queries $\mathcal{L}_K$ where $\mathcal{L} \subseteq \mathcal{L}_G$ is connected
           and $K = Var(\mathcal{L}) \cap \mathcal{K}_G$ is not empty
 1.  Initialize $\overline{\mathcal{G}^*} = \emptyset$, $n = 1$ and $\overline{\mathcal{G}^*_1} = \emptyset$
 2.  For each literal $L \in \mathcal{L}_G$ such that $Var(L) \cap \mathcal{K}_G \neq \emptyset$
 3.      Add the $K$-query $L_K$ to $\overline{\mathcal{G}^*_1}$ where $K = Var(L) \cap \mathcal{K}_G$
 4.  $\overline{\mathcal{G}^*} = \overline{\mathcal{G}^*} \cup \overline{\mathcal{G}^*_1}$
 5.  While $n < N$ where $N$ is the cardinality of $\mathcal{L}_G$ do
 6.      Initialize $\overline{\mathcal{G}^*_{n+1}} = \emptyset$
 7.      For each $K$-query $\mathcal{L}_K \in \overline{\mathcal{G}^*_n}$
 8.          For each literal $L \in \mathcal{L}_G$ such that $L \notin \mathcal{L}_K$
 9.              If $(Var(\mathcal{L}_K) \cap Var(L) \neq \emptyset)$
 10.                 Add the $K$-query $(\mathcal{L} \wedge L)_{K'}$ to $\overline{\mathcal{G}^*_{n+1}}$ where $K' = Var(\mathcal{L} \wedge L) \cap \mathcal{K}_G$
 11.     $\overline{\mathcal{G}^*} = \overline{\mathcal{G}^*} \cup \overline{\mathcal{G}^*_{n+1}}$
 12.     Increment $n$
 13. End
 14. Return $\overline{\mathcal{G}^*}$
```

Figure 2: The MIQ-Init Algorithm

```
Algorithm: MIQ-Eval
 Input:    a set $\mathcal{C}^d$ of $P$ candidate $K$-queries $(\mathcal{L} \wedge \Gamma_j)_K$, $j = 1 \ldots P$
 Output:   the supports of $K$-queries $(\mathcal{L} \wedge \Gamma_j)_K \in \mathcal{C}^d$
 Uses:     a data set $S$ and the set $\Theta(K)$ of all substitutions $\sigma$ such that
           $(\forall x \in K)(\sigma(x) \in dom(x, \mathcal{L}))$
 1.  For each $K$-query $(\mathcal{L} \wedge \Gamma_j)_K \in \mathcal{C}^d$
 2.      Initialize support counter $c_j = 0$ and boolean $b_j = false$
 3.  For each substitution $\sigma \in \Theta(K)$ do
 4.      For each substitution $\theta \in \Theta(\mathcal{L})$ such that $\theta_{|K} = \sigma$ and $\mathcal{L}\theta \subseteq S$ do
 5.          For each $K$-query $(\mathcal{L} \wedge \Gamma_j)_K \in \mathcal{C}^d$
 6.              If $b_j = false$ and $\Gamma_j\theta$ is true then
 7.                  $b_j = true$ and increment $c_j$
 8.      End
 9.      For each $K$-query $(\mathcal{L} \wedge \Gamma_j)_K \in \mathcal{C}^d$ do  $b_j = false$
 10. End
 11. For each $K$-query $(\mathcal{L} \wedge \Gamma_j)_K \in \mathcal{C}^d$ return $c_j/|\theta(K)|$
```

Figure 3: The MIQ-Eval Algorithm

```
Algorithm: MIQ-Gen
Input:   a set $\mathcal{C}^d$ of interesting $K$-queries $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ where $\mathcal{L}_K$ is a $K$-query
         of cardinality $n+1$ in $\overline{\mathcal{G}^*}$ and $\Gamma \subseteq \Gamma_G$
Output:  a set $\mathcal{C}^{d+1}$ of candidate $K$-queries $\mathcal{Q}'_K = (\mathcal{L} \wedge \Gamma')_K$ where $\mathcal{L}_K$ is a $K$-
         query of cardinality $n+1$ in $\overline{\mathcal{G}^*}$ and $\Gamma' \subseteq \Gamma_G$
Uses:    all the sets $\lambda_{K'}(\mathcal{L}')$ where $K' \subseteq K$, $\mathcal{L}' \subseteq \mathcal{L}$ and $\mathcal{L}'_{K'}$ is a $K$-query of
         cardinality $n$ in $\overline{\mathcal{G}^*}$
1.  Initialize level $\mathcal{C}^{d+1} = \emptyset$
2.  For each $K$-query $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ in $\mathcal{C}^d$ do
3.    For each $K$-query $\mathcal{Q}'_K = (\mathcal{L} \wedge \Gamma')_K$ in $\rho_C(\mathcal{Q}_K)$
4.      Add the $K$-query $\mathcal{Q}'_K$ into $\mathcal{C}^{d+1}$ unless
5.        • There exists a $K$-query $\mathcal{P}_K$ such that $\mathcal{Q}'_K \in \rho_C(\mathcal{P}_K)$ and $\mathcal{P}_K \notin \mathcal{C}^d$ or
6.        • There exists $\Gamma' \subseteq \Gamma_G$ such that $Sol(\Gamma) \subseteq Sol(\Gamma')$ and
            $K' \subseteq K$, $\mathcal{L}' \subseteq \mathcal{L}$ such that $\mathcal{L}'_{K'} \in \overline{\mathcal{G}^*_n}$ and $(\mathcal{L}' \wedge \Gamma')_{K'} \notin \lambda_{K'}(\mathcal{L}')$
7.  Return $\mathcal{C}^{d+1}$
```

Figure 4: The MIQ-Gen Algorithm

3.3 Discovering Significant K-rules

In this section, we propose an algorithm (see Figure 5) that mines significant K-rules from a pre-computed set of interesting K-queries. This algorithm discovers only significant K-rules with one atom H in the head. Moreover, we consider only K-rules $(\mathcal{B} \wedge \Gamma_B \rightarrow H \wedge \Gamma_H)_K$ such that:

- the constraint Γ_H concerns only variables that occur in the head H, but not in the body $\mathcal{B}$ (see Step 7.),
- the set K is a subset of $Var(\mathcal{B})$ (see Step 6). According to this condition, it follows that any significant K-rule discovered by algorithm FSR could be transformed into a *safe* clause. We recall here that a rule is safe if every variable appearing in the head also appears in the body (see (Ullman, 1989)).

3.4 Implementation Issues

The algorithms presented in Sections 3.2 and 3.3 are general purpose tools. In this section, we disuss possible optimizations in case of implementation in a relational database environment. In such an environment, we first recall that the facts in a data set S are stored as tuples in associated tables. We denote by RP_i the table associated to a predicate p_i.

We first consider the possible optimizations of the candidate evaluation algorithm MIQ-Eval given a grammar $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$. Let $\mathcal{L}_K = (L_1 \wedge \ldots \wedge L_n)_K$ be a

```
Algorithm: FSR
Input:   the set $IQ$ of all interesting $K$-queries in $\mathcal{G}^*$
Output:  the set $SR$ of significant $K$-rules
Uses:    a minimum confidence threshold $minconf$ and a data set $S$
1.    $SR = \emptyset$
2.    for each $K$-query $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K \in IQ$ do
3.       for each literal $H \in \mathcal{L}$ do
4.          Let $\mathcal{R}_K = (\mathcal{B} \wedge \Gamma_B \rightarrow H \wedge \Gamma_H)_K$ be the $K$-rule such that:
5.             • $\mathcal{L} = \mathcal{B} \wedge H$ and $\Gamma = \Gamma_B \wedge \Gamma_H$ and
6.             • $K \subseteq Var(\mathcal{B})$ and
7.             • $(\forall \gamma \in \Gamma)(Var(\gamma) \subseteq Var(\mathcal{B}) \Rightarrow \gamma \in \Gamma_B)$
8.          If $Conf(\mathcal{R}_K, S) \geq minconf$
9.             Add the $K$-rule $\mathcal{R}_K$ to $SR$
10.      end
11.   end
12.   Return $SR$
```

Figure 5: The FSR Algorithm

K-query in $\overline{\mathcal{G}^*}$ where each L_i is a literal over predicate p_i $(1 \leq i \leq n)$, and $\mathcal{C}^d$ be a set of candidate K-queries $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$ where $\Gamma \subseteq \Gamma_G$. In order to evaluate the supports of K-queries in $\mathcal{C}^d$, algorithm MIQ-Eval has to make one pass through the join of the relations RP_i, $i = 1 \ldots n$, assuming that the join is sorted by the attributes corresponding to the variables in K. The cost of processing this sorted join can be improved if ordered join indices (Li and Ross, 1998) on the join attributes are available, and we note that this set of join indices can be determined from $\overline{\mathcal{G}^*}$. For instance, in the context of our Running Example 7, the table $RSale$ can be seen as a join indice between the tables $RCust$ and $RProd$. Therefore, the computation of supports is efficient if the table $RSale$ is sorted by the attribute that corresponds to k_1.
As another possible optimization of algorithm MIQ-Eval, we also emphasize that the set of candidate K-queries in $\mathcal{C}^d$ have to be structured. Currently, given a substitution $\theta \in \Theta(\mathcal{L})$, algorithm MIQ-Eval tests independently for each candidate K-query $\mathcal{Q}_K = (\mathcal{L} \wedge \Gamma)_K$, if $\Gamma\theta$ is true. However, the candidate K-queries in $\mathcal{C}^d$ may have a lot of elementary constraints in common. For that reason, it should be more efficient to organize the candidate K-queries in $\mathcal{C}^d$ based on a hash tree, as algorithm Apriori does for the candidate itemsets (Agrawal et al, 1996).

We now argue that, in some practical cases, algorithm MIQ has not to compute all the sets $\lambda_K(\mathcal{L})$ where $\mathcal{L}_K$ is a K-query without constraints in $\mathcal{G}^*$. Let us consider again our Running Example 7. We may assume that the attribute K_1 of the relation $RCust(K_1, X_1, X_2)$ is a foreign key referencing the attribute K_1 of the relation

$RSale(K_1, K_2)$, whereas the attribute K_2 of the relation $RSale(K_1, K_2)$ is a foreign key referencing the attribute K_2 of the relation $RProd(K_1, Y_2)$. If so, the data set S satisfies:

$$(\forall k_1, x_1, x_2)(Cust(k_1, x_1, x_2) \rightarrow (\exists k_2)(Sale(k_1, k_2))) \text{ and}$$
$$(\forall k_1, k_2)(Sale(k_1, k_2) \rightarrow (\exists y_2)(Prod(k_2, y_2)))$$

which means that only customers who buy products are registered, and that all products that are sold are registered. Consequently, for every K-query $(Cust(k_1, x_1, x_2) \wedge \Gamma)_{k_1}$ where $\Gamma \subseteq \Gamma_G$, we have:

$$\begin{aligned}&Sup((Cust(k_1, x_1, x_2) \wedge \Gamma)_{k_1}, S)\\ &= Sup((Cust(k_1, x_1, x_2) \wedge Sale(k_1, k_2) \wedge \Gamma)_{k_1}, S)\\ &= Sup((Cust(k_1, x_1, x_2) \wedge Sale(k_1, k_2) \wedge Prod(k_2, y_2) \wedge \Gamma)_{k_1}, S)\end{aligned}$$

It follows that all interesting K-queries in $\lambda_{k_1}(Cust(k_1, x_1, x_2))$ can be obtained when computing $\lambda_{k_1}(Cust(k_1, x_1, x_2) \wedge Sale(k_1, k_2) \wedge Prod(k_2, y_2))$. However, the computation of $\lambda_{k_1}(Cust(k_1, x_1, x_2))$ can reduce the number of candidate K-queries in $\lambda_{k_1}(Cust(k_1, x_1, x_2) \wedge Sale(k_1, k_2) \wedge Prod(k_2, y_2))$. Nevertheless, the benefit of this reduction is important only if the size of the relation $RCust$ is much smaller than the size of the join of the relations $RCust$, $RSale$ and $RProd$.
More generally, given a grammar $\mathcal{G} =< \mathcal{K}_G, \mathcal{L}_G, \Gamma_G >$, let $\mathcal{L}_K$ and $(\mathcal{L} \wedge L)_K$ be two K-queries in $\overline{\mathcal{G}^*}$. If for every data set S, S is a model of $(\forall \vec{X})(\mathcal{L} \rightarrow (\exists \vec{Y})L)$ where $\vec{X}$ and $\vec{Y}$ are the vectors of variables in $Var(\mathcal{L})$ and $Var(L) \setminus Var(\mathcal{L})$ respectively, then the computation of $\lambda_K(\mathcal{L})$ before $\lambda_K(\mathcal{L} \wedge L)$ is relevant only if the size of the relation that corresponds to $\mathcal{L}$ is much smaller than the size of the join of the relations that correspond to $\mathcal{L}$ and L.

4 Concluding Remarks

In this paper, we have considered the problem of interesting pattern discovery in the general framework of first-order logics. The main impact of this generalization with respect to other approaches is, on the one hand that we can handle data sets that are stored in multiple tables, not just in a single table, and on the other hand, that the rules that are discovered can be transformed into Datalog rules.

Several open problems remain. In this paper, we considered only two measures to evaluate the interestingness of K-rules, namely support and confidence. We have now to study and investigate the use of other measures such as *relative accuracy* (Lavrac et al, 1999), *lift* or *conviction* (Brin et al, 1997). As these measures can be expressed in terms of probability, we note that they can be defined in the general framework of first-order logics using the formalism of (Halpern, 1990).

Another perspective is the study of *negation*. In this paper, we considered data sets that contain only positive facts. If we consider data sets that contain both positive and negative facts, representing true and false informations respectively, we can extend the notions of K-query and K-rule so that they contain both positive and negative literals. The problem here is that the main property of confidence does not hold any more. Indeed, in the presence of negative informations, we can not consider the Closed World Assumption, implying that Proposition 6 does not hold (i.e. the confidence of a K-rule can be equal to 1 although the data set is not a model of this rule). Hence, we have to propose new measures of confidence.

In this paper, we considered only elementary constraints of the form $x_i = a_i$, $x_i \leq a_i$ or $x_i > b_i$, where a_i and b_i are constants and x_i is a variable. An interesting question would be to investigate other forms of constraints. These investigations should use the propositions of (Lakshmanan et al, 1998) and (Srikant et al, 1997).

Finally, we are currently considering two other directions of research. The first concerns the implementation of our algorithms in an OLAP (On-Line Analytical Processing) and Data Warehouse environment as in (Kamber et al, 1997; Han, 1997), where data sets are stored in *star schema* structures (Kimball, 1996). The second line of research concerns updates. When new facts are inserted or existing facts are deleted, it may be the case that new rules become significant, while others are no more significant. This aspect of rule discovery is considered in (Laurent and Vrain, 1996), and is currently under investigation in the framework of the present approach.

References

R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A.I. Verkamo (1996). *Fast Discovery of Association Rules.* In Advances in Knowledge Discovery and Data Mining, pp 309-328, AAAI-MIT Press.

S. Brin, R. Motwani, J. Ullman, S. Tsur (1997). *Dynamic Itemset Counting and Implication Rules for Market Basket Data.* ACM SIGMOD'97, pp 255-264.

K. L. Clark (1978). *Negation as failure.* Logic and Data Bases, Plemum Press, Eds. H. Gallaire and J. Minker.

J. Han (1997). *Olap Mining: An Integration of Olap with Data Mining.* IFIP 1997, Conf. on Data Semantics, Leysain, Switzerland, pp.1-11.

L. Dehaspe (1998). *Frequent Pattern Discovery in First-Order Logic.* PhD Thesis, Department of Computer Science, Katholieke Universiteit Leuven.

A. Faye, A. Giacometti, D. Laurent and N. Spyratos (1998). *Mining Significant Rules from Databases.* Networking and Information Systems Journal, Vol. 1, No. 1, pp. 653-682.

A. Faye, A. Giacometti, D. Laurent and N. Spyratos (1999). *Discovering And Updating Rules From Databases.* In Computing Anticipatory Systems:

CASYS'98 - Second International Conference, edited by Daniel M. Dubois, American Institute Of Physics, Woodbury, New-York, Conference Proceedings 465, pp. 321-243.

U. Fayyad, D. Haussler, and P. Stolorz (1996). *Mining Scientific Data.* Communications of the ACM, November 1996, Vol. 39, No. 11.

J. Y. Halpern (1990). *An analysis of first-order logics of probability.* Artificial Intelligence, No. 46, pp 311-350.

M. Kamber, J. Han and J.Y. Chiang (1997). *Metarule-guided mining of multi-dimensional association rules using data cubes.* In Proc. KDD'97, pp 207-210, Newport Beach, California.

R. Kimball (1996). *The data warehouse toolkit.* John Willey & Sons.

L.V.S. Lakshmanan, R. Ng, J. Han and A. Pang (1998). *Optimization of constrained frequent set queries: 2-var constraints.* Tech. Report, Dept. of Computer Science, University of British Columbia.

D. Laurent and Ch. Vrain (1996). *Learning Query-Rules for Optimizing Databases with Update-Rules.* Proc. LID'96, LNCS 1154, Springer-Verlag, p-p 153-172.

N. Lavrač, P. Flach et B. Zupan (1999). *Rule Evaluation Measures: a Unifying View.* Proc. of ILP'99 (to appear).

Z. Li and K.A. Ross (1998). *Fast Joins Using Join Indices.* VLDB Journal, Vol. 7, Number 4.

H. Mannila and H. Toivonen (1997). *Levelwise Search and Borders of Theories in Knowledge Discovery.* Techn. Rep. C-1997-8, University of Helsinki.

D. Maier (1983) *The Theory of Relational Databases.* Computer Science Press.

F. Mizogushi and H. Ohawada (1995). *Constrained relative least general generalization for inducing constrain logic programm.* New Generation Computing, 13 (3-4), pp 335-368.

G.D. Plotkin (1970). *A note on induction generaliazation.* Machine Intelligence, 5, 1970.

R. Srikant, Q. Vu and R. Agrawal (1997). *Mining Association Rules with Item Constraints.* In Proc. KDD'97, Newport Beach, California, pp 407-419.

J.D. Ullman (1989). *Principles of Databases and Knowledge-Base Systems.* Tome 2, Computer Science Press.

I. Weber (1998). *A declarative language bias for levelwise search of first-order regularities.* Proc. FGML-98, Eds Fritz Wysotzki, Peter Geibel, and Christina Schadler, TU Berlin, 1998.