Iceberg Query Lattices for Datalog

Gerd Stumme

Knowledge & Data Engineering Group
Department of Mathematics & Computer Science
University of Kassel, D–34121 Kassel, Germany
www.kde.cs.uni-kassel.de, stumme@cs.uni-kassel.de

Abstract. In this paper we study two orthogonal extensions of the classical data mining problem of mining association rules, and show how they naturally interact. The first is the extension from a propositional representation to datalog, and the second is the condensed representation of frequent itemsets by means of Formal Concept Analysis (FCA). We combine the notion of *frequent datalog queries* with *iceberg concept lattices* (also called *closed itemsets*) of FCA and introduce two kinds of *iceberg query lattices* as condensed representations of frequent datalog queries. We demonstrate that iceberg query lattices provide a natural way to visualize relational association rules in a non-redundant way.

1 Introduction

Mining association rules is a popular knowledge discovery problem. Since the problem was stated [1], various approaches have been proposed for an increased efficiency of rule discovery in very large databases [2, 6, 8, 23, 24]. In parallel, researchers have extended the original problem to knowledge representations that are either related to and/or more expressive than the original representation in propositional logic of itemsets. These include for instance generalized association rules [28], or frequent patterns within time series [3].

In this paper, we consider the extension to first order logic as introduced by L. de Haspe and H. Toivonen in [10] and [11]. Instead of considering rules of the form $X \to Y$ where X and Y are sets of attributes (items; e. g., products of a supermarket) which may or may not apply to objects (e. g., to transactions), they consider X and Y to be datalog queries. This allows specifically to consider relations between objects, and thus enhances the expressiveness of the association rules which can be discovered. The resulting *relational association rules*, however, suffer to an even larger extent the main problem of classical association rule mining: even for a small dataset, the number of resulting rules is very high, and there are many uninteresting and even redundant rules in the result.

In the classical scenario, several solutions (for instance measures of "usefulness" [7]) have been proposed. These approaches can also be applied to relational association rules, but they all have in common that they lose some information.

A complementary approach is based on *Formal Concept Analysis* (*FCA*) [37, 15]. Its simplest data structure, a 'formal context', fits exactly the scenario of classical association rule mining. It turned out that the concepts of the concept lattice provide a condensed representation for frequent itemsets.

This observation was independently made by three research groups around 1997/98: L. Lakhal and his database group in Clermont–Ferrand [22], M. Zaki in Troy, NY [39], and the author in Darmstadt [31]. The first algorithm based on this idea was Close [24], followed by A-Close [23], ChARM [40], PASCAL [5], Closet [26], TITANIC [33, 35], and others, each having its own way to exploit the closure system which is hidden in the data. Then different researchers started working on 'condensed representations' of frequent itemsets and association rules: closed sets, free sets, k-free sets, etc. Some of them were well-known in FCA for quite a while, (e. g., closed sets as concept intents — see for instance [14] — and free sets as minimal generators), others (for instance k-free sets) truly extended the set of condensed representations.

In this paper, we discuss how these representations can be applied to datalog queries and to relational association rules. Our approach is as follows: first we re-formulate the problem of mining relational association rules in terms of Formal Concept Analysis. Then we are able to apply in a straightforward manner the full arsenal of FCA-based condensed representations (closed sets, pseudo-closed sets, free sets) to frequent queries. From them, we define *iceberg query lattices* and show by an example that they are adequate for visualizing relational association rules.

This paper continues our work on iceberg concept lattices presented in [35]. It is organized as follows. After giving an introduction to Datalog and describing the problem of mining relational association rules in Section 2, we recall the basics of mining (classical) association rules with Formal Concept Analysis in Section 3. In Section 4 we will restate the relational mining problem in terms of FCA and introduce *iceberg query lattices*. In Section 5, we discuss their use for visualizing relational association rules, before we conclude in Section 6.

2 Relational Association Rules

Relational association rules were inspired by adopting the classical task of mining association rules to ILP (Inductive Logic Programming). First we recall the classical problem, before introducing datalog and relational association rules.

2.1 Association Rules

The problem of mining *association rules* has been discussed for a decade now. It can be stated as follows.

Problem 1 (Association rule mining [1]). Let M be a set of items,[1] G a set of transaction IDs, and I a binary relation between G and M, indicating which items have been bought in which transactions. An *association rule* is a pair $X \rightarrow Y$ of subsets of M with $X \subseteq Y$.[2] Its *support* is the relative frequency of the transactions containing Y

[1] In the scenario of warehouse basket analysis, the items are products sold by the supermarket.

[2] Usually, one requires head and body of the rules to be disjoint, but both statements are equivalent, since $X \rightarrow Y$ and $X \rightarrow Y \setminus X$ have the same support and the same confidence. Our version of the problem statement is both closer to the way association rules are computed and more adequate to the problem statement for relational association rules.

among all transactions, and its *confidence* is the relative frequency of all transactions containing Y among those containing X.

The problem of mining association rules is to compute, for given thresholds *minsupp* and *minconf* $\in [0,1]$, all association rules such that their support and confidence are above (or equal to) the thresholds *minsupp* and *minconf*, resp.

In a supermarket database, for instance, the rule 'salmon $\rightarrow$ white_wine (conf = 73 %, supp = 3 %)' would indicate that 73 % of all customers buying salmon also buy white wine, and that this combination is bought in 3 % of all transactions.

The standard approach to the problem is to compute first all frequent itemsets Y (i. e., all itemsets with a support above the threshold *minsupp*), and then check for each of them and for each of its subsets X if $conf(X \rightarrow Y) \geq$ *minconf*. The first step is the expensive one, as it requires (multiple) access(es) to the database. Therefore, most research in the domain focuses on this first step.

Relational Association Rules have been introduced by L. Dehaspe and H. Toivonen in [11], following their work on the discovery of frequent datalog patterns [10]. Relational association rules are defined within the framework of *datalog*. This enhances thus the expressiveness of the rules that can be discovered. The basic idea is to replace G and I by a datalog database, and M by a set of datalog queries. Before describing this idea in detail, we recall some basic datalog definitions.

2.2 Datalog

A deductive *datalog database* **r** is a set of definite clauses, i. e., of universally quantified formulas of the form $\forall \vec{x}.l_o \leftarrow (l_1 \wedge \ldots \wedge l_n)$ with l_i being positive literals, $n \geq 0$, and $\vec{x}$ consisting of all variables appearing in the literals l_i. We abbreviate the clauses by $l_o \leftarrow l_1, \ldots, l_n$. A clause with $n = 0$ which does not contain any variables is called a *fact*.

A *substitution* θ is a set $\{X_1/a_1, \ldots, X_m/a_m\}$ of bindings of variables X_i to terms a_i. The *instance* $C\theta$ of a clause C for a substitution θ is obtained from C by replacing all occurrences of the variables X_i by the terms a_i, resp. If $C\theta$ is ground (i. e., if it contains only constants as terms, no variables), then it is called *ground instance* of C, and θ is a *grounding substitution*.

A *datalog query* Q is a logical expression of the form $? - l_1, \ldots, l_n$ with $n \geq 1$, where the l_i are *literals* (i. e., predicates or negated predicates). A query Q *succeeds* for a database **r** if there exists a grounding substitution θ for Q such that the conjunction $(l_1 \wedge \ldots \wedge l_n)\theta$ holds within the database. θ is then called *answering substitution for* Q. The set of all answering substitutions for Q in **r** is denoted by *answerset*$(Q, \mathbf{r})$.

In order to avoid some logical problems related to negation, we restrict ourselves to *range-restricted* queries, i. e., to queries where all variables that occur in negative literals also occur in at least one positive literal.

Example 1. We illustrate these concepts using an example from [11], which we will use as running example throughout the paper. The datalog database is shown in Table 1. It consists of facts only. It could be (and usually is) extended by so-called intensional relations as, e. g., *grandparent*$(X, Z) \leftarrow$ *parent*(X, Y), *parent*(Y, Z). which provides an intensional definition of the predicate *grandparent*.

Table 1. Example datalog database **r**

customer(*allen*)	parent(*allen, bill*)	buys (*allen, wine*)
customer(*bill*)	parent(*allen, carol*)	buys (*bill, cola*)
customer(*carol*)	parent(*bill, zoe*)	buys (*bill, pizza*)
customer(*diana*)	parent(*carol, diana*)	buys (*diana, pizza*)

Consider now the query

$$Q \;\; := \;\; ?-\mathit{customer}(X), \mathit{parent}(X, Y), \mathit{buys}(Y, \mathit{pizza}).$$

Applied to the database **r**, it will return all couples (a_1, a_2) of instances such that the three facts $\mathit{customer}(a_1)$, $\mathit{parent}(a_1, a_2)$, and $\mathit{buys}(a_2, \mathit{pizza})$ are all in the database. The result is thus $\mathit{answerset}(Q, \mathbf{r}) := \{(X/\mathit{allen}, Y/\mathit{bill}), (X/\mathit{carol}, Y/\mathit{diana}\}$.

2.3 Relational Association Rules

Although the intuition of 'datalog association rules' is quite straightforward, there are some subtleties to be resolved. For instance, it is not clear from the start what exactly is to be counted. These aspects have been discussed in [11] and led to the following definition of the problem.

Definition: A *relational association rule* (or *query extension*) is an implication of the form

$$?-l_1, \ldots, l_m. \;\; \to \; ?-l_1, \ldots, l_m, l_{m+1}, \ldots, l_n.$$

with $1 \leq m < n$, where both parts separately are existentially quantified. The rule may be abbreviated by

$$?-l_1, \ldots, l_m \rightsquigarrow l_{m+1}, \ldots, l_n.$$

$?-l_1, \ldots, l_m.$ is the *body* of the rule, and the subquery $l_{m+1}, \ldots, l_n$ is the *head* of the rule. The *conclusion* of the rule is $?-l_1, \ldots, l_m, l_{m+1}, \ldots, l_n$. We will sometimes write the rule in the form $Q_1 \to Q_2$, where Q_1 stands for the body of the rule, and Q_2 for the conclusion of the rule.

Note that relational association rules consist of two queries, where the second one extends the first. In the sequel, we will consider the conclusion of the rule as query rather than the head, as only this guarantees that the scope of the existential quantifier is as it is intended. Different ways of 'misinterpreting' this have been discussed in [11].

In the standard case of association rule mining, transactions are counted to define support and confidence. The transaction id is a key in the database (modeled as set G in Problem 1), so that counting becomes unambiguous. For relational association rules, this has to be simulated. In [11], one of the relations of the datalog database (called *key*) is taking over this role.

Definition: Let **r** be a datalog database and Q be a query containing an atom *key*. Then the *support* (or *relative frequency*) of query Q with respect to database **r** given *key* is

$$\mathit{supp}(Q, \mathbf{r}, \mathit{key}) := \frac{|\{\theta \in \mathit{answerset}(?-\mathit{key}, \mathbf{r}) \mid Q\theta \text{ succeeds w.r.t. } \mathbf{r}\}|}{|\mathit{answerset}(?-\mathit{key}, \mathbf{r})|} \;\; .$$

The *support* of a relational association rule is given by the support of the conclusion of the rule. Its confidence is the support of the conclusion of the rule divided by the support of the body of the rule.

Now we are able to formally state the problem of mining all frequent relational association rules.

Problem 2 (Relational Association Rule Mining [11]). Let **r** be a datalog database and $\mathcal{L}$ a set of datalog queries that all contain an atom *key*. Let *minsupp* and *minconf* be two (user-defined) thresholds between 0 and 1. The task is then to discover among the relational association rules which can be built from $\mathcal{L}$ all rules with support and confidence above the given thresholds.

Lemma 1. *The standard problem of mining association rules is a special case of this scenario.*

Proof: Let M be a set of items, G a set of transaction IDs, and $I \subseteq G \times M$. We introduce a unary predicate *key* which holds for all $g \in G$. We consider each item $m \in M$ as a constant, and introduce a binary predicate *contains*(g, m) that holds whenever transaction g contains item m. If the set $\mathcal{L}$ contains all relational queries composed of the literal *key*(X) and any combination of literals of the form *contains*(X, m), then mining all relational association rules is equivalent to mining all association rules in the classical sense. □

Example 2. In our running example, we are able to discover for instance the following rules. In brackets are shown support (as decimal number) and confidence (as fraction). They are based on the *customer* predicate as key.

$$? - \mathit{customer}(A), \mathit{buys}(A, \mathit{wine}) \rightsquigarrow \mathit{parent}(A, B), \mathit{parent}(B, C) \qquad (0.25, 1/1)$$
$$? - \mathit{customer}(A), \mathit{parent}(A, B) \rightsquigarrow \mathit{buys}(B, \mathit{cola}) \qquad (0.25, 1/3)$$
$$? - \mathit{customer}(A), \mathit{parent}(A, B) \rightsquigarrow \mathit{buys}(B, \mathit{cola}), \mathit{buys}(B, \mathit{pizza}) \qquad (0.25, 1/3)$$

The first rule states that each customer buying wine is also a grandparent. The second rule states that a third of all customers having a child also have a child buying cola. (Remark that this statement is different from the following: a third of all children is buying cola. In the first case, the parents are counted, and in the second the children.) The last statement says that a third of all customers having a child also have a child buying cola and pizza.

The declarative language bias used in this example restricts the construction of queries in the following way:[3] Variable A is bound by the *customer* predicate, and may serve as input in the first position of *parent* and/or *buys*. The *parent* predicate may be iterated, while *buys* is forced to have one of the constants *cola*, *pizza*, or *wine* at the

[3] In WARMODE format, this is stated as "warmode_key(customer(-)). warmode(parent(+,-)). warmode(buys(+, cola)). warmode(buys(+,pizza)). warmode(buys(+,wine))." '-' indicates that the variable is bound by the atom, and '+' means that the variable is bound before the atom is called.

second position. This prohibits for instance queries like ‘return all customers buying the same item as one of their children’. This bias allows to fine-tune the trade-off between the size of the set of rules and the expressive power of the rules.

As in the classical case, this problem is naturally split in two sub-problems: first compute all frequent queries $?-\ l_1, \ldots, l_n.$, and then check the support of all rules of the form $?-l_1, \ldots, l_m \leadsto l_{m+1}, \ldots, l_n.$ by considering all possible partitions of the set of literals. In this paper, we focus on the first step.

In [11], an algorithm for computing frequent queries, called WARMR, is presented, which basically is a first order variant of the well known Apriori algorithm [2]. In order to reduce the resulting rules to a set of ‘useful rules’, WARMR makes additional use of a *declarative language bias* as known from Inductive Logic Programming (ILP). The basic idea is to fix the order in which the variables are bound. Details can be found in[11].

As in the case of classic association rule mining, a major problem for mining relational association rules is the high number of resulting rules. (In fact, the problem is much larger in the new scenario.) In the classical case, a number of additional measures for ‘interestingness’ have been introduced to reduce further the number of rules (see for instance [7]). These measures can of course also be applied to relational association rules. In [11], Dehaspe and Toivonen additionally discuss the statistical significance of the confidence, and the declarative language bias discussed above to further reduce the number of rules. All of these approaches reduce the number of rules, but for the price of loosing some information.

In the next section, we will discuss how quite a number of frequent queries — and subsequently of rules — can be pruned *without* loosing any information. Only if our pruning does not sufficiently reduce the number of rules, then additional means are needed to continue pruning (with loss of information).

The basic idea of our approach is based on the observation that some rules talk about exactly the same set of instances (and thus with exactly the same support and the same confidence) , but on different levels of detail. In that case, the more specific rule can be pruned without loss of information. In Example 2, for instance, the second rule is comprised by the third, and both rules are talking about exactly the same instance, namely *allen*. Hence it is sufficient to present the third rule to the user; the second can be pruned without loosing any information.[4] In the next section, we discuss the theoretical background for this kind of pruning.

3 Formal Concepts and Association Rules

Consider again the classical problem of mining association rules (Problem 1). Assume that there are two itemsets which both describe exactly the same set of transactions. So if we know the support of one of them, we do not need to count the support of the other one in the database. In fact, we can introduce an equivalence relation Ψ on the set of itemsets. Two itemsets are said to be *equivalent with respect to a database* if and only if

[4] More precisely, the second rule can derived from the third rule together with the exact rules discussed in Section 5.

they are contained in exactly the same set of transactions. If we knew the relation from the beginning, it would be sufficient to count the support of one itemset of each class only — all other supports can then be derived.

Of course one does not know the equivalence relation in advance, but one can determine it along the computation. It turns out that one usually has to count the support of more than one query of each class, but normally not of all of them. The percentage of queries to be considered depends on how correlated the data are: the more correlated the data are, the fewer counts have to be performed.

3.1 Formal Concept Analysis

Condensed representations of frequent itemsets (e. g., free or closed sets) were inspired by the theory of Formal Concept Analyis [22, 39, 31]. We assume the reader to be familiar with the basic notions of *Formal Concept Analysis (FCA)* and refer to [15] otherwise. For keeping notations consistent, however, we recall the most important definitions.

Definition: A *(formal) context* $\mathbb{K} := (G, M, I)$ consists of a set G of objects, a set M of attributes, and a binary relation $I \subseteq G \times M$.

The mapping $\cdot^{\uparrow}\colon \mathfrak{P}(G) \to \mathfrak{P}(M)$ is given by $A^{\uparrow} := \{m \in M \mid \forall g \in A\colon (g,m) \in I\}$. The mapping $\cdot^{\downarrow}\colon \mathfrak{P}(M) \to \mathfrak{P}(G)$ is defined dually by $B^{\downarrow} := \{g \in G \mid \forall m \in B\colon (g,m) \in I\}$. If it is clear from the context whether the first or the second mapping is meant, then we abbreviate both $\cdot^{\uparrow}$ and $\cdot^{\downarrow}$ just by $\cdot'$. In particular, B'' stands for $(B^{\downarrow})^{\uparrow}$.

A *(formal) concept* is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. A is called *extent* and B is called *intent* of the concept. The set $\underline{\mathfrak{B}}(\mathbb{K})$ of all concepts of a formal context $\mathbb{K}$ together with the partial order $(A_1, B_1) \leq (A_2, B_2) :\Leftrightarrow A_1 \subseteq A_2$ (which is equivalent to $B_1 \supseteq B_2$) is called *concept lattice* of $\mathbb{K}$.

3.2 Mining association rules with FCA

Obviously, a formal context is just the right data structure for describing the problem of classical association rule mining (Problem 1), and FCA provides indeed a natural framework for the task of mining association rules. The equivalence relation Ψ mentioned above is formalized as follows: for $X, Y \in \mathfrak{P}(M)$, let $X \Psi Y$ if and only if $X' = Y'$. It turns out that each concept intent is exactly the largest itemset of its equivalence class. The concept intents are also called *closed itemsets*, as they are exactly the closed sets of the canonical closure operator $B \mapsto B''$ on $\mathfrak{P}(M)$. For any itemset $B \in \mathfrak{P}(M)$, its closure is the set B'', which is just the concept intent of its equivalence class. Note that in particular holds, for any $\hat{B} \in \mathfrak{P}(M)$ with $B \Psi \hat{B}$, $supp(\hat{B}) = supp(B) = supp(B'')$. The (frequent) concept intents/closed itemsets can hence be considered as 'normal forms' of the (frequent) itemsets.

The *support* of a formal concept (A, B) of $\mathbb{K}$ is the support of the itemset B (which is equal to the ratio $\frac{|A|}{|G|}$). The concepts with a support above or equal to a user-defined threshold *minsupp* $\in [0, 1]$ are called *frequent concepts*, and the set of all frequent concepts is called *iceberg concept lattice* [35]. While it is not of particular interest to study the set of all frequent (closed and non-closed) itemsets, the iceberg lattice provides interesting insights into the data. In [4, 34, 25], we showed how the number of association rules can be reduced by using the iceberg concept lattice, and how the latter can be used for visualizing the rules.

In particular, the iceberg concept lattice contains all information needed to derive the support of all (frequent) itemsets. This observation has been exploited first in the Close algorithm [22] to improve the efficiency of algorithms for mining frequent itemsets. Instead of using the maximal elements of the equivalence classes, one can also use their *minimal generators* (which are now called *free sets* or *key sets* in data mining) [4, 34]. They can be computed, together with the closed sets, e. g., with the algorithm TITANIC [35]. Rather than recalling these results here, we directly apply them to the task of relational association rule mining.

4 Iceberg Concept Lattices for Datalog Queries

Let us now come back to the problem of mining frequent datalog queries.

Example 3. Figure 1 shows all 32 queries which follow the declarative language bias introduced above, and which have at least one answering substitution.[5] Each node in the diagram stands for one query, which consists of all literals that are attached at the node itself, or at some node above (following straight lines only). A query succeeds in our example database with all answering substitutions which are attached to the node of the query itself or to any node below in the hierarchy. If parts of the query are logically redundant, then only the relevant part of the answering substitution is given. The numbers in the nodes indicate the support as discussed in Section 4.2. For the moment, we just ignore them.

For instance, the right-most lower node stands for the query

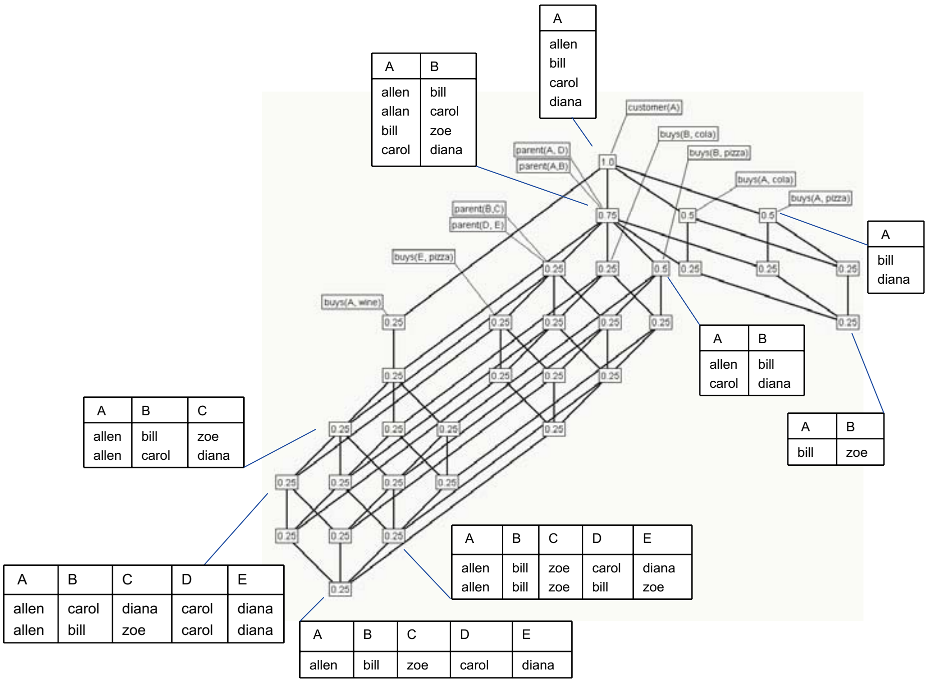
$$?-\,\mathit{customer}(A), \mathit{parent}(A,B), \mathit{parent}(A,D), \mathit{buys}(A,\mathit{cola}), \mathit{buys}(A,\mathit{pizza}).$$

In this case, the literal $\mathit{parent}(A,D)$ is logically redundant, and may be removed. It is needed on the left side of the diagram, though, where we have to distinguish between the different grandchildren of *allen*.

Note that we do not talk about sets of queries (as we would talk about sets of items in the classical case), but only about single queries. This stems from the fact that (unlike in the classical situation where the combination of items is not an item itself) the combination of queries is again a query, since the set of datalog queries is closed under conjunction. We can hence *identify* any finite set of queries with the conjunction of its queries.[6] Therefore, we assume in the sequel that the set $\mathcal{L}$ of datalog queries that we consider is always closed under conjunction.

[5] Remark that [11] lists only 26 of these frequent queries; all queries where both $\mathit{parent}(A,B)$ and $\mathit{parent}(A,D)$ are required are omitted. The reason seems to be that WARMR prunes logically redundant queries immediately when it passes them, even though they may be needed for building up more specific queries, as, e. g., the query represented by the left-most lower node.

[6] Eventually variable renaming has to be performed (in the usual first order logic way) before the conjunction is computed; in order to respect the range of the existential quantifiers of the individual queries.



A
allen bill carol diana

A	B
allen allan bill carol	bill carol zoe diana

A
bill diana

A	B
allen carol	bill diana

A	B
bill	zoe

A	B	C
allen allen	bill carol	zoe diana

A	B	C	D	E
allen allen	carol bill	diana zoe	carol carol	diana diana

A	B	C	D	E
allen allen	bill bill	zoe zoe	carol bill	diana zoe

A	B	C	D	E
allen	bill	zoe	carol	diana

Fig. 1. All queries following the declarative language bias which have at least one answering substitution.

While it is not really informative to study the set of all frequent queries, the situation changes when we consider the *closed queries* among them only. In order to define them formally, we first have to bring together the notions of datalog and FCA. We will consider two formal contexts that canonically arise from our scenario. Both definitions give rise to different understandings of 'closed queries'. They are discussed in the two following subsections, resp.

4.1 Iceberg Query Lattices of Datalog Databases

Definition: Let **r** be a datalog database and $\mathcal{L}$ a set of datalog queries. The *formal context associated to* **r** *and* $\mathcal{L}$ is defined by $\mathbb{K}_{\mathbf{r},\mathcal{L}} := (G_{\mathbf{r},\mathcal{L}}, M_{\mathbf{r},\mathcal{L}}, I_{\mathbf{r},\mathcal{L}})$ where $G_{\mathbf{r},\mathcal{L}} := \{\theta \mid \theta$ is a grounding substitution for all $L \in \mathcal{L}\}$, $M_{\mathbf{r},\mathcal{L}} := \mathcal{L}$, and $(\theta, Q) \in I_{\mathbf{r},\mathcal{L}}$ if and only if $\theta \in$ *answerset*$(Q, \mathbf{r})$.

From this formal context, one can compute the concept lattice as usual. As discussed above, we may identify the intent of a formal concept (A, B) of this lattice with the query $\bigwedge B$, which stands for the conjunction of all queries contained in B. Such a query is also called *closed query with respect to* **r** *and* $\mathcal{L}$, as it is related to the closed set B. Like in the classical scenario, one can introduce an equivalence relation $\Psi_{\mathbf{r}}$ on the set of queries. Two queries Q_1 and Q_2 are said to be *equivalent with respect to database*

$\mathbf{r}$ if and only if $\textit{answerset}(Q_1, \mathbf{r}) = \textit{answerset}(Q_2, \mathbf{r})$. The most specific query of each equivalence class is then just the closed query which is assigned to the corresponding formal concept of the context $\mathbb{K}_{\mathbf{r},\mathcal{L}}$.

Definition: Let $\mathbf{r}$ be a datalog database and $\mathcal{L}$ a set of datalog queries. The *iceberg query lattice of* $\mathbf{r}$ *and* $\mathcal{L}$ for *minsupp* $\in [0, 1]$ is given by $\underline{\mathfrak{B}}(\mathbf{r}, \mathcal{L}) := (\{Q \in \mathcal{L} \mid Q$ is closed with respect to $\mathbf{r}$ and $\mathcal{L},$ and the corresponding concept is frequent$\}, \models)$, where $\models$ is the usual logical implication.

Example 4. Figure 2 shows all frequent closed queries of our running example, where ‘frequent’ means support strictly larger than 0. The diagram is read in the same way as in the previous figure: a node represents the query which consists of all literals labeled at it and at all higher nodes. As in Figure 1, the diagram shows the relevant parts of the answering substitutions for each query. The bold nodes are discussed in the next subsection.

In Figure 1, the nine frequent closed queries are exactly those which are labeled by an answering substitution. Each of the 32 frequent queries belongs to the same equivalence class of $\Psi_{\mathbf{r}}$ as the highest closed query which is below it (i. e., to its most general closed specialization). The right-most upper query $?-\ \textit{customer}(A), \textit{buys}(A, \textit{cola}), \textit{buys}(A, \textit{pizza}).$, for instance, is in the same class as the closed query which is just below it (and which has as additional literal $\textit{parent}(A, B)$).

Without any loss of information, the diagram in Figure 2 gives a much better insight into the database. It shows for instance that being grandparent and buying wine is equivalent in our example, since $\textit{buys}(A, \textit{wine})$ and $\textit{parent}(B, C)$ generate the same node. It also shows that any *customer* buying *cola* also buys *pizza* and is *parent* of someone. This *implication* (or *exact association rule*) is indicated by the fact that the node labeled by $\textit{buys}(A, \textit{cola})$ is below the nodes labeled by $\textit{parent}(A, B)$ and $\textit{buys}(A, \textit{pizza})$, resp., in the diagram. This is the general way implications are read in concept lattices.

It is obvious that the restriction to frequent closed queries gives a much better insight into the content of the database. One drawback, however, — at least for the association rule scenario — still exists: the meaning of counting objects is not intuitively clear. As the size of the ‘relevant part’ of an answering substitution depends on the number of variables involved, it is not clear what exactly has to be counted. If one requires the user to provide meaningful values for *minsupp* and *minconf* for the mining task, then this question has to be answered. That is the reason why Dehaspe and Toivonen introduced an explicit *key* predicate in [11]. We discuss their approach in the light of FCA next.

4.2 Iceberg Query Lattices of Datalog Databases with Respect to a Key Predicate

Again, we first transform the datalog database into a formal context. The difference to the approach discussed above is that we now consider only the instances of the *key* predicate as objects.

Definition: Let $\mathbf{r}$ be a datalog database and $\mathcal{L}$ be a set of datalog queries which all contain an atom *key*. The *formal context associated to* $\mathbf{r}$, $\mathcal{L}$, *and key* is defined by

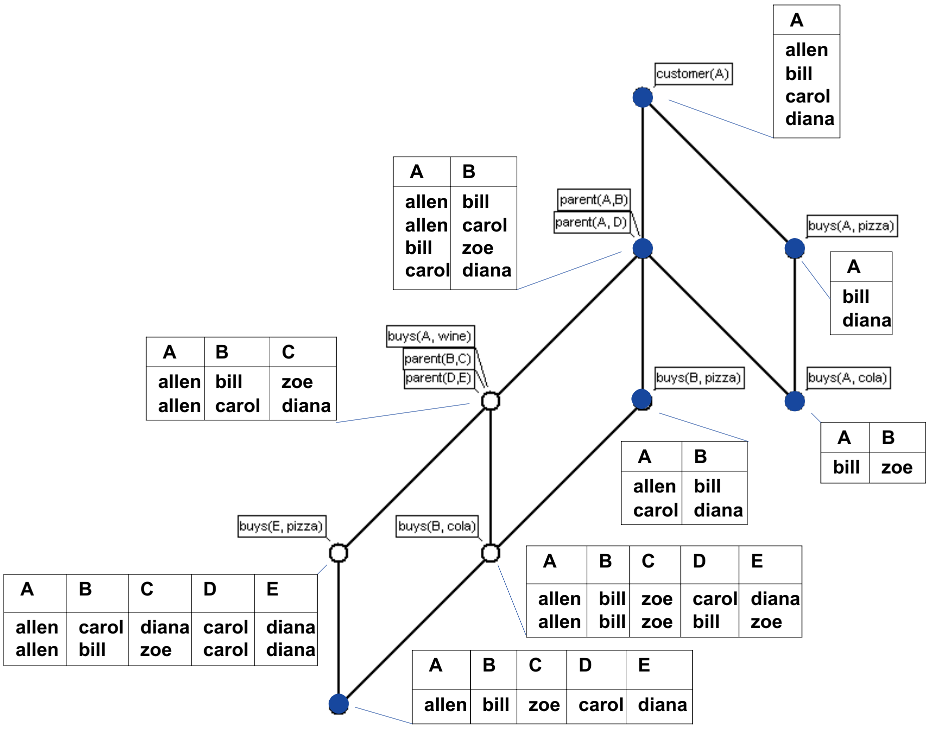


Fig. 2. All frequent closed queries.

$\mathbb{K}_{\mathbf{r},\mathcal{L},key} := (G_{\mathbf{r},\mathcal{L},key}, M_{\mathbf{r},\mathcal{L},key}, I_{\mathbf{r},\mathcal{L},key})$ where $G_{\mathbf{r},\mathcal{L},key} := answerset(?-key, \mathbf{r})$, $M_{\mathbf{r},\mathcal{L},key} := \mathcal{L}$, and $(\theta, Q) \in I_{\mathbf{r},\mathcal{L},key}$ if and only if there exists a grounding substitution $\hat{\theta}$ for Q with $\theta \subseteq \hat{\theta}$.

Two queries $Q_1, Q_2 \in \mathcal{L}$ are *equivalent with respect to* $\mathbf{r}$ *and key* (denoted $Q_1 \Psi_{\mathbf{r},key} Q_2$) if $Q_1' = Q_2'$ holds in $\mathbb{K}_{\mathbf{r},\mathcal{L},key}$. *Closed queries of* $\mathbf{r}$ *and* $\mathcal{L}$ *with respect to key and minsupp* $\in [0,1]$ are defined as above. The *iceberg query lattice* of $\mathbf{r}$, $\mathcal{L}$, and *key* for *minsupp* is $\underline{\mathfrak{B}}(\mathbf{r}, \mathcal{L}, key) := (\{Q \in \mathcal{L} \mid Q \text{ is closed and } supp(Q, \mathbf{r}, key) \geq minsupp\}, \models)$.

Example 5. Figure 3 shows all six queries which are frequent according to this definition for any *minsupp* $\in (0, 0.25]$. The closed queries displayed here are also closed queries of the context $\mathbb{K}_{\mathbf{r},\mathcal{L}}$. In Figure 2, they are the ones marked by filled nodes. Theorem 1 below shows that this containment holds in general.

As in the previous example, we can read off implications between queries from the diagram. In particular, we rediscover the implications discussed in Figure 4: being grandparent and buying wine is equivalent; and any *customer* buying *cola* also buys *pizza* and is *parent* of someone. But as we have now a coarser look to the data, there are more equivalences to be discovered in this representation. Indeed, we focus on the *customer*s, and do not distinguish explicitly between the different family lines of customer

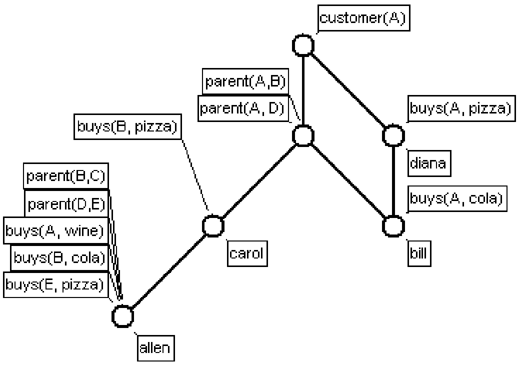


Fig. 3. All frequent closed queries for the *key* predicate '*customer*'.

allen any more: the diagram shows that having a grandchild buying *pizza* is equivalent to having a child buying *cola* (who needs not be *parent* of the grandchild). So by choosing which of the contexts $\mathbb{K}_{\mathbf{r},\mathcal{L}}$ and $\mathbb{K}_{\mathbf{r},\mathcal{L},\mathit{key}}$ to study, we can decide how close to look at the relations between the instances.

The numbers in Figure 1 show the support of each query measured in the context $\mathbb{K}_{\mathbf{r},\mathcal{L},\mathit{key}}$. If a query Q_1 logically subsumes another query Q_2 (i. e., Q_1 is below Q_2 in the diagram) and $\mathit{supp}(Q_1) = \mathit{supp}(Q_2)$ holds, then both queries have the same closure (and are in the same class of $\Psi_{\mathbf{r},\mathit{key}}$). As the closed queries are the most specific in their equivalence classes, they are exactly those queries whose support is different from all supports of the queries which are immediately below it (see [35] for details). In Figure 1, the six queries which are closed with respect to *key* predicate '*customer*' are thus: the top-most query, the one immediately below it, the queries labeled by *buys*$(A,$ *pizza*$)$, *buys*$(B,$ *pizza*$)$, and the two queries which do not have any lower neighbors.

The following theorem shows the general relationship between the (iceberg) query lattices of the formal contexts introduced in this and in the previous subsection.

Theorem 1 *Let* **r** *be a datalog database and $\mathcal{L}$ be a set of datalog queries closed under conjunction where all queries contain an atom key. $\underline{\mathfrak{B}}(\mathbf{r},\mathcal{L},\mathit{key})$ is a $\vee$-sub-semi-lattice of $\underline{\mathfrak{B}}(\mathbf{r},\mathcal{L})$. Here, '$\vee$' can be read both as join (supremum) in the concept lattice or as operator returning the most specific common generalization of two queries.*

Proof: For $\mathbb{K}_{\mathbf{r},\mathcal{L}}$ and $\mathbb{K}_{\mathbf{r},\mathcal{L},\mathit{key}}$, we have $M_{\mathbf{r},\mathcal{L}} = M_{\mathbf{r},\mathcal{L},\mathit{key}}$. We show that for each $\theta \in G_{\mathbf{r},\mathcal{L},\mathit{key}}$ exists a $\hat{\theta} \in G_{\mathbf{r},\mathcal{L}}$ with $\{\theta\}' = \{\hat{\theta}\}'$ where $\cdot'$ is computed in the corresponding context, resp. This proves the theorem by Lemma 31 of [15].

Let $\theta \in G_{\mathbf{r},\mathcal{L},\mathit{key}}$, and let Q be the most specific query in $\mathcal{L}$ returning θ for $\mathit{key}(\vec{x}) \leftarrow Q(\vec{x})$. Q exists since $\mathcal{L}$ is closed under conjunction. Let $\hat{\theta}$ be the answering substitution for Q. Then $\{\theta\}' = \{\hat{\theta}\}'$ holds. □

We conclude this section by showing that the definition of the formal context associated to **r**, $\mathcal{L}$, and *key* is the right way to model Problem 2: this problem is indeed a specific instance of Problem 1 for the context $\mathbb{K}_{\mathbf{r},\mathcal{L},key}$. As the confidence of a rule is always derived from the support of the itemsets/queries involved, it is sufficient to consider the itemsets/queries rather than the (relational) association rules. The proof of the following result is straightforward.

Theorem 2 *Let* $\mathbf{r}$ *be a datalog database,* $\mathcal{L}$ *be a set of datalog queries that all contain an atom key, and let minsupp be a (user-defined) threshold between 0 and 1. Then the set of frequent queries (in the sense of Problem 2) is equal to the set of frequent items of* $\mathbb{K}_{\mathbf{r},\mathcal{L},key}$ *(in the sense of Problem 1).*

5 Visualizations of Relational Association Rules in Iceberg Query Lattices

In [4, 25][7] and [34], we showed how the number of (classical) association rules can be reduced without any loss of information by applying FCA. While the first approach is based on free sets (i. e., the head of a rule is a free set, while the conclusion is a closed set), the second approach is based on closed sets (i. e., both head and conclusion are closed sets). In this paper, we transfer the results of [34] to relational association rules, and show how they can be used for visualizing relational association rules within iceberg query lattices.

We distinguish between two types of rules. *Exact rules* (or *implications*) hold with 100 % confidence, while the confidence of *approximate rules* is strictly lower. In the following two subsections, we discuss how the two kinds of rules can be visualized in the same diagram. Because of space restrictions, we can only provide examples here.

5.1 Visualizing the Exact Rules

The visualization of implications in the (iceberg) concept lattice is a powerful tool for communication, which has been used in FCA all along its twenty-five years history. It is straightforward to apply it to iceberg query lattices: a relational association rule $?- l_1, \ldots, l_m \rightsquigarrow l_{m+1}, \ldots, l_n.$ is an exact rule if and only if the largest node which is below all nodes labeled by the literals l_1, …, l_m of the body of the rule is also below all labels l_{m+1}, …, l_n of the head of the rule.

Example 6. Consider again Figure 3. The first rule from Example 2 holds with confidence 1/1, since the largest node below the two literals *customer*(A) and *buys*$(A,$ *wine*$)$ is the node labeled by *allen*, and this node is also below the labels *parent*(A, B) and *parent*(B, C). Similarly, the rule

$$?- \mathit{customer}(A), \mathit{parent}(A, B), \mathit{buys}(A, \mathit{pizza}) \rightsquigarrow \mathit{buys}(A, \mathit{cola})$$

holds, since the largest node below *customer*(A), *parent*(A, B), and *buys*$(A,$ *pizza*$)$ is the one labeled by *bill*, and this node is also below (more precisely: at) the label *buys*$(A,$ *cola*$)$. There is no frequent exact rule having

$$?- \mathit{customer}(A), \mathit{parent}(A, B), \mathit{buys}(B, \mathit{pizza}), \mathit{buys}(A, \mathit{cola})$$

as head, as there is no node below these literals.

[7] Similar results have been presented independently in [41].

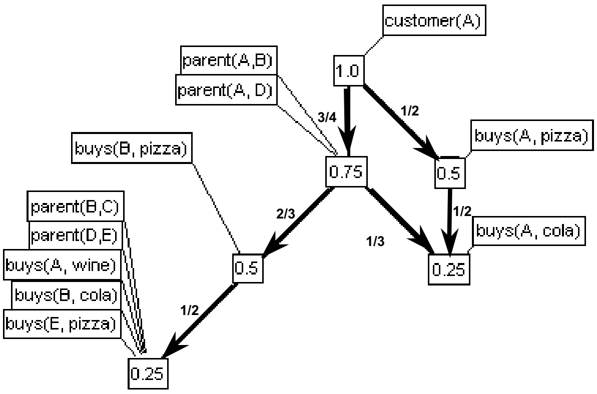


Fig. 4. All frequent relational association rules for the *key* predicate '*customer*'.

5.2 Visualizing the Approximate Rules

In [34], we show that it is sufficient to consider only rules $Q_1 \to Q_2$ where both Q_1 and Q_2 are closed and where Q_2 is an immediate specialization of Q_1. From these, all other frequent rules can be derived.

By considering only these specific rules, they all correspond to edges in the line diagram of the iceberg query lattice $\underline{\mathfrak{B}}(\mathbf{r}, \mathcal{L}, \mathit{key})$. Therefore, we can label each such edge by the confidence of the rule, and each node by the support of the corresponding query. The support of a rule is then the support labeled at the node the rule is pointing to.

Example 7. Figure 4 shows the Luxenburger basis for our running example. The arrow labeled with '2/3', for instance, stands for the rule $?- \mathit{customer}(A), \mathit{parent}(A, B) \rightsquigarrow \mathit{buys}(B, \mathit{pizza})$., which holds with confidence 2/3 and support 0.5. The third rule of Example 2 is given by the arrow labeled with '1/3'.

Rules can also be composed. For instance, the rule $?{-}\mathit{customer}(A), \mathit{parent}(A, B) \rightsquigarrow \mathit{buys}(A, \mathit{wine})$. is composed of the two rules pointing to the left. It has thus confidence $2/3 \cdot 1/2 = 1/3$ and support 0.25.

6 Conclusion

In this paper, we introduced two kinds of iceberg query lattices as different condensed representations of frequent datalog queries. We argued that by switching between them one can decide how close to analyze the relations between instances. We also demonstrated that iceberg query lattices provide a natural way to visualize relational association rules in a non-redundant way.

References

1. R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. *Proc. SIGMOD Conf.*, 1993, 207–216

2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. VLDB Conf.*, 1994, 478–499 (Expanded version in IBM Report RJ9839)
3. R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. 11th Intl. Conference on Data Engineering (ICDE '95),* Taipeh, Taiwan, March 1995, 3–14
4. Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, L. Lakhal: Mining minimal non-redundant association rules using frequent closed itemsets. In: J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.–K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, P. J. Stuckey (Eds.): *Computational Logic — CL 2000*. Proc. 1st Intl. Conf. on CL (6th Intl. Conf. on Database Systems). LNAI **1861**, Springer, Heidelberg 2000, 972–986
5. Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, L. Lakhal: Mining Frequent Patterns with Counting Inference. *SIGKDD Explorations* **2**(2), Special Issue on Scalable Algorithms, 2000, 66–75
6. R. J. Bayardo. Efficiently mining long patterns from databases. *Proc. SIGMOD Conf.*, 1998, 85–93
7. R. J. Bayardo, R. Agrawal, D. Gunopulos. Constraint-based rule mining in large, dense databases. *Proc. ICDE Conf.*, 1999, 188–197
8. S. Brin, R. Motwani, J. D. Ullman, S. Tsur: Dynamic itemset counting and implication rules for market basket data. *Proc. SIGMOD Conf.*, 1997, 255–264
9. C. Carpineto, G. Romano: GALOIS: An Order-Theoretic Approach to Conceptual Clustering. *Machine Learning*. Proc. ICML 1993, Morgan Kaufmann Prublishers 1993, 33–40
10. L. Dehaspe, H. Toivonen: Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery* **3**, 1999, 7–36
11. L. Dehaspe, H. Toivonen: Discovery of Relational Association Rules. In: S. Džeroski, N. Lavrač (Eds.): *Relational Data Mining*. Springer, Heidelberg 2001, 189–212
12. H. Dicky, C. Dony, M. Huchard, T Libourel: On automatic class insertion with overloading. *OOPSLA 1996*, 251–267
13. U. M. Fayyad, G. Piatetsky–Shapiro, P. Smyth, R. Uthurusamy (eds.): Advances in knowledge discovery and data mining. AAAI Press, Cambridge 1996
14. B. Ganter, K. Reuter: Finding all closed sets: A general approach. *Order*. Kluwer Academic Publishers, 1991, 283–290
15. B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations*. Springer, Heidelberg 1999
16. R. Godin, H. Mili, G. Mineau, R. Missaoui, A. Arfi, T. Chau: Design of class hierarchies based on concept (Galois) lattices. *TAPOS* **4**(2), 1998, 117–134
17. J.-L. Guigues, V. Duquenne: Famille minimale d'implication informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines* 24(95), 1986, 5–18
18. M. Luxenburger: Implications partielles dans un contexte. *Mathématiques, Informatique et Sciences Humaines*, 29(113), 1991, 35–55
19. M. Luxenburger: Partial implications. Part I of *Implikationen, Abhängigkeiten und Galois Abbildungen*. PhD thesis, TU Darmstadt. Shaker, Aachen 1993
20. G. Mineau, G., R. Godin: Automatic Structuring of Knowledge Bases by Conceptual Clustering. *IEEE Transactions on Knowledge and Data Engineering* **7**(5), 1995, 824–829
21. M. Missikoff, M. Scholl: An algorithm for insertion into a lattice: application to type classification. *Proc. 3rd Intl. Conf. FODO 1989*. LNCS **367**, Springer, Heidelberg 1989, 64–82
22. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Pruning closed itemset lattices for association rules. *Proc. 14ièmes Journées Bases de Données Avancées (BDA '98)*, Hammamet, Tunisie, 177–196
23. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Discovering frequent closed itemsets for association rules. *Proc. ICDT Conf.*, 1999, 398–416
24. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Efficient mining of association rules using closed itemset lattices. *Journal of Information Systems*, 24(1), 1999, 25–46

25. N. Pasquier, R. Taouil, Y. Bastide, G. Stumme, L. Lakhal: Generating a Condensed Representation for Association Rules. *J. Intelligent Information Systems (JIIS)* (accepted)
26. J. Pei, J. Han, R. Mao: CLOSET: An efficient algorithm for mining frequent closed itemsets. *Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery 2000,* 21–30
27. I. Schmitt, G. Saake: Merging inheritance hierarchies for database integration. *Proc. 3rd IFCIS Intl. Conf. on Cooperative Information Systems*, New York City, Nework, USA, August 20-22, 1998, 122–131
28. R. Srikant, R. Agrawal: Mining generalized association rules. *Proc. VLDB Conf.*, 1995, 407–419
29. R. Srikant, Q. Vu, R. Agrawal: Mining association rules with item constraints. *Proc. KDD Conf.*, 1997, 67–73
30. S. Strahringer, R. Wille: Conceptual clustering via convex-ordinal structures. In: O. Opitz, B. Lausen, R. Klar (eds.): *Information and Classification*. Springer, Berlin-Heidelberg 1993, 85–98
31. G. Stumme: *Conceptual Knowledge Discovery with Frequent Concept Lattices*. FB4-Preprint **2043**, TU Darmstadt 1999
32. G. Stumme: Off to New Shores — Conceptual Knowledge Discovery and Processing. *Intl. J. Human–Comuter Studies (IJHCS)* **59**(3), September 2003, 287–325
33. G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal: Fast Computation of Concept Lattices Using Data Mining Techniques. *Proc. 7th Intl. Workshop on Knowledge Representation Meets Databases*, Berlin, 21–22. August 2000. CEUR-Workshop Proceeding. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/
34. G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal: Intelligent Structuring and Reducing of Association Rules with Formal Concept Analysis. In: F. Baader. G. Brewker, T. Eiter (Eds.): *KI 2001: Advances in Artificial Intelligence*. Proc. KI 2001. LNAI **2174**, Springer, Heidelberg 2001, 335–350
35. G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal: Computing Iceberg Concept Lattices with Titanic. *J. on Knowledge and Data Engineering (KDE)* **42**(2), 2002, 189–222
36. K. Waiyamai, R. Taouil, L. Lakhal: Towards an object database approach for managing concept lattices. *Proc. 16th Intl. Conf. on Conceptual Modeling,* LNCS **1331**, Springer, Heidelberg 1997, 299–312
37. R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.). *Ordered sets*. Reidel, Dordrecht–Boston 1982, 445–470
38. A. Yahia, L. Lakhal, J. P. Bordat, R. Cicchetti: iO2: An algorithmic method for building inheritance graphs in object database design. *Proc. 15th Intl. Conf. on Conceptual Modeling*. LNCS **1157**, Springer, Heidelberg 1996, 422–437
39. M. J. Zaki, M. Ogihara: Theoretical Foundations of Association Rules, *3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD),* Seattle, WA, June 1998, 7:1–7:8
40. M. J. Zaki, C.–J. Hsiao: ChARM: An efficient algorithm for closed association rule mining. Technical Report 99–10, Computer Science Dept., Rensselaer Polytechnic Institute, October 1999
41. M. J. Zaki: Generating non-redundant association rules. *Proc. KDD 2000*. 34–43