


Optimization of Association Rules Extraction Through Exploitation of Context Dependent

View metadata, citation and similar papers at core.ac.uk

brought to you by  CORE

provided by Institutional Research Information System University of...

Arianna Gallo, Roberto Esposito, Rosa Meo, and Marco Botta

Dipartimento di Informatica, Università di Torino, Italy
{gallo, esposito, meo, botta}@di.unito.it

Abstract. In recent years, the KDD process has been advocated to be an iterative and interactive process. It is seldom the case that a user is able to answer immediately with a single query all his questions on data. On the contrary, the workflow of the typical user consists in several steps, in which he/she iteratively refines the extracted knowledge by inspecting previous results and posing new queries. Given this view of the KDD process, it becomes crucial to have KDD systems that are able to exploit past results thus minimizing computational effort. This is especially true in environments in which the system knowledge base is the result of many discoveries on data made separately by the collaborative effort of different users. In this paper, we consider the problem of mining frequent association rules from database relations. We model a general, constraint-based, mining language for this task and study its properties w.r.t. the problem of re-using past results. In particular, we individuate two class of query constraints, namely “item dependent” and “context dependent” ones, and show that the latter are more difficult than the former ones. Then, we propose two newly developed algorithms which allow the exploitation of past results in the two cases. Finally, we show that the approach is both effective and viable by experimenting on some datasets.

1 Introduction

The problem of mining association rules and, more generally, that of extracting frequent sets from large databases has been widely investigated in the last decade [1,2,3,4,5,6,7]. These researches addressed two major issues: on one hand, performance and efficiency of the extraction algorithms; on the other hand, the exploitation of user preferences about the patterns to be extracted, expressed in terms of constraints.

Constraints are widely exploited also in data mining languages, such as in [8,9,10,4,7] where the user specifies in each data mining query, not only the constraints that the items must satisfy, but also different criteria to create groups of tuples from which itemsets will be extracted. Constraint-based mining languages are also the main key factor of inductive databases [11], proposed in order to leverage decision support systems. In inductive databases, the user explores

the domain of a mining problem submitting to the system many mining queries in sequence, in which subsequent queries are very often a refinement of previous ones. This constitutes for the system a huge computational workload and becomes a problem even more severe considering that these queries are typically instances of iceberg queries [12]. In such systems the intelligent exploitation of the queries previously submitted by the user becomes the key factor for a successful exploration of the problem search space [13]. Analogously, in inductive databases, it makes sense to try to exploit the effort already done by the DBMS in order to speed up the execution time of new queries. Furthermore, we suppose that the mining engine works in an environment similar to a data warehouse, in which database content updates occur rarely and in known periods of time. Thus, previous results are up to date and can be usefully exploited to speed up the execution of current queries.

In this paper, we propose and evaluate an “incremental” approach to mining that exploits the results of previous queries in order to reduce the response time to new queries. Of course, we suppose that the system relies on an optimizer who is entitled to recognize query equivalence and query containment relationships. [14] describes a prototype of such an optimizer and shows that its execution time is negligible (in the order of milliseconds).

We notice that several “incremental” algorithms have been developed in the data mining area [15,16], but they address a different issue: how to efficiently revise the result set of a mining query when the database source relations get updated with new data.

In all the previous works in constraint-based mining, a somewhat implicit assumption has always been made: properties on which users define constraints are functionally dependent on the item to be extracted, i.e., the property is either always true or always false for all the occurrences of a certain item in the database. In this case, it is possible to establish the satisfaction of the constraint considering only the properties of the item itself, that is, separately from the context of the database in which the item is found (e.g., typically the database transaction). In [14], we characterized the constraints on attributes that are functionally dependent on the items extracted and called them *item dependent* (ID). The exploitation of these constraints proves to be extremely useful for incremental algorithms.

In [14] another class of constraints, namely *context dependent* (CD), was introduced as well. CD constraints proved to be very difficult to be dealt with because, even when they hold within a transaction for a particular itemset, they do not necessarily hold for the same itemset but within another transaction.

Unfortunately, most of the state of the art algorithms [3,17,18], assume precisely that constraints do or do not hold for a given itemset database wide.

2 Preliminary Definitions and Notation

Let us consider a database instance D and let T be a database relation having the schema $TS = \{A_1, A_2, \dots, A_n\}$. A given set of functional dependencies Σ over

the attribute domains $dom(A_i)$, $i = 1..n$ is assumed to be known. We denote with Σ_{A_i} the set of attributes that are in the RHS of the functional dependencies with A_i as LHS.

For the sake of exemplification, we consider a fixed instance of the application domain. In particular, we will refer to a market basket analysis application in which T is a **Purchase** relation that contains data about customer purchases. In this context, TS is given by $\{\mathbf{tr}, \mathbf{date}, \mathbf{customer}, \mathbf{product}, \mathbf{category}, \mathbf{brand}, \mathbf{price}, \mathbf{qty}, \mathbf{discount}\}$, where: \mathbf{tr} is the purchase transaction identifier and the meaning of the other columns is the usual one for this kind of application. The Σ relation is $\{\mathbf{product} \rightarrow \mathbf{price}, \mathbf{product} \rightarrow \mathbf{category}, \mathbf{product} \rightarrow \mathbf{brand}, \{\mathbf{tr}, \mathbf{product}\} \rightarrow \mathbf{qty}, \mathbf{tr} \rightarrow \mathbf{date}, \mathbf{tr} \rightarrow \mathbf{customer}, \{\mathbf{tr}, \mathbf{product}\} \rightarrow \mathbf{discount}\}$. $\Sigma_{product}$, the set of attributes whose values depend on product is $\{\mathbf{price}, \mathbf{category}, \mathbf{brand}\}$.

In writing a mining query, the user must specify, among the others, the following parameters:

- The *item attributes*, a set of attributes whose values constitute an item, i.e., an element of an itemset. In the language it is allowed to specify possibly different sets of attributes, one for the antecedent of association rules (body), and one for the consequent (head).
- The *grouping attributes* needed in order to decide how tuples are grouped for the formation of each itemset. This grouping, for the sake of generality and expressiveness of the language, can be decided differently in each query according to the purposes of the analysis.
- The *mining constraints* specify how to decide whether an association rule meets the user needs. Since we want to allow different constraints on the body and on the head of the association rules, we admit a distinct constraint expressions for each part of the rule.
- An expression over a number of *statistical measures* used to reduce the size of the result set and to increase the relevance of the results. This evaluation measures are evaluated only on the occurrences of the itemsets that satisfy the mining constraints.

By summarizing, a mining query may be described as

$$Q = (T, G, I_B, I_H, \Gamma_B, \Gamma_H, \Xi)$$

where: T is the database table; G is the set of grouping attributes; I_B and I_H are the set of item attributes respectively for the body and the head of association rules; Γ_B and Γ_H are boolean expressions of atomic predicates specifying constraints for the body and for the head of association rules; and Ξ is an expression on some statistical measures used for the evaluation of each rule.

We define an atomic predicate to be an expression in the form:

$$A_i \theta v_{A_i}$$

where θ is a relational operator such as $<$, \leq , $=$, $>$, \geq , \neq , and v_{A_i} is a value from the domain of attribute A_i . Ξ is defined to be a boolean expression in which each term has the form

$$\xi\theta v$$

where ξ is a statistical measure for the itemset evaluation, v is a real value, and θ is defined as above.

Examples of ξ are **support count** and **frequency**. The support count is the counting of the distinct groups containing the itemset. The itemset frequency is computed as the ratio between the itemset support count and the total number of database groups.

A mining engine, takes a query Q_i defined on an input relation T and generates a result set R_i .

Example 1. *The query*

$$Q=(Purchase, \{tr\}, \{product\}, \{product\}, \\ category='clothes', category='clothes' \text{ and } discount \geq 10\%, \\ support\ count \geq 20 \text{ AND } confidence \geq 0.5)$$

over the Purchase relation (first parameter) extracts rules formed by products in the body (third parameter) associated to products in the head (fourth parameter), where all the products in the rule have been sold in the same transaction (second parameter). Moreover, each product in the body must be of type “clothes” (fifth parameter) and be associated with discounted clothes (sixth parameter). Finally, the support count of the returned rules must be at least 20 and the confidence of the rules at least 0.5.

An item dependent constraint is a predicate on an attribute A_i which lies in the dependency set of the schema of the rules (here denoted as $I_{BH} = I_B \cup I_H$, i.e. the union of the schema of the body and of the head), i.e., $\Sigma_{I_{BH}}$. As a consequence, being A_i in the dependency set of I_{BH} , its value can be determined directly (or indirectly, i.e., transitively) from the value of the association rules. In other words, the verification of this kind of constraint depends on the elements of the rule itself and not on other information stored in the transaction that make up the “context” of the rule. For instance, if we extract association rules on the values of the products frequently sold together in transactions, the category of the products does not depend on the transactions, but only on the products themselves. As explained in [14] (Lemma 1), an itemset satisfies an ID constraint either in all the instances of the database or in none of them.

On the contrary, the verification of a context dependent constraint depends on the contextual information that accompany the rules elements in the database. For instance, in our running example, the quantity of a product sold in a particular transaction depends on the product and on the transaction together. Therefore, a predicate on quantity is said to be a context dependent constraint and, in fact, its satisfaction changes depending on the particular transaction (the context). This implies that the support count might take any value ranging from 0 to the number of occurrences of that itemset in the database. In other words, when context dependent constraints are involved, we are obliged to evaluate the constraints on the fact table, where the contextual information can be retrieved.

3 An Incremental Algorithm for Item Dependent Constraints

In a previous work [14], we showed that in case a query contains only ID constraints, then we can obtain the result of a newly posed query Q_2 by means of set operations (unions and intersections) on the results of previously executed queries. We qualify this approach to itemset mining as *incremental* because instead of computing the itemsets from scratch it starts from a set of previous results. In this paper, we are interested, in particular, to study the situation of query containment, that is, when query Q_2 has a more restrictive set of constraints with respect to previous queries. In this case, it suffices to identify those rules in the previous results which satisfy the new constraints and report them all. We call this simple algorithm the ID incremental algorithm.

4 An Incremental Algorithm for Context Dependent Constraints

In this section we propose a new incremental algorithm, aiming at deriving the result of a new mining query Q_2 starting from a previous result R_1 . This algorithm is able to deal with context dependent constraints, which, at the best of our knowledge, have not been tackled yet by any previous data mining algorithm [2,19,3,20,21,6].

Here we give a brief account of the algorithm behavior, describing it in greater details in the forthcoming sections. Initially, the algorithm reads rules from R_1 and builds a data structure to keep track of them. We call this structure the *BHF* (Body-Head Forest) (see Section 4.1). Then, the algorithm considers the items which satisfy the mining constraints in each group, and uses this information to update the BHF accordingly.

4.1 Description and Construction of the BHF

A BHF is a forest containing a distinguished tree (the body tree) and a number of other trees (head trees). The body tree is intended to contain the itemsets which are candidates for being in the body part of the rules. An important property of body trees is that an itemset B is represented as a single path in the tree and vice versa. The end of each path in the tree is associated to a head tree and to a (body) support counter.

The head tree rooted at the ending node of the path corresponding to itemset B is meant to keep track of those itemsets H that can possibly be used to form a rule $B \rightarrow H$. A head tree is similar to a body tree with the notable exception that there is no link pointing to further heads. A path in a head tree corresponds to an itemset H and is associated to a counter which stores the support of the rule.

Figure 1 gives a schematic representation of a BHF.

In the following we will make use of the following notation: given a node n belonging to a body tree or to a head tree, we denote with $n.child(i)$ the

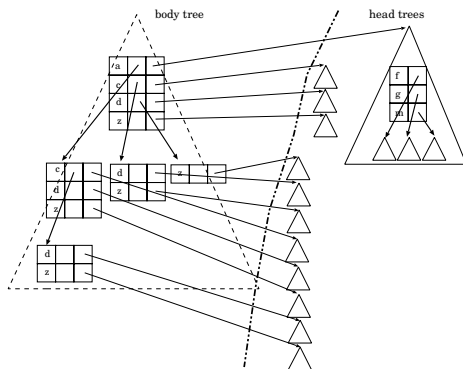


Fig. 1. Example of BHF

body (respectively the head) tree rooted in the node n in correspondence of the item i . For instance, in the root node of the BHF reported in Figure 1, there are four items, and three non-empty children; $root.child(a)$ denotes the body node containing the items c, d , and z . In a similar way we denote the head tree corresponding to a particular item i in a node n using the notation $n.head(i)$. We also assume that itemsets are sorted in an unspecified but fixed order. We denote with $I[k]$ the k -th element of the itemset I w.r.t. this ordering. Finally, in many places we adopt the standard notation used for sets in order to deal with BHF nodes. For instance, we write $i \in n$ in order to specify that item i is present in node n . Procedure `insertRule` describes how a rule is inserted in the

Procedure `insertRule`

Data : `root` : the BHF root node
 $B \rightarrow H$: the rule to be inserted
`headTree` \leftarrow `insertBody`(`root`, B , 1) ;
`insertHead`(`headTree`, H , 1);

BHF structure. The algorithm consists in two steps. In the first one the body of the rule is inserted in the body tree (see Function `insertBody`). In the second one the head is inserted and attached to the path found by the former function call (see Procedure `insertHead`). We notice that the hierarchical structure of the BHF describes a compressed version of a rule set. In fact, two rules $B_1 \rightarrow H_1$ and $B_2 \rightarrow H_2$ share a sub path in the body tree provided that B_1 and B_2 have a common prefix. Analogously they share a sub path in a head tree provided that $B_1 \equiv B_2$ and H_1 and H_2 have a common prefix.

4.2 Description of the Incremental Algorithm

Here, we assume that a BHF has been built out of the previous result set R_1 . We explain how the counters in the structure are updated in order to reflect the support counters implied by Q_2 .

Function insertBody

```

Data :  $n$  : a BHF node;  $B$  : an itemset;  $k$  : an integer
if  $B[k] \notin n$  then
   $n \leftarrow n \cup B[k]$ 
end
if  $k < \text{size}(B)$  then
  insertBody( $n.\text{child}(B[k]), B, k + 1$ )
else
  return  $n.\text{head}(B[k])$ 
end

```

Procedure insertHead

```

Data :  $n$  : a BHF node;  $H$  : an itemset;  $k$  : an integer
if  $H[k] \notin n$  then
   $n \leftarrow n \cup H[k]$ 
end
if  $k < \text{size}(H)$  then
  insertHead( $n.\text{child}(H[k]), H, k + 1$ )
end

```

In the following we will denote with:

- $T'_b[g] \equiv \{i \mid (g, i) \in T'_b\}$ and with $T'_h[g] \equiv \{i \mid (g, i) \in T'_h\}$ the set of items in group g that satisfy the body constraints and the head constraints.
- $\Pi_{\text{GID}}(T'_b) \equiv \{g \mid (g, i) \in T'_b\}$ and with $\Pi_{\text{GID}}(T'_h) \equiv \{g \mid (g, i) \in T'_h\}$ the set of GIDs in T'_b and in T'_h .
- τ the support threshold chosen by the user
- $B(r)$ the body of rule r and with $H(r)$ the head of rule r

For the sake of readability, we reported in Algorithm 4 a simplified version of the incremental algorithm which has the advantage of making its intended behavior clear. In fact, the implemented version greatly improves on the reported one by exploiting the hierarchical structure of BHF and the fact that there exists a single path in BHF for each B and at most $|B|$ paths for H . This allows the function to require $O(|B||H|)$ time in the worst case.

5 Results

The two incremental algorithms presented in this paper have been assessed on a database instance, describing retail data, generated semi-automatically. We generated a first approximation of the fact table (**purchases**) using the synthetic data generation program described in [22]. This data generation program has been run using parameters $|T| = 25$, $|I| = 10$, $N = 1000$, $|\mathcal{D}| = 10,000$, i.e., the average transaction size is 25, the average size of potentially large itemsets is 10, the number of distinct items is 1000 and the total number of transactions

Algorithm 4: Context Dependent (CD) incremental algorithm

```

Data : BHF; pointers to  $T'_b, T'_h$ 
Result :  $R_2$ 
for all  $GID\ g \in \Pi_{GID}(T'_b)$  do
  | incrRuleSupp(BHF,  $T'_b[g], T'_h[g]$  )
end
for all rule  $r \in BHF$  do
  | if  $\Xi(r)$  then
  | |  $R_2 \leftarrow R_2 \cup r$ 
  | end
end

```

Procedure incrRuleSupp

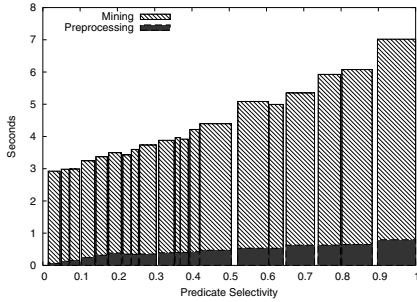
```

Data : BHF;  $I_B$ : items in current transaction satisfying  $\Gamma_B$ ;  $I_A$ : items in
  current transaction satisfying  $\Gamma_A$ 
Result : It updates the support counters in the BHF
for all  $r \in BHF$  do
  | if  $B(r) \subseteq T'_b[g]$  then
  | | support(B(r))++;
  | | if  $H(r) \subseteq T'_h[g]$  then
  | | | support(r)++;
  | | end
  | end
end

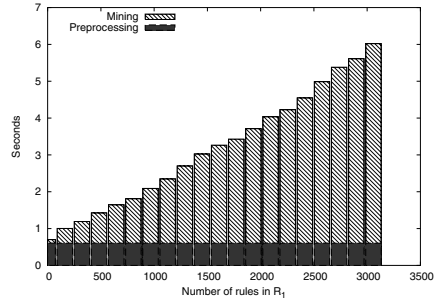
```

is 10.000. Then, we updated this initial table adding some more attributes, constituting the description (and the contextual information) on sales: some item dependent features (such as category of product and price) and some context dependent features (such as discount and quantity of sales). We generated these attributes values randomly, using a uniform distribution defined on the respective domains.

We note here how a single fact table suffices for the objectives of our experimentation. In fact, the important parameters from the viewpoint of the performance study of incremental algorithms are the selectivity of the mining constraints (which determine the volume of data to be processed from the given database instance) and the size of the previous result set. Figure 2(a) reports the performances of the item dependent incremental algorithm (ID) as the selectivity of the mining constraints changes. We experimented different constraints on the item dependent attributes, letting the constraints selectivity vary from 0% to 100% of the total number of items. In Figure 2(a) we sampled twenty points. Figure 2(b) tests the same algorithm, but it lets vary the number of rules in the previous result set. Again we sampled twenty points (in the range 0...3220). The two figures report the total amount of time needed by the algorithm to complete, subdividing it in the preprocessing time (spent in querying the database

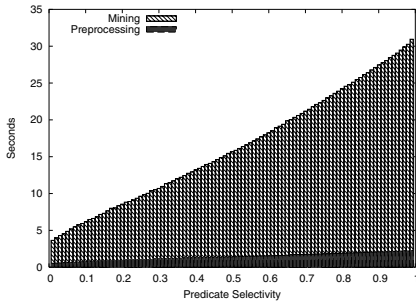


(a) Execution time vs constraint selectivity

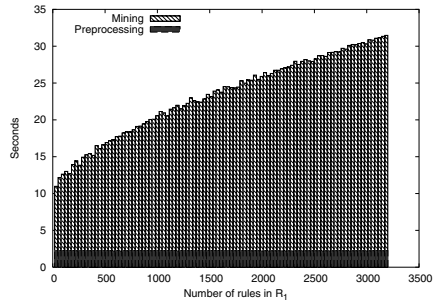


(b) Execution time vs volume of previous result

Fig. 2. Empirical evaluation of the item dependent (ID) incremental algorithm



(a) CD vs selectivity



(b) CD vs nrules

Fig. 3. Empirical evaluation of the CD incremental algorithm

to retrieve and store in main memory the items that satisfy the constraints), and the core mining time (needed by the algorithm to read the previous result set and to filter out those rules that do not satisfy the constraints any more).

Figures 3(a) and 3(b) report the results on performances of the context dependent (CD) algorithm. The figures report again the total execution time, specifying how much time was spent for preprocessing and for the core mining task.

A couple of points are worth noting. The execution times of both algorithms increase almost linearly with the increase of the two parameters (constraints selectivity and previous results), but as it was expected the item dependent incremental algorithm runs much faster than its counterpart. In addition, evidence from another set of experiments (not reported here for space reasons) shows that the algorithms highly improve on our baseline miner algorithm that solves the problem of constraint-based mining in its most general version. Due to lack of space we do not describe this algorithm in details. In brief, the algorithm evaluates the mining constraints directly on the fact table, and supports the following features: fully support constraints (SQL-like predicates) on bodies and heads

and correctly solves context dependent queries. This is an Apriori-like algorithm that keeps track of the groups in which each item satisfies the mining constraints. It adopts a work-flow similar to Partition [23] but uses BHF as data structure. Moreover, in order to support CD constraints the generalized algorithm builds a temporary source table which is usually much bigger than the original one. This algorithm takes about 700 seconds in the average case to build the complete result set which is more than an order magnitude higher than the time spent by the CD incremental algorithm on the same task.

We also ran a version of FPGrowth [24] on the same dataset, but using no constraints at all (since, at the best of our knowledge, CD constraints are not supported by any algorithm proposed so far in the literature). It takes, on average, about 21 seconds to complete. This is three times faster than the worst performance of the CD incremental algorithm and three times slower than the ID incremental one. We agree that this is only a very rough comparison and that things can change substantially accordingly to the size of the previous result set as well as with the support parameter given to FPGrowth. However, it is interesting to notice how suggests that it may be possible to combine an efficient algorithm like FPGrowth to build an initial result set, and an incremental one (supporting CD constraints) in order to solve constraint-based queries.

This further motivates the interest in incremental algorithms, since it is then possible to obtain an execution time that is still much smaller than the one required by the generalized constraint-based algorithm, and still allowing a general class of constraints to be supported.

6 Conclusions

In this paper we proposed a novel “incremental” approach to constraint-based mining which makes use of the information contained in previous results to answer new queries. The beneficial factors of the approach are that it uses both: the previous results and the mining constraints, in order to reduce the itemsets search space.

We presented two incremental algorithms. The first one deals with item dependent constraints, while the second one with context dependent ones. We note how the latter kind of constraints has been identified only recently and that there is very little support for them in current mining algorithms. However, the difficulty to solve mining queries with context dependent constraints can be partially overcome by combining the “traditional” algorithms proposed so far in the literature, and the context dependent incremental algorithm proposed in this paper.

In Section 5, we evaluated the incremental algorithms on a pretty large dataset. The results show that the approach reduces drastically the overall execution time. We believe the improvement to be absolutely necessary in many practical data mining applications, in data warehouses and inductive database systems.

An interesting direction for future research is the integration of condensed representations (which have been heavily studied in recent years) with the incremental techniques presented here. In fact, it would be desirable to take advantage

of both: the improved readability of condensed patterns, and the speeds improvements of incremental algorithms. Moreover, to store condensed patterns means that incremental algorithms need to deal with smaller result sets. Then, it may be possible to obtain an even faster processing of incremental queries.

To this regard, the main problem to be faced is the interaction between novel (more restrictive) constraints and condensed patterns. In fact, even though the representative of a condensed set may not satisfy the new constraint, this not necessarily hold for all the elements of the condensed set. This means that, when such a situation occurs, it is not possible to simply remove the representative from the previous result set. On the contrary, the old dataset must undergo careful rewriting. While we expect that efficient solutions to the problem could be found, the extension of our algorithms to deal with these issues is not straightforward and deserve additional work.

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., eds.: Knowledge Discovery in Databases. Volume 2. AAAI/MIT Press, Santiago, Chile (1995)
2. Srikant, R., Vu, Q., Agrawal, R.: Mining association rules with item constraints. In: Proceedings of 1997 ACM KDD. (1997) 67–73
3. Ng, R.T., Lakshmanan, L.V.S., Han, J., Pang, A.: Exploratory mining and pruning optimizations of constrained associations rules. In: Proc. of 1998 ACM SIGMOD Int. Conf. Management of Data. (1998) 13–24
4. Tsur, D., Ullman, J.D., Abiteboul, S., Clifton, C., Motwani, R., Nestorov, S., Rosenthal, A.: Query flocks: A generalization of association-rule mining. In: Proceedings of 1998 ACM SIGMOD Int. Conf. Management of Data. (1998)
5. Chaudhuri, S., Narasayya, V., Sarawagi, S.: Efficient evaluation of queries with mining predicates. In: Proc. of the 18th Int'l Conference on Data Engineering (ICDE), San Jose, USA (2002)
6. Perng, C.S., Wang, H., Ma, S., Hellerstein, J.L.: Discovery in multi-attribute data with user-defined constraints. ACM SIGKDD Explorations **4** (2002) 56–64
7. Wang, H., Zaniolo, C.: User defined aggregates for logical data languages. In: Proc. of DDLP. (1998) 85–97
8. Imielinski, T., Virmani, A., Abdoulghani, A.: Datamine: Application programming interface and query language for database mining. KDD-96 (1996) 256–260
9. Meo, R., Psaila, G., Ceri, S.: A new SQL-like operator for mining association rules. In: Proceedings of the 22st VLDB Conference, Bombay, India (1996)
10. Han, J., Fu, Y., Wang, W., Koperski, K., Zaiane, O.: DMQL: A data mining query language for relational databases. In Proc. of SIGMOD-96 Workshop on Research Issues on Data Mining and Knowledge Discovery (1996)
11. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Communications of the ACM **39** (1996) 58–64
12. Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R., Ullman, J.: Computing iceberg queries efficiently. In: Proceeding of VLDB '98. (1998)
13. Sarawagi, S.: User-adaptive exploration of multidimensional data. In: Proc. of the 26th Int'l Conf. on Very Large Databases (VLDB), Cairo, Egypt (2000) 307–316

14. Meo, R., Botta, M., Esposito, R.: Query rewriting in itemset mining. In: Proceedings of the 6th International Conference On Flexible Query Answering Systems. LNAI (to appear), Springer (2004)
15. Cheung, D.W., Han, J., Ng, V.T., Wong, C.Y.: Maintenance of discovered association rules in large databases: an incremental updating technique. In: ICDE96 12th Int'l Conf. on Data Engineering, New Orleans, Louisiana, USA (1996)
16. Labio, W., Yang, J., Cui, Y., Garcia-Molina, H., Widom, J.: Performance issues in incremental warehouse maintenance. In: Proceedings of Twenty-Sixth International Conference on Very Large Data Bases. (2000) 461–472
17. Leung, C.K.S., Lakshmanan, L.V.S., Ng, R.T.: Exploiting succinct constraints using fp-trees. *ACM SIGKDD Explorations* 4 (2002) 40–49
18. Bucila, C., Gehrke, J., Kifer, D., White, W.M.: Dualminer: a dual-pruning algorithm for itemsets with constraints. In: Proc. of 2002 ACM KDD. (2002) 42–51
19. Bayardo, R., Agrawal, R., Gunopulos, D.: Constraint-based rule mining in large, dense databases. In: Proc. of the 15th Int'l Conf. on Data Engineering. (1999)
20. Lakshmanan, L.V.S., Ng, R., Han, J., Pang, A.: Optimization of constrained frequent set queries with 2-variable constraints. In: Proceedings of 1999 ACM SIGMOD Int. Conf. Management of Data. (1999) 157–168
21. Raedt, L.D.: A perspective on inductive databases. *ACM SIGKDD Explorations* 4 (2002) 69–77
22. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th VLDB Conference, Santiago, Chile (1994)
23. Savasere, A., Omiecinski, E., Navathe, S.: An efficient algorithm for mining association rules in large databases. In: Proc. of the 21st VLDB Conf. (1995)
24. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proc. of ACM SIGMOD 2000, Dallas, TX, USA (2000)