INCREMENTAL EXTRACTION OF ASSOCIATION RULES IN APPLICATIVE DOMAINS

Arianna Gallo Roberto Esposito Rosa Meo Marco Botta *

August 29, 2006

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, in order to reduce the computational effort, it becomes crucial to have KDD systems that are able to exploit past results. This is expecially 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 first model a general, constraint-based, mining language for this task. Then, we propose an algorithm that answers such queries re-using past results. In particular, this solution is effective for a new class of constraints, named *context dependent*, which are more difficult than the traditionally studied *item dependent* constraints. Nevertheless, we show that some typical queries of important application domains - such as market stock trading, analysis of web logs and gene microarrays in bioinformatics - have context dependent constraints. We show with a set of experiments in these application domains that the proposed solution with an incremental approach is both effective and viable.

1 Introduction

The problem of mining association rules and frequent sets from large databases has been widely investigated so far [1, 2, 3, 4, 5, 6, 7]. On one hand, researchers increased the performance of the extraction algorithms, on the other hand, exploited user preferences about the patterns, expressed in terms of constraints.

Constraints are widely exploited also in data mining languages, such as in [8, 9, 10, 4, 7] and in inductive databases [11], a new generation of database systems that provide support to the

*Dipartimento di Informatica, Università di Torino, Italy. e-mail: {gallo,esposito,meo,botta}@di.unito.it

KDD process. In data warehouses and 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 the problem becomes even more severe considering that these queries are typically instances of iceberg queries [12].

In such systems, since the storage space is less critic than the reduction of the computational workload in the exploration of the problem search space, the result of some queries is materialized. As a consequence, the intelligent exploitation of these queries becomes the key factor [13]. Analogously, in order to speed up the execution time of new queries, it makes sense to also exploit the effort already done by the system with previous queries. In this context, 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 materialized results 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, 17, 18], 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. The exploitation of these constraints proves to be extremely useful for algorithms because 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). An example is a constraint that requires products in a set of purchases to be of a certain color, brand or type. Most of the state of the art algorithms [3, 19, 20], assume precisely that constraints are item dependent. In a previous work [14], we showed that in case a query contains only ID constraints, then we can either obtain the result of a newly posed query by means of set operations (unions and intersections) on the results of previously executed queries or simply identify those rules in the previous results which satisfy the new constraints. This is immediate since constraints can be evaluated directly on the itemsets themselves. 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.

As we will show in Section 5, in many practical application domains, such as stock market trading, weather forecast, web log mining and bioinformatics, it is often necessary to enforce on

the patterns some properties that do necessarily depend on the context in which the pattern is found [21]. In [14] this class of constraints, namely *context dependent* (CD), was introduced as well.

CD constraints proved to be very difficult to deal with because, even when they hold within a transaction for a particular itemset, they do not necessarily hold for the same itemset within another transaction. Therefore, with CD constraints, it is not possible to establish the satisfaction of the constraint separately from the context of the database. For instance, co-occurrence of a set of items in the same transaction is the most famous and used constraint of such type. With context dependent constraints, the algorithms can work in two ways: either partitioning the space of the possible patterns and performing repeated scans on the database for each partition in order to evaluate the constraint (such as Apriori [22]), or by "projection" of the database, i.e. by keeping for each pattern the context identifiers (i.e., transaction identifiers) in which it satisfies the constraints [23, 24]. Often, this is a prohibitive amount of data, given the dimensions of the problem.

In this paper we present an algorithm that is able to deal with CD constraints with an incremental solution. The incremental approach is beneficial because it allows to reduce the search space to the space of previous results. This reduction allows us to propose a third kind of solution: to keep in memory the search space but avoid both the projection of the database and the repeated scans of the database, as well. In particular, we study the situation of query containment, that is, when the current query 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: in doing this, with CD constraints, we are forced to scan the database.

This work is organised as follows. Section 2 gives the preliminary definitions and introduces the constraint-based mining query language. Section 3 describes the incremental algorithm while Section 4 and Section 5 give an account of the experimental evaluation respectively in synthetic data and in the mentioned real applicative domains. Finally, Section 6 draws the conclusions.

2 Preliminary Definitions and Notation

Let us consider a database instance D and let T be a database relation having the schema TS=$\{A_1, A_2, \ldots, A_n\}$. 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 {`tr, date, customer, product, category, brand, price, qty,discount`}, where: `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 {`product`→ `price`, `product` → `category`, `product` → `brand`,

{tr, product} → qty, tr → date, tr → customer, {tr,product} → discount}. $\Sigma_{product}$, the set of attributes whose values depend on product is {price, category, brand}.

The problem of association rule mining can be informally stated as the one of finding implications having the form $B \Rightarrow H$ occurring frequently in the transactions of the analysed database. In this context, B and H are called the body and the head of the rule. B and H are *itemsets*, i.e., collections of objects (*items*) whose precise meaning depends on the purposes of the analysis. An association rule mined from a given database instance D can be interpreted as the discovery: it is frequent that "if itemset B is present in a transaction, then itemset H also belongs to that transaction".

In this paper we allow the user to constrain the search for association rules in order to focus on more interesting items and to decrease the search complexity of the problem. The user interacts with the inductive database using a query language which we characterize in the following.

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. 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 expression 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. These 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.

An atomic predicate is an expression in the form: $\mathtt{A_i}\theta\mathtt{v_{Ai}}$ where θ is a relational operator such as $<, \leq, =, >, \geq, \neq$, and $\mathtt{v_{Ai}}$ is a value from the domain of attribute A_i. Ξ is a boolean expression in which each term has the form $\xi\theta\mathtt{v}$ 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*

$$\begin{aligned} Q = (&\boldsymbol{T}\text{=}Purchase,\ \boldsymbol{G}\text{=}\{tr\},\ I_B\text{=}\{product\},\ I_H\text{=}\{product\},\\ &\Gamma_B\text{= }(category\text{='clothes'}) \wedge (discount\geq 10\%),\\ &\Gamma_H\text{= }(category\text{='shoes'}) \wedge (discount\geq 10\%),\\ &\Xi\ \text{= }(support\ count\geq 20) \wedge (confidence\geq 0.5)) \end{aligned}$$

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" and be discounted (fifth parameter) and be associated with discounted shoes (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.*

Let us denote the schema of the rules as $I_{BH} = I_B \cup I_H$. The dependency set of the schema of the rules is denoted as $\Sigma_{\mathtt{I_{BH}}}$.

Definition 1 *An* item dependent *(ID) constraint is a predicate on an attribute $A_i \in \Sigma_{\mathtt{I_{BH}}}$.*

As a consequence of the above definition, 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 makes 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.

Definition 2 *A* context dependent *(CD) constraint is a predicate on an attribute $A_i \notin \Sigma_{\mathtt{I_{BH}}}$.*

The verification of a context dependent constraint depends on the contextual information that accompanies 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 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, 25, 3, 26, 27, 6].

The algorithm we present in this paper leverages past results so to dramatically reduce the computational burden faced by the system. In so doing, it exploits the fact that, under precise conditions (see [14]) the new result is guaranteed to be contained in a past one. Here, we do not address the problem of how to find suitable previous results (see instead [28]) and focus on the problem of deriving the new result from the past knowledge.

As already mentioned in Section 1, the beneficial features of the proposed algorithm are that it is able to scan the database only once while it can keep in memory the search space without any projection of the database because the search space is reduced to the previous result. In cases the previous result does not fit in memory - and this could happen for very dense data-sets - some solutions with a condensed representation like in [29] can be pursued.

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 *BHF* (Body-Head Forest) (see Section 3.1). Then, the algorithm considers the items which satisfy the mining constraints in each group, and uses this information to update BHF accordingly.

3.1 Description and construction of the BHF

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 the body support counter and to a head tree.

This 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 notably 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(c) 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 $n.\text{child}(i)$ the body (respectively the head) tree rooted in the node n in correspondence of the item i. In a similar way we denote the head tree of item i in node n as $n.\text{head}(i)$. For instance, in root node of body tree in the BHF reported in Figure 1(c), there are three items (a,b,c). Following the paths in the body tree rooted in a,

root.child(a), we obtain two body itemsets $\{a,b\}$ and $\{a,c\}$ from which we reach the rules $\{a,b\} \Rightarrow \{c\}$ and $\{a,c\} \Rightarrow \{b\}$. 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 set notation (for instance, we write $i \in n$ to specify that item i is present in node n). Procedure insertRule describes how a rule is inserted in the BHF

Procedure insertRule

Data : *root* : the BHF root node
$B \rightarrow H$: the rule to be inserted
Result: Inserts a new rule into the BHF structure
$headTree \leftarrow \text{insertBody}(root, B, 1)$;
$\text{insertHead}(headTree, H, 1)$;

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

Function insertBody

Data : n : a BHF node; B : an itemset; k : an integer
Result: Recursively inserts the last $|B| - k + 1$ items of B into the body subtree rooted in n. It returns the pointer to the head tree found in BHF node on which the insertion terminates.
if $B[k] \notin n$ **then**
  $n \leftarrow n \cup B[k]$
end
if $k < \text{size}(B)$ **then**
  $\text{insertBody}(n.\text{child}(B[k]), B, k+1)$
else
  return $n.\text{head}(B[k])$
end

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.

3.2 Description of the incremental algorithm

Here, we assume that a BHF has been built out of the previous result set R_1 (see, for instance, Figure 1(c)). We explain how the counters in the structure are updated in order to reflect the support counters implied by Q_2. For ease of description, we will adopt the following notation:

- $Q_2 = (T, G, I_B, I_H, \Gamma_B, \Gamma_H, \Xi)$ denotes the query to be evaluated incrementally.
- $T_b = \{< i, g >\} \equiv \Pi_{I_B,G}(\sigma_{\Gamma_B}(T))$ and $T_h = \{< i, g >\} \equiv \Pi_{I_H,G}(\sigma_{\Gamma_H}(T))$ are the items

Procedure insertHead

Data : n : a BHF node; H : an itemset; k : an integer

Result: Recursively inserts the last $|H| - k + 1$ items of H into the head subtree rooted in n.

```
if H[k] ∉ n then
    n ← n ∪ H[k]
end
if k < size(H) then
    insertHead(n.child(H[k]), H, k + 1)
end
```

and group identifiers of the fact table that satisfy constraints of Q_2 for body and head, respectively.

- $T_b^i[g] \equiv \{i \mid (i,g) \in T_b\}$ and $T_h^i[g] \equiv \{i \mid (i,g) \in T_h\}$ are the set of items in a specific group g that satisfy body and head constraints, respectively.
- $T_b^g[i] \equiv \{g \mid (i,g) \in T_b\}$ and $T_h^g[i] \equiv \{i \mid (i,g) \in T_h\}$ are the set of group identifiers in which a specific item i satisfies body and head constraints, respectively.
- τ is the support threshold chosen by the user
- $B(r)$ is the body of rule r and $H(r)$ is 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. It checks if any database group g contains a body and a head satisfying constraints (by means of T_b and T_h). For the corresponding rule, function `incrRuleSupp` increases its support count.

The implemented version greatly improves on the reported one by exploiting the hierarchical structure of BHF. In fact, the set of possible heads of B is a subset of the heads already associated to a subset of B and can be "inherited" along the hierarchical structure. Since there exists a single path in BHF for each body B and each head H, this allows the function to require $O(|B||H|)$ time in the worst case. Notice also that the algorithm correctly allows the increment of the support counter of a body even in those groups in which no head appears.

Figure 1 shows a toy example reporting both the BHF built from an initial, previous result R_1 and its pruned version, after the application of a new query constraints.

4 Experimental results

The incremental algorithm presented in this paper has been assessed both on synthetic and real data-sets. In this section we provide an extensive experimental evaluation with a synthetic data-set. In Section 5 instead we give a brief experimental comparison of the execution times of an incremental algorithm with a traditional one in some real applicative domains.

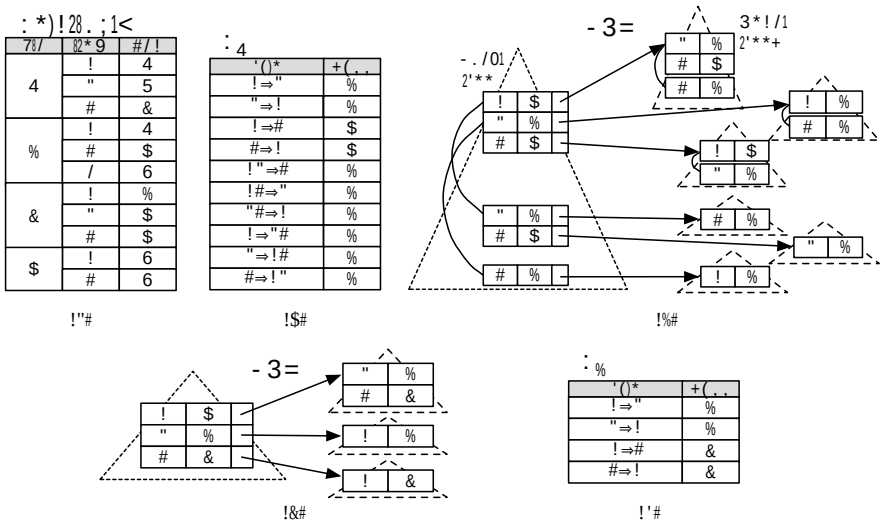


Figure 1: Example of using the BHF based incremental algorithm. (a) A toy relation T having the schema {gid,item,cda}, where: "gid" is the grouping attribute; item is the item attribute; cda (context dependent attribute) is an attribute not belonging to Σ_{item}. (b) Result set R_1 satisfying a query without constraints and support count > 1. The incremental algorithm exploits R_1 as the starting point to solve the new, more constrained, query Q_2: "mine rules having support count $>$ 1 and cda > 3". The BHF shown in (c) is built by processing R_1 and pruned to obtain (d). The new result set is reported in (e).

4.1 Synthetic dataset

The synthetic data-set describes retail data and was generated semi-automatically. We generated a first approximation of the fact table (`purchases`) using the synthetic data generation program described in [22]. It 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 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.

We report here the results on performances of the incremental algorithm with context dependent constraints. We experimented different constraints on the context dependent attributes, letting the constraints selectivity vary from 0% to 100% of the table. In Figure 2(a) we sampled one hundred points. Figure 2(b) tests the same algorithm, but it lets vary the number of rules in

Algorithm 4: Context Dependent (CD) incremental algorithm

```
Data  : BHF; pointers to T_b, T_h
Result: R_2
for all g ∈ T_b do
    incrRuleSupp(BHF,T_b[g], T_h[g] )
end
for all rule r ∈ BHF do
    if Ξ(r) then
        R_2 ← R_2 ∪ r
    end
end
```

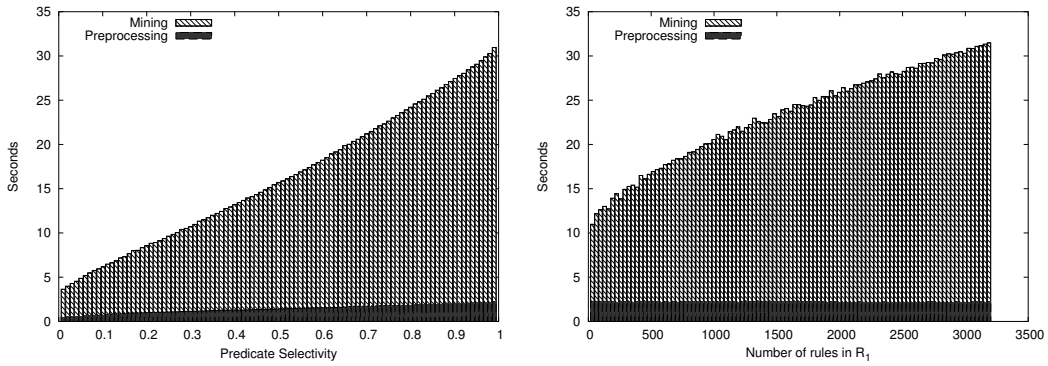
Procedure `incrRuleSupp`

```
Data  : BHF;
        i_b: set of items in current group satisfying Γ_B;
        i_h: set of items in current group satisfying Γ_H
Result: It updates the support counters in the BHF
for all r ∈ BHF such that B(r)⊆ i_b do
    support(B(r))++;
    if H(r)⊆ i_h then
        support(r)++;
    end
end
```

the previous result set. Again we sampled one hundred points. The two figures report the total amount of time needed by the algorithm to complete, subdividing it in the preprocessing time (spent in retrieving the past result and building the BHF data structure), and the core mining time (needed by the algorithm to read the fact table to filter out those rules that do not satisfy the constraints any more).

A couple of points are worth noting. The execution times of the algorithm increase almost linearly with the increase of the two parameters (constraints selectivity and previous results). 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. 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 incremental algorithm on the same task.

We also ran a version of FPGrowth [30] 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



(a) Evaluation on constraint selectivity

(b) Evaluation on # previous rules

Figure 2: Empirical evaluation of the incremental algorithm with CD constraints.

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 a version of the incremental algorithm specifically optimized for dealing with ID constraints. 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 that it may be possible to combine an efficient algorithm like FPGrowth in order 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, but allowing a general class of constraints to be supported.

5 Query examples in three application domains

In this section, we introduce three important and popular application domains: market stock trading, web log mining and analysis of gene correlations in microarray data. We show now examples of typical queries that users could submit to explore the data and extract some relevant patterns. We want to show that the incremental solution is the most immediate and natural in such an environment, expecially if the materializations of queries and the knowledge base are the result of a collaborative effort of a group of different users. In Section 5.4 we provide an account of the execution time of the reported queries with a comparison between incremental and traditional execution. This will allow us to identify the cases in which an incremental solution is the winning choice.

5.1 Analysis of Dow Jones's Candlesticks

Dow Jones 30 contains 30 stocks of companies strictly connected to production activities in USA, such as Microsoft, Intel, General Motors, etc., and still used as a meter of judgement on the evolution and wealth of the American economy.

It contains for any date of the trade exchange, the following information for any stock: *ticker* (symbol of the stock); *open* value (value of the stock quote that is used at the opening time of the trading date); *close* value (value of the stock quote that is reached at the closing time of the date); *minimum* and *maximum* quote value of the stock; *volume* (the total number of exchanged stocks of a ticker in the date). We used a data-set containing the Dow Jones index for one year (2002) that contains 7439 rows.

Technical analysis has been often used by market analyzers and is based on analysis of the time series determined by the stock quotes at certain periods of time. In technical analysis some patterns, known as *candlesticks*, are frequently used to search a signal of inversion in the stock quote trend. The ability to predict these signals can be very important for the stock market analyzers because helps investors to determine the best time to sell or buy stocks. Candlesticks have been originally proposed by Japanese market stock analyzers to study the rice market. A single candlestick represents the synthesis of the exchanged stocks occurred in one period of time for a given stock.

A candlestick is colored as follows. **Black candlestick** represents a time period in which the open value of a stock is higher than the close value. This identifies a period in which the stock lost part of its value. On the contrary, **white candlestick** represents a time period in which the open value is lower than the close value and this identifies a period in which the stock gained value.

First query. Let us consider the following query that finds the co-occurrences of tickers that frequently have black candlesticks in the same date. This query is relevant to discover sets of tickers with the same (negative) behaviour in time.

$$Q = (\boldsymbol{T}= \textit{dj30quotes},\ \boldsymbol{G}= \{\textit{date}\},\ I_B= \{\textit{ticker}\},\ I_H= \{\textit{ticker}\},$$
$$\Gamma_B= \textit{(candlestick-type="black")},\ \Gamma_H= \textit{(candlestick-type="black")},$$
$$\Xi=\textit{support} \geq 0.015 \wedge \textit{confidence} \geq 0.45)$$

In this mining query, support and confidence are based on the dates. A rule {`t1, t2`}⇒ {`t3, t4`} with support=0.015 and confidence=0.45 means that stocks {`t1, t2, t3, t4`} presented a black candlestick in 1.5% of the total dates. Moreover, in the 45% of the dates in which {`t1, t2`} have a black candlestick also {`t3, t4`} have a black candlestick.

This query contains constraints that select 51% of the table. It produces around 13 thousands rules. (In this result, {Microsoft}⇒ {Intel} is among the rules with the highest confidence).

Second query. The following query finds the co-occurrences of tickers that frequently have black candlesticks in the same date but with a moderate percentage variation in quote (< 5%

and computed with respect to the annual variation). This query is relevant to discover sets of tickers with a similar behavior in time, negative but moderate.

$$Q_{moderate} = (\boldsymbol{T}= dj30quotes,\ \boldsymbol{G}= \{date\},\ I_B = \{ticker\},\ I_H = \{ticker\},$$
$$\Gamma_B = (candlestick\text{-}type=\text{"black"} \wedge varp<5),\ \Gamma_H = (candlestick\text{-}type=\text{"black"} \wedge varp<5),$$
$$\Xi = support \geq 0.01 \wedge confidence \geq 0.40)$$

This query contains constraints that select 51% of the table. With respect to the first query, you can notice that the second query has some additional constraints; the system could usefully exploit this situation with an incremental execution starting its computation from the result of the first query. However, the second query still produces a volume of rules similar to previous result.

5.2 Web Log Analysis

Web navigation patterns show strong regularities which, if discovered, can give clues about users' interests and/or habits. Typically, Web servers store these patterns in a standard log file. This information is very valuable and can be used for cross-marketing, for improving the performance of the web application, for personalization of the user interface, etc.

The web log has the form of a relational table (*WebLog*) that typically contains at least the following attributes: *RequestID* (the identifier of the request); *IPcaller* (IP address from which the request is originated); *Date* (of the request); *TS* (time stamp); *Operation* (GET, POST, etc); *Page URL* (of the requested page); *Protocol* (transfer protocol, such as HTTP version); *Return Code* (returned by the Web server); *Dimension* (in Bytes transfered); *Session* (identifier of the user crawling session).

We used the log of the web application site of a University Department. It contains almost 6 thousand rows (one per page request) spanning a time period of 3 months with an average number of pages requested per session equal to 12.

First query. The following query discovers sets of pages that were all requested in the same crawling sessions by a high number of users. In other terms, it searches for pages frequently visited together.

$$Q = (\boldsymbol{T}= WebLog,\ \boldsymbol{G}= \{Session\},\ I_B = \{PageUrl\},\ I_H = \{PageUrl\},$$
$$\Xi = (support \geq 0.01) \wedge (confidence \geq 0.01))$$

In this query, support and confidence are based on the user visits. For instance, the rule {`page1, page2`}⇒ {`page3, page4`} with support=0.01 and confidence=0.4 means that pages {`page 1, page2, page3, page4`} were visited in 1% of the total user sessions and 1% of the user sessions that visit pages {`page1, page2`} visit also {`page3, page4`}.

This query produces around 478800 rules.

Second query. Let us analyze the pages visited during office hours.

$$Q_{office-hours}=(\boldsymbol{T}=\ WebLog,\ \boldsymbol{G}=\{Session\},\ I_B=\{PageUrl\},\ I_H=\{PageUrl\},$$
$$\Gamma_B=(hour>=9 \wedge hour<=19),\ \Gamma_H=\ (hour>=9 \wedge hour<=19),$$
$$\Xi=\ (support \geq 0.01) \wedge\ (confidence \geq 0.01))$$

Constraints in this query select 57% of the table. Also in this case, the successive result can be obtained by refinement of the previous one and produces around 184700 rules.

5.3 Genes Classification by Micro-Array Experiments

Micro-arrays are a relatively new technology that allow massively parallel measurements of relative gene expressions between biological samples, with applications in all areas of the life and biomedical sciences. Micro-array data has generated intense interest in mathematics and statistics in recent years.

We consider information on a single micro-array experiment containing data on several samples of biological tissue tied to correspondent probes on a silicon chip. Each sample is treated in various ways and under different experimental conditions; these can determine the over-expression or the under-expression of a set of genes if they are, respectively, active or inactive in the experimental conditions.

A micro-array typically contains hundreds of samples, and for each sample, several thousands of genes are measured. Thus, input relation *MicroArray* contains the following information: the identifier *SampleID* of the sample of biological tissue tied to a probe on the microchip; the identifier *GeneId* of the gene measured in the sample; the identifier *TreatmentConditionId* of the experimental conditions under which the sample has been treated; the measured value *FoldChange*. The fold change can be calculated in relation to a selected baseline array, or in relation to the minimal expression value of the gene (also determined by the user). If the fold change is higher than a threshold T_2, the genes are over-expressed; if it is lower than another threshold T_1, genes are under-expressed.

The data-set contains almost 500 thousands rows (that are the genes) in 7615 different biological situations.

First query. We want to discover sets of genes similarly expressed (for instance, under-expressed) in the same experimental conditions:

$$Q\ =\ (\boldsymbol{T}=\ MicroArray,\ \boldsymbol{G}=\{SampleId,\ TreatmentConditionId\},$$
$$I_B=\{GeneId\},\ I_H=\{GeneId\},\Gamma_B=\ FoldChange<T1,\ \Gamma_H=\ FoldChange<T1,$$
$$\Xi=\ (support \geq 0.2) \wedge\ (confidence \geq 0.40))$$

In this query, while support determines the portion of samples in which sets of genes are expressed similarly, confidence determines how strongly the two sets of genes are correlated.

This statement might help biologists to discover the sets of genes that are responsible of the production of proteins involved in the development of certain diseases (e.g., cancer).

This query produces 1 million and 42 thousands rules.

Second query. A Gene Ontology (GO) is a hierarchy that describes gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. Now, we want to discover genes in correlation with a set of specific genes whose expression levels are over-expressed, and have the same collocation in the ontology. This can be done by evaluation of a given attribute *GOId* having for instance the value "GO:0000731" (this value is related to genes involved in the DNA synthesis during the DNA repair). *GOId* was retrieved from another relation which was joined with *MicroArray* obtaining *MicroArrayWithGO*: it associates to each gene in the microarray its collocation in the ontology. The new mining query is:

$$
\begin{aligned}
Q_{GO} = (&\boldsymbol{T}=MicroArrayWithGO,\ \boldsymbol{G}=\{SampleId,\ TreatmentConditionId\},\\
&I_B=\{GeneId\},\ I_H=\{GeneId\},\\
&\Gamma_B=\ (FoldChange<T1)\ \wedge\ (GOId=\text{“}GO{:}0000731\text{”}),\\
&\Gamma_H=\ (FoldChange<T1)\ \wedge\ (GOId=\text{“}GO{:}0000731\text{”}),\\
&\Xi=\ (support \geq 0.2)\ \wedge\ (confidence \geq 0.40))
\end{aligned}
$$

Also in this case, the new result can be obtained starting from the result of a previous query producing almost 52 thousands rules.

5.4 Experimental results

In Figure 3 we show the comparison between the execution times of the incremental algorithm and a traditional one that works from scratch. For each application discussed, we test the algorithms on the queries denoted as *Second query* and use the result of the *First query* as the starting point for the incremental algorithm. You can notice that execution times are highly dependent on the number of resulting rules. However, it is clear that in almost all the cases the incremental algorithm is the winning choice because the high selectivity of constraints in the second query allows to reduce effectively the result set. An exception is the query on candlesticks, in which the selectivity of constraints in the first and second query does not differ very much. Moreover, by an insight in the intermediate computations of the algorithms, we discovered that the previous result did not allow to

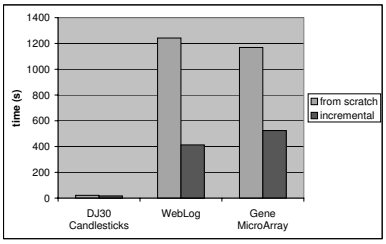


Figure 3: Comparison between execution times from scratch and by the incremental algorithm.

reduce significantly the search space of the candidate itemsets. In this precise case, the incremental solution should not be the preferred strategy: indeed, the I/O operations that are necessary to read the previous result already constitute a workload *per se* and are not counterbalanced by an effective advantage.

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. Furthermore, doing this way, the source table is only scanned once, without the need of maintaining in memory transaction ids, as done in Partition or Eclat.

We note that context dependent constraints have 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 literature, and the context dependent incremental algorithm proposed in this paper.

In Sections 4 and 5, we evaluated the incremental algorithms on some pretty large datasets, both synthetic and real. The results show that the approach reduces drastically the overall execution time. We believe the improvement to be absolutely necessary in many other practical data mining applications, in data warehouses and inductive database systems.

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 Conference 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 Answeringd 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 International Conference on Data Engineering, New Orleans, Louisiana, USA (1996)

[16] Thomas, S., Bodagala, S., Alsabti, K., Ranka, S.: An efficient algorithm for the incremental updation of association rules in large databases. In: Proceedings of the third ACM SIGKDD International Conference on Knowledge discovery and data mining. (1997)

[17] Ayan, N.F., Tansel, A.U., Arkun, E.: An efficient algorithm to update large itemsets with early pruning. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, San Diego, California, United States (1999)

[18] 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

[19] Leung, C.K.S., Lakshmanan, L.V.S., Ng, R.T.: Exploiting succinct constraints using fp-trees. ACM SIGKDD Explorations **4** (2002) 40–49

[20] Bucila, C., Gehrke, J., Kifer, D., White, W.M.: Dualminer: a dual-pruning algorithm for itemsets with constraints. In: Proceedings of 2002 ACM KDD. (2002) 42–51

[21] Feng, L., Dillon, T.S., Liu, J.: Inter-transactional association rules for multi-dimensional contexts for prediction and their application to studying meteorological data. Data Knowledge Engineering **37** (2001) 85–115

[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 Conference, Zurich, Swizerland (1995)

[24] Agarwal, R.C., Aggarwal, C.C., Prasad, V.V.V.: A tree projection algorithm for generation of frequent item sets. Jornal of Parallel Distrib. Comput. **61** (2001)

[25] Bayardo, R., Agrawal, R., Gunopulos, D.: Constraint-based rule mining in large, dense databases. In: Proceedings of the 15th Int'l Conf. on Data Engineering, Sydney, Australia (1999)

[26] 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

[27] Raedt, L.D.: A perspective on inductive databases. ACM SIGKDD Explorations **4** (2002) 69–77

[28] Botta, M., Esposito, R., Gallo, A., Meo, R.: Rt79-2004: Optimizing inductive queries in frequent itemset mining. Technical report, Dipartimento di Informatica, Università di Torino (2004)

[29] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. Lecture Notes in Computer Science **1540** (1999) 398–416

[30] Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proc. of ACM SIGMOD 2000, Dallas, TX, USA (2000)