Query Rewriting in Itemset Mining

Rosa Meo, Marco Botta, and Roberto Esposito

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

Abstract. In recent years, researchers have begun to study *inductive databases*, a new generation of databases for leveraging decision support applications. In this context, the user interacts with the DBMS using advanced, constraint-based languages for data mining where constraints have been specifically introduced to increase the relevance of the results and, at the same time, to reduce its volume.
In this paper we study the problem of mining frequent itemsets using an inductive database[1]. We propose a technique for query answering which consists in rewriting the query in terms of union and intersection of the result sets of other queries, previously executed and materialized. Unfortunately, the exploitation of past queries is not always applicable. We then present sufficient conditions for the optimization to apply and show that these conditions are strictly connected with the presence of functional dependencies between the attributes involved in the queries. We show some experiments on an initial prototype of an optimizer which demonstrates that this approach to query answering is not only viable but in many practical cases absolutely necessary since it reduces drastically the execution time.

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, 21, 15, 23, 4, 18, 24]. 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. For instance, Ng et al. [15] proposed a constrained frequent set mining framework within which the user can use a rich set of constraints that must be satisfied by the searched rules and that include SQL-style aggregate and non-aggregate predicates. These constraints can be exploited to guide the mining process, by pushing them deeply in the mining algorithms, in order to prune the search space of frequent itemsets as early as possible.

Constraints are widely exploited also in data mining languages, such as in [10, 14, 8, 23, 24] 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

[1] This work has been funded by EU FET project cInQ **c**onsortium on discoverying knowledge by **In**ductive **Q**ueries (IST-2000-26469).

H. Christiansen et al. (Eds.): FQAS 2004, LNAI 3055, pp. 111–124, 2004.

of tuples from which itemsets will be extracted. Constraint-based mining languages are also the main key factor of inductive databases proposed by Mannila and Imielinski in [9], in order to leverage decision support systems. These new promising approaches to mining will become really effective only when efficient optimizers for the mining languages will be available, i.e., if it will be possible to execute a query exploiting the available information in the database, such as the constraints in the schema, the indices or the results of previously executed queries.

In this paper, we introduce a very generic constraint-based language for the extraction of frequent itemsets and study the conditions under which query rewriting in the constraint-based mining language is possible.

By query rewriting we mean the determination of a relational expression on a set of queries whose result is equivalent to the result of a given query for every database on the same schema. Query rewriting is usually performed by query optimizers because the execution plan of the DBMS for the query in the rewritten form is better in terms of execution costs than for the original query. In the past, query rewriting has been widely used in relational databases, in data warehouses and in statistical database systems [3, 7, 6, 11, 25, 17]. In these works, query rewriting of a query computing aggregate functions is performed in terms of union of other queries whose results have already been materialized. In particular, [25] suggests that the choice of the materializations (the summaries) should be made according to the user frequent requests. We also make this assumption for the choice of the data mining materializations. Interestingly, [6] introduces iceberg queries and observes that a query which searches for itemsets is an example of this kind of queries. Iceberg queries are generally very expensive to compute since they require several scans of huge relations. As a consequence, in order to speed up the execution time, it makes sense to try to factorize the effort already done by the DBMS. [13, 11] search for the conditions under which query rewriting of queries with aggregate functions is possible. In [16] the problem of recognizing equivalent queries in multidimensional databases has been addressed.

1.1 Storage and Exploitation of the Results of Other Queries

We imagine that we can store the result sets of some query in the database. We do this because our aim is to reduce as much as possible the computational times of the data mining engine since, nowadays, the storage space is critic to a lesser extent.

Furthermore, we suppose to work 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 considered up to date and can be usefully exploited to speed up the execution of current queries. Suppose, for instance, that the optimizer recognizes that the current query is equivalent to a previous one whose result is available in the database. This allows the system to completely avoid huge computational effort: the exploration of the lattice search space of the frequent itemsets, and several scans of the database which are needed to

compute the aggregate functions on the itemsets (notably support count). In Section 4, we show that this approach is feasible and advantageous, by means of some experiments with a prototype optimizer. At the moment, the implemented optimizer recognizes equivalent queries, and exploits such equivalences to avoid heavy computations. However, it can be easily extended to recognize when the result of the current query is contained in a materialized result, thus saving computation in this case as well.

In this paper, of course, we start answering a little part of the complex and interesting issues arisen by the usage and maintainance of materializations in data mining. We start finding conditions under which the composition by means of set union and intersection of previous query results gives the solution of a current query. Not surprisingly, these conditions are related to the existence of functional dependences between the attributes on which itemsets are defined and the attributes on which the constraints are expressed. These results are similar, but more general than those presented in [20, 19] where the problem of query decomposition in inductive databases has been studied in terms of convex version spaces and seems to be linked to the monotonicity property of constraints.

1.2 Item Dependent and Context Dependent Constraints

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 truth 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., the purchase transaction). In this paper, we will characterize the constraints that are functionally dependent on the item extracted and call them *item dependent*. The exploitation of these constraints proves to be extremely useful from the viewpoint of the optimization of languages for data mining. In fact, the result set of queries that are constrained only on item dependent properties can be directly derived by the result set of related queries. In particular, if an optimizer can decompose the query into sub-queries that have been already executed by the system, the system can drastically reduce the workload by means of relational operations on the previous results.

In contrast to item dependent constraint, we present a new class of constraints for which the assumption of functional dependence is not valid; in other words, constraints whose satisfaction depends on the transactions in the database. We call these constraints, *context dependent*. We believe that these constraints are very difficult to manage. Indeed, they still might show the same properties of monotonicity and anti-monotonicity studied so far in literature but still cannot be embedded in algorithms as already done in the literature [15, 12].

Then, we will characterize a kind of queries which can possibly contain context dependent constraints and do not present any negative impact on the proposed optimization method.

The rest of the paper is organized as follows. Section 2 presents some preliminary definitions such as item dependent and context dependent constraints. Section 3 states the main results of the paper: sufficient conditions for query rewriting based on the materialization of other queries. Finally Section 4 shows some experimental results. These results further motivate the proposed approach. Section 5 draws some 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.

For the sake of exemplification, let us also 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`}, where: `tr` is the purchase transaction identifier, `customer` is the customer identifier, `date` is the date in which the purchase transaction occurred, `product` is the purchased product identifier, `category` is the category to which the product belongs, `brand` is the manufacturer of the product, `price` is the product price, and `qty` is the quantity purchased in transaction `tr`. The Σ relation is {`product`→`price`, `product`→`category`, `product`→`brand`, {`tr, product`}→`qty`, `tr`→`date`, `tr`→`customer`}. It should be noted, however, that the validity of the framework is general, and it does depend on neither the mining query language nor the running database example.

Of course, the above schema could also be represented over a set of relations and dimensions adopting the usual data warehouse star schema. Nonetheless, we keep the database in this non normalized form since it is very usual to mine data from the result of a pre-processing step of the whole data warehouse content (by means of selection and join over fact relation and one or more dimension relations).

The following equivalence relation will prove to be useful for the forthcoming discussion:

Definition 1. *Grouping equivalence relationship: two sets of attributes K_1 and K_2 are said to be* `grouping equivalent` *if and only if for any relation T defined on TS:*

$$\forall t_1, t_2 \in T : t_1[K_1] = t_2[K_1] \Leftrightarrow t_1[K_2] = t_2[K_2]$$

where $t_1[K_1]$ is the projection of the tuple t_1 on the attributes in K_1.

Sets of attributes that are grouping equivalent form a grouping equivalence class E. Each set of attributes belonging to the same grouping equivalence class partitions a database relation T in the same groups. We assume to know about a set of grouping equivalence classes $E_1 \ldots E_j$.

Example 1. In the `Purchase` example, the following non trivial equivalence class may be found: $E_1 = \{\{\texttt{tr}\}, \{\texttt{date}, \texttt{customer}\}, \{\texttt{tr}, \texttt{date}\}, \{\texttt{tr}, \texttt{customer}\}, \{\texttt{tr}, \texttt{date}, \texttt{customer}\}\}$.

Let us denote by $X \rightarrow Y$ a functional dependency (FD) between two attribute sets X (LHS) and Y (RHS) in the database schema TS.

Definition 2. *A* dependency set *of a set of attributes X contains all the possible RHS that can be obtained from X following a FD in Σ (direct or transitive) such that there is no $X' \subset X$ such that $X' \rightarrow Y$.*

As we did for equivalence classes, we assume to know about a set of dependency sets.

Example 2. The dependency set of {`product`} is {`category, price, brand`}. The dependency set of {`tr`} is {`customer, date`} while the dependency set of {`tr, product`} is {`qty`}.

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

- The *item attributes*, a set of attributes whose values constitute an item, i.e., an element of an itemset.
- The *grouping attributes* needed in order to decide how tuples are grouped for the formation of each itemset.
- The *mining constraints* which may be based either on the values of any of the attributes in TS (e.g., kind of product, price or quantity) or on aggregate values (e.g., sum of prices of products in an itemset)
- 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.

Usually in market basket analysis, when the user/analyst wants to describe by means of itemsets the most frequent sales occurred in purchase transactions, the grouping attribute is `tr` (the transaction identifier) and the itemsets are formed by the projection on `product` of sets of tuples selected from one group. However, for the sake of generality and of the expressive power of the mining language, grouping can be decided differently in each query. For instance, if the analyst wants to study the buying behavior of customers, grouping can be done using the `customer` attribute, or if the user wants to study the sales behaviour over time he/she can group by `date` or by week or month in the case these attributes were defined.

Users may exploit the mining constraints in order to discard uninteresting itemsets and to improve the performances of the mining algorithm.

By summarizing, a mining query may be described as

$$Q = (T, G, I, \Gamma(M), \Xi)$$

where T is the database relation, G is the set of grouping attributes, I is the set of item attributes, Γ is a boolean expression of atomic predicates over a set M

of attributes (the *mining attributes*) with $M \subset TS$, and Ξ is an expression on some statistical measures used for the evaluation of each itemset.

An atomic predicate can be any of the following:

1. $\mathtt{A_i}\theta\mathtt{v_{Ai}}$
2. $\mathtt{agg(A_i)}\theta\mathtt{v}$

where θ is a relational operator such as $<, <=, =, >, >=, <>$, $\mathtt{v_{Ai}}$ is a value from the domain of attribute A_i, $\mathtt{agg(A_i)}$ is the result of an aggregate function on the set of values of A_i which appear in the tuples from which the itemset has been extracted, $\mathtt{v}$ is a value from the natural or rational domain. Finally, Ξ is a boolean expression in which each term has the form

$$\xi(\nu)\theta\mathtt{v}$$

where ξ is a statistical measure for the itemset evaluation and θ and $\mathtt{v}$ are defined as above. We require that the evaluation measure $\xi : \mathbb{N} \rightarrow \mathbb{R}$ of an itemset J is a function applied to the number of the groups in which J satisfies the constraints.

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 defined on an input relation T and generates a result set R [2].

Example 3. The query

$$Q = (\ \mathtt{Purchase}, \{\mathtt{tr}\}, \{\mathtt{product}\},$$
$$\mathtt{price} > 100 \wedge \mathtt{count(product)} >= 2, \mathtt{support\ count} >= 20)$$

over the `Purchase` relation (first parameter) extracts itemsets formed by products (third parameter), where all the products in the itemset have been sold in the same transaction (second parameter). Moreover, each product in the itemset must have price greater than 100 and the itemset must contain at least two products (aggregate function $\mathtt{count(product)} >= 2$ on the candidate itemset). Finally, support count of the returned itemsets must be at least 20.

Example 4. The following query over the `Purchase` relation extracts itemsets formed by occurrences of products in sales grouped by date where the selected products are sold in a low quantity (less than 10 units each) and where the itemset support count is at least 30.

$$Q = (\mathtt{Purchase}, \{\mathtt{date}\}, \{\mathtt{product}\}, \mathtt{qty} < 10, \mathtt{support\ count} >= 30)$$

Now that we have seen how constraint-based mining queries are formed, let us define two particular types of predicates in constraints: the *item dependent* constraints and the *context dependent* ones.

[2] In particular, the mining engine stores R on the same DB from which R was mined as a pair of normalized relations R_{summary} and R_{detail}. R_{summary} has the schema {Itemset_id, Ξ} where Itemset_id is the identifier of the itemset and Ξ is its statistical evaluation expression. The second relation R_{detail}, over the schema {Itemset_id, I}, contains the detail of each itemset in terms of its constituting items.

Definition 3. *Let us consider the query*

$$Q = (T, G, I, \Gamma(M), \Xi)$$

An atomic predicate $\mathtt{P}(\mathtt{A_i}) \in \Gamma(M), \mathtt{A_i} \in M$ *is defined as an* item dependent constraint *if and only if* A_i *belongs to the dependency set of at least one attribute subset of* I*. Otherwise, it is defined as a* context dependent constraint*.*

Example 5. Predicate `price>100` in the query of Example 3 is an item dependent constraint because `price` is in the dependency set of `product`. Predicate `support count>=20` is not an item dependent constraint because it is not applied on an attribute in the dependency set of `product` and also because it is not a predicate in a constrained expression, but in a statistical evaluation expression.

Example 6. Predicate `qty>100` in query of Example 4 is a context dependent constraint because `qty` depends on either `product` or `tr` alone.

We notice that an itemset $\mathcal{I}$ satisfies an item dependent constraint either in any database group in which it occurs, or in none. This immediately implies the following:

Lemma 1. *An itemset* $\mathcal{I}$ *that satisfies an item dependent constraint in a mining query has a statistical measure that is a function of the total number of groups in which* I *occurs in the given database instance.*

On the contrary, a context dependent constraint might be satisfied by some occurrences of itemset $\mathcal{I}$, but not all of them. Then, its statistical measure is based on the number of groups in which the itemset satisfies the constraint (that might be less than the total number of groups in which it appears).

3 Query Rewriting

We suppose that a certain set S of other queries Q_i have been already executed by the inductive database management system and that their results have been stored in distinct result sets R_i.

Definition 4. *A query* Q_0 *on* TS *is rewritable in terms of a set* $S' = \{Q_j\} \subseteq S$ *of other queries if it is possible to rebuild the result set* R_0 *using only intersection and union operations of the result sets* R_j *and this is true regardless of the database instance.*

Certainly, the query rewriting of Q_0 on queries in S is possible for every database instance on the given schema, if certain hypothesis are fulfilled.

Definition 5. *A set of queries* $S' = \{Q_i\} \subseteq S$ *is a candidate rewriting of* Q_0 *if the following conditions hold for all* $Q_i \in S'$*:*

1. G_i *is in the same grouping equivalence class of* G_0
2. I_i *is in the same grouping equivalence class of* I_0*.*
3. Ξ_0 *and* Ξ_i *are logically equivalent expressions on the same statistical evaluation measures and values.*

Condition 1 guarantees that grouping attributes in Q_0 and in Q_i partition the input relation T into the same groups for every database defined on TS.

Condition 2 allows the result sets of different queries R_i, R_j to be intersected and joined even when $I_i \neq I_j$. In such a situation it is necessary to rewrite the two result sets in terms of a common element in the grouping equivalence class to which I_i and I_j both belong.

Condition 3 guarantees that any result element in R_i has been evaluated by the same statistical evaluation measures as in Q_0. If those measures were not the same for all the queries in S', it would be necessary to recompute them for each result element and this would require to access again the input relation. This should be avoided in order to save computational work, otherwise most of the benefits gained would be wasted.

Now we can establish the main result of this paper. It defines how the various query results can be composed together.

Theorem 1. *Let us consider a query Q and let $S' = \{Q_i, Q_j\}$ be a candidate rewriting of Q. If both M_i and M_j are in the dependency set of any two subsets of I, then the following relationships hold.*

$$\Gamma(M) = \Gamma_i(M_i) \wedge \Gamma_j(M_j) \Rightarrow R = R_i \cap R_j$$
$$\Gamma(M) = \Gamma_i(M_i) \vee \Gamma_j(M_j) \Rightarrow R = R_i \cup R_j$$

Proof. (sketch) We first notice that being Q_i and Q_j in a candidate rewriting of Q, then their results may be composed together since the semantics of the itemsets extracted by Q_i and Q_j are the same. Moreover, let us recall that an itemset appears in a result set iff it satisfies both the query constraints and the statistical expression. The result follows from the following considerations:

- an itemset which satisfies the conjunction (respectively the disjunction) of $\Gamma_i(M_i)$ and $\Gamma_j(M_j)$ will be included in the intersection (respectively the union) of R_i and R_j.
- the atomic predicates involved are item dependent constraints and hence, as a consequence of Lemma 1, the statistical measures associated to them does not depend on the constraints themselves. Hence, its value is the same in R_i, R_j, and R.

Item dependent constraints in a query Q guarantee that a query rewriting candidate $\{Q_i,Q_j\}$ of Q can be used for answering query Q if there is a logical equivalence between mining predicates of Q and the conjunction (disjunction) of mining predicates in Q_i and Q_j. In these cases, the itemsets found respectively in the intersection (union) of the results R_i and R_j are also in R and viceversa. This Theorem has important consequences, from the computational viewpoint, because not only we can predict which itemsets we will found in the result of query Q without coming back to the database but also the value of their statistical measures.

3.1 Query Rewriting for Mining Queries on Context Dependent Constraints

In order to explain how it is possible to do query rewriting, even for context dependent constraints, let us consider a sample query Q and one of its candidate rewritings $Q' = \{Q_i, Q_j\}$. Let J be an itemset which appears in both Q_i and Q_j, and let us denote with O_i and O_j the sets of database groups that contain J in which J satisfies the mining constraints in Q_i and Q_j respectively. As we will discuss below, O_i and O_j are not necessarily the same set and hence the set of groups which satisfy both the mining conditions $\Gamma_i(M_i)$ and $\Gamma_j(M_j)$ should be computed as $O_i \cap O_j$. The important implication of this fact is that the number of groups that satisfy both queries cannot be foretold directly from the results of Q_i and Q_j, when only the size of O_i and the size of O_j are known: one usually needs to access the database and retrieve the entire group lists O_i and O_j.

Interestingly, Theorem 1 states it is sufficient that the query constraints are item dependent for query rewriting to be possible (without making accesses to the original database relation T). Under the theorem assumptions, in fact, whether J satisfies the constraints or not uniquely depends on J itself. If J satisfies the constraints, it does it for all the groups in which J occurs. This implies that, for any item dependent constraint, the set of groups O_i and O_j are either equal or empty.

On the other hand, when a mining query is based on context dependent constraints, the occurrences of an itemset J may satisfy the mining constraints in certain database groups *but not in others*. This is the core characteristic of context dependent constraints: whether they are satisfied by an itemset depends on the database context in which the itemset occurs. As a consequence, in order to answer the mining query, it is usually necessary to retrieve all the database groups in which the itemset occurs in order to verify case by case the constraints (or typically maintaining O_i and O_j for each itemset J, and then by means of their intersection and union).

Interestingly, as it is stated below, there are situations in which this can be avoided even when context dependent constraints are involved.

Theorem 2. *Let us consider a query Q and let $S' = \{Q_i, Q_j\}$ be a candidate rewriting of Q. Let us also assume that G and I are such that an itemset appears at most once in each group. If $\Gamma_i(X) \Rightarrow \neg\Gamma_j(Y)$ (i.e., $\Gamma_j(Y)$ is always false for the itemsets that satisfy $\Gamma_i(X)$ regardless the database instance T) then the following relationships hold:*

$$\Gamma(M) = \Gamma_i(M_i) \wedge \Gamma_j(M_j) \Rightarrow R = \emptyset$$
$$\Gamma(M) = \Gamma_i(M_i) \vee \Gamma_j(M_j) \Rightarrow R = R_i \cup R_j$$

For each itemset J in R and for each statistical measure ξ_k that contributes to Ξ, the new value of $\xi_k(\nu)$ is computed by setting:

$$\nu = \nu_i + \nu_j$$

where ν_i and ν_j are the number of groups in which J appears and that satisfy the mining constraints of the query Q_i and Q_j respectively.

Proof. (sketch) We first notice that since we do not assume item dependent constraints, the statistical measure of an itemset may change accordingly to the constraints involved in the queries.

The first part of the theorem follows since $\Gamma_i(M_i) \Rightarrow \neg\Gamma_j(M_j)$ implies that an itemset in relation T cannot satisfy $\Gamma_i(M_i) \wedge \Gamma_j(M_j)$.

For the second part, the union of R_i and R_j contains all itemsets which satisfy either $\Gamma_i(M_i)$ or $\Gamma_j(M_j)$. Moreover, since any group contains at most one occurrence of a fixed itemset J and it cannot satisfy both Γ_i and Γ_j, then that group contributes either to the count of J in R_i or to its count in R_j, but not to both. Hence, the evaluation measure in R resulting from the sum of the counts in R_i and R_j is correct.

4 Experiments

The query rewriting of a query Q requires the identification of a set of queries which satisfy either Theorem 1 or Theorem 2 and the foreseen algebraic operations between their result sets. This is advantageous provided that their overall cost is less than the cost of executing the query from scratch. Indeed union and join of two results by means of SQL queries are well known to be very efficient operations provided that the needed indices are defined.

Hence, an important issue involved in the optimization problem is the identification of which queries among previous ones may be used as Q_1 and Q_2. In the most general setting, the solution to the problem is not trivial at all, since it requires to be able to recognize the equivalence of logical formulae. The problem is even more complex than this, since when Q increases in size, the number of different ways in which we can choose Q_i and Q_j increases exponentially. A complete solution to this problem is clearly out of the scope of this paper. Anyway, at the present time we already implemented an optimizer that tests for the equivalence of two, possibly complex, queries. In the forthcoming work, we plan to implement a heuristic strategy based on the greedy covering strategy in order to find approximate solutions to the general problem.

In the following we explain how the optimizer checks for the logical equivalence between a query Q and one of its candidate rewritings Q_i. In brief, the following operations are performed:

1. Rewriting of the mining constraints of the two queries in disjunctive normal form. This step is necessary in order to make it easier to identify equivalent logical expressions.
2. Substitution of some mining constraints with some other equivalent ones or elimination of redundant ones. This is done by the exploitation of the functional dependencies in the database schema.
3. Generation of a compact representation of the truth table of the two resulting expressions.
4. Check the equivalence between the two predicates expressions on the table.

Example 7. Consider queries

$$
\begin{aligned}
Q = (\ & \texttt{Purchase}, \{\texttt{tr}\}, \{\texttt{product}\},\\
& \texttt{category} = \texttt{'computer'} \wedge \texttt{brand} = \texttt{'XX'}\\
& \vee \texttt{category} = \texttt{'hi-fi'} \wedge \texttt{price} < 200,\\
& \texttt{support count} >= 20)\\
Q_i = (& \texttt{Purchase}, \{\texttt{tr}\}, \{\texttt{product}\},\\
& \texttt{category} = \texttt{'computer'} \wedge (\neg \texttt{category} = \texttt{'hi-fi'} \vee \texttt{price} \geq 200)\\
& \vee \texttt{category} = \texttt{'hi-fi'} \wedge \texttt{price} > 100 \wedge \texttt{price} < 200),\\
& \texttt{support count} >= 20)
\end{aligned}
$$

and suppose that the following functional dependencies are known to hold:

- no product in category `hi-fi` costs less than 150 (that is $\texttt{category} = \texttt{'hi-fi'} \rightarrow \texttt{price} \geq 150$).
- all products that have 'XX' brand are in category `'computer'` (that is $\texttt{brand} = \texttt{'XX'} \rightarrow \texttt{category} = \texttt{'computer'}$).

The optimizer performs the following transformations:

1. it transforms constraints in disjunctive form. Q is already in disjunctive form. Q_i becomes:

$$
\begin{aligned}
Q_i = (\ & \texttt{Purchase}, \{\texttt{tr}\}, \{\texttt{product}\},\\
& \texttt{category} = \texttt{'computer'} \wedge \neg \texttt{category} = \texttt{'hi-fi'}\\
& \vee \texttt{category} = \texttt{'computer'} \wedge \texttt{price} \geq 200\\
& \vee \texttt{category} = \texttt{'hi-fi'} \wedge \texttt{price} > 100 \wedge \texttt{price} < 200,\\
& \texttt{support count} >= 20)
\end{aligned}
$$

2. it exploits the known functional dependencies and some basic property of range predicates in an ordered domain. As a result query Q becomes

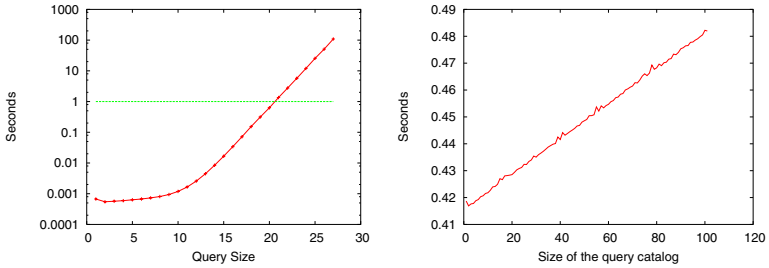
$$
\begin{aligned}
Q = (\ & \texttt{Purchase}, \{\texttt{tr}\}, \{\texttt{product}\},\\
& \texttt{category} = \texttt{'computer'} \vee \texttt{category} = \texttt{'hi-fi'} \wedge \texttt{price} < 200,\\
& \texttt{support count} >= 20)
\end{aligned}
$$

and query Q_i becomes

$$
\begin{aligned}
Q_i = (\ & \texttt{Purchase}, \{\texttt{tr}\}, \{\texttt{product}\}, \texttt{category} = \texttt{'computer'}\\
& \vee \texttt{category} = \texttt{'computer'} \wedge \texttt{price} \geq 200\\
& \vee \texttt{category} = \texttt{'hi-fi'} \wedge \texttt{price} < 200),\\
& \texttt{support count} >= 20)
\end{aligned}
$$

3. it builds the truth table of the mining constraints expressions in Q and in Q_i from step 2 and verifies that they are logically equivalent. In this case this is easily checked to be true.

The time complexity of the solution is exponential in the number of distinct predicates involved; anyway, a careful implementation keeps the problem tractable for most reasonable queries. In order to test the last assertion, we



(a) (b)

Fig. 1. Optimization times as a function of (a) number of atomic predicates and (b) the size of the query catalog.

randomly built queries of increasing length and ran the testing procedure over them. The results are reported in logarithmic scale in Figure 1 (a). It is worth noting that, even if the algorithm is clearly exponential, whenever the number of predicates involved in the queries is not larger than 20, the time spent to process the query is less than $10^{-0.2} \approx 0.63$ seconds[3]. Since typical query lengths are much smaller than 20, we can argue that in common situations the processing time for the equivalence checking is actually acceptable.

The algorithm which checks the entire database catalog of previous queries simply repeats the equivalence checking algorithm once for each past query until an equivalence is found. Hence, the complexity increases linearly with the number of previous queries as it can be seen in Figure 1 (b). Of course, a number of "cleaning" policies can be devised in order to keep the size of the query catalog relatively small.

Let us consider an experiment we made on a sample database having $\approx 250,000$ records containing $\approx 10,000$ sales transactions composed of 25 items on average, randomly extracted from a total of 939 items. The system took about one millisecond for the optimization steps, 12 seconds to prepare the dataset for the mining algorithm and 543 seconds to build the final result using a Partition-like algorithm (at the end, the result set contained 42 association rules).

The system ran on a setting which corresponds to a nearby of the origin of the two Figures 1 (a) and (b). The results prove the feasibility of the proposed approach since they make evident that the time saved in case the optimization succeeds is dramatically smaller than the execution time of the mining algorithm.

[3] The testing program was written entirely in C++. The testing machine running Linux (Kernel v.2.4) sports two Pentium III processors (1.4GHz each) and 1Gb of RAM. The mining engine connects to a MySQL server (v.3.23.53) by means of an ODBC connection.

Notice that, even in the case that the query catalog has 100 queries with 15 atomic predicates on average, the time spent by the optimizer is still less than four seconds, which is acceptable.

5 Conclusions

In this paper we studied the problem of query rewriting for queries that mine frequent itemsets. The proposed technique consists in rewriting the query in terms of union and intersection of the result sets of other queries, previously executed and materialized. We found sufficient conditions for the optimization to apply and shown by means of some experiments that this approach is viable because it reduces drastically the execution time. This seems absolutely necessary in many data mining applications.

In the future we plan to consider also fuzzy evaluation techniques for the problem of mining frequent itemset from a database. Previous results in this direction already exist (the so called condensed ϵ-adequate representations developed by Boulicaut and Rigotti in [2]) that consider approximate solutions to the identification of the statistical evaluation measure of an itemset.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Knowledge Discovery in Databases*, volume 2. AAAI/MIT Press, Santiago, Chile, September 1995.
2. J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, 2003.
3. S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of 11th ICDE*, March 1995.
4. S. Chaudhuri, V. Narasayya, and S. Sarawagi. Efficient evaluation of queries with mining predicates. In *Proc. of the 18th Int'l Conference on Data Engineering (ICDE)*, San Jose, USA, April 2002.
5. S. Chaudhuri and K. Shim. Optimizing queries with aggregate views. In *Proc. of EDBT*, 1996.
6. M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. Computing iceberg queries efficiently. In *Proceeding of VLDB '98*, 1998.
7. A. Gupta, V. Harinarayan, and D. Quass. Aggregate query processing in data warehousing environments. In *Proceedings of VLDB '95*, pages 358–369, 1995.
8. J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational databases. *In Proceedings of SIGMOD-96 Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1996.
9. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, November 1996.
10. T. Imielinski, A. Virmani, and A. Abdoulghani. Datamine: Application programming interface and query language for database mining. *KDD-96*, pages 256–260, 1996.

11. H.-J. Lenz and A. Shoshani. Summarizability in olap and statistical data bases. In *Proceedings Ninth International Conference on Scientific and Statistical Database Management*, pages 132–143. IEEE Computer Society, August 1997.
12. C. K.-S. Leung, L. V. S. Lakshmanan, and R. T. Ng. Exploiting succinct constraints using fp-trees. *ACM SIGKDD Explorations*, 4(1):40–49, June 2002.
13. F. M. Malvestuto. The derivation problem for summary data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 82–89, 1988.
14. R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proceedings of the 22st VLDB Conference*, Bombay, India, September 1996.
15. R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of 1998 ACM SIGMOD International Conference Management of Data*, pages 13–24, 1998.
16. W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalence among aggregate queries. In *Proceedings of ACM PODS*, pages 214–223, 1998.
17. C.-S. Park, M. H. Kim, and Y.-J. Lee. Rewriting olap queries using materialized views and dimension hierarchies in data warehouses. In *Proceeding of ICDE'01*, 2001.
18. C.-S. Perng, H. Wang, S. Ma, and J. L. Hellerstein. Discovery in multi-attribute data with user-defined constraints. *ACM SIGKDD Explorations*, 4(1):56–64, 2002.
19. L. D. Raedt. A perspective on inductive databases. *ACM SIGKDD Explorations*, 4(2):69–77, December 2002.
20. L. D. Raedt, M. Jaeger, S. D. Lee, and H. Mannila. A theory of inductive query answering. In *Proceedings of IEEE International Conference on Data Mining*, pages 123–130. IEEE Computer Society, December 2002.
21. R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proceedings of 1997 ACM KDD*, pages 67–73, 1997.
22. D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. In *Proceeding of VLDB '96*, 1996.
23. D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: A generalization of association-rule mining. In *Proceedings of 1998 ACM SIGMOD International Conference Management of Data*, 1998.
24. H. Wang and C. Zaniolo. User defined aggregates for logical data languages. In *Proc. of DDLP*, pages 85–97, 1998.
25. X. S. Wang and C. Li. Deriving orthogonality to optimize the search for summary data. *Information Systems*, 24(1):47–65, 1999.
26. Y. Zhao, P. M.Deshpande, J. F. Naughton, and A. Shukla. Simultaneous optimization and evaluation of multiple dimensional queries. In *Proceedings of ACM SIGMOD '98*, pages 271–282, 1998.