Optimization of a Language for Data Mining

Rosa Meo
Università degli Studi di Torino - Dipartimento di Informatica
corso Svizzera 185 - 10149 - Torino - Italy
e-mail: meo@di.unito.it

ABSTRACT
Constraint-based mining has attracted in recent years the interest of the data mining research community because it increases the relevance of the result set, reduces its volume and the amount of workload. However, constrained-based mining will be completely feasible only when efficient optimizers for mining languages will be available.

This paper is a first step towards the construction of optimizers for a constraint-based mining language. It provides the guidelines for the comparison of classes of statements by means of the relationships existing between their result sets. Furthermore it identifies as useful information to the optimization the presence of unique constraints and functional dependencies in the schema of the database. We show the practical implications of the discussed principles with a set of algorithms designed for a specific mining language. These algorithms use also a new designed index, called mining index that allows to reduce the portion of the database to be read in response to some classes of queries. In these cases the workload of the mining engine is greatly reduced or completely avoided in a significant subset of the cases.

Categories and Subject Descriptors
H.2.8 [Database Management]: Database Applications—data mining; H.2.3 [Languages]: query languages; H.2.4 [Systems]: query processing; D.3.4 [Processors]: Compilers, Optimization

General Terms
Languages, Theory, Algorithms

Keywords
Association rules, constraints, query optimization

1. INTRODUCTION
In data mining, especially if the volumes of the data are very large, there exist many data patterns that satisfy a certain problem. If the discovered patterns do not seem to be perfectly tailored to the expectations of the user, he/she can further refine his/her query in order to restrict the result set in a meaningful way to the more interesting data patterns. However, the drawback is that the overall computational work might be prohibitive if each new request submitted by the user is processed by the system ex-novo because of the tremendous number of data patterns involved. On the contrary, it can be adopted with success if the query language and the data pattern extraction engine are able to work in an incremental fashion, in the sense that they compute on the fly the requested result on the basis of a previously generated result set or intermediate data set. In this way the overall mass of work is accordingly limited. A level-wise algorithm that tackles the related problem of the incremental maintenance of the rule base when the database is updated is presented in [3].

In the last years more interest has been focused on constraint-based mining [4, 6, 12, 13, 15, 14] for several reasons. Constrained mining helps to reduce the volume of the result set; it helps the user, analyst to specify better the typology of the desired result and therefore increases its relevance; it reduces the amount of computational work performed by the system, provided that constraints can be used to reduce the search-space of the patterns solving a certain problem [12, 13, 14]. Finally, constraint-based mining is at the basis of iterative, interactive mining [6, 8, 12]. In particular, in the context of inductive databases proposed by Mannila and Imielinski in [5], interactive, constrained mining occurs by means of the use of special-purpose query languages. 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 (intermediate) results of other queries, previously executed.

This paper is a first step towards the construction of optimizers of query languages for mining. It provides the guidelines for the comparison of classes of queries by means of the existing relationships between their result sets. As we will see each of these relationships (dominance, inclusion and equivalence) foresees a class of algorithms for the extraction of data mining patterns that presents a decreasing degree of coupling with the database: with dominance both the database and the result sets of some other queries must be read and processed; with inclusion only the result set must be read and processed, while finally with equivalence, only the result set must be read with no need of processing. These properties are particularly interesting.
when we want to speed-up the execution of a succession of several correlated queries and reduce the volume of data to be read in response to each of them, as in the case of constraint based, iterative mining. In this paper we identify the conditions (presence of keys and functionally dependent attributes) under which a certain relationship can be implemented with an algorithm characterized by a lower degree of coupling with the database. In particular, the relationship of dominance between two queries (which occurs most frequently) can be implemented with an algorithm that requires only the input of the result set of some other queries and not the database. The presented guidelines are general and not restricted to a certain language. They are useful to identify the relationships between classes of queries under the cited conditions and provide the premises to reuse the result sets of previously executed queries.

In this paper we show that these guidelines are valid adopting them for a specific mining language, the MINE RULE operator [10, 11]. Thus we show with a set of sample algorithms that an incremental approach for the extraction of association rules from a relational database is feasible. We show also the necessity and importance of storing intermediate results during query processing. We identify for these purposes a specific data structure, called mining index. In this paper, the design of the mining index is presented in such a way that the results of previous queries are exploited. We show that when a mining index has been created for query \( Q \), and the system evaluates a query \( Q' \), if certain relationships exist between \( Q \) and \( Q' \), the system will only use the mining index and the results of \( Q \). This reduces the number of I/O operations on the source database and the overall computational effort. Finally, we show the use of mining indices and of constraints in database schema in some simple algorithms that address some typical cases of MINE RULE queries.

The paper is organized as follows: Section 2 presents the MINE RULE operator, Section 3 studies the relationships between two mining queries and applies its results to the MINE RULE language. Section 4 discusses the results and applies them in a set of new algorithms. Finally Section 5 draws the conclusions.

## 2. MINE RULE SYNTAX

We briefly present here the MINE RULE operator. For a complete description refer to [10, 11]. The syntax of a MINE RULE query is the following:

\[
\text{MINE RULE} \langle \text{OutputTable} \rangle \text{ AS SELECT DISTINCT 1..n \langle \text{bodyAttrList} \rangle AS BODY, 1..m \langle \text{headAttrList} \rangle AS HEAD, \text{ SUPPORT, CONFIDENCE WHERE} \langle \text{miningCondition} \rangle \text{ FROM} \langle \text{sourceTable} \rangle \text{ GROUP BY} \langle \text{groupAttrList} \rangle \text{ [HAVING} \langle \text{groupCondition} \rangle \text{] CLUSTER BY} \langle \text{clusterAttrList} \rangle \text{ [HAVING} \langle \text{clusterCondition} \rangle \text{] EXTRACTING RULES WITH SUPPORT:} \langle \text{minSup} \rangle, \text{ CONFIDENCE:} \langle \text{minConf} \rangle \text{].}
\]

The association rules are extracted by performing the following steps:

**Group computation.** The GROUP BY clause logically partitions the source relation into groups, such that all tuples in a group have the same value of the grouping attributes groupAttrList.

**Group filtering.** The optional HAVING clause associated to the GROUP BY clause says that only groups in which all tuples satisfy the groupCondition are considered for rule extraction.

**Cluster identification.** The optional CLUSTER BY clause further partitions each group into sub-groups called clusters such that tuples in a cluster have the same value for the clustering attributes clusterAttrList. The body (respectively head) of a rule is extracted from clusters and not from entire groups. Thus elements in the body (respectively head) share the same value of the clustering attributes; if clusters are not specified, body and head are extracted from the trivial cluster, i.e. the entire group.

**Cluster coupling.** To compose rules, every pair of clusters (one for the body and one for the head) inside the same group is considered. Furthermore, the optional HAVING clause of the CLUSTER BY clause selects the cluster pairs that should be considered for extracting rules, that are those that satisfy the clusterCondition.

**Rule extraction.** From each group and each cluster pair, the SELECT clause extracts all possible associations of an unlimited set of bodyAttrList (clause 1..n bodyAttrList AS BODY), representing the body of rules, with an unlimited set of headAttrList (clause 1..m headAttrList AS HEAD), representing the head of rules. Attributes appearing in the rules are called rule attributes.

**Mining condition.** The (optional) WHERE clause following the SELECT clause forces rule extraction to consider only tuples that satisfy the miningCondition.

**Support and confidence evaluation.** The support of a rule is the number of groups from which the rule is extracted divided by the total number of groups generated by the GROUP BY clause. The confidence is the number of groups from which the rule is extracted divided by the number of groups that contain the body in some cluster. When support or confidence are lower than the respective minimum thresholds (minSup and minConf in our sample statement), the rule is discarded.

[11] pointed out that it is possible to write every MINE RULE query considering certain design criteria (which are the entities that association rules describe, how do we want to describe them, etc.). These criteria correspond to orthogonal dimensions in a cube, and in turn these dimensions correspond to the clauses of the MINE RULE statement: the GROUP BY and grouping attributes, the SELECT and rule attributes which are mandatory; the mining attributes (and relative predicates in mining condition), the clustering attributes (and relative predicates in clustering condition) which are optional. For the purposes of the discussion we will refer to these attributes and conditions with symbols. The correspondences between attributes and symbols are indicated in Table 1. These latter considerations are of course valid also for the other query languages: for instance, in SQL the orthogonal dimensions are given by SELECT, WHERE, GROUP BY and ORDER BY clauses in which only SELECT is mandatory.
Table 1: Symbols for the main syntactic features of interest in MINE RULE

<table>
<thead>
<tr>
<th>attributes or predicate</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>grouping attributes</td>
<td>ga</td>
</tr>
<tr>
<td>grouping condition</td>
<td>gc</td>
</tr>
<tr>
<td>rule attributes</td>
<td>ra</td>
</tr>
<tr>
<td>mining condition</td>
<td>mc</td>
</tr>
<tr>
<td>clustering attributes</td>
<td>ca</td>
</tr>
<tr>
<td>clustering condition</td>
<td>cc</td>
</tr>
</tbody>
</table>

3. THE RELATIONSHIPS BETWEEN TWO MINE RULE QUERIES

As already said, the previous work on the containment relationships between two MINE RULE queries can be found in [1]. The main relationships between two queries are described in the following.

Equivalence Let M and M' be two mining queries, extracting from the same source data rule sets R and R' respectively. M and M' are equivalent (M ≡ M') if, for all instances of the source data, each rule r in R is also in R' and vice versa, with the same value of support and confidence.

Examples of equivalence are provided by the queries that are identical apart from the FROM clause in which two equivalent relations are retrieved (according to the relational algebra equivalence relationships). Other examples will be provided in presence of candidate keys or functional dependence relationships between attributes, and will be described in the paper.

Inclusion Let M and M' be two mining queries, extracting from the same source data rule sets R and R' respectively. M includes M' (M ⊇ M') if, for all instances of the source data, each rule r in R' is also in R with the same value of support and confidence.

Examples of inclusion are provided by the queries M and M' that are identical apart from the minimum support and confidence thresholds and the constraints on body and head minimum and maximum cardinalities [1].

Dominance Let M and M' be two mining queries, extracting from the same source data rule sets R and R' respectively, with s, c and s', c' the support and confidence values of a rule in R and R'. Query M dominates M' (M ⊄ M') if, for all instances of the source data, each rule r in R' is also in R with a support s' ≤ s and a confidence c' ≤ c.

Examples of dominance are provided by the following theorem.

Theorem 1: Two queries, M and M', identical apart from the respective mining conditions m and m', if m' ⊃ m (i.e., m' entails m) then M ⊃ M'. M ⊃ M' occurs also if the implication relationship (m' → m) exists between the cluster conditions (b and b') of two mining queries M and M', which are identical apart from the cluster condition predicates. □

3.0.1 Identifying the Dominant Queries

We present here some general results that apply to MINE RULE queries. These results have been used also to elaborate the Sections that follow.

Theorem 2: Given queries M and M', M ⊃ M' if M' is identical to M apart from an additional clause. □

As an example of the above Theorem, consider the following MINE RULE queries that extract association rules from the source table Purchases(tr, cust, item, date, price, discount) collecting purchase data of the customers transactions. In the example, query M' has an additional clustering clause.

M: MINE RULE Output_for_query_M AS
SELECT DISTINCT 1..n item AS BODY,
WHERE BODY.price>100
FROM Purchases
GROUP BY cust
EXTRACTING RULES WITH SUPPORT:0.1, CONFIDENCE:0.4

M': MINE RULE Output_for_query_M' AS
SELECT DISTINCT 1..n item AS BODY,
WHERE BODY.price>100
FROM Purchases
GROUP BY cust
CLUSTER BY date
EXTRACTING RULES WITH SUPPORT:0.1, CONFIDENCE:0.4

The proof of the above theorem as well as of the theorems that will follow are omitted for lack of space but can be found in [2].

Theorem 3: Given a query M and another query M' identical to M apart from an additional predicate in gc, cc or mc: M ⊃ M' if the additional predicate is connected with the conjunction operator; M ⊃ M if it is connected with the disjunction operator. □

We start with some definitions and theorems. We will refer to them for the results that will follow.

Definition 1: We denote by a ⊳ b two candidate keys, i.e. two non null attributes of the same database relation on which a unique constraint is defined (unique(a) and unique(b)). In other words 2 two functions F, F' such that F ⊃ F' if \( \forall (v_a, v_b), v_a \in \text{Dom}_a \text{ (the domain of } a), v_b \in \text{Dom}_b \text{ (the domain of } b), \text{ if } F(v_a) = F'(v_b) \text{ then } v_b = F'^{-1}(v_a) \).

Theorem 4: Let be a ⊳ b and M a query with a ∈ ga (or ca or ra). Let M' be a query identical to M apart from ga (or ca or ra) in which b is added to the attribute list (for instance, ga_M = ga_M.b). Under these conditions, M' ⊃ M. The same holds if M' is obtained from M by substitution of a with b. □

Definition 2: Let be a ⊳ b and P(a) and Q(b) two predicates such that \( \forall (v_a, v_b), v_a \in \text{Dom}_a, v_b \in \text{Dom}_b, \text{ if } P(v_a) = Q(v_b) =\)}
Table 2: The relationships between two MINE RULE queries in presence of candidate keys.

<table>
<thead>
<tr>
<th>M:</th>
<th>M'</th>
<th>q(b)</th>
<th>ga</th>
<th>ca</th>
<th>ra</th>
<th>gc</th>
<th>cc</th>
<th>mc</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td>GA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>ca</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>ra</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>M'</td>
<td></td>
<td></td>
<td>GC</td>
<td>MC</td>
<td>MC</td>
<td>MC</td>
<td>MC</td>
<td>MC</td>
</tr>
<tr>
<td>P(a)</td>
<td>G</td>
<td>NA</td>
<td>NA</td>
<td>GA</td>
<td>GA</td>
<td>GA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

\begin{align}
\text{v}_a \text{ then } P(v_a) = Q(v_a). \text{ We call } P(a) \text{ and } Q(b) \text{ equivalent predicates.} \quad & \square \\
\text{Theorem 5: Let be } a \leftrightarrow b, P(a) \text{ and } Q(b) \text{ two equivalent predicates, and } M \text{ be a query with } P(a) \text{ in } gc \text{ (or } cc \text{ or } mc). \text{ Let } M' \text{ be a query identical to } M \text{ apart from } gc \text{ (or } cc \text{ or } mc) \text{ in which } Q(b) \text{ is added to the condition } (gc_M = gc_M \land Q(b)). \text{ Under these conditions, } M' = M. \text{ The same holds if } M' \text{ is obtained from } M \text{ by substitution of } Q(b) \text{ with } Q(b). \quad & \square \\
\end{align}

3.1.1 Relationships between queries with reference to candidate keys

In the analysis that follows, when we consider predicates in gc, mc or cc of the MINE RULE queries we will refer to equivalent predicates, denoted by P(a) and Q(b). In Table 2 we report the relationships between two classes of MINE RULE queries when candidate keys are present. The two classes are identical apart from a single instantiated clause in each of them. Each of the instantiated clauses is indicated in one dimension of Table 2: the first query class, named M, is specified by the vertical dimension and its instantiated clause is defined with the assignment of a to the attribute list ga (or ca or ra) or with the predicate P(a) to the condition gc (or cc or mc). The second query class, named M', is specified by the horizontal dimension and its instantiated clause is defined with the assignment of b to ga (or ca or ra) or with the assignment of Q(b) to the condition gc (or cc or mc). Furthermore, note that when P(a) is assigned to gc (or cc) it is an aggregate function expressed on the groups (clusters) and therefore it expects the grouping (clustering) is made by a. This means that ga=a (or ca=a). The analogous reasoning holds for P(b) and b.

To explain Table 2 we take as example the cell in the sixth row and second column (identified by the pair of assignments mc=P(a) and ca=b) that is used to compare the following two classes of MINE RULE queries.

\begin{align}
M: & \quad \text{MINE RULE <OutputTable_for_M> AS SELECT DISTINCT 1..n <ruleAttr> AS BODY, 1..n <ruleAttr> AS HEAD, SUPPORT, CONFIDENCE WHERE Q(b)} \\
& \text{FROM <sourceTable> GROUP BY <groupAttr> [HAVING <groupCondition>] EXTRACTING RULES WITH SUPPORT:<mc>, CONFIDENCE:<mC>}

M': & \quad \text{MINE RULE <OutputTable_for_M> AS SELECT DISTINCT 1..n <ruleAttr> AS BODY, 1..n <ruleAttr> AS HEAD, SUPPORT, CONFIDENCE WHERE Q(b)} \\
& \text{FROM <sourceTable> GROUP BY <groupAttr> [HAVING <groupCondition>] CLUSTER BY b EXTRACTING RULES WITH SUPPORT:<mc>, CONFIDENCE:<mC>}
\end{align}

Inside each cell of Table 2 we report the relationship between M and M'. When the cell is left empty nothing can be stated. If a cell contains NA it means that the comparison between the two classes of queries is Not Applicable because sufficient information is not available. In the case of the MINE RULE operator this is the case of the columns (rows) of ga and ra: if one of them is completely specified in a query class and not in the other class, it is impossible to compare the result sets of the two queries (unless supposed they are equal). Indeed, these two classes (grouping and rule attribute selection) are the mandatory clauses without which a MINE RULE query is not meaningful.

With Table 2 we compare all the pairs of simplest, "basic" query classes and report their relationships. For the comparison of more complex queries, that exhibit a greater number of differing clauses you need to apply Theorems 2 and 3. This methodological way of comparing two classes of queries is valid for whichever query language and is not limited at all to the only MINE RULE operator. Indeed you need just to reserve one row/column of the table for each clause of the statement and a dimension of the table for each of the two query classes that you want to compare.

The detailed explanation of the content of Table 2 (and of the tables that follow) is omitted for lack of space; in [2] you can find the complete discussion. However it is important to observe that the equivalence relationship holds for every cell in the diagonal of Table 2 and this can be proved with the aid of Theorems 4 and 5. Notice also that the table is symmetrical with respect to the diagonal. This is easy to understand for the symmetrical nature of the candidate keys and of the equivalent predicates.

3.1.2 Determination of equivalent queries with reference to candidate keys

Now, we analyze the case in which the same query contains references to both candidate keys (a and b). In this case, we consider the classes of queries that are obtained by placing the two attributes in all the possible pairs of clauses. Table 3 shows all these pairs of clauses and therefore each cell of Table 3 identifies a single class of queries. For instance, the cell in the upper right corner corresponding to ga=a and mc=Q(b), identifies the following class of MINE RULE queries.

\begin{align}
\text{MINE RULE <OutputTable> AS SELECT DISTINCT 1..n <ruleAttr> AS BODY, 1..n <ruleAttr> AS HEAD, SUPPORT, CONFIDENCE WHERE Q(b)} \\
& \text{FROM <sourceTable> GROUP BY a [HAVING <groupCondition>] CLUSTER BY <clustAttr> [HAVING <clustCondition>] EXTRACTING RULES WITH SUPPORT:<mc>, CONFIDENCE:<mC>}
\end{align}

Inside each cell of Table 3 we specify the modifications to the query that allow us to obtain a new class of queries, equivalent to the previous class, i.e., such that their instances return the same result set, and this holds for every database. In particular, we specify in the row labelled out which of the two classes can be omitted and in the row labelled in which new clause can be inserted in order to obtain the equivalent query class. If a cell is empty it means that nothing can be obtained for that combination of clauses. If a cell contains NA it means that the combinations of clauses is Not Applicable because it constitutes a meaningless combination of clauses. This is the case of ga=a and ra=b and vice versa (and also gc=a and ra=b and
Table 3: Determining the equivalent MINE RULE queries in presence of candidate keys.

Table 4: The relationships between two MINE RULE queries in presence of functional dependence.

In this section we study the relationships between two MINE RULE queries when a functional dependence exists between two attributes, a and b that belong to the same table (or view) in the FROM clause. Furthermore, we suppose that each query refers by means of one of its clauses to at least one of them (a or b).

Definition 3: We denote by a → b a functional dependence between a and b. In other words, we mean that there exists a function F such that given a value v_a ∈ Dom_a there exists a single value v_b ∈ Dom_b such that F(v_a) = v_b. □

Definition 4: Let be a → b. We call P(a) and Q(b) functional preserving predicates if for each pair of values (v_a ∈ Dom_a, v_b ∈ Dom_b) such that P(v_a) = v_a then P(v_a) → Q(v_b). □
ogous reasoning of Table 2. However, you can refer to [2] for a complete discussion.

Now, we analyze the case in which the same query contains both attributes related by a functional dependence (Table 5). In this case, we identify classes of queries that are obtained by placing the two attributes in all the possible pairs of clauses. Table 5 is analogous to Table 3, but designed for functionally dependent attributes and functional preserving predicates. For lack of space we will not discuss in detail the content of Table 5 which can be easily derived applying analogous reasoning of Table 3. However, you can refer to [2] for a complete discussion.

The results contained in the Tables 2–5, provide us important information that allow to determine precisely which queries are equivalent or with a dominance relationship. In the next Sections we will discuss in detail the classes of algorithms that can be adopted for these classes of queries. Furthermore, they allow us to observe that the cases of inclusion relationships are not frequent. These results are the premise that allow us to start with the construction of an MINE RULE optimizer and more in general to afford the optimization of mining queries with constraints.

4. THE ALGORITHMS FOR SOME TYPICAL CASES

The problem of association rules extraction typically requires tasks that are computationally expensive; therefore it is important to reduce them whenever possible. The incremental computation of the result set of a query \( M \) starting from the result set of a previously executed query \( M \) can provide significant reduction of the computational work performed by the system. In the general case, the problem is difficult to solve for several reasons. One of the reasons is that the association rules extraction is executed by a set of highly optimized procedures. Another reason is that pattern extraction in knowledge discovery is a task that performs aggregation of data. Under these conditions it is difficult to restore the original values from the aggregated ones, unless suitable intermediate information is maintained. This occurs in the data warehouse framework in which summarized data are maintained so that the computation of the aggregated data can be performed efficiently and at the same time, original data can be restored from them. This occurs also in the database mining framework in which given an itemset with an aggregated value (its support value) we need to efficiently retrieve all the rows of the original relation in which the itemset is present. This problem occurs typically during the incremental evaluation of a query \( M' \) from the result set of a previously executed query \( M \): the itemset whose rows have to be retrieved belongs to the result set of \( M \), and the user-defined constraint on the attributes of those rows (such as the mining condition in MINE RULE) belongs to \( M' \). This issue will be discussed in detail in the following section on the intermediate data structure design.

However, in some particular case, incremental computation of the association rules is possible and decidable with ease. This occurs when certain relationships hold between the queries and certain conditions that we will discuss in the following are met. In particular,

**Equivalence:** if \( M \equiv M' \) then no computation is required because \( R = R' \).

Inclusion: if \( M \sqsupseteq M' \) then \( R' \) can be obtained by \( R \) without scanning the source data, but only with a scan of the rules in \( R \) in order to select those rules that are also in \( R' \). Recall that this scan of the rule set \( R \) is sufficient because \( M \sqsupseteq M' \) occurs when the changed requirements of \( M' \) w.r.t. \( M \) (like cardinality or support) can be checked directly in the relation containing the result set \( R \).

Dominance: if \( M \succ M' \) then \( R' \) can be obtained by \( R \) with a scan of \( R \) and a single scan of the source data, needed in order to derive the correct values of support and confidence of the rules in \( R' \).

In [1, 2] the schema of an incremental computation for the case of dominance between two queries is reported. The algorithm considers each rule of \( R \) as a candidate rule for \( R' \) and recomputes the rule support for all the groups in which the rule can be found. If the resulting support or confidence of the rule are not sufficient the rule is not included in the output rule set \( R' \).

This algorithm presents the problem that it performs a full scan of the database with the only purpose to computing the support and confidence values of a given set of rules. In theory, it would be sufficient to read only the portion of the database in which the given rules are present. This would be possible only if we were able to provide the system with more sophisticated data structures. This is the issue discussed in the following section. Furthermore, in Section 4.2, we present an enhanced algorithm that makes use of this data structure.

4.1 Design of Intermediate Data Structures: the Mining Indices

In this Section we describe the intermediate data structures that are used by more general incremental algorithms in order to retrieve from the database only those data that are effectively needed. We designed these data structures with the following idea in mind. In traditional databases, an index is used by the system to make efficient access to a tuple searched by the value of some of its attributes. A similar need is present in the case of association rule mining. Given an itemset \( I \), from which a rule \( r \) in \( R \) can be generated, we need to retrieve all the groups in which \( I \) is present. This is because we want to evaluate the constraints of the new query \( M' \) on the rule \( r \) in order to compute its support.

In theory, each itemset could have an entry in this data structure that we call **mining index**. In practice, this is not needed. Recall that the theory on anti-monotone and succinct constraints [12, 13] suggests that if a constraint is both anti-monotone and succinct at the same time (as most of the constraints in MINE RULE), it is sufficient to check it only for the first level of the lattice (i.e., only for the single items). Then, the same constraints are **by sure** also satisfied by all the itemsets that will be derived by combining the items at the first step. In this way, the constraints are effectively used to efficiently prune the lattice of the itemsets.

In conclusion, a mining index is designed as follows. We maintain an efficient and ordered access to the items (for instance by means of a hash table or a tree such as B+tree); for each entry we have all the identifiers of the groups in which that item is present. We indicate this data structure \( I_{in \_m} \) if the index is created on the item attribute. We will see that also other indices, on some other attribute list,
will be useful. For instance, if a MINE RULE query with a mining condition on attribute price (such as \texttt{BODY.price > 100}) is submitted, the system can conveniently use also the mining index \(I_{price}\) on the mining attribute. This index keeps the values of the price attribute ordered and for each value it gives the list of group identifiers in which there is a tuple with that value of price. In this way it is relatively easy to check if a rule in a previous result set still satisfies the constraints of the new query. This issue will be discussed in detail in the following Section.

### 4.2 A New Enhanced Incremental Algorithm

With the above observations, we propose a new enhanced incremental algorithm that works as follows. For the purposes of the description, to be concrete, we make use of a practical example. Suppose that a MINE RULE query \(M\) with the mining condition \(\texttt{BODY.price < 150}\) has been already executed. After some time, another query is submitted, \(M'\), that is identical to \(M\) apart from a tighter mining condition \(\texttt{BODY.price < 100}\). You can see that \(M \Rightarrow M'\) because between the mining conditions of \(M\) and \(M'\) there is an implication relationship.

The intuition tells us that it is not necessary to read all the database in order to select from \(R\) those rules that are valid also in \(R'\). It suffices to read only the groups in which at least a rule of \(R\) is present. In this way to evaluate the constraints of the query \(M'\) on the tuples of the original relation. But perhaps we can do better and exploit the constraints that are present in this particular query. Actually, it might be necessary to read only the groups that contain both a rule in \(R\) and a tuple with price that gets a value that satisfies the query constraint (price < 100 that we denote with \(OK_{price}\)). However, if the cardinality of the complementary domain (100 ≤ price < 150 denoted with \(\texttt{NOprice}\)) is lower a destructive approach is used and the new result set \(R'\) is obtained by elimination of a selected subset of rules from \(R\). In the following (Figure 1), we give the algorithm with the constructive approach and show the use of the mining index.

The algorithm takes as input the result set \(R\) of \(M\) and gives in output the result set of \(M'\): at first, searching in \(I_{price}\) it builds the gid-list \(g\text{-list-p}\) containing the group id of groups with at least a tuple with a valid price. Then, for each rule \(r\) in \(R\), obtain \(g\text{-list-temp}\) with the group id of groups satisfying both \(r\) and a valid price. Then the gid-list \(g\text{-list-db}\) containing the id of the groups to be read from the database is formed. During the reading phase of the database only this portion of the database is scanned and from it the rules satisfying \(M'\) are generated (in \(RS\)).

Among these rules, only those ones that are already in \(R\) are inserted in \(R'\) and kept if their support and confidence are sufficient (these ones can be determined by the gid-list associated to the rule).

This algorithm is more efficient than the original one in [1] whose complexity is \(O(N)\) (with \(N\) the dimension of the database, see [16]), while the discussed algorithm has complexity \(O(N*f)\) where \(f\) is the selectivity of the constraints in \(M'\) that are evaluated when searching groups in the mining index.

<table>
<thead>
<tr>
<th>(a)</th>
<th>(ga)</th>
<th>(b)</th>
<th>(ca)</th>
<th>(ra)</th>
<th>(gc)</th>
<th>(Q(b))</th>
<th>(oc)</th>
<th>(mc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ga)</td>
<td>out</td>
<td>(b)</td>
<td>(b)</td>
<td>(NA)</td>
<td>(Q(b))</td>
<td>(ca=b, Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
</tr>
<tr>
<td>(ca)</td>
<td>out</td>
<td>(b)</td>
<td>(NA)</td>
<td>(b)</td>
<td>(NA)</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
</tr>
<tr>
<td>(ra)</td>
<td>out</td>
<td>(NA)</td>
<td>(b)</td>
<td>(NA)</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
</tr>
<tr>
<td>(gc)</td>
<td>out</td>
<td>(b)</td>
<td>(b)</td>
<td>(NA)</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
</tr>
<tr>
<td>(cc)</td>
<td>out</td>
<td>(gc=Q(b))</td>
<td>(b)</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td></td>
</tr>
<tr>
<td>(mc)</td>
<td>out</td>
<td>(gc=Q(b))</td>
<td>(cc=Q(b))</td>
<td>(mc=Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td>(Q(b))</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(F(a))</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Determining the equivalent MINE RULE queries in presence of functional dependence.
database in favour of a scan of the FD table (whose cardinality is far lower than the cardinality of the database since \( |FD| = \Pi_{\text{sup}}(\text{price}) \)).

We report in Figure 2 the sketch of the algorithm, in general, there may be two subsets (possibly disjoint) of the values of the price domain that are allowed by the query \( M \): one for the body elements (denoted by \( \text{OK}_{\text{item}, \text{price}} \)), and another for the head elements (\( \text{OK}_{\text{item}, \text{price}} \)). For each value of price in these two sets we obtain the set of values of item that are allowed in the body or in the head of the rules (resp. \( i\text{-list-b} \) and \( i\text{-list-h} \)). This is an immediate operation that is performed by looking in the table \( FD(\text{item}, \text{price}) \). Then the algorithm performs one scan over \( R \) and copies in \( R' \) the rules whose items are found in the previously identified lists. Notice that in this case the algorithm performs only a scan of the rule set \( R \) and completely avoids the scan of the database. In other words, the algorithm is transformed from a "dominance" algorithm into a (more efficient) "inclusion" algorithm that is characterized by a lower degree of coupling with the database.

![Figure 2: The generation of \( R' \) from \( R \) when functional dependencies are present.](image)

```plaintext
Input: \( R \); Output: \( R' \).
for all \( p \in \text{OK}_{\text{item}, \text{price}} \) do // cycle for the body
    \( i\text{-list-b} = i\text{-list-b} \cup \sigma_{\text{price}(FD(\text{item}, \text{price}))} \);
for all \( p \in \text{OK}_{\text{item}, \text{price}} \) do // cycle for the head
    \( i\text{-list-h} = i\text{-list-h} \cup \sigma_{\text{price}(FD(\text{item}, \text{price}))} \);
for all rules \( r \in R \) do
    if all items in \( r \) body are in \( i\text{-list-b} \) and all items in \( r \) head are in \( i\text{-list-h} \) then
        add \( r \) to \( R' \);
```

5. CONCLUSIONS

This paper is a first step towards the construction of optimizers for a constraint based mining language. It provides the guidelines for the comparison of classes of queries by means of the existing relationships between their result sets. This is the premise to allow the reuse of the result sets of the queries when certain conditions are met. Furthermore in this paper we identify the database schema informations (unique constraints and functional dependencies) that can be exploited for these purposes. In these cases we show that the volume of the database be processed by the mining engine is greatly reduced or completely avoided in a large subset of the cases. Furthermore, we identify a set of intermediate data structures called the mining indices specifically designed in order to exploit previous query executions. We show the practical implications of the discussed principles with a set of algorithms designed for a specific mining language, the \textsc{Mine Rule} operator.

Future work is to provide an implementation of the set of algorithms proposed here, taking into consideration the previous work on constraint-based mining and condensed representations. These ones, indeed, exploit the specific properties of each constraints (such as anti-monotonicity and succinctness) in order to reduce at a minimum the global workload.

6. ACKNOWLEDGMENTS

This work has been funded by EU FET project clinQ.

7. REFERENCES


