A Relational Query Primitive for Constraint-Based Pattern Mining

Francesco Bonchi[1], Fosca Giannotti[1], and Dino Pedreschi[2]

[1] ISTI - CNR, Area della Ricerca di Pisa, Via Giuseppe Moruzzi, 1 - 56124 Pisa, Italy
[2] Dipartimento di Informatica, Via F. Buonarroti 2, 56127 Pisa, Italy

Abstract. As a step towards the design of an Inductive Database System, in this paper we present a primitive for constraint-based frequent pattern mining, which represents a careful trade-off between expressiveness and efficiency. Such primitive is a simple mechanism which takes a relational table in input and extracts from it all frequent patterns which satisfy a given set of user-defined constraints. Despite its simplicity, the proposed primitive is expressive enough to deal with a broad range of interesting constraint-based frequent pattern queries,using a comprehensive repertoire of constraints defined over SQL aggregates. Thanks to its simplicity, the proposed primitive is amenable to be smoothly embedded in a variety of data mining query languages and be efficiently executed, by the state-of-the-art optimization techniques based on pushing the various form of constraints by means of data reduction.

1 Introduction

Typically, two different kinds of structures are sought in data mining: *models* and *patterns* [31]. Models are high level, global, descriptive summaries of data sets. Patterns, on the other hand, are local descriptive structures. Patterns may be regarded as *local models*, and may involve just a few points or variables; i.e., they are descriptions of small fragments of the data, instead of overall descriptions. Accordingly, *Pattern Discovery* has a distinguished role within data mining technology. In particular, since *frequency* provides support to any extracted knowledge, it is the most used measure of interest for the extracted patterns. Therefore during the last decade a lot of researchers have focussed their studies on the computational problem of *Frequent Pattern Discovery*, i.e., mining patterns which satisfy a user-defined minimum threshold of frequency [3, 30].

The simplest form of frequent pattern is the frequent itemset: given a database of transactions (a transaction is a set of items) we want to find those subsets of transactions (itemsets) which appear together frequently.

Definition 1 (Frequent Itemset Mining). Let $\mathcal{I} = \{x_1, ..., x_n\}$ be a set of distinct literals, usually called *items*, where an item is an object with some predefined attributes (e.g., price, type, etc.). An *itemset* X is a non-empty subset of $\mathcal{I}$. If $|X| = k$ then X is called a *k-itemset*. A *transaction database* $\mathcal{D}$ is a bag of itemsets $t \in 2^{\mathcal{I}}$, usually called *transactions*. The *support* of an itemset X in

J.-F. Boulicaut et al. (Eds.): Constraint-Based Mining, LNAI 3848, pp. 14–37, 2005.

date	cust	item
11-2-97	cust1	beer
11-2-97	cust1	chips
11-2-97	cust1	wine
11-2-97	cust2	wine
11-2-97	cust2	beer
11-2-97	cust2	pasta
11-2-97	cust2	chips
13-2-97	cust1	chips
13-2-97	cust1	beer
13-2-97	cust2	jackets
13-2-97	cust2	col_shirts
13-2-97	cust3	wine
13-2-97	cust3	beer
15-2-97	cust1	pasta
15-2-97	cust1	chips
16-2-97	cust1	jackets
16-2-97	cust2	wine
16-2-97	cust2	pasta
16-2-97	cust3	chips
16-2-97	cust3	col_shirts
16-2-97	cust3	brown_shirts
18-2-97	cust1	pasta
18-2-97	cust1	wine
18-2-97	cust1	chips
18-2-97	cust1	beer
18-2-97	cust2	beer
18-2-97	cust2	beer
18-2-97	cust2	chips
18-2-97	cust2	chips
18-2-97	cust3	pasta

(a)

date	cust	itemset
11-2-97	cust1	{beer,chips,wine}
11-2-97	cust2	{wine,beer,pasta,chips}
13-2-97	cust1	{chips,beer}
13-2-97	cust2	{jackets,col_shirts}
13-2-97	cust3	{wine,beer}
15-2-97	cust1	{pasta,chips}
16-2-97	cust1	{jackets}
16-2-97	cust2	{wine,pasta}
16-2-97	cust3	{chips,col_shirts,brown_shirts}
18-2-97	cust1	{pasta,wine,chips,beer}
18-2-97	cust2	{beer,chips}
18-2-97	cust3	{pasta}

(b)

name	price	type
beer	10	beverage
chips	3	snack
wine	20	beverage
pasta	2	food
jackets	100	clothes
col_shirt	30	clothes
brown_shirt	25	clothes

(c)

Fig. 1. (a) A sample `sales` table, (b) its transactional representation and (c) the `product` table

database $\mathcal{D}$, denoted $supp_{\mathcal{D}}(X)$, is the number of transactions in $\mathcal{D}$ which are superset of X. Given a user-defined *minimum support* σ, an itemset X is called *frequent* in $\mathcal{D}$ if $supp_{\mathcal{D}}(X) \geq \sigma$.

The problem of mining all frequent itemsets in a database is the basis of the well-known association rule mining task [1]. However, frequent itemsets are meaningful not only in the context of association rules: they can be used as a basic mechanism in many other kind of analysis, ranging from classification [39, 40] to clustering [50, 58].

Example 2 (Market Basket Analysis). The classical context for association rule mining, as well as the most natural way to think about a transaction database, is the market basket setting, where we have a *sales database* of a retail store recording the content of each basket appearing at the cash register. In this context a transaction represent the content of a basket. In Figure 1(a) we have a relational table where a transaction or basket identifier is not explicitly given; however one could reconstruct transactions, for instance, grouping items by the pair (`date`, `cust`) as represented in Figure 1(b).

In principle, it is possible to express a query to count frequent itemsets in conventional SQL. This approach is examined in [57, 32, 52, 2]. For example,

in Figure 2 it is shown how to compute all 2-itemsets which are frequent in a relational database. We join `sales` with itself, with the condition that `tID` (transaction identifier) is the same, and the names of the two items are lexicographically different. We group the joined relation by the pair of items involved and check in the having clause that the group has at least 2 transactions.

```
SELECT  i1.item, i2.item
FROM  sales AS i1, sales AS i2
WHERE i1.item < i2.item AND
      i1.tID = i2.tID
GROUP BY i1.item, i2.item
HAVING 2 <= COUNT(i1.tID)
```

Fig. 2. Frequent 2-itemsets computation in SQL

The problem with this approach is that the right optimizations and "tricks" are beyond the state-of-the-art of conventional optimizers in commercial DBMS. Therefore a DBMS-based approach can not compete with specific algorithms employing ad hoc data structures. This is one of the main reasons, together with the stronger requirements of expressiveness, that justifies the need for specialized primitives and query languages for knowledge discovery.

1.1 Constraint-Based Pattern Discovery

Recently the research community has turned its attention to more complex kinds of frequent patterns extracted from more structured data: sequences, trees, and graphs. All these different kinds of patterns have different peculiarities and application fields, but they all share the same computational aspects: a usually very large input, an exponential search space, and a too large solution set. This situation – too many data yielding too many patterns – is harmful for two reasons. First, performance degrades: mining generally becomes inefficient or, often, simply unfeasible. Second, the identification of the fragments of interesting knowledge, blurred within a huge quantity of mostly useless patterns, is difficult.

Therefore, the paradigm of *constraint-based mining* was introduced. Constraints provide focus on the interesting knowledge, thus reducing the number of patterns extracted to those of potential interest. Additionally, they can be pushed deep inside the pattern discovery algorithm in order to achieve better performance [7, 4, 6, 9, 36, 37, 19, 25, 29, 38, 48, 49, 56, 44].

Definition 3 (Constrained Frequent Itemset Mining). A constraint on itemsets is a function $\mathcal{C} : 2^{\mathcal{I}} \rightarrow \{true, false\}$. We say that an itemset I satisfies a constraint if and only if $\mathcal{C}(I) = true$. We define the *theory* of a constraint as the set of itemsets which satisfy the constraint: $Th(\mathcal{C}) = \{X \in 2^{\mathcal{I}} \mid \mathcal{C}(X)\}$. Thus with this notation, the *frequent itemsets mining problem* requires to compute the set of all frequent itemsets $Th(\mathcal{C}_{freq[\mathcal{D},\sigma]})$. In general, given a conjunction of constraints $\mathcal{C}$ the *constrained frequent itemsets mining problem* requires to compute $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$.

According to the constraint-based mining paradigm, the data analyst must have a high-level vision of the pattern discovery system, without worrying about the details of the computational engine, in the very same way a database designer has not to worry about query optimizations. The analyst must be provided with a set of primitives to be used to communicate with the pattern discovery system, using a *Pattern Discovery Query Language*. The analyst just needs to declaratively specify in the pattern discovery query how the desired patterns should look like and which conditions they should obey (a set of constraints). Such rigorous interaction between the analyst and the pattern discovery system, can be achieved by means of a set of *pattern discovery primitives*, that should include:

- the specification of the source data,
- the kind of pattern to be mined,
- background or domain knowledge,
- the representation of the extracted patterns,
- constraints that interesting patterns must satisfy,
- interestingness measures for patterns evaluation.

Providing a query language capable to incorporate all these features may result, like in the case of relational databases, in a high degree of expressiveness in the specification of pattern discovery tasks, a clear and well-defined separation of concerns between logical specification and physical implementation of such tasks, and easy integration with heterogeneous information sources.

Clearly, the implementation of this vision presents a great challenge. A path to this goal is indicated in [41] where Mannila introduces an elegant formalization for the notion of interactive mining process: the term *inductive database* refers to a relational database plus the set of all sentences from a specified class of sentences that are true w.r.t. the data. In other words, the inductive database is a database framework which integrates the raw data with the knowledge extracted from the data and materialized in the form of patterns. In this way, the knowledge discovery process consists essentially in an iterative querying process, enabled by a query language that can deal either with raw data or patterns.

Definition 4. Given an instance $\mathbf{r}$ of a relation $\mathbf{R}$, a class $\mathcal{L}$ of sentences (patterns), and a selection predicate q, a pattern discovery task is to find a theory

$$\mathcal{T}h(\mathcal{L}, \mathbf{r}, q) = \{s \in \mathcal{L} | q(\mathbf{r}, s) \text{ is true}\}$$

The selection predicate q indicates whether a pattern s is considered interesting. In the constraint-based paradigm, such selection predicate q is defined by a conjunction of constraints.

1.2 Paper Contribution and Organization

As a step towards the design of an Inductive Database System, in this paper we present a primitive for constraint-based frequent pattern mining in a relational context. Such primitive is a simple mechanism which takes a relational table

in input and extracts from it all frequent patterns which satisfy a given set of user-defined constraints. Despite its simplicity, the proposed primitive is expressive enough to deal with a broad range of interesting constraint-based frequent pattern queries,using a comprehensive repertoire of constraints defined over SQL aggregates.

In this paper we do not deal specifically with the Data Mining Query Language issue, as our primitive can be embedded in any kind of query language based on the relational paradigm (SQL-like databases, logic-based deductive databases); basically we can view our primitive as an operator of the relational algebra.

Therefore, after introducing the primitive for constraint-based pattern mining we informally assess its expressiveness by means of example queries in SQL-like and DATALOG-like syntax. Then we concentrate on optimization issues: we summarize and amalgamates all the algorithmic results in constraint-based frequent pattern discovery obtained in the last years (2003-05) at Pisa KDD Laboratory, thus achieving an optimization framework.

The paper is organized as follows. In the next Section we define our primitive going through a rigorous identification of its basic components and of the constraints handled. In Section 3, we advocate the expressiveness and versatility of our proposal, showing how it can be embedded in query languages of different flavour. In Section 4 we recall the state-of-the-art classification of constraints and their properties, which can be exploited in order to achieve an efficient computation. In Section 5 we compose the state-of-the-art of the constraint pushing techniques in a breadth-first Apriori-like computation, achieving a very efficient evaluation strategy for our primitive. In particular, we adopt the strategy of pushing constraints in the computation mainly by means of data-reduction techniques: this approach enables us to exploit the different properties of constraints all together, and the total benefit is always greater than the sum of the individual benefits. Finally in Section 6 we conclude by describing on-going work on constrained frequent pattern discovery at Pisa KDD Laboratory, and drawing some future research paths.

2 A Primitive for Constraint-Based Mining

In this Section, going through a rigorous identification of all its basic components, we provide a definition of a primitive for constraint-based frequent pattern mining task over a relational database DB.

The first needed component is the data source: which table must be mined for frequent patterns, and which attributed do identify transactions and items.

Definition 5 (Mining View). Given a database DB any relational expression $\mathcal{V}$ on *preds*(DB) can be selected as data source, and named *mining view*.

Definition 6 (Transaction id). Given a database DB and a relation $\mathcal{V}$ derived from *preds*(DB). Let $\mathcal{V}$ with attributes $sch(\mathcal{V})$ be our mining view. Any subset of attributes $\mathcal{T} \subset sch(\mathcal{V})$ can be selected as transaction identifier, and named *transaction id*.

Definition 7 (Item attribute). Given a database DB and a relation $\mathcal{V}$ derived from *preds*(DB). Let $\mathcal{V}$ with attributes $sch(\mathcal{V})$ be our mining view. Given a subset of attributes $\mathcal{T} \subset sch(\mathcal{V})$ as transaction id, let $Y = \{y | y \in sch(\mathcal{V}) \backslash \mathcal{T} \wedge \mathcal{T} \rightarrow y \textit{ does not hold}\}$; we define an attribute $\mathcal{I} \in Y$ an *item attribute* provided the functional dependency $\mathcal{T}\mathcal{I} \rightarrow Y \backslash \mathcal{I}$ holds in DB.

Proposition 8. *Given a relational database* DB*, a triple* $\langle \mathcal{V}, \mathcal{T}, \mathcal{I} \rangle$ *denoting the mining view* $\mathcal{V}$*, the transaction id* $\mathcal{T}$*, the item attribute* $\mathcal{I}$*, uniquely identifies a transactional database, as defined in Definition 1.*

We next distinguish between attributes which describe items (descriptive attributes), from attribute which describe transactions (circumstance attributes).

Definition 9 (Circumstance attribute). Given a database DB and a relation $\mathcal{V}$ derived from *preds*(DB). Let $\mathcal{V}$ with attributes $sch(\mathcal{V})$ be our mining view. Given a subset of attributes $\mathcal{T} \subset sch(\mathcal{V})$ as transaction id, we define any attribute $\mathcal{A} \in sch(R)$ where R is a relation in *preds*(DB) *circumstance attribute* provided that $\mathcal{A} \notin \mathcal{T}$ and the functional dependency $\mathcal{T} \rightarrow \mathcal{A}$ holds in DB.

Definition 10 (Descriptive attribute). Given a database DB and a relation $\mathcal{V}$ derived from *preds*(DB). Let $\mathcal{V}$ with attributes $sch(\mathcal{V})$ be our mining view. Given a subset of attributes $\mathcal{T} \subset sch(\mathcal{V})$ as transaction id, and given $\mathcal{I}$ as item attribute; we define *descriptive attribute* any attribute $\mathcal{A} \in sch(R)$ where R is a relation in *preds*(DB), provided the functional dependency $\mathcal{I} \rightarrow \mathcal{A}$ holds in DB.

Consider the mining view: `sales(tID, locationID, time, product, price)` where each attribute has the intended semantics of its name and with `tID` acting as the transaction id. Since the functional dependency {`tID`} → {`locationID`} holds, `locationID` is a circumstance attribute. The same is true for `time`. We also have {`tID, product`} → {`price`}, and {`product`} → {`price`}, thus `product` is an item attribute, while `price` is a descriptive attribute.

Note that, from the previous definitions, *transaction id* and the *item attribute* must be part of the mining view, while circumstance and descriptive attributes could be also in other relations.

Constraints, as introduced in the previous Section (see Definition 3), describes properties of itemsets, i.e., a constraint $\mathcal{C}$ is a boolean function over the domain of itemsets: $\mathcal{C} : 2^{\mathcal{I}} \rightarrow \{true, false\}$. According to this view, constraints are only those ones defined on item attributes (Definition 7) or descriptive attributes (Definition 10).

Constraints defined over the *transaction id* (Definition 6) or over circumstance attributes (Definition 9) are not constraints in the strict sense. Indeed, they can be seen as selection conditions on the transactions to be mined and thus they can be satisfied in the definition of the *mining view*. Consider the relation: `sales(tID, locationID, time, product, price)` where each attribute has the intended semantics of its name and with `tID` acting as the transaction id. Since the functional dependency {`tID`} → {`locationID`} holds, `locationID` is a circumstance attribute. The constraints `locationID` ∈ {`Florence, Milan,`

`Rome`} is not a real constraint of the frequent pattern extraction, indeed it is a condition in the mining view definition, i.e., it is satisfied by imposing such condition in the relational expression defining the mining view (a `select` statement if we embed our primitive in a SQL-like language).

The following Definition lists all kinds of constraints that we consider as possible input for our primitive.

Definition 11 (Constraints on itemsets). In Table 1 all constraints admitted in our primitive are listed. The following notation is adopted:

- s *is an itemset;*
- $a_1, \ldots, a_n$ *are items;*
- d *is a descriptive attribute;*
- $d_1, \ldots, d_n$ *are values of a descriptive attribute;*
- m *is a numeric constant;*
- $\theta \in \{\leq, \geq, =\}$
- $aggr \in \{min, max, sum, avg, count, range, avg, median, var, std, md\}$

Table 1. Constraints admitted in our primitive

Constraint	Description
$s \supseteq \{a_1, \ldots, a_n\}$	itemset contains
$s \subseteq \{a_1, \ldots, a_n\}$	itemset domain
$count(s)\,\theta\,m$	itemset cardinality
$s.d \supseteq \{d_1, \ldots, d_n\}$	descriptive attribute contains
$s.d \subseteq \{d_1, \ldots, d_n\}$	descriptive attribute domain
$aggr(s.d)\,\theta\,m$	aggregate on descriptive attribute

We have provided all the needed components of our primitive, thus we are now ready to introduce it.

Definition 12 (Primitive for constraint-based itemset mining). Given a database DB, let the quintuple $\langle \mathcal{V}, \mathcal{T}, \mathcal{I}, \sigma, \mathcal{C} \rangle$ denotes the mining view $\mathcal{V}$, the transaction id $\mathcal{T}$, the item attribute $\mathcal{I}$, the minimum support threshold σ, and a conjunction of constraints on itemsets $\mathcal{C}$.

The primitive for constraint-based itemset mining takes in input such quintuple and returns a binary relation recording the set of itemsets which satisfy $\mathcal{C}$ and are frequent (w.r.t. σ) in the transaction database $\langle \mathcal{V}, \mathcal{T}, \mathcal{I} \rangle$, and their supports:

$$freq(\mathcal{V}, \mathcal{T}, \mathcal{I}, \sigma, \mathcal{C}) = \{(I, S) \mid \mathcal{C}(I) \ \wedge \ supp_{\langle \mathcal{V}, \mathcal{T}, \mathcal{I} \rangle}(I) = S \ \wedge \ S \geq \sigma\}$$

Example 13. A frequent pattern query for the `sales` table in Figure 1 (a), and the `product` table in Figure 1 (c), querying itemsets having a support ≥ 3 (transactions are made grouping by customer and date), and having a total price ≥ 30, could be simply defined as:

$freq(sales, \{date, cust\}, item, 3, sum\{p \mid i \in I \wedge product(i,p,t)\} \geq 30) = \{(I,S) \mid$
$sum\{p \mid i \in I \wedge product(i,p,t)\} \geq 30 \wedge supp_{\langle sales,\{date,cust\},item\rangle}(I) = S \wedge S \geq 3\}$
The result of such query is a relation `(I,S)` with the following two entries: `({beer, wine},4)` and `({beer, wine, chips},3)`.

3 Embedding the Primitive into a Query Language

The issue of designing a query language capable of dealing with all requirements of knowledge discovery process, including definition of interestingness measures for extracted patterns and ad hoc exploitation of the application specific background knowledge, is a prominent research goal in data mining. This issue has been tackled both by a database perspective and a machine learning perspective (see [24] for a thorough discussion).

The proposal by a database perspective [33] is to combine relational query languages with data mining primitives in an overall framework capable of specifying data mining problems as an iterative and interactive querying session, where queries can involve both data and extracted models or patterns, and the result of a query becomes available for further querying (*closure principle*). In such a knowledge discovery system, identified by the term *inductive database*, query optimization and execution techniques typically rely on advanced ad hoc data mining algorithms. Past efforts for developing such languages can be classified in two categories:

- Developing mining tools tightly integrated with SQL DBMSs, representing both the source data and the induced patterns in database relations. Mining queries are specified in an SQL-like language [42, 43, 34, 35, 16, 27, 26, 55, 28, 29, 44].
- Exploiting logic to encode ad hoc data mining tasks and to specify background knowledge using the same language: typically some DATALOG extension [57, 53, 54, 21, 22, 14, 23].

On the machine learning side the main effort has been devoted to upgrading existing "propositional" data mining techniques to first order logic. This approach is denoted Inductive Logic Programming (ILP). ILP systems construct logic programs from examples (both positive and negative) and background knowledge: the challenge is to find a hypothesis that is consistent (w.r.t. negative examples) and complete (w.r.t. positive examples). The background knowledge and the hypothesis are expressed in two (not necessarily distinct) languages that are fixed in advance. In recent years, ILP has broadened its scope to cover standard data mining tasks such as classification, regression, clustering and association rules. This research is triggered by the need to pass from single-relational to *multirelational data mining* [20, 18, 17], i.e., a learning setting where every example is a set of facts or, equivalently, a (small) relational database.

An interesting approach would be to integrate the two different perspective in an overall framework, where multirelational data mining approach can meet

the potentials offered by the integration of mining and querying which is typical of the inductive database approach. A nice tentative in this direction is the logic language RDM [51] which uses terms for conjunctive queries: both constants and variables can be conjunctive queries, thus RDM queries can be regarded as higher order queries that seeks for standard conjunctive queries which satisfy some given constraints.

Although we are aware of the importance of designing a powerful query language for knowledge discovery, in this paper we do not provide yet another tentative in this direction. Less ambitiously, we provide a simple primitive for constraint-based frequent pattern queries, which can be embedded in any kind of query language and system coherent with the relational paradigm.

The rationale for this aim, is in our intention to focalize on the crucial point of the *Data Mining Query Language* problem: the hot-spot where an adequate trade-off between efficiency and expressiveness of the query language has to be found. The potential efficient evaluation of the proposed primitive is discussed in Section 4 and 5; here we advocate its adequate expressiveness and versatility showing how it can be embedded in query languages of different flavour.

Example 14 (Embedding the primitive in a SQL-like language). Consider the constraint-based frequent pattern query of Example 13. The following is some SQL-like syntactic sugar to express such query.

```
MINE PATTERNS Freq_pat, Support
FROM sales
GROUPING item BY day,cust
MINIMUM SUPPORT: 3
CONSTRAINTS ON name
FROM product
HAVING SUM(price) >= 30,
```

In the first line we define the name of the two output attributes `Freq_pat` and `Support` corresponding to the variables I and S of the query in Example 13. In the `FROM` clause we indicate the data source, or, in other words, the relation which plays the role of the mining view ($\mathcal{V}$): note that according to Definition 5 the mining view can be defined by a relational expression; this means that in the `FROM` clause we can have a full SQL `SELECT` clause. In the third line we indicate that transactions are created grouping the attribute `item` by the two attributes `day,cust`. In other word, we indicate the attribute which plays the role of item ($\mathcal{I}$), and a list of attributes which play the role of transaction identifier ($\mathcal{T}$). Note that such grouping is not really performed by the underlying DBMS: this is just syntactic sugar to express the input parameters for our primitive. In the fourth line we define the minimum support threshold σ. The last three lines define the additional constraint on the sum of prices. Since the attribute `price` is recorded in a relation different from the mining view we must indicate such relation (`product`) and the name of the item attribute in such relation (`name`).

An alternative could be to join the two tables in the definition of the mining view, as follows:

```
MINE PATTERNS Freq_pat, Support
FROM (SELECT *
      FROM sales JOIN product ON item = name
      )
GROUPING item BY day,cust
MINIMUM SUPPORT: 3
HAVING SUM(price) >= 30,
```

However, this requires that the underlying DBMS joins two tables, a costly operation not really necessary since our primitive can handle constraints defined on different relations. In general we can have how many clauses of the form "`CONSTRAINTS ON` *attribute* `FROM` *table* `HAVING` *constraint*" as necessary.

The simple query language whose syntax has been exemplified above, and whose semantics is indirectly given by Definition 12, seems to be an adequate trade-off between efficiency and expressiveness. It inherits its efficiency by our primitive, which is implemented by an ad hoc optimized mining algorithm (see Section 4 and 5). It is expressive since it allows full SQL definition of the source data, it can exploit a wide variety of different constraints, and more importantly, it is geared on frequent itemsets which is a primitive task for many complex queries. The most famous SQL-like data mining query language, MINE RULE [42], has not the same flexibility, being geared on association rules, nor the same efficiency, since it does not exploit constrain-pushing optimizations as those described in the next Section. MINE RULE allows to express queries for the mining of association rules whose body and head satisfy some structural constraints. In the following Example we show how such template-based queries can be expressed in a DATALOG-like query language which embeds our primitive as optimized mining engine.

Example 15 (Embedding the primitive in a DATALOG-like language). Consider again the relation in Figure 1(a) and (c). Suppose we want to compute simple association rules having support greater than 5 and confidence greater than 0.4 with exactly two items in the head (one of type `beverage` and one of type `snack`) and at least 3 items in the body.

As usually we can divide the association rule mining problem in two parts: the mining of frequent itemsets and the subsequent generation of valid rules. For the first subproblem we can use an inductive query which exploits our primitive in order to have an efficient mining, while the second post-processing subproblem can be solved by a simple DATALOG query.

The first inductive rule requires to compute itemsets with the proper support and containing at least one item of type `beverage` and one of type `snack`.

$$\begin{aligned}\texttt{frequentPatterns}(\texttt{Itemset},\texttt{Support}) \leftarrow\ & \texttt{Support} = \texttt{freq}(\texttt{Itemset},\texttt{X}),\\ & \texttt{X} = \langle \texttt{I} | \{\texttt{D},\texttt{C}\}\rangle,\\ & \texttt{sales}(\texttt{D},\texttt{C},\texttt{I}),\\ & \texttt{Support} >= 5,\\ & \texttt{product}(\texttt{L},\_,\texttt{beverage}),\\ & \texttt{product}(\texttt{J},\_,\texttt{snacks}),\\ & \{\texttt{L},\texttt{J}\} \subset \texttt{Itemset}.\end{aligned}$$

In the head of the rule we have the two output attributes `Itemset` and `Support`. The first clause in the body of the rule states that the variable `Support` stores the support of `Itemset` in the transaction database `X`. Such database is obtained from relation `sales(D,C,I)` (third clause) grouping `I` by `{D,C}` (second clause). The last three clauses in the body of the rule define the required constraints.

The second rule is a deductive DATALOG rule which computes the required association rules from the frequent itemsets by finding frequent itemsets of cardinality at least 5, having a frequent subset composed by two items to use as head of the association rule.

$$\begin{aligned}\texttt{rules}(\texttt{L},\texttt{R},\texttt{S},\texttt{C}) \leftarrow\ & \texttt{frequentPatterns}(\texttt{I},\texttt{S}),\\ & \texttt{cardinality}(\texttt{I}) >= 5,\\ & \texttt{frequentPatterns}(\texttt{R},\texttt{S}_1),\\ & \texttt{cardinality}(\texttt{R}) = 2,\\ & \texttt{subset}(\texttt{R},\texttt{I}), \texttt{difference}(\texttt{I},\texttt{R},\texttt{L}),\\ & \texttt{C} = \texttt{S}/\texttt{S1}, \texttt{C} >= 0.4.\end{aligned}$$

As pointed out by these examples, our primitive is a propositional mechanism which works on a single relation, the input mining view. However, it is also possible in principle to embed the primitive into a multirelational data mining query language, such as RDM [51], equipped with high-order mechanisms to generate all possible mining views coherent with the specified item and transaction IDs, thus obtaining repeated invocation to the primitive on the various admissible mining views.

4 Constraint Properties and How to Exploit Them

Constrained frequent pattern mining can be seen as a query optimization problem: given a mining query $\mathcal{Q}$ containing a set of constraints $\mathcal{C}$, provide an efficient evaluation strategy for $\mathcal{Q}$ which is sound and complete (i.e. it finds all and only itemsets in $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$). A naïve solution to such a problem is to first find all frequent patterns ($Th(\mathcal{C}_{freq})$) and then test them for constraints satisfaction. However more efficient solutions can be found by analyzing the property of constraints comprehensively, and exploiting such properties in order to push constraints in the frequent pattern computation. Following this methodology, some classes of constraints which exhibit nice properties have been individuated. In this Section, by reviewing all basic works on the constrained frequent itemsets mining problem, we recall a classification of constraints and their properties.

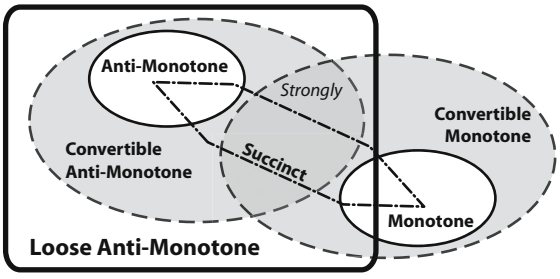

Fig. 3. Characterization of the classes of commonly used constraints

4.1 Anti-monotone and Succinct Constraints

A first work defining classes of constraints which exhibit nice properties is [44]. In that paper is introduced an Apriori-like algorithm, named CAP, which exploits two properties of constraints, namely *anti-monotonicity* and *succinctness*, in order to reduce the frequent itemsets computation. Four classes of constraints, each one with its own associated computational strategy, are defined:

1. Anti-monotone but not succinct constraints;
2. Anti-monotone and succinct constraints;
3. Succinct but not anti-monotone constraints;
4. Constraints that are neither.

Given an itemset X, a constraint $\mathcal{C}_{AM}$ is *anti-monotone* if $\forall Y \subseteq X : \mathcal{C}_{AM}(X) \Rightarrow \mathcal{C}_{AM}(Y)$. The frequency constraint is the most known example of a $\mathcal{C}_{AM}$ constraint. This property, *the anti-monotonicity of frequency*, is used by the Apriori [3] algorithm with the following heuristic: if an itemset X does not satisfy $\mathcal{C}_{freq}$, then no superset of X can satisfy $\mathcal{C}_{freq}$, and hence they can be pruned. This pruning can affect a large part of the search space, since itemsets form a lattice. Therefore the Apriori algorithm (see Algorithm1) operates in a level-wise fashion moving bottom-up on the itemset lattice, from small to large itemsets. At each iteration k Apriori counts the support of *candidate* itemsets (i.e. itemsets which have all subsets frequent) of size k, which are denoted by C_k.

Algorithm 1 Apriori

Input: $\mathcal{D}, \sigma$
Output: $Th(\mathcal{C}_{freq[\mathcal{D},\sigma]})$
1: $C_1 \leftarrow \{\{i\} \mid i \in \mathcal{I}\}$; $k \leftarrow 1$
2: **while** $C_k \neq \emptyset$ **do**
3: $L_k \leftarrow count(\mathcal{D}, C_k)$
4: $C_{k+1} \leftarrow generate\_apriori(L_k)$
5: $k++$
6: $Th(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \leftarrow \bigcup_k L_k$

Those ones which have a support greater than the minimum support threshold σ are frequent itemsets. From the set of frequent itemsets of size k (denoted by L_k) the set of candidates for the next iteration C_{k+1} is generated by the *generate_apriori* procedure. Other $\mathcal{C}_{AM}$ constraints can easily be pushed deeply down into the frequent itemsets mining computation since they behave exactly as $\mathcal{C}_{freq}$: if they are not satisfiable at an early level (small itemsets), they have no hope of becoming satisfiable later (larger itemsets). Conjoining other $\mathcal{C}_{AM}$ constraints to $\mathcal{C}_{freq}$ we just obtain a more selective anti-monotone constraint.

A succinct constraint $\mathcal{C}_S$ is such that, whether an itemset X satisfies it or not, can be determined based on the singleton items which are in X. Informally, given A_1, the set of singleton items satisfying a succinct constraint $\mathcal{C}_S$, then any set X satisfying $\mathcal{C}_S$ is based on A_1 , i.e. X contains a subset belonging to A_1 (for the formal definition of succinct constraints see [44]). A $\mathcal{C}_S$ constraint is *pre-counting pushable*, i.e. it can be satisfied at candidate-generation time: these constraints are pushed in the level-wise computation by substituting the usual *generate_apriori* procedure, with the proper (w.r.t. $\mathcal{C}_S$) candidate generation procedure. For instance, consider the constraint $\mathcal{C}_S \equiv min(X.price) \leq v$, which is a succinct but not anti-monotone constraint. Given $A_1 = \{i \in \mathcal{I} \mid i.price \leq v\}$, we have that $Th(\mathcal{C}_S) = \{X \in 2^{\mathcal{I}} \mid \exists i \in X : i \in A_1\}$. Therefore this constraint can be satisfied at candidate-generation time. This can be done using a special candidate generation procedure, which takes care of the kind of the given constraint, and produces only candidates which satisfy it. Constraints that are both anti-monotone and succinct can be pushed completely in the level-wise computation before it starts (at pre-processing time). For instance, consider the constraint $min(X.price) \geq v$: if we start with the first set of candidates formed by all singleton items having price greater than v, during the computation we will generate only itemsets satisfying the given constraint. Constraints that are neither succinct nor anti-monotone are pushed in the CAP [44] computation by inducing weaker constraints which are either anti-monotone and/or succinct.

4.2 Monotone Constraints

Monotone constraints work the opposite way of anti-monotone constraints. Given an itemset X, a constraint $\mathcal{C}_M$ is monotone if: $\forall Y \supseteq X : \mathcal{C}_M(X) \Rightarrow \mathcal{C}_M(Y)$. Since the frequent itemset computation is geared on $\mathcal{C}_{freq}$, which is anti-monotone, $\mathcal{C}_M$ constraints have been considered more hard to be pushed in the computation and less effective in pruning the search space. In fact, many works [4, 19, 15, 13] have studied the computational problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$, proposing some smart exploration of its search space, but all facing the inherent difficulty of the computational problem: the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *tradeoff*. Such tradeoff can be described as follows. Suppose that an itemset has been removed from the search space because it does not satisfy a monotone constraint. This pruning avoids checking support for this itemset, but on the other hand, if we check its support and find it smaller than the frequency threshold, we may prune away all the supersets of this itemset. In other words, by monotone pruning we risk to lose anti-monotone pruning opportunities given by the pruned itemset. The tradeoff is clear: pushing

monotone constraint can save frequency tests, however the results of these tests could have lead to more effective anti-monotone pruning.

In [7] a completely new approach to exploit monotone constraints by means of data-reduction is introduced. The *ExAnte Property* [7, 8] is obtained by shifting attention from the pattern search space to the input data. Indeed, the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *tradeoff* exists only if we focus exclusively on the search space of the problem, while if exploited properly, monotone constraints can reduce dramatically the data in input, in turn strengthening the anti-monotonicity pruning power. With data reduction techniques we exploit the effectiveness of a $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *synergy*.

The ExAnte property states that a transaction which does not satisfy the given monotone constraint can be deleted from the input database since it will never contribute to the support of any itemset satisfying the constraint.

Proposition 16 (ExAnte property [7]). *Given a transaction database $\mathcal{D}$ and a conjunction of monotone constraints $\mathcal{C}_M$, we define the μ-reduction of $\mathcal{D}$ as the dataset resulting from pruning the transactions that do not satisfy $\mathcal{C}_M$: $\mu_{\mathcal{C}_M}(\mathcal{D}) = \{t \in \mathcal{D} \mid t \in Th(\mathcal{C}_M)\}$.*
It holds that this data reduction does not affect the support of solution itemsets:

$$\forall X \in Th(\mathcal{C}_M) : supp_{\mathcal{D}}(X) = supp_{\mu_{\mathcal{C}_M}(\mathcal{D})}(X).$$

A major consequence of reducing the input database in this way is that it implicitly reduces the support of a large amount of itemsets that do not satisfy $\mathcal{C}_M$ as well, resulting in a reduced number of candidate itemsets generated during the mining algorithm. Even a small reduction in the database can cause a huge cut in the search space, because all supersets of infrequent itemsets are pruned from the search space as well. In other words, monotonicity-based data-reduction of transactions strengthens the anti-monotonicity-based pruning of the search space. This is not the whole story, in fact, infrequent singleton items can not only be removed from the search space together with all their supersets, for the same anti-monotonicity property they also can be deleted from all transactions in the input database (this anti-monotonicity-based data-reduction is named α-*reduction*). Removing items from transactions offers another positive effect: reducing the size of a transaction which satisfies $\mathcal{C}_M$ can make the transaction violate it. Therefore a growing number of transactions which do not satisfy $\mathcal{C}_M$ can be found. Obviously, we are inside a loop where two different kinds of pruning (α and μ) cooperate to reduce the search space and the input dataset, strengthening each other step by step until no more pruning is possible (a fix-point has been reached).

The ExAMiner Algorithm. The recently introduced algorithm ExAMiner [6, 5], generalizes the ExAnte idea to reduce the problem dimensions at all levels of a level-wise Apriori-like computation. In this way, the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *synergy* is effectively exploited at each iteration of the mining algorithm, and not only at pre-processing as done by ExAnte, resulting in significant performance improvements. The idea is to generalize ExAnte's α-reduction from singletons level to the generic level k. This generalization results in the following set of data reduc-

tion techniques, which are based on the anti-monotonicity of $\mathcal{C}_{freq}$ (see [6] for the proof of correctness).

$\mathcal{G}_k(i)$**:** an item which is not subset of at least k frequent k-itemsets can be pruned away from all transactions in $\mathcal{D}$.

$\mathcal{T}_k(t)$**:** a transaction which is not superset of at least $k+1$ frequent k-itemsets can be removed from $\mathcal{D}$.

$\mathcal{L}_k(i)$**:** given an item i and a transaction t, if the number of frequent k-itemsets which are superset of i and subset of t is less than k, then i can be pruned away from transaction t.

Algorithm 2 *count&reduce*

Input: $\mathcal{D}_k, \sigma, \mathcal{C}_M, C_k, V_{k-1}$
Output: $\mathcal{D}_{k+1}, V_k, L_k$

forall $i \in \mathcal{I}$ **do** $V_k[i] \leftarrow 0$
forall tuples t in $\mathcal{D}_k$ **do**
  forall $i \in t$ **do if** $V_{k-1}[i] < k-1$
    then $t \leftarrow t \setminus i$
    else $i.count \leftarrow 0$
  if $|t| \geq k$ **and** $\mathcal{C}_M(t)$ **then**
    forall $X \in C_k, X \subseteq t$ **do**
      $X.count{+}{+}$; $t.count{+}{+}$
      forall $i \in X$ **do** $i.count{+}{+}$
      if $X.count = \sigma$ **then**
        $L_k \leftarrow L_k \cup \{X\}$
        forall $i \in X$ **do** $V_k[i]{+}{+}$
    if $|t| \geq k+1$ **and** $t.count \geq k+1$ **then**
      forall $i \in t$ **if** $i.count < k$
        then $t \leftarrow t \setminus i$
        if $|t| \geq k+1$ **and** $\mathcal{C}_M(t)$ **then**
          write t in $\mathcal{D}_{k+1}$

In ExAMiner [6] these data reductions are coupled with the μ-reduction for $\mathcal{C}_M$ constraints as described in Proposition 16. Essentially ExAMiner is an Apriori-like algorithm, which at each iteration $k-1$ produces a reduced dataset $\mathcal{D}_k$ to be used at the subsequent iteration k. Each transaction in $\mathcal{D}_k$, before participating to the support count of candidate itemsets, is reduced as much as possible by means of $\mathcal{C}_{freq}$-based data reduction, and only if it survives to this phase, it is effectively used in the counting phase. Each transaction which arrives to the counting phase, is then tested against the $\mathcal{C}_M$ (μ-reduction) , and reduced again as much as possible, and only if it survives to this second set of reductions, it is written to the transaction database for the next iteration $\mathcal{D}_{k+1}$. The procedure we have just described, is named *count&reduce* (see Algorithm 2), and substitutes the usual support counting procedure of Apriori (Algorithm 1).

In Algorithm 2 in order to implement the data-reduction $\mathcal{G}_k(i)$ we use an array of integers V_k (of the size of $Items$), which records for each item the number of frequent k-itemsets in which it appears. This information is then exploited during the subsequent iteration $k+1$ for the global pruning of items from all transaction in $\mathcal{D}_{k+1}$ (lines 3 and 4 of the pseudo-code). On the contrary, data reductions $\mathcal{T}_k(t)$ and $\mathcal{L}_k(i)$ are put into effect during the same iteration in which the information is collected. Unfortunately, they require information (the frequent itemsets of cardinality k) that is available only at the end of the actual counting (when all transactions have been used). However, since the set of frequent k-itemsets is a subset of the set of candidates C_k, we can use such data reductions in a relaxed version: we just check the number of candidate itemsets X which are subset of t ($t.count$ in the pseudo-code, lines 10 and 18) and which are superset of i ($i.count$ in the pseudo-code, lines 9 and 14).

4.3 Convertible Constraints

In [48, 49] the class of convertible constraints is introduced, and an FP-growth based methodology to push such constraints is proposed. A constraint $\mathcal{C}_{CAM}$ is convertible anti-monotone provided there is an order $\mathcal{R}$ on items such that whenever an itemset X satisfies $\mathcal{C}_{CAM}$, so does any prefix of X. A constraint $\mathcal{C}_{CM}$ is convertible monotone provided there is an order $\mathcal{R}$ on items such that whenever an itemset X violates $\mathcal{C}_{CM}$, so does any prefix of X. In [48, 49], two FP-growth based algorithms are introduced: $\mathcal{FIC}^{\mathcal{A}}$ to mine $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_{CAM})$, and $\mathcal{FIC}^{\mathcal{M}}$ to mine $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_{CM})$. A major limitation of any FP-growth based algorithm is that the initial database (internally compressed in the prefix-tree structure) and all intermediate projected databases must fit into main memory. If this requirement cannot be met, these approaches can simply not be applied anymore. This problem is even harder with $\mathcal{FIC}^{\mathcal{A}}$ and $\mathcal{FIC}^{\mathcal{M}}$: in fact, using an order on items different from the frequency-based one, makes the prefix-tree lose its compressing power. Thus we have to manage much greater data structures, requiring a lot more main memory which might not be available. Another important drawback of this approach is that it is not possible to take full advantage of a conjunction of different constraints, since each constraint in the conjunction could require a different ordering of items.

4.4 Loose Anti-monotone Constraints

In [12] a new class of tougher constraints, which is a proper superclass of convertible anti-monotone, is introduced together with an Apriori-like algorithm which exploit such constraints by means of data reduction.

Example 17 (var constraint is not convertible). Calculating the variance is an important task of many statistical analysis: it is a measure of how spread out a distribution is. The variance of a set of number X is defined as:

$$var(X) = \frac{\sum_{i \in X}(i - avg(X))^2}{|X|}$$

A constraint based on *var* is not convertible. Otherwise there is an order $\mathcal{R}$ of items such that $var(X)$ is a prefix increasing (or decreasing) function. Consider a small dataset with only four items $\mathcal{I} = \{A, B, C, D\}$ with associated prices $P = \{10, 11, 19, 20\}$. The lexicographic order $\mathcal{R}_1 = \{ABCD\}$ is such that $var(A) \leq var(AB) \leq var(ABC) \leq var(ABCD)$, and it is easy to see that we have only other three orders with the same property: $\mathcal{R}_2 = \{BACD\}, \mathcal{R}_3 = \{DCBA\}, \mathcal{R}_4 = \{CDBA\}$. But, for $\mathcal{R}_1$, we have that $var(BC) \not\leq var(BCD)$, which means that *var* is not a prefix increasing function w.r.t. $\mathcal{R}_1$. Moreover, since the same holds for $\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$, we can assert that there is no order $\mathcal{R}$ such that *var* is prefix increasing. An analogous reasoning can be used to show that it neither exists an order which makes *var* a prefix decreasing function.

Following a similar reasoning it can be shown that other interesting constraints, such as for instance those ones based on *standard deviation* (*std*) or *unbiased variance estimator* (var_{N-1}) or *mean deviation* (*md*), are not convertible as well. Luckily, all these constraints share a nice property that named *"Loose Anti-monotonicity"* [12].

While an anti-monotone constraint is such that, if satisfied by an itemset then it is satisfied by *all* its subsets, a loose anti-monotone constraint as such that, if it is satisfied by an itemset of cardinality k then it is satisfied by *at least one* of its subsets of cardinality $k-1$. Since some of these interesting constraints make sense only on sets of cardinality at least 2, in order to get rid of such details, we shift the definition of loose anti-monotone constraint to avoid considering singleton items.

Definition 18 (Loose Anti-monotone constraint). Given an itemset X with $|X| > 2$, a constraint is *loose anti-monotone* (denoted $\mathcal{C}_{LAM}$) if: $\mathcal{C}_{LAM}(X) \Rightarrow \exists i \in X \ : \ \mathcal{C}_{LAM}(X \setminus \{i\})$

The next proposition and the subsequent example state that the class of $\mathcal{C}_{LAM}$ constraints is a proper superclass of $\mathcal{C}_{CAM}$ (convertible anti-monotone constraints).

Proposition 19. *Any convertible anti-monotone constraint is trivially loose anti-monotone: if a k-itemset satisfies the constraint so does its* $(k-1)$*-prefix itemset.*

Example 20. We show that the constraint $var(X.A) \leq v$ is a $\mathcal{C}_{LAM}$ constraint. Given an itemset X, if it satisfies the constraint so trivially does $X \setminus \{i\}$, where i is the element of X which has associated a value of A which is the most far away from $avg(X.A)$. In fact, we have that $var(\{X \setminus \{i\}\}.A) \leq var(X.A) \leq v$, until $|X| > 2$.Taking the element of X which has associated a value of A which is the closest to $avg(X.A)$ we can show that also $var(X.A) \geq v$ is a $\mathcal{C}_{LAM}$ constraint. Since the standard deviation *std* is the square root of the variance, it is straightforward to see that $std(X.A) \leq v$ and $std(X.A) \geq v$ are $\mathcal{C}_{LAM}$. The mean deviation is defined as: $md(X) = (\sum_{i \in X} |i - avg(X)|) \ / \ |X|$. Once again, we have that $md(X.A) \leq v$ and $md(X.A) \geq v$ are loose anti-monotone. It is

easy to prove that also constraints defined on the unbiased variance estimator, $var_{N-1} = (\sum_{i \in X}(i - avg(X))^2) \ / \ (|X| - 1)$ are loose anti-monotone.

The next Proposition (see [12] for the proof) indicates how a $\mathcal{C}_{LAM}$ constraint can be exploited in a level-wise Apriori-like computation by means of data-reduction. It states that if at any iteration $k \geq 2$ a transaction is not superset of at least one frequent k-itemset which satisfy the $\mathcal{C}_{LAM}$ constraint (a solution), then the transaction can be deleted from the database.

Proposition 21. *Given a transaction database $\mathcal{D}$, a minimum support threshold σ, and a $\mathcal{C}_{LAM}$ constraint, at the iteration $k \geq 2$ of the level-wise computation, a transaction $t \in \mathcal{D}$ such that: $\nexists X \subseteq t, |X| = k, X \in Th(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \cap Th(\mathcal{C}_{LAM})$ can be pruned away from $\mathcal{D}$, since it will never be superset of any solution itemsets of cardinality $> k$.*

As in ExAMiner [6] the anti-monotonicity based data reductions are coupled with the μ-reduction for $\mathcal{C}_M$ constraints, similarly we can exploit the above Proposition for $\mathcal{C}_{LAM}$ constraints, by embedding such loose anti-monotonicity based data reduction with-in the *count&reduce* procedure (see [12]).

5 Constraint Pushing Optimization

In this section we define an ad hoc optimized algorithm for the evaluation of our primitive on the basis of the state-of-the-art of constraint pushing techniques described in the previous Section. The proposed algorithm, is a breadth-first Apriori-like computation based on data-reduction techniques.

Adopting this kind of algorithmic architecture, i.e., moving level-wise and reducing data as much as possible, we can exploit different properties of constraints all together, and the global reduction benefit is always greater than the sum of the individual benefits.

In Table 2 we report the properties that our algorithm exploits for each constraint admitted in our framework. In the Table we do not report convertibility property since we do not exploit it. Convertibility, in fact, is well suited for FP-tree based depth-first algorithms [30, 48, 49] while we adopt an Apriori-like breadth-first computation. In our framework, the constraint based on the avg aggregate, which is the prototypical convertible constraint, is pushed in the computation by means of loose anti-monotonicity data reduction, obtaining stronger benefits. For a deeper discussion on this issue and for empirical comparison of the two different strategies see [12].

Algorithm 3 implements our primitive. Given $freq(\mathcal{V}, \mathcal{T}, \mathcal{I}, \sigma, \mathcal{C})$ Algorithm 3 computes $Th(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \cap Th(\mathcal{C})$ where $\mathcal{D}$ is given by the triple $\langle \mathcal{V}, \mathcal{T}, \mathcal{I} \rangle$ as described in Section 2. In the pseudo-code the constraints in the conjunction $\mathcal{C}$ are partitioned in groups w.r.t. their properties. In particular:

- $\mathcal{C}_{AM}$ is the conjunction of constraints in $\mathcal{C}$ which are anti-monotone but not succinct;

Table 2. Properties of the constraints admitted in our primitive

Constraint	Properties
$S \subseteq V$	$\mathcal{C}_{AM}, \mathcal{C}_S, \mathcal{C}_{LAM}$
$S \supseteq V$	$\mathcal{C}_M, \mathcal{C}_S, \mathcal{C}_{LAM}$
$S.A \subseteq V$	$\mathcal{C}_{AM}, \mathcal{C}_S, \mathcal{C}_{LAM}$
$S.A \supseteq V$	$\mathcal{C}_M, \mathcal{C}_S, \mathcal{C}_{LAM}$
$min(S.A) \geq v$	$\mathcal{C}_{AM}, \mathcal{C}_S, \mathcal{C}_{LAM}$
$min(S.A) \leq v$	$\mathcal{C}_M, \mathcal{C}_S, \mathcal{C}_{LAM}$
$max(S.A) \geq v$	$\mathcal{C}_M, \mathcal{C}_S, \mathcal{C}_{LAM}$
$max(S.A) \leq v$	$\mathcal{C}_{AM}, \mathcal{C}_S, \mathcal{C}_{LAM}$
$count(S) \leq v$	$\mathcal{C}_{AM}, \mathcal{C}_{LAM}$
$count(S) \geq v$	$\mathcal{C}_M$
$count(S.A) \leq v$	$\mathcal{C}_{AM}, \mathcal{C}_{LAM}$
$count(S.A) \geq v$	$\mathcal{C}_M$
$sum(S.A) \leq v\ (\forall i \in S, i.A \geq 0)$	$\mathcal{C}_{AM}, \mathcal{C}_{LAM}$
$sum(S.A) \geq v\ (\forall i \in S, i.A \geq 0)$	$\mathcal{C}_M$
$range(S.A) \leq v$	$\mathcal{C}_{AM}, \mathcal{C}_{LAM}$
$range(S.A) \geq v$	$\mathcal{C}_M, \mathcal{C}_{LAM}$
$avg(S.A)\theta v$	$\mathcal{C}_{LAM}$
$median(S.A)\theta v$	$\mathcal{C}_{LAM}$
$var(S.A)\theta v$	$\mathcal{C}_{LAM}$
$std(S.A)\theta v$	$\mathcal{C}_{LAM}$
$md(S.A)\theta v$	$\mathcal{C}_{LAM}$

- $\mathcal{C}_{AMS}$ is the conjunction of constraints in $\mathcal{C}$ which are both anti-monotone and succinct;
- $\mathcal{C}_M$ is the conjunction of constraints in $\mathcal{C}$ which are monotone but not succinct;
- $\mathcal{C}_{MS}$ is the conjunction of constraints in $\mathcal{C}$ which are both monotone and succinct;
- $\mathcal{C}_{LAM}$ is the conjunction of constraints in $\mathcal{C}$ which are loose anti-monotone.

Note that this groups of constraints are not necessarily disjoint.

Example 22. The constraint $range(S.A) \geq v \equiv max(S.A) - min(S.A) \geq v$, is both monotone and loose anti-monotone. Thus, when we mine frequent itemsets which satisfy such constraint we can exploit the benefit of having together, in the same *count&reduce* procedure, the $\mathcal{C}_{freq}$-based data reductions, the μ-reduction for monotone constraints, and the reduction based on $\mathcal{C}_{LAM}$.

The possibility of exploiting different properties of constraints all together, exists not only for $\mathcal{C}_M$ and $\mathcal{C}_{LAM}$ constraints (as seen in Example 22), but also for any other kind of constraints. In fact, all the properties that we exploit are orthogonal and thus can be combined.

Example 23. Consider now the constraint $max(S.A) \geq v$. This constraint is monotone, succinct and loose anti-monotone. This means that we can exploit all

these properties by using it as a succinct constraint at candidate generation time, and using it as a monotone constraint and as a loose anti-monotone constraint by means of data-reduction at counting time.

Algorithm 3 Constraint-based Frequent Pattern Mining

Input: $\mathcal{D}, \sigma, \mathcal{C}$
Output: $Th(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \cap Th(\mathcal{C})$

1: $L_1 \leftarrow \mathcal{I}$
2: $C_1 \leftarrow \{\{i\} \mid i \in \mathcal{I} \wedge \mathcal{C}_{AMS}(\{i\}) \wedge \mathcal{C}_{AM}(\{i\})\}$
3: $\mathcal{D}_1 \leftarrow \pi_{C_1}(\mathcal{D})$
4: $L_1, \mathcal{D}_1 \leftarrow count\_first\_iteration(\mathcal{D}_1, \sigma, C_1, \mathcal{C}_M, \mathcal{C}_{MS})$
5: **while** $L_1 \neq C_1$ **do**
6: $\quad C_1 \leftarrow L_1$;
7: $\quad L_1, \mathcal{D}_1 \leftarrow count\_first\_iteration(\mathcal{D}_1, \sigma, C_1, \mathcal{C}_M, \mathcal{C}_{MS})$
8: $C_2 \leftarrow generate(L_1, \mathcal{C}_{AM}, \mathcal{C}_{MS})$
9: **forall** $i \in L_1$ **do** $V_1[i] \leftarrow 0$
10: $k \leftarrow 2$
11: **while** $C_k \neq \emptyset$ **do**
12: $\quad L_k, \mathcal{D}_{k+1}, V_k \leftarrow count\&reduce(\mathcal{D}_k, \sigma, \mathcal{C}_M, \mathcal{C}_{MS}, \mathcal{C}_{LAM}, C_k, V_{k-1})$
13: $\quad C_{k+1} \leftarrow generate(L_k, \mathcal{C}_{AM}, \mathcal{C}_{MS})$
14: $\quad k++$
15: **for**$(i = 0; i \leq k; i++)$ **do**
16: $\quad$ **forall** $X \in L_i$ **do**
17: $\quad\quad$ **if** $\mathcal{C}_M(X) \wedge \mathcal{C}_{MS}(X) \wedge \mathcal{C}_{LAM}(X)$ **then return** X

Let us briefly describe the pseudo-code in Algorithm 3. First of all, note that as stated in Section 3 constraints which are both anti-monotone and succinct are pushed once and for all, at preprocessing, simply by considering in the forthcoming computation singleton items which satisfy them (Line 2). Lines from 3 to 7 together with procedure *count_first_iteration* (Algorithm 4), implement the Ex-Ante pre-processing [7]. Lines from 11 to 14 implements the typical central loop of the Apriori algorithm, where the *generate* procedure exploits succinctness and anti-monotonicity to reduce the set of candidates, and the *count&reduce* procedure exploits monotonicity and loose anti-monotonicity. Finally, lines from 15 to 17 implement the post-processing, where possible solution itemsets are check for satisfaction of those kinds of constraints, for which satisfaction not already guaranteed.

Our algorithm, by means of data-reduction, exploits a real synergy of all constraints that the user defines for the pattern extraction: each constraint does not only play its part in reducing the data, but this reduction in turns strengthens the pruning power of the other constraints. Moreover data-reduction induces a pruning of the search space, and the pruning of the search space in turn strengthens future data reductions.

The orthogonality of the exploited constraint pushing techniques has a twofold benefit: on one hand all the techniques can be amalgamated together achieving a

Algorithm 4 *count_first_iteration*

Input: $\mathcal{D}, \sigma, C, \mathcal{C}_M, \mathcal{C}_{MS}$
Output: $\mathcal{D}_1, L_1$

1: $L_1 \leftarrow \emptyset; \mathcal{D}_1 \leftarrow \emptyset$
2: **forall** $t \in \mathcal{D}$ **do**
3: **if** $\mathcal{C}_M(t) \wedge \mathcal{C}_{MS}(t)$ **then**
4: **forall** $i \in t$ **do** $i.count++$; **if** $i.count == \sigma$ **then** $L_1 \leftarrow L_1 \cup \{i\}$
5: $\mathcal{D}_1 \leftarrow \mathcal{D}_1 \cup t$
6: $\mathcal{D}_1 \leftarrow \pi_{L_1}(\mathcal{D}_1)$

very efficient computation (for the empirical evaluation of each single technique we address the interested reader to the respective paper cited in Section 4); on the other hand the framework can be easily extended to handle to other constraints.

Another positive effect of adopting an Apriori-like algorithm, is that in the implementation we can exploit all coding tricks and smart data structure that have been developed in the last decade for the Apriori algorithm (see, for instance, [45, 46]).

At Pisa KDD Laboratory developed a prototype of the optimized computational framework in tight collaboration with the authors of [45, 46], within the P^3D project[1].

6 Conclusions and Future Work

In this paper, we introduced a primitive for constraint-based pattern discovery, which represents a trade-off between simplicity and generality, as well as between expressiveness and efficiency. The versatility of the primitive is witnessed by its easy adaption within query languages of different nature; its efficiency is witnessed by the availability of systematic optimization methods, based on the properties of the specified constraints. We believe that this trade-off ia a step forward in the road to a realistic inductive database system. We are also aware that many issues remain open, and deserve further research:

- how to support user-defined constraints;
- how to integrate condensed representations of patterns in the constraint-based mining framework [47, 10, 11];
- how to tightly integrate our primitive within a relational DBMS: his issue is strictly connected with many other open problems, for instance, how to store and index frequent pattern query results;
- defining constraint-based incremental mining techniques, i.e., how to exploit the results of previous queries in order to have a more efficient computation for the forthcoming queries;
- developing a constraint-based mining framework for more complex kinds of patterns such as sequences and graphs.

[1] `http://www-kdd.isti.cnr.it/p3d/index.html`

Our objective is to integrate the results of these investigations in a unified system for exploratory constraint-based pattern discovery.

References

1. R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD'93*.
2. R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In *Proceedings of KDD'96*.
3. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of VLDB'94*.
4. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Adaptive Constraint Pushing in frequent pattern mining. In *Proceedings of PKDD'03*.
5. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Efficient Breadth-first Mining of Frequent Pattern with Monotone Constraints. To appear in *Knowledge and Information Systems - An International Journal (KAIS)*. Springer, Berlin.
6. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In *Proceedings of ICDM'03*.
7. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAnte: Anticipated data reduction in constrained pattern mining. In *Proceedings of PKDD'03*.
8. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Preprocessing for Frequent Pattern Mining through Data Reduction. To appear in *IEEE Intelligent Systems*.
9. F. Bonchi and B. Goethals. FP-Bonsai: the Art of Growing and Pruning Small FP-trees. In *Proceedings of PAKDD'04*, 2004.
10. F. Bonchi and C. Lucchese. On closed constrained frequent pattern mining. In *Proceedings of ICDM'04*.
11. F. Bonchi and C. Lucchese. On Condensed Representations of Constrained Frequent Patterns. To appear in *Knowledge and Information Systems - An International Journal (KAIS)*. Springer, Berlin.
12. F. Bonchi and C. Lucchese. Pushing tougher constraints in frequent pattern mining. In *Proceedings of PAKDD'05*, 2005.
13. J.F. Boulicaut and B. Jeudy. Using constraints during set mining: Should we prune or not? In *Actes des Seizime Journes Bases de Donnes Avances BDA'00*.
14. J.F. Boulicaut, P. Marcel, and C.Rigotti. Query driven knowledge discovery in multidimensional data. In *Proceedings of DOLAP'99*.
15. C. Bucila, J. Gehrke, D. Kifer, and W. White. DualMiner: A dual-pruning algorithm for itemsets with constraints. In *Proceedings of ACM SIGKDD'02*.
16. S. Choenni and A. Siebes. Query Optimization to Support Data Mining. In *Proc. of the Int'l. Workshop on Database and Expert Systems Application 1997*.
17. L. Dehaspe and L. De Raedt. Dlab: A declarative language bias formalism. In *Proceedings of ISMIS'96*.
18. L. Dehaspe and H. Toivonen. Discovery of Frequent Datalog Patterns. *Journal of Knowledge Discovery and Data Mining*, 3(1):7–36, 1999.
19. L. De Raedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *Proceedings of IJCAI'01*.
20. S. Džeroski and N. Lavrač, editors. *Relational Data Mining*. Springer, Berlin, 2001.
21. F. Giannotti and G. Manco. Querying Inductive Databases via Logic-Based User-Defined Aggregates. In *Proceedings of PKDD'99*.

22. F. Giannotti and G. Manco. Making Knowledge Extraction and Reasoning Closer. In T. Terano, editor, *Proceedings of PAKDD'00*.
23. F. Giannotti, G. Manco and F. Turini. Specifying Mining Algorithms with Iterative User-Defined Aggregates. IEEE Trans. Knowl. Data Eng. 16(10): 1232-1246 (2004).
24. F. Giannotti, G. Manco and J. Wijsen. Logical Languages for Data Mining. In *Logics for emerging Applications of Databases*. Springer, Berlin, 2003.
25. G. Grahne, L. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In *Proceedings of ICDE'00)*.
26. J. Han. Towards On-Line Analytical Mining in Large Databases. *Sigmod Records*, 27(1):97–107, 1998.
27. J. Han, S. Chee, and J. Chiand. Issues for On-Line Analytical Mining of Data Warehouses. In *Proceedings of DMKD'98*.
28. J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A Data Mining Query Language for Relational Databases. In *Proceedings of DMKD'96*.
29. J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-based, multidimensional data mining. *Computer*, 32(8):46–50, 1999.
30. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM SIGMOD'00*.
31. D. Hand, H. Mannila, and P. Smyh. *Principles of Data Mining*. The MIT Press, 2001.
32. M. Houtsma and A. Swami. Set-oriented mining for association rules in relational databases. In *Proceedings of ICDE'95*.
33. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Comm. Of The Acm*, 39:58–64, 1996.
34. T. Imielinski and A. Virmani. MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, 1999.
35. T. Imielinski, A. Virmani, and A. Abdulghani. DMajor - Application Programming Interface for Database Mining. *Data Mining and Knowledge Discovery*, 3(4):347–372, 1999.
36. B. Jeudy and J.F. Boulicaut. Optimization of association rule mining queries. *Intelligent Data Analysis Journal*, 6(4):341–357, 2002.
37. S. Kramer, L. De Raedt, and C. Helma. Molecular feature mining in hiv data. In *Proceedings of ACM SIGKDD'01*.
38. L. V. S. Lakshmanan, R. T. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. *SIGMOD Record*, 28(2), 1999.
39. W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *In Proceedings of ICDM'01*.
40. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of KDD'98*.
41. H. Mannila and H. Toivonen. Levelwise Search and Border of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3:241–258, 1997.
42. R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proceedings of VLDB'96*.
43. R. Meo, G. Psaila, and S. Ceri. A Tightly-Coupled Architecture for Data Mining. In *Proceedings of ICDE'98*.
44. R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the ACM SIGMOD'98*.
45. S. Orlando, P. Palmerini, and R. Perego. Enhancing the Apriori Algorithm for Frequent Set Counting. In *Proceedings of DaWak'01*.

46. S. Orlando, P. Palmerini, R. Perego, and F. Silvestri. Adaptive and Resource-Aware Mining of Frequent Sets. In *Proceedings of ICDM'02*.
47. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of ICDT'99*.
48. J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proceedings of ACM SIGKDD'00*.
49. J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *(Proceedings of ICDE'01)*.
50. J. Pei, X. Zhang, M. Cho, H. Wang, and P. Yu. Maple: A fast algorithm for maximal pattern-based clustering. In *Proceedings of ICDM'03*.
51. L. De Raedt. A logical database mining query language. In *Proceedings of ILP'00*.
52. S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proceedings of the ACM SIGMOD'98*.
53. W. Shen and B. Leng. A Metapattern-Based Discovery Loop for Integrated Data Mining - Unsupervised Learning of Relational Patterns. *IEEE Trans. on Knowledge and Data Engineering*, 8(6):898–910, 1996.
54. W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for Data Mining. In *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI Press/The MIT Press, 1996.
55. A. P. J. M. Siebes and M. L. Kersten. Keso: Minimizing Database Interaction. In *Proceedings of KDD'97*.
56. R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proceedings of KDD'97*.
57. 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 ACM SIGMOD'98*.
58. M. L. Yiu and N. Mamoulis. Frequent-pattern based iterative projected clustering. In *Proceedings of ICDM'03*.