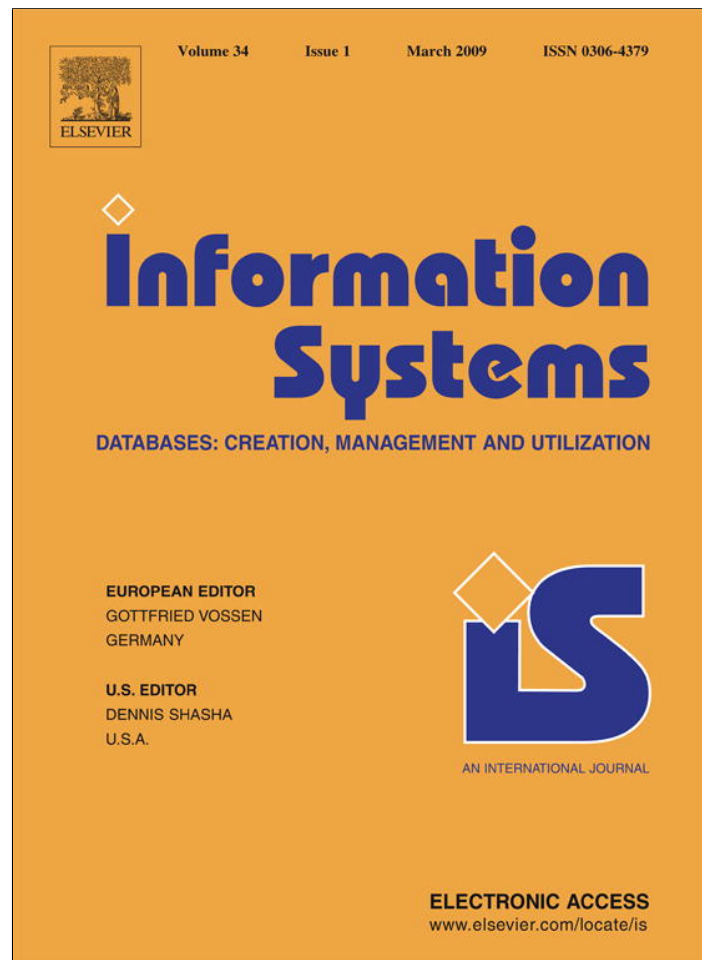


Provided for non-commercial research and education use.  
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

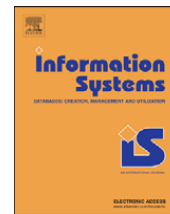
In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

<http://www.elsevier.com/copyright>



Contents lists available at ScienceDirect

Information Systems

journal homepage: [www.elsevier.com/locate/infosys](http://www.elsevier.com/locate/infosys)

## A constraint-based querying system for exploratory pattern discovery

Francesco Bonchi<sup>a,\*</sup>, Fosca Giannotti<sup>b</sup>, Claudio Lucchese<sup>b,c</sup>, Salvatore Orlando<sup>c</sup>,  
Raffaele Perego<sup>b</sup>, Roberto Trasarti<sup>b,d</sup>

<sup>a</sup> Yahoo! Research Barcelona, Ocatà 1, Barcelona, Spain

<sup>b</sup> ISTI-CNR, Area della Ricerca di Pisa, Via Giuseppe Moruzzi 1, Pisa, Italy

<sup>c</sup> Dipartimento di Informatica, Università Ca' Foscari di Venezia, Via Torino 155, Venezia Mestre, Italy

<sup>d</sup> Dipartimento di Informatica, Università di Pisa, Largo Pontecorvo 3, Pisa, Italy

### ARTICLE INFO

#### Article history:

Received 16 July 2007

Received in revised form

28 January 2008

Accepted 21 February 2008

Recommended by J. Van den Bussche

#### Keywords:

Constrained pattern mining

Data mining systems

Inductive databases

Data mining query languages

Interactive data mining

### ABSTRACT

In this article we present CONQUEST, a constraint-based querying system able to support the intrinsically exploratory (i.e., human-guided, interactive and iterative) nature of pattern discovery. Following the *inductive database* vision, our framework provides users with an expressive constraint-based query language, which allows the discovery process to be effectively driven toward potentially interesting patterns. Such constraints are also exploited to reduce the cost of pattern mining computation. CONQUEST is a comprehensive mining system that can access real-world relational databases from which to extract data. Through the interaction with a friendly graphical user interface (GUI), the user can define complex mining queries by means of few clicks. After a pre-processing step, mining queries are answered by an efficient and robust pattern mining engine which entails the state-of-the-art of data and search space reduction techniques. Resulting patterns are then presented to the user in a pattern browsing window, and possibly stored back in the underlying database as relations.

© 2008 Elsevier B.V. All rights reserved.

### 1. Introduction

In this article we present in details CONQUEST, a comprehensive knowledge discovery system for extracting interesting patterns, where the interestingness of the patterns is defined by means of user-defined constraints.

The paradigm of pattern discovery based on constraints was introduced 10 years ago with the aim of providing the user with a tool to drive the discovery

process toward potentially *interesting* information, with the positive side effect of pruning the huge search space thus achieving a more efficient computation [1–5]. Most of the research so far has focussed on developing efficient, sound and complete evaluation strategies for constraint-based mining queries, and regardless some successful applications, e.g., in medical domain [6–8], or in biological domain [9], there is still a lack of research on languages and systems supporting this knowledge discovery paradigm. Indeed, to the best of our knowledge, CONQUEST is the first and only system of this kind.

The basic idea behind CONQUEST is that the task of extracting useful and interesting knowledge from data is an *exploratory* querying process, i.e., human-guided, iterative and interactive. We believe that 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 same way a database

\* Corresponding author. Tel.: +34 935421182; fax: +34 935421150.

E-mail addresses: [bonchi@yahoo-inc.com](mailto:bonchi@yahoo-inc.com) (F. Bonchi),

[fosca.giannotti@isti.cnr.it](mailto:fosca.giannotti@isti.cnr.it) (F. Giannotti), [claudio.lucchese@isti.cnr.it](mailto:claudio.lucchese@isti.cnr.it)

(C. Lucchese), [orlando@dsi.unive.it](mailto:orlando@dsi.unive.it) (S. Orlando),

[raffaele.perego@isti.cnr.it](mailto:raffaele.perego@isti.cnr.it) (R. Perego), [roberto.trasarti@isti.cnr.it](mailto:roberto.trasarti@isti.cnr.it)

(R. Trasarti).

<sup>1</sup> This work was conducted when the first author was a researcher at ISTI-CNR, Italy.

designer need not worry about query optimization. The system must provide the analyst with a set of primitives 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 implemented following the guidelines described in [10], where Mannila introduced an elegant formalization for the notion of interactive mining process, named *inductive database*. This term refers a relational database framework which integrates the raw data with the patterns (or true sentences) extracted from data, and materialized in the form of relations. In such vision the analyst, exploiting an expressive query language, drives the discovery process through a sequence of complex mining queries, extracts patterns satisfying some user-defined constraints, refines the queries, materializes the extracted patterns as first-class citizens in the database, combines the patterns to produce more complex knowledge and cross-over the data and the patterns: the knowledge discovery process consists essentially in an iterative querying process. Therefore, an inductive database system should provide the following features:

*Coupling with a DBMS.* The analyst must be able to retrieve the portion of interesting data (for instance, by means of SQL queries). Moreover, extracted patterns should also be stored in the DBMS in order to be further queried or mined (*closure principle*).

*Expressiveness of the query language.* The analyst must be able to interact with the pattern discovery system by specifying declaratively how the desired patterns should look like, and which conditions they should satisfy. The task of composing all constraints and producing the most efficient mining strategy (execution plan) for a given query should be thus completely demanded to the underlying system.

*Efficiency of the mining engine (ME).* Keeping query response time as small as possible is an important requirement, since this allows us to design a system able to give frequent feedbacks to the user, thus allowing realistic human-guided exploration. Unfortunately, this is a very challenging task, due to the exponential complexity of pattern discovery computations. To this end, data and

search space reduction properties of constraints should be effectively exploited by pushing them within the mining algorithms. Moreover, we can take advantage of the iterative nature of a typical pattern discovery task: a mining session is usually made up of a series of queries (exploration), where each new query adjusts, refines or combines the results of some previous queries. It is thus important for the ME to adopt techniques for incremental mining. For example, by reusing results of previous queries, in order to give a faster response to the last query presented to the system, instead of performing again the mining task from scratch.

*Graphical user interface (GUI).* The exploratory nature of pattern discovery imposes to the system not only to return frequent feedbacks to the user, but also to provide pattern visualization and navigation tools. These tools should help the user in visualizing the continuous feedbacks from the system, allowing an easier and human-based identification of fragments of interesting knowledge. Such tools should also play the role of graphical querying interface. In this way the action of browsing and visualizing patterns should become tightly integrated (both by a conceptual and engineering point of view) with the action of iteratively querying.

Starting from the above requirements we designed CONQUEST, an exploratory pattern discovery system, equipped with a simple, yet powerful, query language (named SPQL that stands for *simple pattern query language*). CONQUEST is now a mature and stable software, providing a larger number of functionalities, compared with [11]. CONQUEST includes a user-friendly interface for accessing the underlying DBMS, and also for data visualization and query formulation. In designing the CONQUEST query language, its architecture and user interface, we have kept in mind all the tasks involved in the typical *knowledge discovery process* [12]: (i) source data selection, (ii) data preparation, pre-processing and transformation and (iii) pattern discovery and model building (see Fig. 1).

The user supervises the whole process not only by defining the parameters of the three tasks, but also by evaluating the quality of the outcome of each step and possibly re-tuning the parameters of any step. Moreover, the user is in charge of interpreting and evaluating the extracted knowledge, even if the system must provide adequate support for this task.

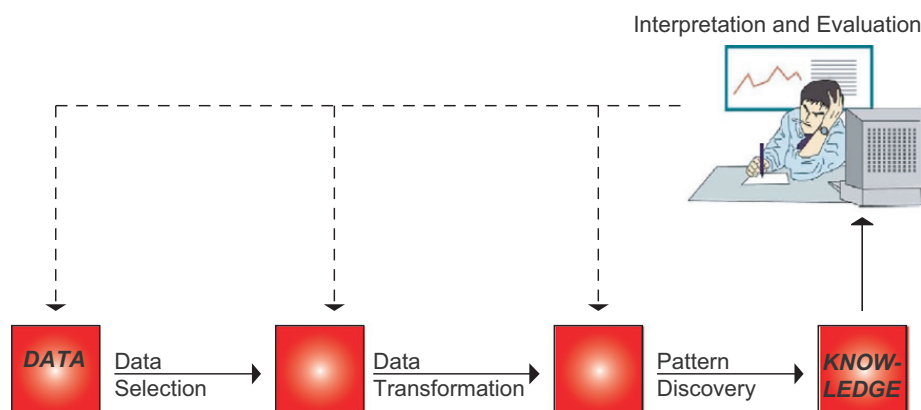


Fig. 1. CONQUEST knowledge discovery process.

---

```

1. MINE PATTERNS WITH SUPP>= 0.5% IN
2. SELECT A1, A2
3. FROM R
4. WHERE R.A3 > 100
5. TRANSACTION A1
6. ITEM A2
7. CONSTRAINED BY length >= 4

```

---

**Fig. 2.** An example  $SPQL$  mining query.

In Fig. 2 an example of  $SPQL$  mining query is shown. We will describe the details of the language later, in Section 3. Here we just want to highlight how the three main tasks of the knowledge discovery process can be expressed using the query language. In fact, a standard  $SPQL$  mining query is essentially made up of three parts:

- (1) The data source selection, by means of an SQL select-from-where statement (lines 2–4 of the query in Fig. 2).
- (2) The data preparation parameters (lines 5 and 6), to transform the relational database into a transactional one before starting the mining process.
- (3) The mining parameters and constraints (line 1 for the minimum frequency constraint, and line 7 for the other constraints).

### 1.1. Paper organization

In the rest of this article we describe in full details the main design choices and features of CONQUEST. The article is organized as follows. In Section 2 we provide the formal definitions and the theoretical framework underlying our system. In Section 3 we discuss the  $SPQL$  query language. Section 4 provides a state-of-the-art of the algorithms for constraint-based frequent pattern mining that leads to a detailed description of the CONQUEST'S ME and the algorithmic choices underlying it. Then a thorough experimental analysis of CONQUEST'S ME is reported. In Section 5 we discuss the overall CONQUEST'S architecture, by also giving several details concerning the main modules that constitute the system. In particular, Section 5.2 describes the GUI and how the interactions between the user and the system actually happens; Section 5.3 describes the query interpreter (QI) and pre-processor modules. Finally, in Section 6 we draw some conclusions.

## 2. Pattern mining from relational databases

In this section we provide the formal definition of the constraint-based pattern mining framework, that is the theoretical background for the data analysis for which CONQUEST has been devised. In particular, we highlight the gap that exists between the theoretical framework definition, which is the basis of all the algorithms presented in the literature, and the real data stored in a commercial DBMS. Closing this gap is one of the main design objectives and contributions of CONQUEST.

Devising fast and scalable algorithms able to crunch huge amount of data has been so far one of the main goal of data mining research. Recently, researchers realized that in many practical cases it does not matter how much efficiently knowledge is extracted, since the volume of the results themselves is often embarrassingly large, and creates a second order mining problem for the human expert. This situation is very common in the case of association rules and frequent pattern mining [13], where the identification of the fragments of interesting knowledge, blurred within a huge quantity of mostly useless patterns, is very difficult.

In traditional frequent pattern mining, the only interest measure is the frequency of a pattern.

**Definition 2.1** (*Frequent itemset mining*). Let  $\mathcal{I} = \{x_1, \dots, x_n\}$  be a set of distinct items, any subset  $X \subseteq \mathcal{I}$  is an *itemset*. Given a *transactional dataset*  $\mathcal{D}$ , i.e., a multiset of transactions/itemsets  $t \subseteq \mathcal{I}$ , the support of an itemset  $X$  is defined as  $supp_{\mathcal{D}}(X) = |\{t \in \mathcal{D} | X \subseteq t\}|$ . Given a *minimum support threshold*  $\delta \geq 1$ , an itemset  $X$  is said to be *frequent* if  $supp_{\mathcal{D}}(X) \geq \delta$ . The *frequent itemset mining problem* requires to compute all the *frequent itemsets* occurring in the transactions of  $\mathcal{D}$ :  $\{X \subseteq \mathcal{I} | supp_{\mathcal{D}}(X) \geq \delta\}$

The minimum support requirement is a particular selection constraint  $\mathcal{C}_{freq}(X) :: 2^{\mathcal{I}} \rightarrow \{0, 1\}$ , depending solely on the transactional dataset  $\mathcal{D}$ , the appearances of  $X$  within it, and the minimum support threshold  $\delta$ . We have that  $\mathcal{C}_{freq}(X) = 1$  (*True*) if  $X$  satisfies the constraint (i.e.,  $supp_{\mathcal{D}}(X) \geq \delta$ ), and  $\mathcal{C}_{freq}(X) = 0$  (*False*) otherwise (i.e.,  $supp_{\mathcal{D}}(X) < \delta$ ). However, many other interesting constraints can be defined over the set of *properties* (e.g., price, weight, category, etc.), characterizing the different items that compose a pattern.

The constraint-based pattern mining paradigm has been recognized as one of the fundamental techniques for inductive databases: by taking into consideration an additional set of user-defined constraints, we can solve the data/pattern abundance problem, allowing the analyst to drive, iteratively and interactively, the mining process toward potentially interesting patterns only. Moreover, constraints can be pushed deep inside a mining algorithm, in order to deal with the exponential search-space curse and achieve better performance.

### 2.1. Constraint-based pattern mining

In the following we formally define the constraint-based pattern mining problem. The input data for such

a

$\mathcal{D}$
{beer, chips, wine}
{wine, beer, pasta, chips}
{chips, beer}
{jackets, col_shirts}
{wine, beer}
{pasta, chips}
{jackets}
{wine, pasta}
{chips, col_shirts, brown_shirts}
{pasta, wine, chips, beer}
{beer, chips}
{pasta}

b

item	$\mathcal{P}$	
	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

Fig. 3. An example of mining view  $MV$  for constraint-based pattern mining.

problem is composed of a transactional dataset and the aforementioned properties of items. We name the combination of these two entities *mining view*, and denote it by  $MV$ .

**Definition 2.2** (*Mining view*). We call *mining view*  $MV$  a pair  $(\mathcal{D}, \mathcal{P})$ , where  $\mathcal{D}$  is a transactional dataset (see Definition 2.1), and  $\mathcal{P} = \{p_1, \dots, p_m\}$  is a set of  $m$  functions ( $m \geq 0$ ), denoting the *properties* of interest for each item in  $\mathcal{I}$ , such that  $p_j(x_i)$  denotes the  $j$ -th property value of the  $i$ -th item. An example of mining view is given in Fig. 3.

We can now define a set of constraints over the item properties  $\mathcal{P}$ :

$$\mathcal{C}_{\mathcal{P}} = \{\mathcal{C}_{\mathcal{P}}^1, \dots, \mathcal{C}_{\mathcal{P}}^n\}$$

where each of these constraints is defined as  $\mathcal{C}_{\mathcal{P}}^h :: 2^{\mathcal{I}} \rightarrow \{0, 1\}$ ,  $\mathcal{C}_{\mathcal{P}}^h(X) = 1$  (*True*) if  $X$  satisfies the constraint, and  $\mathcal{C}_{\mathcal{P}}^h(X) = 0$  (*False*) otherwise. Hereinafter, where it is clear from the context, we omit constraint parameters other than the itemset  $X$  itself.

For the sake of readability of this section, we do not discuss other constraints supported by CONQUEST that are not defined in terms of the property set  $\mathcal{C}_{\mathcal{P}}$ . In particular, *structural and syntactical constraints* that define the *form* of the valid itemsets that must be extracted from  $\mathcal{D}$ . For example, constraints regarding the length of the itemsets extracted, or constraints forcing the presence/absence of item subsets in each itemset extracted.

The *constrained frequent itemset mining* problem can finally be defined as follows:

**Definition 2.3** (*Constrained frequent itemset mining*). Given a mining view  $MV = (\mathcal{D}, \mathcal{P})$ , a minimum support threshold  $\delta \geq 1$ , and a set (possibly empty) of user-defined constraints  $\mathcal{C}_{\mathcal{P}} = \{\mathcal{C}_{\mathcal{P}}^1, \dots, \mathcal{C}_{\mathcal{P}}^n\}$ , the *constrained frequent itemset mining problem* requires to compute all the *valid itemsets* occurring in the transactions of  $\mathcal{D}$ :<sup>2</sup>

$$\{X \subseteq \mathcal{I} \mid \mathcal{C}_{freq}(X) \wedge \mathcal{C}_{\mathcal{P}}^1(X) \wedge \dots \wedge \mathcal{C}_{\mathcal{P}}^n(X)\}$$

<sup>2</sup> Note that the minimum frequency constraint  $\mathcal{C}_{freq}(X)$  must always be part of the conjunction of constraints in the query, at least with a minimum support threshold  $\delta = 1$ .

**Example 1.** The following is an example of constraint-based mining query over the mining view in Fig. 3:

$$supp_{\mathcal{D}}(X) \geq 3 \wedge sum(X.price) \geq 30$$

where the support constraint is  $\mathcal{C}_{freq}(X) :: supp_{\mathcal{D}}(X) \geq 3$ , while the only constraint defined over  $\mathcal{P}$  is  $\mathcal{C}_{\mathcal{P}}^1(X) :: sum(X.price) \geq 30$ .

The result of such query are the two following itemsets  $\{beer, wine\}$  and  $\{beer, wine, chips\}$ .

This is the theoretical setting in which all the research on constrained frequent itemset mining has been developed. Unfortunately, when we come to real-world data stored in relational DB, we find a gap between how data are actually organized and a mining view like the one in Fig. 3.

Firstly, data are stored in relations, and thus we have to transform them in order to build the transactions of our input transactional dataset  $\mathcal{D}$ . Secondly, in the mining view we defined each property of interest of an item as a function. So, for each database attribute selected as an item property, a *functional dependence* with the items must exist, i.e., each property has not to change along the database entries. This is rarely the case in real-world data. As an example consider property *price* of item *beer* in a sales database, during a period of six months: such a property is obviously floating during the period. As discussed below, in this case we have to force a functional dependency before creating the mining view, in order to apply our constraint-based mining framework.

In the following we thus describe how CONQUEST actually closes the gap existing between the formal computational framework introduced above and the actual data stored in relational databases.

## 2.2. Building a mining view from a relational database

Consider the two relational tables in Figs. 4(a) and (b): they contain all the information needed to build the mining view  $MV$  in Fig. 3. We can now formalize this data transformation process.

Let  $\mathcal{R}$  be a relational expression over a relational database DB, such that  $\mathcal{R}$  contains all and only the information needed to answer a given mining query. The

corresponding mining view  $MV$  is uniquely determined by specifying a partition of its attributes, and then generating items and transactions, and the table of the properties associated with each item.

**Definition 2.4** (Mining view definition). Given a relation  $\mathcal{R}$  over a relational database DB, let  $sch(\mathcal{R})$  denote its schema. We can induce a mining view  $MV \equiv \langle \mathcal{D}, \mathcal{P} \rangle$ , in accordance with Definition 2.2, by partitioning its attributes into three sets  $T$ ,  $I$  and  $P$ , i.e.,  $sch(\mathcal{R}) = T \cup I \cup P$ , where  $T \cap I = \emptyset$ ,  $T \cap P = \emptyset$  and  $I \cap P = \emptyset$ . We call this partition and the consequent generation of  $MV$  a *mining view definition*. Moreover we denote it as  $\mathcal{R}_{T,I,P} \equiv MV \equiv \langle \mathcal{D}, \mathcal{P} \rangle$ . In order to formally define  $\mathcal{R}_{T,I,P}$ , in the following we will use  $\sigma$ ,  $\pi$  and  $\times$  to denote the usual selection, projection and cartesian product operators of the relational algebra.

The induced transactional dataset  $\mathcal{D}$  is thus defined as follows. We construct a transaction  $t_{tid}$  for each distinct tuple  $tid$  in  $\pi_T(\mathcal{R})$ :

$$\mathcal{D} = \{t_{tid} | tid \in \pi_T(\mathcal{R})\}$$

In turn, each transaction  $t_{tid}$  is a set of items, where each item is defined in terms of the attribute partition  $I$  of  $\mathcal{R}$ . More specifically, each item is a pair attribute's name–attribute's value, i.e.  $\langle att\_name, value \rangle$ , where  $att\_name \in I$ . Therefore, we have a distinct item  $\langle att\_name, value \rangle$  for each possible distinct value value in the domain of

$att\_name$ . Hence we can define each transaction  $t_{tid} \in \mathcal{D}$  as

$$t_{tid} = \bigcup_{i \in I} (\{i\} \times \pi_i(\sigma_{T=tid}(\mathcal{R})))$$

Finally, the properties functions  $\mathcal{P} = \{p_1, \dots, p_n\}$  are defined as follows, in terms of the attribute partition  $P$ , one for each attribute occurring in the partition  $P$ :

$$p_j(\langle att\_name, value \rangle) = \pi_j(\sigma_{att\_name=value}(\mathcal{R}))$$

$$att\_name \in I, j \in P$$

**Example 2** (Defining a mining view). Let  $\mathcal{R}$  be the relation  $sales \bowtie_{item=name} product$  in Fig. 4(c). Given the following partition of its attributes:

$$T = \{ date, cust \}; \quad I = \{ name \};$$

$$P = \{ price, type \}$$

we obtain  $\mathcal{R}_{T,I,P}$  that corresponds to the mining view of Fig. 3. Note that, for the sake of readability, instead of representing the various items with a pair  $\langle att\_name, value \rangle$ , in Fig. 3 we represent them simply as value. For example, the items  $\langle name, beer \rangle$  and  $\langle name, chips \rangle$  are represented as *beer* and *chips*. In this example we have constructed the transactions grouping tuples by *date* and *customer*, considering the object of our analysis the purchases made by the same customer in the same day. In an alternative analysis we could be interested in the purchases of each customer in the whole period, considering all her/his basket in the period as a unique

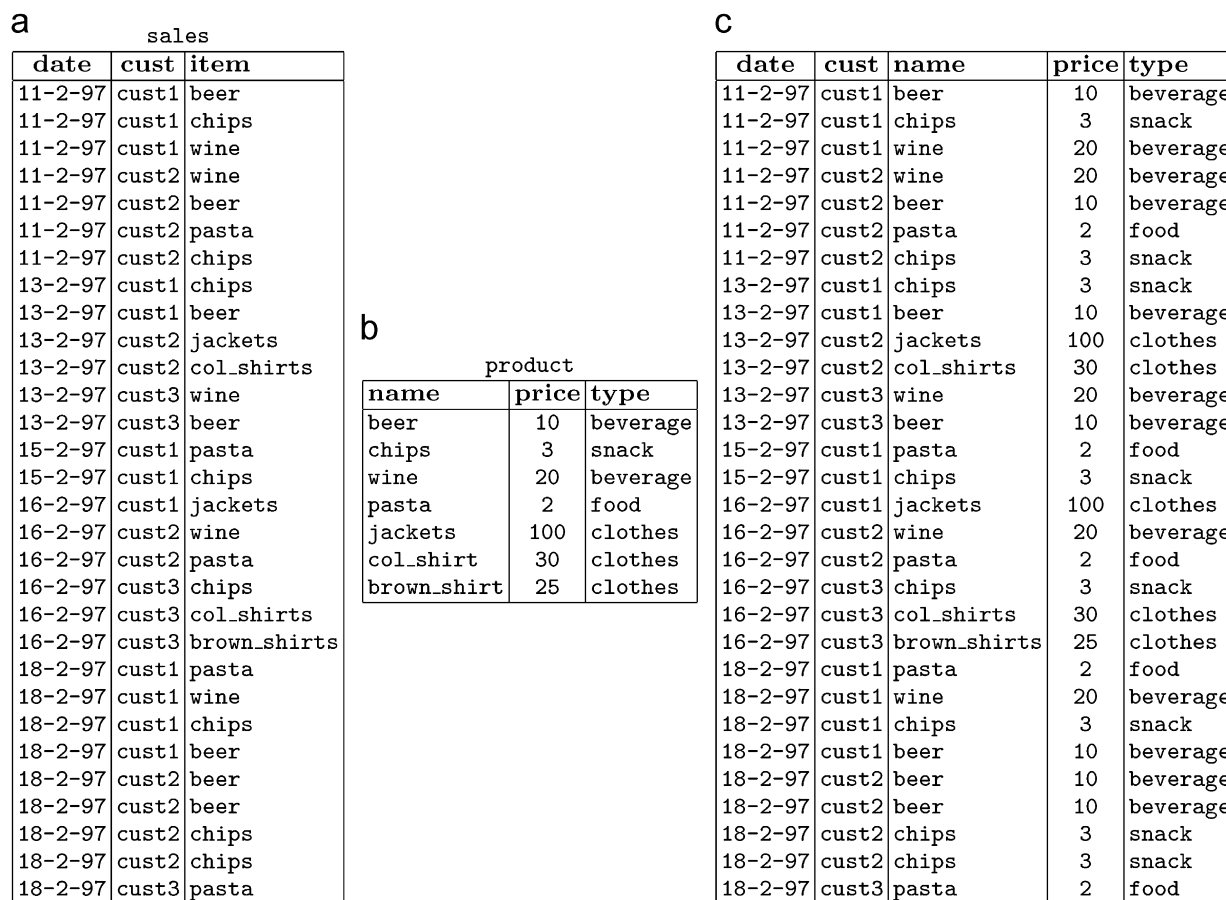


Fig. 4. An example sales table (a), a product table (b) and the table resulting from the join  $sales \bowtie_{item=name} product$  (c).

basket. In this case we would construct transactions by grouping tuples only by customer (i.e.,  $T = \{\text{cust}\}$ ).

It is worth noting that the simple mechanism described above allows CONQUEST to deal with both intra-attribute (as in the example above) and inter-attribute pattern mining (as discussed in the following example). In particular, the last can be easily obtained by specifying more than one attribute as belonging to  $I$ .

**Example 3** (*Inter-attribute pattern discovery*). As an example of inter-attribute pattern discovery, consider each tuple in a relational table as a transaction. Let  $\mathcal{R}$  a table recording information of our customers:

```
customer(cid, name, age, gender,
marital_status, occupation, education),
```

and suppose that we want to compute frequent patterns over the various characteristics of our customers. We can define our mining view  $\mathcal{R}_{T,I,P}$  by setting:

- $T = \{\text{cid}\}$ ,
- $I = \{\text{age, gender, marital\_status, occupation, education}\}$ , and
- $P = \emptyset$ .

In the case a primary key (such as `cid`) to identify transactions is not available, CONQUEST allows the user to define the intrinsic tuple identifier as the transaction identifier `tid`. In this case a distinct transaction will be created from each tuple in the table.

### 2.2.1. Solving conflicts on properties

Finally, as mentioned above, the constraint-based mining paradigm only considers properties  $P$  in *functional dependence* with items. When we select a property attribute in relation  $\mathcal{R}$  that does not satisfy such requirement, we have a property conflict.

**Definition 2.5** (*Property conflict*). Given a relation  $\mathcal{R}$ , and a mining view definition  $\mathcal{R}_{T,I,P}$ , we get a property conflict whenever  $\exists p \in P$  such that the functional dependency  $I \rightarrow p$  does not hold in  $\mathcal{R}$ .

In order to mine patterns according to the constrained frequent itemset mining framework of Definition 2.3, we have to modify input data in order to force the functional dependency for each  $p \in P$ . For example, suppose that, in the tuples of  $\mathcal{R}$ , attribute  $p_j$  takes different values whenever each attribute in  $I$  assumes a given value. In order to resolve the conflict we will assign to  $p_j$  the value:

$$p_j(\text{att\_name, value}) = f(\pi_j(\sigma_{\text{att\_name}=\text{value}}(\mathcal{R})))$$

$$\text{att\_name} \in I, j \in P$$

where the function  $f$  can be for instance minimum, maximum, average. Other methods are possible to solve the same conflict.

### 2.3. Crisp versus soft constraints

In Section 2.1 we presented a classical framework for constrained frequent itemsets mining: all exploited con-

straints correspond to *crisp* Boolean functions, whose codomain is either 0 (*False*) or 1 (*True*).

In CONQUEST we are also interested in exploiting *soft* constraints, according to the new paradigm for pattern discovery introduced in [14]. The codomain of such constraints is the continuous interval  $[0, 1]$ . Roughly speaking, the use of soft constraints allows the user to describe what is the “*shape*” of the patterns of interest, and receive back those patterns that “*mostly*” exhibits such shape.

The paradigm of pattern discovery based on soft constraints has various merits over the crisp ones:

- It is not rigid: a potentially interesting pattern is not discarded for just a slight violation of a constraint.
- It can order patterns with respect to interestingness (level of constraints satisfaction): this allows us to say that a pattern is *more interesting* than another, instead of strictly dividing patterns in interesting and not interesting.
- From the previous point, it follows that our paradigm allows for naturally expressing *top-k* queries based on constraints: e.g., the data analyst can ask for the *top-10* patterns with respect to a given description, like a conjunction of soft constraints.
- Alternatively, we can ask to the system to return all and only the patterns which exhibit an interest level larger than a given threshold  $\lambda$ .

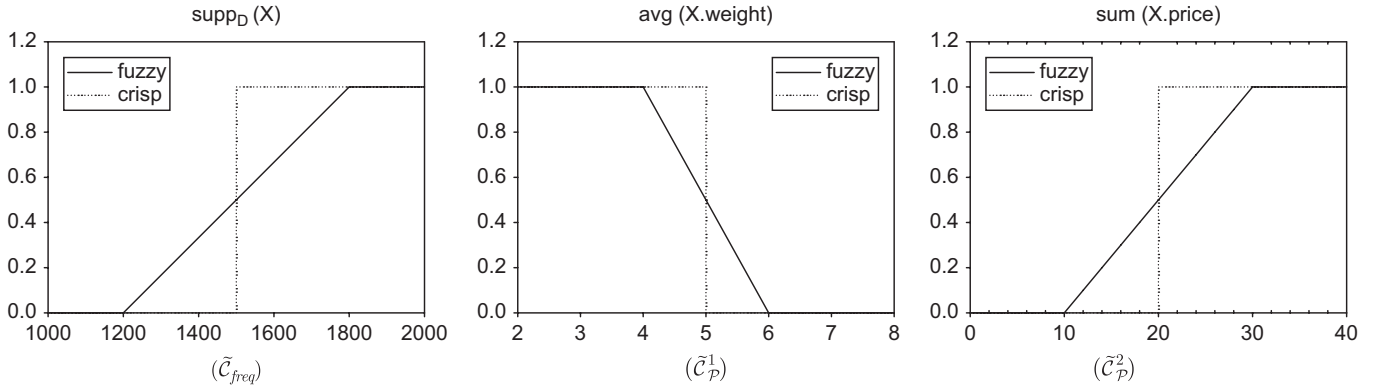
The paradigm introduced in [14] is based on the mathematical concept of *semiring*. In this paper we avoid entering in unnecessary details. We just mention two possible instantiations of the framework for soft-constrained frequent patterns based on the *fuzzy* and the *probabilistic* semirings, as described in the following. The main difference between the two semirings is the way the soft constraints are combined to determine the interestingness of an itemset.

**Definition 2.6** (*Soft constraints*). Given a mining view  $MV \equiv \langle \mathcal{D}, \mathcal{P} \rangle$ , a *soft frequency constraint* is a function  $\tilde{c}_{\text{freq}}(X) :: 2^{\mathcal{I}} \rightarrow [0, 1]$ . Similarly we can define a set of *soft constraints* over  $\mathcal{P}$ :  $\tilde{c}_{\mathcal{P}} = \{\tilde{c}_{\mathcal{P}}^1, \dots, \tilde{c}_{\mathcal{P}}^n\}$ , where  $\tilde{c}_{\mathcal{P}}^h :: 2^{\mathcal{I}} \rightarrow [0, 1]$ .

Therefore, a soft constraints returns a value in the interval  $[0, 1]$ , denoting the level of *constraint satisfaction* or *interestingness*.

For sake of simplicity, we restrict our system to constraints which behave as those ones in Fig. 5. They return a value which grows linearly from 0 (*False*) to 1 (*True*) in a certain interval, while they are 0 before the interval and equal to 1 after the interval.

To describe such a simple behavior, we just need two parameters: a value  $t$ , associated with the center of the interval (corresponding to an interest value of 0.5), and a parameter  $\alpha$  to adjust the gradient of the function that is named *softness parameter*. Therefore, the interval in which the constraint turns out to be satisfied, with a level of interestingness ranging from 0 to 1, is  $[t - \alpha t, t + \alpha t]$ .



**Fig. 5.** Graphical representation of possible fuzzy/probabilistic instances of three constraints. The dotted lines represent the crisp version of the same constraints.

**Example 4.** In Fig. 5 we provide graphical representations of the following soft constraints:

- $\tilde{\mathcal{C}}_{freq} :: \text{supp}_{\mathcal{D}}(X) \geq 1500$  ( $\alpha = 0.2$ ),
- $\tilde{\mathcal{C}}_{\mathcal{P}}^1 :: \text{avg}(X.\text{weight}) \leq 5$  ( $\alpha = 0.2$ ),
- $\tilde{\mathcal{C}}_{\mathcal{P}}^2 :: \text{sum}(X.\text{price}) \geq 20$  ( $\alpha = 0.5$ ).

Note that when the softness parameter  $\alpha$  is equal to 0 we obtain the crisp version of the constraint.

Given a set of soft constraints, we need to define how the global interestingness value is computed when a combination of soft constraints are provided in a mining query. To this end, we will exploit the two semirings, the *fuzzy* and the *probabilistic* ones.

**Definition 2.7.** Given a combination of soft constraints  $\otimes \tilde{\mathcal{C}} \equiv \tilde{\mathcal{C}}_1 \otimes \dots \otimes \tilde{\mathcal{C}}_n$ , we define the interest level of an itemset  $X \in 2^I$  as:

- $\otimes \tilde{\mathcal{C}}(X) = \min(\tilde{\mathcal{C}}_1(X), \dots, \tilde{\mathcal{C}}_n(X))$  in a *fuzzy* query,
- $\otimes \tilde{\mathcal{C}}(X) = \times \tilde{\mathcal{C}}_1(X), \dots, \tilde{\mathcal{C}}_n(X)$  in a *probabilistic* query.

**Example 5.** In this example, we use for the patterns the notation  $p: \langle v_1, v_2, v_3 \rangle$ , where  $p$  is an itemset, and  $\langle v_1, v_2, v_3 \rangle$  denote, respectively, the three values  $\text{supp}_{\mathcal{D}}(p)$ ,  $\text{avg}(p.\text{weight})$  and  $\text{sum}(p.\text{price})$ , corresponding to the three constraints in Fig. 5.

Consider, for example, the following three patterns:

- $p_1 : \langle 1700, 0.8, 19 \rangle$ ,
- $p_2 : \langle 1550, 4.8, 54 \rangle$ ,
- $p_3 : \langle 1550, 2.2, 26 \rangle$ .

For the pattern  $p_1 : \langle 1700, 0.8, 19 \rangle$  we obtain that:  $\tilde{\mathcal{C}}_{freq}(p_1) = 0.83$ ,  $\tilde{\mathcal{C}}_{\mathcal{P}}^1(p_1) = 1$  and  $\tilde{\mathcal{C}}_{\mathcal{P}}^2(p_1) = 0.45$ . Since in a *fuzzy query* the constraint combination operator is  $\min$ , we get that the interest level of  $p_1$  is 0.45. In the same way we obtain the interest levels for  $p_2$  and  $p_3$ :

- $p_1 : \tilde{\mathcal{C}}_{freq} \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^1 \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^2(1700, 0.8, 19) = \min(0.83, 1, 0.45) = 0.45$ ,
- $p_2 : \tilde{\mathcal{C}}_{freq} \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^1 \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^2(1550, 4.8, 54) = \min(0.58, 0.6, 1) = 0.58$ ,

- $p_3 : \tilde{\mathcal{C}}_{freq} \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^1 \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^2(1550, 2.2, 26) = \min(0.58, 1, 0.8) = 0.58$ .

Therefore, with this particular instance we obtain that  $p_2$  and  $p_3$  are the most interesting patterns among the three.

Similarly, since in a *probabilistic query* the constraint combination operator is the arithmetic multiplication  $\times$ , we get that the interest level of  $p_1$  is 0.37. In the same way we obtain the interest levels for  $p_2$  and  $p_3$ :

- $p_1 : \tilde{\mathcal{C}}_{freq} \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^1 \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^2(1700, 0.8, 19) = \times(0.83, 1, 0.45) = 0.37$ ,
- $p_2 : \tilde{\mathcal{C}}_{freq} \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^1 \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^2(1550, 4.8, 54) = \times(0.58, 0.6, 1) = 0.35$ ,
- $p_3 : \tilde{\mathcal{C}}_{freq} \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^1 \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^3(1550, 2.2, 26) = \times(0.58, 1, 0.8) = 0.46$ .

Therefore, with this particular instance we obtain that  $p_3$  is the most interesting pattern.

We are now ready to define a framework for *soft-constrained* frequent itemset mining, by extending Definition 2.3, concerning the *crisp-constrained* counterpart.

**Definition 2.8** (*Soft-constrained frequent itemset mining*). Given a mining view  $MV = (\mathcal{D}, \mathcal{P})$ , a minimum support threshold  $\delta \geq 1$ , and a (possibly empty) set of user-defined constraints  $\tilde{\mathcal{C}}_{\mathcal{P}} = \{\tilde{\mathcal{C}}_{\mathcal{P}}^1, \dots, \tilde{\mathcal{C}}_{\mathcal{P}}^n\}$ , the *soft-constrained frequent itemset mining problem* has to mine part of the set  $\mathcal{V}$  of *valid itemsets* occurring in  $\mathcal{D}$  according to the soft constraints:

$$\mathcal{V} = \{X \in 2^{\mathcal{I}} \mid \otimes \tilde{\mathcal{C}}(X) > 0\}$$

where  $\otimes \tilde{\mathcal{C}}(X) = (\tilde{\mathcal{C}}_{freq}(X) \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^1(X) \otimes \dots \otimes \tilde{\mathcal{C}}_{\mathcal{P}}^n(X))$ , and  $\otimes$  can be either  $\min$  (fuzzy) or  $\times$  (probabilistic) according to Definition 2.7.

In particular, by specifying two further parameters  $\lambda$  and  $k$ , we can state the following two sub-problems, which only select part of all the soft-constrained valid itemsets  $\mathcal{V}$ .

$\lambda$ -interesting: given a *minimum interest threshold*  $\lambda \in (0, 1]$ , it is required to compute all  $\lambda$ -interesting patterns, i.e.,  $\{X \in \mathcal{V} \mid \otimes \tilde{\mathcal{C}}(X) \geq \lambda\}$ .



*top-k*: given a threshold  $k \in \mathbb{N}$ , it is required to mine the top- $k$  patterns  $X \in \mathcal{V}$  with respect to the order determined by measure  $\otimes_{\tilde{C}}(X)$ , i.e., the most interesting itemsets.

### 3. The SPQL mining query language

In this section we propose a query language for pattern discovery named SPQL. We extend a preliminary work of us [15] by introducing new features in the language. In particular, two main kinds of SPQL queries exist: standard mining queries with *crisp* constraints, and mining queries with *soft* constraints. Moreover, SPQL is also used for defining the mining view, solving problems as the functional dependency of the property attributes from the item ones, or for the discretization of numerical continuous attributes.

As discussed in Section 5.2, CONQUEST offers simple GUI mechanisms to facilitate SPQL query definition, as well as to solve conflict regarding property conflicts or to discretize continuous items attributes. However, all these knowledge discovery tasks can directly be specified in SPQL.

#### 3.1. SPQL mining queries with crisp constraints

In Section 2 we have provided the theoretical background definitions needed to specify a constraint-based mining queries on a relational database (see Definitions 2.3 and 2.4). In the following definition we summarize all the elements that play a role in such queries, used to specify a complete knowledge discovery process, namely data selection, data preparation and transformation, and finally pattern discovery.

**Definition 3.1** (*Crisp constraint-based frequent pattern query*). A constraint-based frequent pattern query  $\mathcal{Q}$  over a relational database DB, is defined by six elements  $\mathcal{Q} \equiv \langle \mathcal{R}, T, I, P, \delta, \mathcal{C}_{\mathcal{P}} \rangle$ : a relation  $\mathcal{R}$  over DB, a mining view definition  $\mathcal{R}_{T,I,P}$ , a minimum support threshold  $\delta$  and a conjunction of constraints  $\mathcal{C}_{\mathcal{P}}$ .

The result of the query is the set of itemsets  $X$  (along with their supports), that are frequent (with respect to  $\delta$ ) in  $\mathcal{D}$  where  $\mathcal{M}_{T,I,P} \equiv \langle \mathcal{D}, \mathcal{P} \rangle$ , and that satisfy the conjunction of constraints  $\mathcal{C}_{\mathcal{P}}$ :

$$\{(X, \text{supp}_{\mathcal{Q}}(X)) \mid X \in 2^{\mathcal{I}} \text{ and } \text{supp}_{\mathcal{Q}}(X) \geq \delta \wedge \mathcal{C}_{\mathcal{P}}(X)\}$$

The result of such mining query can be materialized as relations in DB. In particular, such materialization creates three tables:

ITEMS		
item_id	item	price
1	$\langle \text{name, beer} \rangle$	10
2	$\langle \text{name, wine} \rangle$	20
3	$\langle \text{name, chips} \rangle$	3

ITEMSETS	
itemset_id	item_id
1	1
1	2
2	1
2	2
2	3

SUPPORTS	
itemset_id	support
1	4
2	3

Fig. 6. Materialization of the result of the query in Example 1.

ITEMS: a table containing for each item  $i \in \mathcal{I}$ , belonging to at least one of the returned itemsets, an automatically generated identifier `item_id`, the literal associated to the `item`, and a value for each attribute property  $p_j(i)$ . The literal associated with each item  $i \in \mathcal{I}$  should have the form  $\langle \text{att\_name, value} \rangle$ , where `att_name`  $\in I$ , and `value` is a value assumed by `att_name` in  $\mathcal{R}$ .

SUPPORTS: a table modeling the inclusion relation of the various items (`item_id`) within each returned itemset, in turn identified by an automatically generated identifier `itemset_id`.

ITEMSETS: a table containing, for each returned itemset  $X$  (`itemset_id`), its support  $\text{supp}_{\mathcal{Q}}(X)$ .

**Example 6.** Consider the mining query in Example 1. The result of that query is materialized as shown in Fig. 6.

An SPQL query  $\mathcal{Q}$  adopts an SQL-like syntax. An example has already been shown in Fig. 2, for which we have identified how the three main tasks of the knowledge discovery process (i.e., source data selection, data pre-processing and pattern discovery) can be expressed.

In Fig. 7 we provide a more complex example query: in the following we summarize the correspondence between all the parameters of  $\mathcal{Q} \equiv \langle \mathcal{R}, T, I, P, \delta, \mathcal{C}_{\mathcal{P}} \rangle$  and the various statements of a standard SPQL query:

- (1) The minimum frequency threshold  $\delta$  (line 1 of the query in Table 7).
- (2) The relational table  $\mathcal{R}$  from which we can define the mining view  $\mathcal{R}_{T,I,P}$  (lines 2–4).
- (3) The partition of the attributes of  $\mathcal{R}$  in the sets  $T, I$  and  $P$ , or, in other terms, the mining view definition  $\mathcal{R}_{T,I,P}$  (lines 5–7).
- (4) A conjunction  $\mathcal{C}_{\mathcal{P}}$  of constraints defined over the item properties (line 8).

As stated in Section 2, if we select as item property an attribute that is not in functional dependency with an item attribute, we have a property conflict. CONQUEST is able to handle such constraints appropriately. It first analyzes the mining view definition given in the query, and if there is any conflict, it raises a warning. The user can either take the conflict into consideration, and resolving it by using one of the method provided by the system, or proceed to the query evaluation without caring the warning. In the latter case CONQUEST automatically applies the default method to resolve the conflict. In more details, given a conflicting property attribute  $P_i$ , the defaults method is *take average*, which re-assigns to  $P_i$  the average

---

```

1. MINE PATTERNS WITH SUPP>= 5 IN
2. SELECT product.product_name, product.gross_weight, sales_fact_1998.time_id,
   sales_fact_1998.customer_id, sales_fact_1998.store_id
3. FROM [product], [sales_fact_1998]
4. WHERE sales_fact_1998.product_id=product.product_id
5. TRANSACTION sales_fact_1998.time_id, sales_fact_1998.customer_id,
   sales_fact_1998.store_id
6. ITEM product.product_name
7. PROPERTY product.gross_weight
8. CONSTRAINED BY Sum(product.gross_weight)<=30

```

---

**Fig. 7.** An example SPQL mining query defined within CONQUEST on the famous foodmart2000 datamart.

of all its original values that were associated with the item attribute. Other available methods to solve conflicts are *take maximum*, *take minimum*, *sum*, *first* or *last*.

The user can specify one of these methods to solve conflicts in the SPQL query, by adding a TAKE clause in the property attribute definition. For example, consider the query in Fig. 7. In the property attribute definition in line 7, we could have:

```
PROPERTY product.gross_weight TAKE Avg
```

In CONQUEST we have chosen a well-defined set of classes of constraints. Many of them are defined in terms of the property attributes  $P$ , others concern the structural and syntactical forms of the valid itemsets to extract from  $\mathcal{D}$ . These constraints have been deeply studied and analyzed in the past few years, in order to find nice properties that can be used at mining time to reduce the computational cost. In particular, as discussed later in Section 4, our system is able to deal with anti-monotone, succinct [2], monotone [16], convertible [17] and loose anti-monotone [18] constraints. Such classes include all the constraints based on the aggregates listed in Table 1.

So far we have presented an SPQL query in its basilar form. Other SPQL queries can be expressed: in particular queries that specify *soft constraints* [14], and *discretization tasks*, as presented in the following subsections.

Finally, note that CONQUEST's SPQL is a superset of SQL, in a double sense: first any SPQL query contains an SQL query needed to define the data source; second, in CONQUEST we allow the user to define any SQL query in place of an SPQL query, which could be useful, for instance, to pre-process the data or post-process the extracted patterns.

### 3.2. SPQL mining queries with soft constraint

In CONQUEST we have introduced the possibility of defining queries according to the new paradigm of pattern discovery based on *soft constraints* [14], according to the framework discussed in Section 2.3.

A soft-constrained frequent pattern query  $\mathcal{Q}$  over a relational database DB can be either  $\lambda$ -interesting or top- $k$ , but also *probabilistic* or *fuzzy*, depending on the way the soft constraints must be combined. Thus, besides defining the soft constraints along with the associated softness levels, in  $\mathcal{Q}$  we have to specify the query kind. The SPQL

**Table 1**  
The set of available constraints

<b>subset</b>	subset	<b>supset</b>	superset
<b>asubset</b>	attributes are subset	<b>len</b>	length
<b>asupset</b>	attributes are superset	<b>acount</b>	attributes count
<b>min</b>	minimum	<b>max</b>	maximum
<b>range</b>	range	<b>sum</b>	sum
<b>avg</b>	average	<b>var</b>	variance
<b>std</b>	standard deviation	<b>spv</b>	sample variance
<b>md</b>	mean deviation	<b>med</b>	median

syntactic sugar to define such queries is provided by means of the following example.

**Example 7.** Fig. 8 shows a complex SPQL query exploiting the soft constraint paradigm. In particular it is a *probabilistic* query requiring to mine the top five patterns with respect to a given combination of three soft constraints: the frequency constraint, support larger than 5 with 0.4 softness, plus two aggregate soft constraints defined over the properties `product.gross_weight` and `product.units_per_case`. This is a true mining query, defined within CONQUEST on the famous foodmart2000 datamart.

In line 1 we specify the query type definition (in this case we have a *top- $k$*  one with the appropriate threshold) and the semiring (in this case we have a *probabilistic* one) in which the query must be evaluated. In line 2 a minimum frequency constraint is defined with threshold 5 and 0.4 softness level. From lines 3 to 5 we have the usual SQL select-from-where statement, defining the data source  $\mathcal{D}$  for the query. Lines from 6 to 8 contain the transactional mining view definition. Line 9 contains the two other constraints defined over the item properties with their associated softness parameters. How queries based on soft constraints are evaluated is described in Section 4.

### 3.3. Discretization queries

Discretization is often a needed step when preparing data for associative analysis. CONQUEST provides the user with a functionality for discretizing continuous attributes

---

```

1. MINE TOP 5.0 PROBABILISTIC PATTERNS
2. WITH SUPP>= 5.0 SOFT 0.4 IN
3. SELECT product.product_name, product.gross_weight,
product.units_per_case, sales_fact_1998.time_id,
sales_fact_1998.customer_id, sales_fact_1998.store_id
4. FROM [product], [sales_fact_1998]
5. WHERE sales_fact_1998.product_id=product.product_id
6. TRANSACTION sales_fact_1998.time_id, sales_fact_1998.customer_id, sales_fact_1998.store_id
7. ITEM product.product_name
8. PROPERTY product.gross_weight, product.units_per_case
9. CONSTRAINED BY Average(product.gross_weight)<=20 SOFT 0.8
AND Sum(product.units_per_case)>=50 SOFT 0.5

```

---

**Fig. 8.** An example SPQL probabilistic mining query defined within CONQUEST on the foodmart2000 datamart.

---

```

1. DISCRETIZE old_attribute AS new_attribute
2. FROM table
3. IN number_of_bins [DISCRETIZATION METHOD] BINS
4. SMOOTHING BY [SMOOTHING METHOD]

1. DISCRETIZE old_attribute AS new_attribute
2. FROM table
3. IN (l1,u1), ..., (ln,un) BINS
4. SMOOTHING BY [SMOOTHING METHOD]

```

---

**Fig. 9.** An SPQL equal-width or equal-depth discretization query (above), and an SPQL free-partitioning discretization query (below):  $(l_1, u_1), \dots, (l_n, u_n)$  are the user-defined bin boundaries.

of a table. The following discretization methods are provided:

*Equal width:* the domain of a continuous attribute is partitioned into bins of the same length.

*Equal depth:* the domain of a continuous attribute is partitioned into bins containing same number of elements.

*Free partitioning:* the partition is defined by the user.

The smoothing defines the type of information that will be stored in the new attribute generated by the discretization. The following smoothing methods are provided:

*Bin boundaries:* the bin boundaries are stored as text.

*Average:* the average value for the elements in the bin.

*Count:* the count of the elements in the bin.

Given these methods, the syntax of possible CONQUEST discretization queries is described in Fig. 9.

### 3.4. Related work on data mining query languages

In this section we discuss other approaches to the data mining query language definition issue. For the sake of presentation we focus only on few, most relevant, approaches: we are aware that this presentation does not exhaustively cover the wide state-of-the-art of the research (and also the development) on data mining systems and query languages.

The problem of providing an effective interface between data sources and data mining tasks has been a

primary concern in data mining. There are several perspectives upon which this interface is desirable, the most important ones being (i) to provide a standard formalization of the desired patterns and the constraints they should obey to; and (ii) to achieve a tighter integration between the data sources and the relational databases (which likely accommodate them). The common ground of most of the approaches can be summarized as follows:

- Create and manipulate data mining models through an SQL-based interface (thus implementing a “command-driven” data mining metaphor).
- Abstract away the algorithmic particulars.
- Allow mining tasks to be performed on data in the database (thus avoiding the need to export to a special-purpose environment).

Approaches differ on what kinds of models should be created (which patterns are of interest), and what operations we should be able to perform (which constraints the patterns should satisfy). The query language proposed in [19,20] extends SQL with the new operator `MINE RULE`, which allows the computation and coding of associations in a relational format. Let us consider the relation `transaction(Date, CustID, Item, Value)` that contains the transactions of a sales representative.

The following rule allows the extraction of the rules with support 20% and confidence 50%:

```
MINE RULE Associations AS
  SELECT DISTINCT 1..n Item AS BODY,
    1..1 Item AS HEAD, SUPPORT, CONFIDENCE
  WHERE BODY.Value > 100 AND HEAD.Value > 100
  FROM transaction
  GROUP BY CustID
    HAVING COUNT(Item) > 4
  CLUSTER BY Date
    HAVING BODY.Date < HEAD.Date
  EXTRACTING RULES WITH SUPPORT: 0.2, CONFIDENCE: 0.5
```

The above expression specifies the mining of associations of purchased items such that the right part of the rule (consisting of only one item) has been purchased after the left part of the rule (that can consist of more than one item), and related to those customers who bought more than four items. Moreover, we consider items only with a value greater than 100.

The above approach reflects the following features:

- The source data is specified as a relational entity, and data preparation is accomplished by means of the usual relational operators. For example, the source table can be specified by means of usual join operations, selections and projections.
- The extended query language allows mining of uni-dimensional association rules. The `GROUP BY` keyword allows the specification of the transaction identifier, while the item description is specified in the `SELECT` part of the operator.
- Limited forms of background knowledge can be specified, by imposing some conditions over the admitted values of `BODY` and `HEAD`, and by using multiple source tables. Notice, however, that relying directly on SQL does not allow direct specification of more expressive constructs, such as, e.g., concept hierarchies. A limited form of data reorganization is specified by the `CLUSTER` keyword, that allows the specification of *topology* constraints (i.e., membership constraints of the components of rules to clusters).
- Concerning interestingness measures, the above operator allows the specification of the usual support and confidence constraints, and further constraints over the contents of the rules (in particular, the `SELECT` keyword allows the specification of cardinality constraints).
- Extracted knowledge is represented by means of relational tables, containing the specification of four attributes: Body, Head, Support and Confidence.

Similarly to `MINE RULE`, the `DMQL` language [21,22] is designed as an extension of SQL that allows to select the primary source knowledge in SQL-like form. However, the emphasis here is on the kind of patterns to be extracted. Indeed, `DMQL` supports several mining tasks

involving rules: characteristic, discriminant, classification and association rules. The following query:

```
use database university_database find
characteristic rules
related to gpa, birth_place, address, count(*)%
from student where status = "graduate" and major =
"cs"
and birth_place = "Canada"
with noise threshold = 0.05
```

specifies that the database used to extract the rules is the university database (use database university\_data-base), and the kind of rules you are interested in are characteristic rules (find characteristic rules) with respect to attributes `gpa`, `birth_place`, and `address` (related to ...). The query specifies also that this rules are extracted on the students who are graduated in computer science and born in Canada. As for `MINE RULE`, the specification of primary source knowledge is made explicit in the `from` and `where` clauses.

`DMQL` exploits a decoupled approach between specification and implementation, since the extraction is accomplished by external algorithms, and the specification of the query has the main objective of preparing the data and encoding them in a format suitable for the algorithms. Interestingly, `DMQL` allows the manipulation of a limited form of background knowledge, by allowing the direct specification of concept hierarchies.

Unfortunately, neither `MINE RULE` nor `DMQL` provide operators to further query the extracted patterns. The closure principle is marginally considered in `MINE RULE` (the mining result is stored into a relational table and can be further queried), but not considered at all within `DMQL`. By contrast, Imielinski and others [23] propose a data mining query language (`MSQL`) which seeks to provide a language both to selectively generate patterns, and to separately query them. `MSQL` allows the extraction of association rules only, and can be seen as an extension of `MINE RULE`. The pattern language of `MSQL` is based on multidimensional propositional rules, which are specified by means of *descriptors*. A descriptor is an expression of the form  $A_i = a_{ij}$ , where  $A_i$  is an attribute, and  $a_{ij}$  is either a value or a range of values in the domain of  $A_i$ . Hence, the rules extracted by `MSQL` have the form  $Body \Rightarrow Consequent$ , where *Body* is a conjunctset (i.e., the conjunction of an arbitrary number of descriptors such that each descriptor in the set refers to a different attribute) and *Consequent* is a single descriptor. Rules are generated by means of a `GetRules` statement which, apart from syntax issues, has similar features as `MINE RULE` and `DMQL`. In addition, `MSQL` allows for nested queries, that is, queries containing subqueries.

The extracted rules are stored in a *RuleBase*, from which they can be further queried, by means of the `SelectRules` statement. It is possible to select a subset of the generated rules that verify a certain condition

```
SelectRules(R)
where Body has { (Age = *), (Sex = *) }
and Consequent is { (Address = *) }
```

as well as to select the tuples of the input database that violate (satisfy) all (any of) the extracted rules:

```
Select* from Actor where VIOLATES ALL(
  GetRules(Actor)
  where Body is { (Age = *) }
  and Consequent is { (Sex = *) }
  and confidence > 0.3
)
```

A novel and completely different perspective to inductive databases querying has been devised in [24]. The basic intuition is that, if the pattern language  $\mathcal{L}$  were stored within relational tables, any constraint predicate  $Q$  could be specified by means of a relational algebra expression, and the DBMS could take care of implementing the best strategy for computing the solution space. Assume, for example, that sequence patterns are stored within a relational engine by means of the following relations:

- *Sequences* (*sid*, *item*, *pos*), representing each sequence by means of a sequence identifier, an item and its relative position within the sequence.
- *Supports* (*sid*, *supp*) which specifies, for each sequence, its frequency.

Then, the following SQL query, asking for sequences such that either their frequencies are greater than 60%, or item *a* occurs before item *b* within transactions, can be expressed as follows:

```
SELECT Supports.sid
FROM Sequences S1, Sequences S2, Supports
WHERE S1.sid = Supports.sid AND S2.sid = S1.sid
  AND Supports.supp > 60
  OR (S1.item = a AND S2.item = b AND S1.pos <
      S2.pos)
```

Clearly, the pattern language can be extremely huge, and hence it is quite unpractical to effectively store it. Indeed, the pattern language is represented as a *virtual* table, i.e., an empty table which has to be populated. In the above example, although the *Sequences* and *Supports* tables are exploited within the query, they are assumed to be virtual tables, i.e., no materialization actually exists for them within the DBMS. The idea here is that, whenever the user queries such pattern tables, an efficient data mining algorithm is triggered by the DBMS, which materializes those tuples needed to answer the query. Afterwards, the query can be effectively executed. Thus, the core of the approach is a constraint extraction procedure, which analyzes a given SQL query and identifies the relevant constraints. The procedure builds, for each SQL query, the corresponding relational algebra tree. Since virtual tables appear in the leaf nodes of the tree, a bottom-up traversal of the tree allows the detection of the necessary constraints. Finally, specific calls to a ME can be raised in order to populate those nodes representing virtual tables.

This approach has the merit of providing a real tight coupling between the ME and the DBMS, or, in other

terms, between the mining queries and the database queries. Indeed, this approach does not even require the definition of a data mining query language, since it is SQL itself to play such role. However, it is not clear how such approach could support a complex knowledge discovery process. For instance, the pre-processing step is completely overlooked by this approach: preparing data for mining would require long and complex SQL queries. Moreover, since we got no reference to the source data, it is not clear how the mining view could be defined and/or changed within a mining session. Consider again the *Sequences* and *Supports* relations in the above example, and suppose that the support of sequences patterns are computed with respect to a database of sequences of events with a *weekly* timescale: what does it happen if the analyst decides to move to the *daily* timescale?

The problem of providing little support to the pre-processing and evaluation phase of the knowledge discovery process, is common to all the query languages discussed above. In CONQUEST, differently from the pre-processing phase (e.g., easy mining view definition, attributes discretization), the evaluation phase is not supported at the language level. However, as described later in Section 5, we provide pattern browser capabilities such as the *on-the-fly constraints tuning*. More sophisticated post-processing of the extracted patterns, visualization and reasoning techniques, will be the subject of our on-going and future work.

#### 4. Algorithms and ME

Several researchers have focused their efforts in designing new frequent itemsets mining algorithms during the last decades, resulting in a overwhelming spectrum of completely different approaches as well as tiny effective optimizations. However, none of them showed to be faster or better (whatever this may mean) than all the others. The goodness of an algorithm depends on the characteristics of data being analyzed, the parameters of the mining task, and many other features of the computational environment.

While developing the CONQUEST's ME, one of our main goals was efficiency, in order to provide a prompt interaction with users. But also robustness, since we aimed at developing a software able to mine every kinds of datasets on every kinds of hardware. More importantly, our software must be able to exploit the high expressiveness of the SPQL language, thus pushing all these constraints deep down in the frequent pattern computation, rather than performing a mere post-processing filtering.

CONQUEST's ME exploits the state-of-the-art frequent pattern algorithms and constraint pushing techniques, as those ones developed in the last three years in our labs [25,26,16,27,18].

##### 4.1. Frequent itemsets mining

The CONQUEST ME is based on direct count and intersect (DCI) [25], a state-of-the-art high performance frequent itemsets mining algorithm.

DCI explores the search space level-wise, like Apriori, by first discovering the frequent itemsets of length one, then the frequent itemsets of length two, and so on until no longer frequent itemset exists. As other Apriori-like algorithms, DCI reiterates two basic steps. Given a collection of frequent itemsets of length  $k$ , a new set of possibly frequent itemsets of length  $k + 1$  is produced via the *candidate generation* function. Then, their actual support is calculated by scanning the dataset via the *support counting* function. These two steps are repeated until all the frequent itemsets have been discovered.

Although the depth-first visit is now considered a more fashionable strategy, DCI is as fast as other algorithm thanks to its internal vertical bitmap representation of the dataset, and the associated *list-intersection* counting method, based on fast bitwise operations. In more detail, this counting method permits DCI to compute the supports of candidate itemsets on-the-fly, also exploiting a small cache storing the most recently used list intersections. In the next section we will show how the level-wise visit of the search space adopted by DCI allowed us to adopt a unifying framework for constrained itemsets mining.

The added value that really makes DCI different from other algorithms is its nice feature of being *resource and data aware*.

- It is *resource aware* because, unlike other algorithms, it performs the first iterations out-of-core, and at each step prunes useless information from the original dataset thus reducing the amount of data to be used in the following iterations. When the reduced dataset is small enough to be loaded in main memory, it is converted and stored in-core as a vertical bitmap. The compact vertical representation allows a fruitful use of CPU's cache due to the spatial and temporal locality in accessing data.
- It is *data aware* because its behavior changes in presence of sparse or dense datasets. It uses an ad hoc representation (similar to the run length encoding) in the case of sparse datasets, and detects highly correlated items to save bitwise works in the case of dense datasets.

Being resource and data aware is not only a nice feature of DCI, but it is also a strong requirement of CONQUEST, due to the need of quickly mining real world datasets. This is the reason of our choice of DCI as the main building block of our ME. CONQUEST, by inheriting the same characteristics as DCI, turns out to be an extremely robust and fast software.

#### 4.2. A generalized unifying framework for constrained pattern mining

Constraint-based frequent pattern mining has been studied as a query optimization problem, i.e., developing efficient, sound and complete evaluation strategies for constraint-based mining queries. To this aim, properties of constraints have been studied comprehensively, e.g., *anti-monotonicity*, *succinctness* [2,28], *monotonicity* [16,29,30], *convertibility* [31], *loose anti-monotonicity* [18], and on the

basis of such properties efficient computational strategies have been defined. However, the proposed strategies cannot be exploited altogether in a comprehensive algorithm. A preliminary effort to propose a general framework for constrained frequent itemsets mining is [32]: CONQUEST is the actual realization of that framework, and thus it is the first software that is able to deal with all of these classes of constraints at the same time.

In the following, together with an exhaustive presentation of the different constraints classes, we will review the algorithms proposed to exploit constraints properties in order to reduce the computational cost of data mining algorithms by means of *search-space* and *data* reduction.

The first interesting property is the *anti-monotonicity*, which was already introduced with the Apriori [13] algorithm, since the minimum frequency is actually an anti-monotone constrains.

**Definition 4.1** (*Anti-monotone constraint*). Given an itemset  $X$ , a constraint  $\mathcal{C}_{AM}$  is *anti-monotone* if  $\forall Y \subseteq X : \mathcal{C}_{AM}(X) \Rightarrow \mathcal{C}_{AM}(Y)$ .

This *anti-monotonicity* property can be used to reduce both the search space and the data. A large portion of the search space can be pruned whenever we meet an itemset  $X$  that does not satisfy  $\mathcal{C}_{freq}$ , since no superset of  $X$  can satisfy  $\mathcal{C}_{freq}$ . Additionally, many well-known related properties can be used to shorten the transactions in the dataset. These properties boil down to the following: if an item in a transaction is not included in  $k$  frequent itemsets of length  $k$  supported by that transaction, then it will not be included in any frequent itemset of length  $>k$ . These items can be discarded after the  $k$ -iteration of a level-wise algorithm. Eventually the whole transaction can be pruned from the dataset.

**Definition 4.2** (*Succinct constraint*). An itemset  $\mathcal{I}_s \subseteq \mathcal{I}$  is a *succinct set*, if it can be expressed as  $\sigma_p(\mathcal{I})$  for some selection predicate  $p$ , where  $\sigma$  is the selection operator.  $SP \subseteq 2^{\mathcal{I}}$  is a *succinct power-set*, if there is a fixed number of succinct sets  $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_k \subseteq \mathcal{I}$ , such that  $SP$  can be expressed in terms of the strict power-sets of  $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_k$  using union and minus. Finally, a constraint  $\mathcal{C}_s$  is *succinct* provided that the set of itemsets satisfying the constraint is a succinct power-set.

Informally, 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  $X$ . An example of succinct constraints, is  $\mathcal{C}_s(X) \equiv "X \text{ contains items of type food}"$ .

This class of constraints was introduced with the CAP algorithm [2]. In general it is possible to understand if no supersets of a given itemset can satisfy the constraint, and remove those supersets from the search space. However, since supersets of invalid itemsets cannot be pruned, this strategy does not provide an effective reduction as anti-monotone ones. On the other hand, constraints that are both anti-monotone and succinct can be pushed *once and for all* at pre-processing time, by removing invalid items from the dataset.

**Definition 4.3** (*Monotone constraint*). Given an itemset  $X$ , a constraint  $\mathcal{C}_M$  is *monotone* if:  $\forall Y \supseteq X : \mathcal{C}_M(X) \Rightarrow \mathcal{C}_M(Y)$ .

Handling monotone constraints in conjunction with the minimum frequency constraint (that it is always present in any query) and with other anti-monotone constraints, turns out to be very difficult due to their symmetrical behavior: valid patterns w.r.t.  $\mathcal{C}_{freq}$  are in the lower part of the itemsets lattice, while itemsets that satisfy  $\mathcal{C}_M$  are in the upper part. A typical bottom-up as well as a top-down visit introduced by DUAL-MINER [30] algorithm will inevitably traverse many invalid itemsets w.r.t. one of the two classes of constraints.

Also consider that only little work can be saved due to the  $\mathcal{C}_{AM}-\mathcal{C}_M$  trade-off: if an itemset does not satisfy the monotone constraint, we could avoid the expensive frequency count for this itemset. On the other hand, if the same itemset was actually infrequent, it could have been pruned with its supersets from the search space, thus saving many more frequency counts.

Among the many algorithms introduced for mining frequent patterns under a conjunction of monotone and anti-monotone constraints, [30,33–35], a completely orthogonal approach was introduced in [16,26]. In fact, a

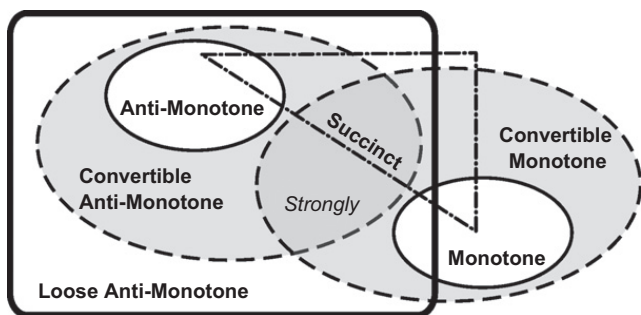


Fig. 10. Characterization of classes of constraints.

transaction that does not satisfy the monotone constraint  $\mathcal{C}_M$ , can be removed since none of its subsets will satisfy  $\mathcal{C}_M$  either, and therefore the transaction cannot support any valid itemsets. This data reduction, in turns reduces the support of other frequent and invalid itemsets thus reducing the search space and improving anti-monotone pruning techniques. This virtuous circle, and in general the synergy between data reduction and search space reduction, will inspire our algorithmic framework.

Another class of constraints was introduced in [31].

**Definition 4.4 (Convertible constraints).** A constraint  $\mathcal{C}_{CAM}$  ( $\mathcal{C}_{CM}$ ) is convertible anti-monotone (monotone) provided there is an order  $\mathcal{R}$  on items such that whenever an itemset  $X$  satisfies  $\mathcal{C}_{CAM}$  (violates  $\mathcal{C}_{CM}$ ), so does any prefix of  $X$ . If a constraint is both convertible monotone and anti-monotone, then it is called *strongly convertible*.

In [31], two FP-GROWTH based algorithms are introduced. During the construction of the initial FP-tree, items are ordered according to  $\mathcal{R}$ , and then a slightly modified FP-GROWTH starts the mining. In presence of a  $\mathcal{C}_{CAM}$  constraint, the visit of the search space can stop whenever an invalid itemset is found. While, in presence of a  $\mathcal{C}_{CM}$  constraint, no pruning is performed, only monotone checks are saved for the supersets of valid itemsets. Unfortunately, the order  $\mathcal{R}$  may significantly affect the performances of the algorithm.

The same classes of constraints can also be exploited in a level-wise framework, as shown in [32]: if transactions are sorted according to the same order  $\mathcal{R}$ , some advanced pruning strategies may take place, since  $\mathcal{C}_{CAM}$  and  $\mathcal{C}_{CM}$  impose stronger requirements for an item of a given

Table 2

A summary of search space and data-reduction strategies for the different classes of constraints

Constraint	Algorithms	Search space	Data
Anti-Monotone $\mathcal{C}_{AM}$	Apriori [13]	Discard supersets of infrequent itemsets	Remove useless items in each transaction
Monotone $\mathcal{C}_M$	ExAnte [26]	Implicitly from the data reduction	Repeatedly remove invalid transactions and singletons that become infrequent
	ExAMiner [16] Dual Miner [30]	Implicitly from the data reduction Top Down visit	Remove invalid transactions –
Succinct anti-monotone $\mathcal{C}_{AMS}$	CAP [2]	Implicitly from the data reduction	Remove all itemsets that do not satisfy the constraint from the database
Succinct monotone $\mathcal{C}_{MS}$	CAP [2]	Remove invalid itemsets that are not subset of a valid one	–
	ExAMiner [16]	Implicitly from the data reduction	The same of a monotone constraint
Convertible $\mathcal{C}_{CAM}, \mathcal{C}_{CM}$	$FIC^d, FIC^h$ [31]	Bottom-up depth-first with item reordering	–
	ExAMiner Lam [18]	Implicitly from the data reduction	Remove useless items from transactions. Exploit loose anti-monotonicity
Loose anti-monotone $\mathcal{C}_{LAM}$	ExAMiner Lam [18]	Implicitly from the data reduction	Remove transactions that do not contain any valid itemset

**Table 3**  
Classification of commonly used constraints

Constraint	Anti-monotone	Monotone	Succinct	Convertible	$\mathcal{C}_{LAM}$
$min(S.A) \geq v$	Yes	No	Yes	Strongly	Yes
$min(S.A) \leq v$	No	Yes	Yes	Strongly	Yes
$max(S.A) \geq v$	No	Yes	Yes	Strongly	Yes
$max(S.A) \leq v$	Yes	No	Yes	Strongly	Yes
$count(S) \leq v$	Yes	No	Weakly	$\mathcal{A}$	Yes
$count(S) \geq v$	No	Yes	Weakly	$\mathcal{M}$	No
$sum(S.A) \leq v (\forall i \in S, i.A \geq 0)$	Yes	No	No	$\mathcal{A}$	Yes
$sum(S.A) \geq v (\forall i \in S, i.A \geq 0)$	No	Yes	No	$\mathcal{M}$	No
$range(S.A) \leq v$	Yes	No	No	Strongly	Yes
$range(S.A) \geq v$	No	Yes	No	Strongly	Yes
$avg(S.A) \leq v$	No	No	No	Strongly	Yes
$avg(S.A) \geq v$	No	No	No	Strongly	Yes
$median(S.A) \leq v$	No	No	No	Strongly	Yes
$median(S.A) \geq v$	No	No	No	Strongly	Yes
$var(S.A) \geq v$	No	No	No	No	Yes
$var(S.A) \leq v$	No	No	No	No	Yes
$std(S.A) \geq v$	No	No	No	No	Yes
$std(S.A) \leq v$	No	No	No	No	Yes
$var_{N-1}(S.A) \geq v$	No	No	No	No	Yes
$var_{N-1}(S.A) \leq v$	No	No	No	No	Yes
$md(S.A) \geq v$	No	No	No	No	Yes
$md(S.A) \leq v$	No	No	No	No	Yes

transaction to be useful in the subsequent iterations of a level-wise strategy.

Another class of constraint has been introduced in [18].

**Definition 4.5** (*Loose Anti-monotone constraint*). Given an itemset  $X$  with  $|X| > 2$ , a constraint is *loose anti-monotone* (denoted  $\mathcal{C}_{LAM}$ ) if:  $\mathcal{C}_{LAM}(X) \Rightarrow \exists i \in X : \mathcal{C}_{LAM}(X \setminus \{i\})$ .

Loose anti-monotone constraints are a proper superset of convertible constraints, also related to many interesting statistical functions. Every constraint in this class can be exploited in a level-wise computation by means of data reduction. In fact, if at level  $k > 1$  a transaction is not a superset of any valid itemset of size  $k$ , then it will not contain any valid larger itemset.

A characterization of the various classes of constraints is given in Fig. 10, while in Table 3 we report the classification of some commonly used constraints.

#### 4.3. A new framework for constrained pattern mining

In Table 2 we summarize the aforementioned classes of constraints and some representative algorithms that exploit their properties. For each constraints, there exist both data reduction and search space reduction algorithm. Recall that our system should be able to answer conjunctive queries possibly containing many constraints belonging to different classes.

Unfortunately, most of the search space reduction based strategy cannot be exploited at the same time. For instance, Dual Miner top-down search can hardly be adapted to traditional bottom-up strategies, and reordering items as  $FIC^{\mathcal{A}}$  and  $FIC^{\mathcal{M}}$  [31] algorithms may not be possible in presence of multiple convertible constraints requiring different orderings.

On the other hand, all the other data-reduction strategies are orthogonal, i.e., they can be applied at the same time independently without producing any interference. Conversely, they will help each other in reducing the size of the mining problem. In fact, the data reduction operated by one constraint, i.e., the shortening of transaction or even their removal, may introduce new pruning chances for other constraints regardless whether they operate on the search space, as we have shown with the  $\mathcal{C}_{AM}$ - $\mathcal{C}_M$  trade-off, or the operate on the data.

Supported by these consideration, we designed the ME of CONQUEST as a level-wise bottom-up data-reduction boosted algorithm for constrained pattern mining, that uses DCI as its computational skeleton. After each iteration, each transaction is subject to every data-reduction opportunity given by the constraints specified by the mining query. Therefore, after each iteration, the data on which the support counting step is exploited becomes smaller and smaller.



**Algorithm 1.** CONQUEST mining engine.

```

Input:  $\mathcal{D}, \delta, \mathcal{C}$   $\triangleright$  where  $\mathcal{C} = \mathcal{C}_{AM} \cup \mathcal{C}_M \cup \mathcal{C}_{MS} \cup \mathcal{C}_{AMS} \cup \mathcal{C}_{CAM} \cup \mathcal{C}_{LAM}$ 
Output:  $\{X \in 2^{\mathcal{I}} \mid \text{supp}_{\mathcal{D}}(X) \geq \delta \wedge \mathcal{C}(X)\}$ 

1:  $k \leftarrow 1$ 
2:  $C_k \leftarrow \{ii \in \mathcal{I} \wedge \mathcal{C}_{AMS}(ii) \wedge \mathcal{C}_{AM}(ii)\}$   $\triangleright$  Candidate itemsets
3:  $L_k \leftarrow \emptyset$   $\triangleright$  Frequent itemsets
4:  $R_k \leftarrow \emptyset$   $\triangleright$  Valid itemsets
5:  $\mathcal{D}_k \leftarrow \pi_{C_k}(\mathcal{D})$   $\triangleright$  Ex-Ante Loop

6: while pruning is possible do
7:    $(C_k, L_k, \mathcal{D}_k) \leftarrow \text{Ex-Ante}(\mathcal{D}_k, \delta, C_k, \mathcal{C}_M)$ 
8: end while
9:  $R_k \leftarrow \{X \in L_k \mid \mathcal{C}(X)\}$   $\triangleright$  Horizontal Loop

10: while  $\mathcal{D}_k$  does not fit in main memory AND  $|L_k| > k$  do
11:    $C_{k+1} \leftarrow \text{gen\_candidates}(L_k, \mathcal{C}_{AM}, \mathcal{C}_{MS}, \mathcal{C}_{CAM}, \mathcal{C}_{LAM})$ 
12:    $(L_{k+1}, \mathcal{D}_{k+1}) \leftarrow \text{horizontal\_count}(\mathcal{D}_k, C_{k+1}, \delta, \mathcal{C}_M, \mathcal{C}_{CAM}, \mathcal{C}_{LAM})$ 
13:    $R_{k+1} \leftarrow \{X \in L_{k+1} \mid \mathcal{C}(X)\}$ 
14:    $k \leftarrow k + 1$ 
15: end while  $\triangleright$  Vertical Loop

16:  $\mathcal{V}\mathcal{D} = \text{build\_vertical}(\mathcal{D}_k)$ 
17: while  $|L_k| > k$  do
18:    $C_{k+1} \leftarrow \text{gen\_candidates}(L_k, \mathcal{C}_{AM}, \mathcal{C}_{MS}, \mathcal{C}_{CAM}, \mathcal{C}_{LAM})$ 
19:    $L_{k+1} \leftarrow \text{vertical\_count}(\mathcal{V}\mathcal{D}, C_{k+1}, \delta)$ 
20:    $R_{k+1} \leftarrow \{X \in L_{k+1} \mid \mathcal{C}(X)\}$ 
21:    $k \leftarrow k + 1$ 
22: end while

23: return  $\bigcup R_i$   $\triangleright$  The collection of interesting itemsets

```

The simplified pseudo-code of CONQUEST ME is given in Alg. 1. Notice how this framework fits and enhances each single algorithmic contribution participating to the ME.

During the first stage (lines 6–9) we exploit a data reduction loop inspired by *ExAnte*: transactions that do not satisfy monotone constraints are removed from the dataset, thus lowering the support of singletons. If any item turns out to be infrequent it can be removed thus providing new chances to prune useless transactions by re-checking monotone constraints. This loop can be repeated until no more pruning is possible. At the end of the process, the algorithm produces the set of frequent singletons and a possibly reduced dataset.

Then the *horizontal loop* starts (lines 10–15). The `horizontal_count` procedure reads the disk resident dataset and calculates the support of the candidate itemsets  $C_{k+1}$  in order to produce the set of frequent itemsets  $L_{k+1}$ . Also, data-reduction properties are exploited in order to remove useless items and transactions, thus producing a new reduced dataset  $\mathcal{D}_{k+1}$  to use during the subsequent iteration.

At every iteration the algorithm will work on a smaller and smaller amount of data. When the dataset is small enough, it will be entirely stored in the main memory by using a vertical bitmap. The bitmap format does not allow to prune the dataset further during the *vertical loop* (lines 16–22). Nonetheless, the compact memory resident image of the dataset will enhance the temporal and spatial locality of DCI, which can complete the mining with its efficient use of CPU cache without the need of additional data reduction.

**Table 4**

Characteristics of the datasets used in our experiments

Dataset	Size	# Items	# Transactions	Avg. Tr. length
<i>retail</i>	4 MB	16,469	88,162	10.3
<i>accidents</i>	34 MB	468	340,183	33.8
<i>webdocs</i>	1.2 GB	5,267,656	1,692,082	177
<i>census</i>	521 MB	396	2,458,295	68

Lastly, as discussed later in Section 5.4, this framework can be easily extended to handle future user-defined constraints by exploiting their data-reduction properties in a similar way.

#### 4.4. Performance evaluation

In our experiments we used four different datasets, which are well known in the pattern mining community. In Table 4 we report their main characteristics.

The first dataset is *retail*, a collection of market baskets coming from the Belgium. The second one is *accidents*, resulting from a Belgian study on car accidents. Dataset *webdocs*, the largest one (about 1.2 GB), comes from a large collection of web documents. The last one, *census*, contains census information about a very small portion of U.S. population. Notice that all of these datasets are not synthetic.

These four datasets are very different one from each other. The first two, respectively a sparse one and a dense one, are quite small. While the latter two are definitively much bigger, and they will show a more complex nature. In evaluating a pattern mining system it is very important to test different datasets because each of them raises different difficulties for a pattern mining algorithm. Sparse and small datasets are usually very easy to mine, since the overall number of extracted frequent itemsets is not large. On the other hand, dense datasets produce a much larger number of frequent itemsets, and since they are usually characterized by having long transactions, it is usually difficult to effectively apply frequency-based or other pruning techniques to them. Additionally, the two large datasets force the algorithm to explore a huge number of candidates, thus increasing its memory requirements. Since these datasets are pure transactional datasets, items have no associated attribute. We thus generated a synthetic attribute for each distinct item by using random values drawn from a uniform distribution in the interval [0, 1000]. This is not a limitation, since the aim of our performance experiments was to assess the pruning power of our data-reduction techniques, to show the absolute effectiveness of the system compared with other specialized constrained frequent pattern mining algorithm, and, finally, to show the goodness coming from the synergy of a powerful data-reduction approach with a fast mining algorithm.

##### 4.4.1. Dataset pruning power

We tested CONQUEST pruning power on the first three datasets by using three different constraints. We forced

the algorithm to perform the horizontal setup only, i.e., the only one where data-reduction is performed. During each iteration the algorithm produces a reduced dataset, whose size is plotted in order to evaluate the effectiveness of the different pruning strategies.

The constraints we tested were  $sum(S.A) \geq v$  (herein-after denoted *sumgeq* for simplicity) that is monotone,  $avg(S.A) \geq v$  (*avggeq*) that is convertible, and  $var(S.A) \leq v$  (*varleq*) that is loose-antimonotone.

Fig. 11 shows the results of these tests, where we varied the threshold values of each constraint. For each set of tests we also report the baseline case, where all the frequent itemsets satisfy the constraint (e.g.,  $sum(X.A) \geq 0$ ). In this way we can compare our data-reduction techniques with the sole anti-monotone frequency based pruning.

With the *retail* dataset, the *avggeq* and *varleq* were effective only for very selective thresholds, and only starting from the fourth iteration. However, they will soon prune a large number of transactions almost emptying the dataset in a few successive iterations. The *sumgeq* constraint requires more iterations to reduce significantly the dataset, but it immediately removes a

large part of transactions starting from the very first iterations. Moreover, its behavior is more consistent with respect to the selectivity of the constraint.

Since *accidents* is a dense dataset, the pruning here is more difficult. In fact, the frequency-based pruning is not able to remove a valuable number of transactions during the first ten iterations. Even the pruning given by the *sumgeq* constraint is poor for every thresholds used, because it works at a transaction granularity: it removes the whole transaction or does nothing. Conversely, the other two constraints perform much better. The advanced pruning of *avggeq* works at item granularity, and allows to remove many items from each transaction, in turn increasing chances for the removal of the whole transaction. The last constraint *varleq*, behaves pretty well because of its quasi-antimonotone properties: computation may actually end after a few iterations.

In the last dataset, the frequency-based anti-monotone pruning works better than with the *accidents* dataset, but is not effective as with the *retail* one. This means that, while being generally sparse, there are many dense portion of the dataset. As a result the monotone pruning

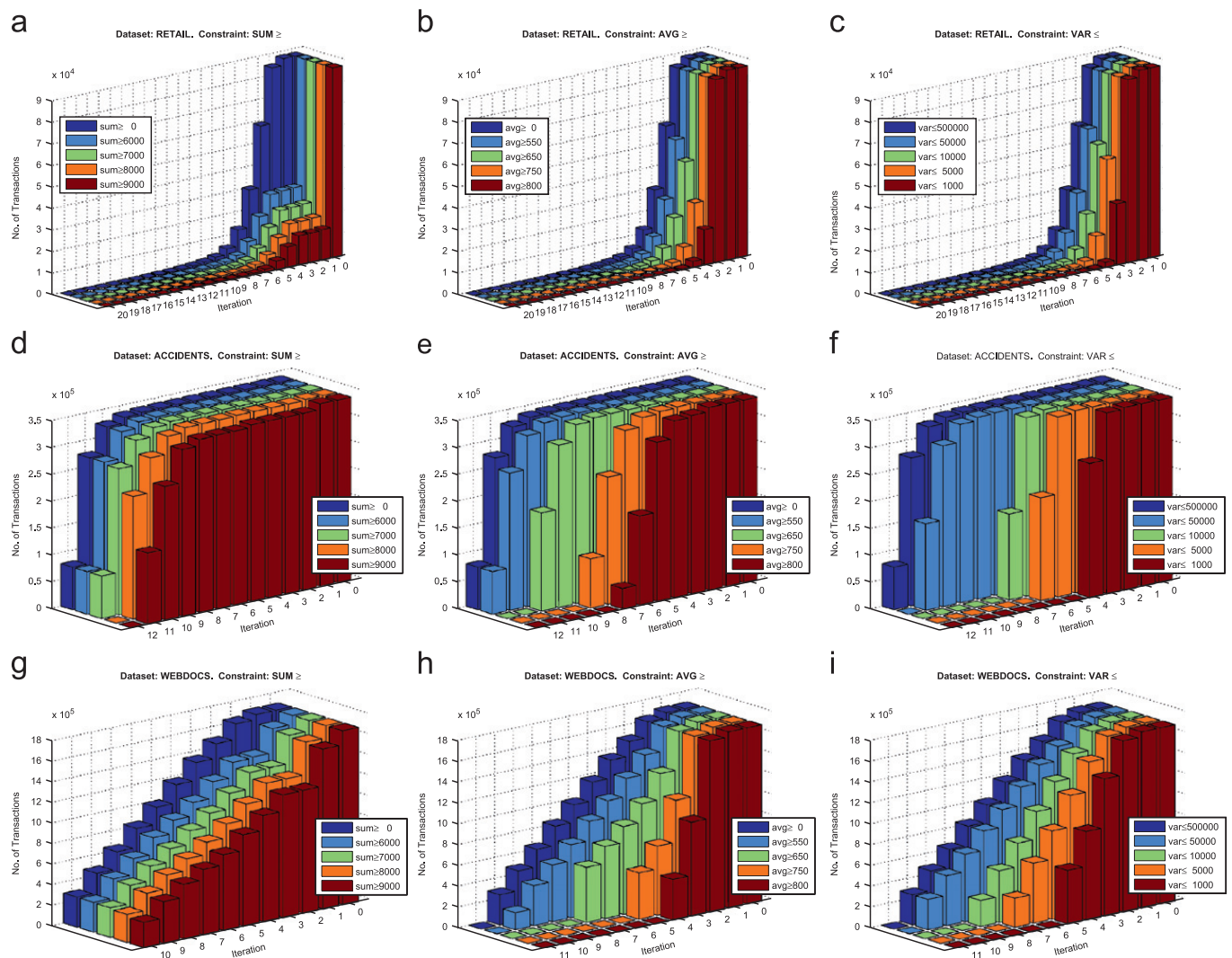


Fig. 11. Effectiveness of the various pruning strategies. We used the following minimum support thresholds:  $\delta = 3$  for retail,  $\delta = 100000$  for accidents and  $\delta = 180000$  for webdocs dataset.

helps in mining the *sumgeq* constraint by removing a small, but significant number of transactions, just from the first iterations. The other two constraints work much better and consistently for every threshold used, even if they are not able to perform any pruning in first two iterations.

As expected, monotone pruning is more effective when the dataset is sufficiently sparse, i.e., when a lot of short transactions can be easily pruned. On the other hand it allows to remove transactions from the very beginning of the algorithm. The other two pruning strategies need instead a few iterations to make their contribution evident. However, after such initial stage, they can immediately remove a large portion of transactions, often allowing the algorithm to end some iterations in advance due to the emptied dataset at that point.

#### 4.4.2. Comparison with other algorithms

In Fig. 12 we compare the performance of CONQUEST with other two state-of-the-art algorithms  $FIC-\delta$  and  $FIC-\mu$ . We used the same three datasets discussed above, and the convertible constraint *avggeq*. In this test we wanted to stress all the algorithms, and for this reason we used highly selective constraint thresholds. In fact, the number of valid patterns is less than 100 in every experiment, having size up to seven items, while the number of candidates, i.e., the size of the search space, is in the order of hundreds of thousand itemsets.

As expected,  $FIC-\mu$  was never able to compete with  $FIC-\delta$ , since, as we discussed above, it does not perform an actual pruning of the search space. Note the bars of Fig. 12 where an exclamation mark appears: in these cases we artificially interrupted  $FIC-\delta$  or  $FIC-\mu$ .

In all the tests with all the three datasets CONQUEST was the clear winner. Even if it was designed to handle several classes of constraints at the same time, still it resulted to be much faster than specialized algorithms such as  $FIC-\delta$ . Great part of this effectiveness has to be credited to the mining core coming from DCI.

Finally, in mining the largest dataset among the three (i.e., *webdocs*) CONQUEST was the only one able to run to completion. Both the other two algorithms terminated because of segmentation fault errors.

#### 4.4.3. Synergy of the two approaches

In the previous two sections we showed the effectiveness of the data-reduction process, and the high perfor-

mance of the pattern mining core. Notice that dataset pruning is possible only during the horizontal iterations, while it is not applicable during vertical iterations.

In this section we will show the synergy of the combination of these two approaches. A few first horizontal iterations will speed up the vertical phase of the algorithm.

For this test we used the *census* dataset and the *avggeq* constraint. Even if this dataset is smaller than *webdocs*, it has a larger number of transactions. Additionally, if we consider that the average transaction length (68) is large compared to the total number of items (396), it is clear that this dataset is very dense. In other words, we chose a large dataset so that to have a large number of possible frequent itemsets. Moreover, since the dataset is dense, we can expect that this feature makes the pruning process very complex.

We run CONQUEST by varying the number of initial horizontal iterations from 2 to 6, and the obtained results are shown in Fig. 13. The rationale was to find out the (possibly small) number of horizontal iterations that improves the overall performance of the mining process.

Consider the constraint *avggeq* with threshold 700: Fig. 13(a) shows that the running time of the algorithm increases dramatically when increasing the number of horizontal iterations. The reason of this behavior is shown in Fig. 13(b), which reports the number of candidate itemsets explored by the algorithm for different number of horizontal iterations performed. For *avggeq* with threshold 700, this number does not change significantly for the various iterations. In this case, the horizontal iterations are not able to introduce a significant pruning, and, being slower than the vertical ones, they only degrade the performance.

When increasing the selectivity of the constraint to *avggeq* with threshold 750, the running time decreases significantly due to the reduced size of the dataset, that also reduces by two or more orders of magnitude the number of candidate itemsets. It is clear that the best choice is to have three horizontal iterations and to exploit the vertical loop from the fourth iteration, since the cost of additional horizontal pass is not worth the dataset reduction provided.

The conclusion is that the horizontal technique is not efficient enough to bear the mining effort, but it can highly reduce the cost of the following phases of the mining process. However, the combination of the two approaches results to be definitely successful.

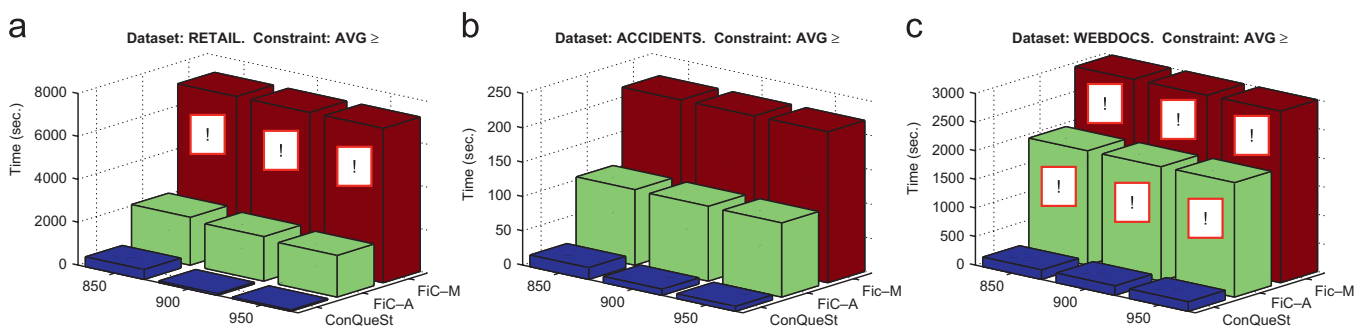


Fig. 12. CONQUEST versus other algorithms. We used the following minimum support thresholds:  $\delta = 3$  for *retail*,  $\delta = 100000$  for *accidents* and  $\delta = 180000$  for *webdocs* dataset. The exclamation mark means that the algorithm aborted before completion.

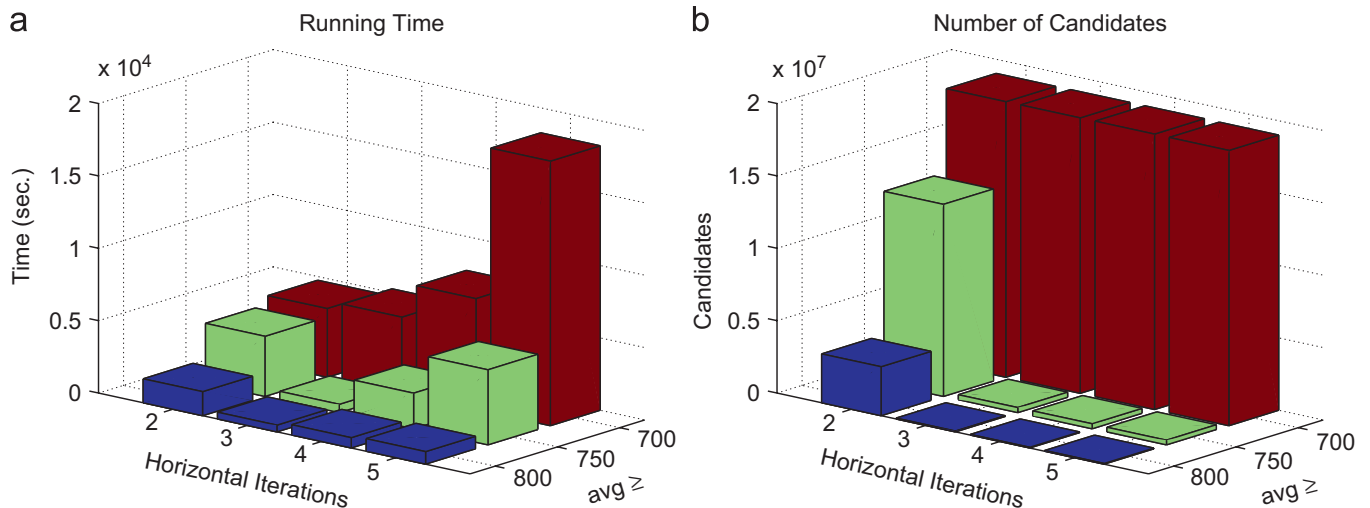


Fig. 13. Effectiveness of the combined horizontal and vertical approach on census dataset with a minimum support threshold  $\delta = 150000$ .

#### 4.5. Mining with soft constraints

In the following we describe how SPQL queries with soft constraints, introduced in Section 3.2, are effectively evaluated in CONQUEST.

We rely on the theoretical results of [14]. In that work it is shown that computing all the  $\lambda$ -interesting patterns can be done by solving a crisp problem where all the constraint instances with interestingness level lower than  $\lambda$  correspond to *false*, and all the instances with interestingness level greater or equal to  $\lambda$  correspond to *true*. In fact, if a pattern does not satisfy such conjunction of crisp constraints, it will not be interesting w.r.t. the soft constraints [14].

Using this theoretical result, and some simple arithmetic we can:

- (1) transform each soft constraint in a corresponding crisp constraint,
- (2) push the crisp constraint in the mining computation to prune uninteresting patterns,
- (3) and when needed, post-process the solution of the crisp problem, to remove uninteresting patterns from it.

**Definition 4.6** (Crisp translation of a soft constraint). Given a soft constraint  $\mathcal{C} \equiv \text{Agg}(Att) \theta t(\alpha)$  (where *Agg* represent the aggregate on which the constraint is defined, *Att* an attribute,  $\theta = \{\geq, \leq\}$ , and  $\alpha$  is the softness parameter), and a minimum interest threshold  $\lambda$ , we define the crisp translation of  $\mathcal{C}$  w.r.t.  $\lambda$  as

$$\mathcal{C}_{crisp}^\lambda \equiv \begin{cases} \text{Agg}(Att) \geq t - \alpha t + 2\lambda\alpha t, & \text{if } \theta = \geq, \\ \text{Agg}(Att) \leq t + \alpha t - 2\lambda\alpha t, & \text{if } \theta = \leq. \end{cases}$$

**Example 8.** The crisp translation of the soft constraint  $\langle \text{sum}, \text{price}, \geq, 20, 0.5 \rangle$  is  $\text{sum}(X.\text{price}) \geq 26$  for  $\lambda = 0.8$ , while it is  $\text{sum}(X.\text{price}) \geq 18$  for  $\lambda = 0.4$ .

This is exactly the translation that CONQUEST performs to transform a query based on soft constraints in a query based on crisp constraints, which can hence be evaluated by the ME described in Section 4.3.

In the fuzzy case, since the combination operator is the *min* which is idempotent, we got the nice property [14] that the solution set resulting from the translated crisp query, will correspond exactly to the solution set of the original soft-constrained query. In fact, all patterns satisfying the conjunction of crisp constraints would for sure reach an interestingness level larger than  $\lambda$ .

Instead, in the probabilistic case, since we must take the arithmetic times of a set of values between 0 and 1, we need some post-processing to select, among the solution set resulting from the translated crisp query, whose pattern really have a total interestingness value larger than  $\lambda$ .

Both the translation from soft to crisp constraints, and the post-processing needed in the probabilistic case, are handled by CONQUEST's QI module (see Section 5.3).

So far we have described how CONQUEST handles  $\lambda$ -interesting queries. It remains to explain how *top-k* are handled.

The main difficult to solve *top-k* queries is that we can know the number of solutions only after the evaluation of a query. Therefore, given  $k$ , the simple idea is to repeatedly run  $\lambda$ -interesting queries with different  $\lambda$  thresholds: we start from extremely selective  $\lambda$  (fast mining) decreasing in selectivity, until we do not extract a solution set which is large enough (more than  $k$ ).

Considering for instance the fuzzy semiring, where the best semiring value is 1: we could start by performing a 0.95-interesting query, and if the query results in a solution set of cardinality larger than  $k$ , then we sort the solution according to their interestingness and return the best  $k$ , otherwise we slowly decrease the threshold, for instance  $\lambda = 0.9$ , and so on. Notice that it is important to start from a very high threshold in order to perform fast mining extractions with small solution sets, and only if needed decrease the threshold to get more solutions at the cost of longer

computations. The user can use the CONQUEST's general options to set the *step* parameter, which determines how  $\lambda$  must be progressively decreased in a *top-k* query evaluation.

## 5. The CONQUEST system

This section illustrates the functionalities of CONQUEST, and the main components of its architecture. We first focus on its user interface, which permits the user not only to access powerful tools for analyzing and visualizing the input data, but also to express queries, involving data selection and mining, in both textual and graphical way. Using the same interface, also the extracted patterns can be visualized and manipulated. The others two main blocks of the overall architectures of CONQUEST roughly corresponds to the QI and the ME. The interpreter has to prepare the data for the mining task by also communicating with the underlying database, to invoke the ME, and, finally, to store the returned data. The ME, which efficiently solves the constrained pattern mining problem, allows the user-provided constraints to be pushed in the mining algorithm.

### 5.1. Architecture overview

The modular architecture of CONQUEST reflects the needs of realizing a complete KDD process, even if limited to the mining of frequent constrained patterns. The main building blocks of the system are the following:

- The GUI is responsible for every interaction between the user and the other modules. Thanks to the GUI, the user is able to navigate any DBMS, to define a mining task and to evaluate its results.
- The QI has the main task of transforming relational tables into a transactional dataset, defined in terms of a mining view of the database. Moreover, since any interaction with the underlying DBMS is caused by the user-provided SPQL queries that are processed by the QI, it is also in charge of other optimization tasks, such as caching, to improve the overall system usability.
- The core of the pattern extraction process is performed by the ME, which encloses the state of the art of high performance and constrain-based pattern mining algorithms in a novel unifying framework.

Fig. 14 depicts the main architectural blocks of the architecture of CONQUEST. While the GUI and the QI were developed in Java, the ME was developed in C++ for efficiency reasons, and wrapped in Java to allow the rest of the software package to invoke its methods. Note that the three main blocks interacts with the DBMS through a JDBC abstraction layer. Using the JDBC interface, the current version of the CONQUEST prototype can exchange data with PostgreSQL, Microsoft Access, Oracle, MySQL and SQL Server. Additional DBMSs can be interfaced by adding the suitable JDBC plug-in.

This subdivision in three distinct blocks reflects the main functionalities of our tool. It provides a better maintainability and extensibility of the software, and even a smart deployment when required. In fact, as a proof of the good separation between independent tasks, the various modules could be disjoint software packages, running on different machines and cooperating through Java RMI. For instance, the GUI may run on the user machine, while the QI may be located in a different site where a fast access to the database is provided and finally the ME may be run on a high performance cluster serving many users with many different tasks. Finally, note that the JDBC layer allows us to ignore the physical location of the DBMS server.

### 5.2. GUI

The GUI helps the user in specifying and tuning every phase of the KDD process, and in evaluating the outcome of the mining process. It is simple and user-friendly. Moreover, many high level information and statistics about data are provided.

Even if the GUI includes a *text area* in which the user can directly specify its SPQL query, it is worth noting that the same task can be carried out just by means of mouse-clicks. All the actions that are specified in the graphical part also affect the textual SPQL query, and vice versa.

In the following, we discuss in depth the features of the CONQUEST GUI, corresponding to each fundamental phase of the KDD process.

#### 5.2.1. Data selection and transformation

As soon as the user connects to the database, a set of information and statistics are collected and presented in many ways. The idea is to provide the user a simple and

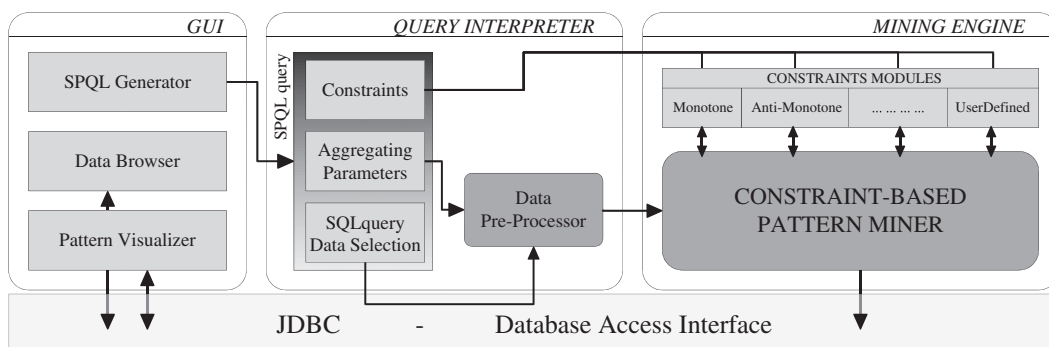


Fig. 14. CONQUEST architecture.

high level mean for navigating the database, select the data source, possibly discretize some numerical fields and define the mining view.

- **Navigating the structure of the database:** The *desk area* (Fig. 15 [1]) occupies the most part of our GUI. Here the tables that are present in the database are shown as a graph structure, where each vertex represents a table, and each edge corresponds to a logical link between a foreign key and a primary key. For each vertex of the graph, the user may choose either to see the fields of the corresponding table, or to visualize it as a simple node. Finally, a *tables list* (Fig. 15 [2]) helps the user to select, hide or un-hide any of the graph vertices. In conclusion, this gives the user a high level view of the database, allowing her/him to grasp all the database relations and connections at a glance, and to focus on the portion of data of interest.
- **Table-field information and statistics:** The currently selected table may be visualized in a *table visualization pane* (Fig. 15 [5]). Since aggregate information is very useful for the analyst in order to define his SPQL query, CONQUEST shows the data type and statistics associated with each selected field (e.g., number of distinct values, average, minimum and maximum) in the *table information pane* (Fig. 15 [3]). In addition, in order to help the analyst to prepare the transactional mining view, and decide the constraints of an SPQL query, CONQUEST visualizes a bar or a pie chart (Fig. 15 [4]) showing the distribution of the values of the selected field. This is important to decide, for example, the discretization of a numeric field.
- **Definition of the mining view:** The user may define the mining view by clicking on the table fields that must play the role of transaction identifiers, items or

properties (i.e., the triple  $T, I, U$  of Definition 2.4) directly on the *desk area*. These fields will be highlighted with different colors in the *desk area* and also reported in the *mining view definition pane* (Fig. 15 [6]). Note that a click-based data selection may implicitly require relational joins, e.g., transaction identifier and items belong to different tables. All these data selection actions and implicit joins are automatically inserted in the first part of the SPQL query, which appears in the *SPQL Query pane* (Fig. 15 [5]).

- **Discretization:** In CONQUEST discretization can be handled at query language as described in Section 3.3, or directly by interacting with an adhoc window of the GUI (Fig. 16). This allows not only to use the common discretization methods, like *equal width* and *equal depth*, but also the discretization task can be done graphically, by clicking in a pane and setting the partitions' boundaries manually.
- **Property conflicts:** Also *property conflicts* (defined in Section 2.2) can be handled not only by the language (as shown in Section 3.1) but also interacting directly with the GUI *desk area*. When the functional dependency between an item and one of its properties (i.e.,  $I \rightarrow P$ ) does not hold, a warning icon appears close to the conflicting property. The user can click with the mouse right button on the attribute, and select one of the resolution methods in a scroll-down window.

- **Mining view preview:** Before executing the SPQL mining query, a preview of the mining view, according to its current definition, is provided in the *mining view pane* (Fig. 15 [5]). This is useful to confirm to the user that the data preparation defined in the query is the one desired, thus avoiding a useless mining.

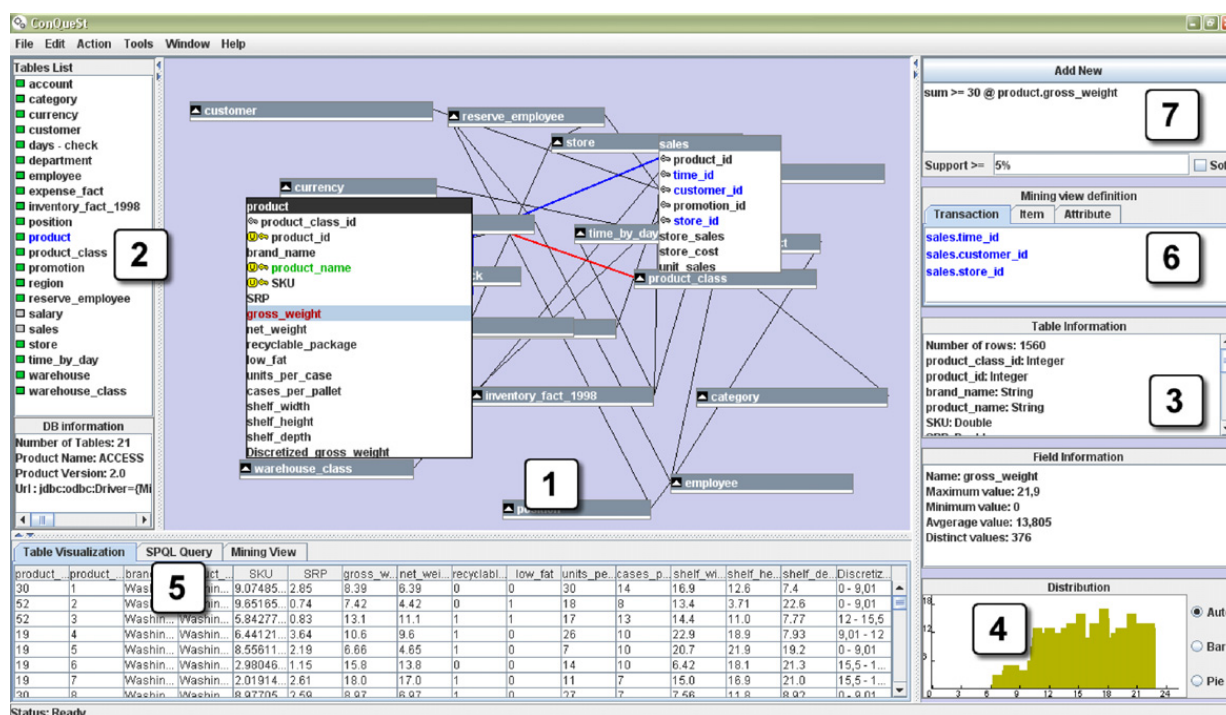


Fig. 15. The GUI main window. [1] Desk area; [2] table list; [3] table information pane; [4] distribution graph; [5] table visualization pane, SPQL query and mining view pane; [6] mining view definition pane; [7] constraints pane.

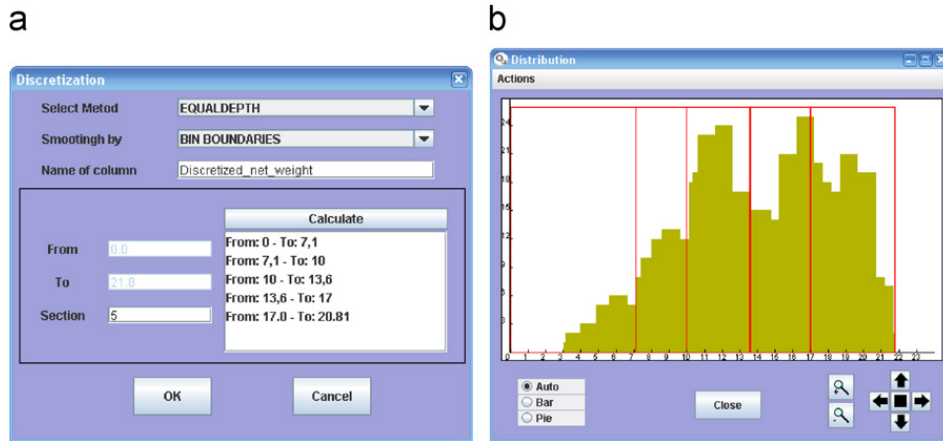


Fig. 16. (a) The discretization window (b) the distribution graph show the partition selected.

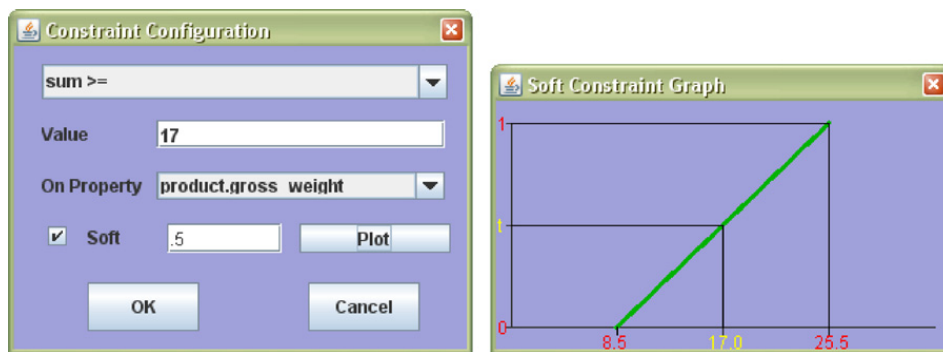


Fig. 17. The constraints definition window and the plot of a soft constraint.

- *Advanced SPQL query definition*: At any moment, the user can edit by hand the query in the SPQL query pane. Any modification of the query will be reflected in the rest of the GUI, e.g., by updating the mining view pane. The possibility to edit directly the query, rather than using the GUI, does not provide any additional expressive power from a mining point of view. Anyway, since part of an SPQL query is pure SQL, we can allow the user to exploit more complex SQL queries and additional constraints that are not part of the mining task, but rather they are part of the data preparation phase. Moreover, any SQL query can be submitted in place of an SPQL query, providing additional control to the analyst.

### 5.2.2. Pattern mining

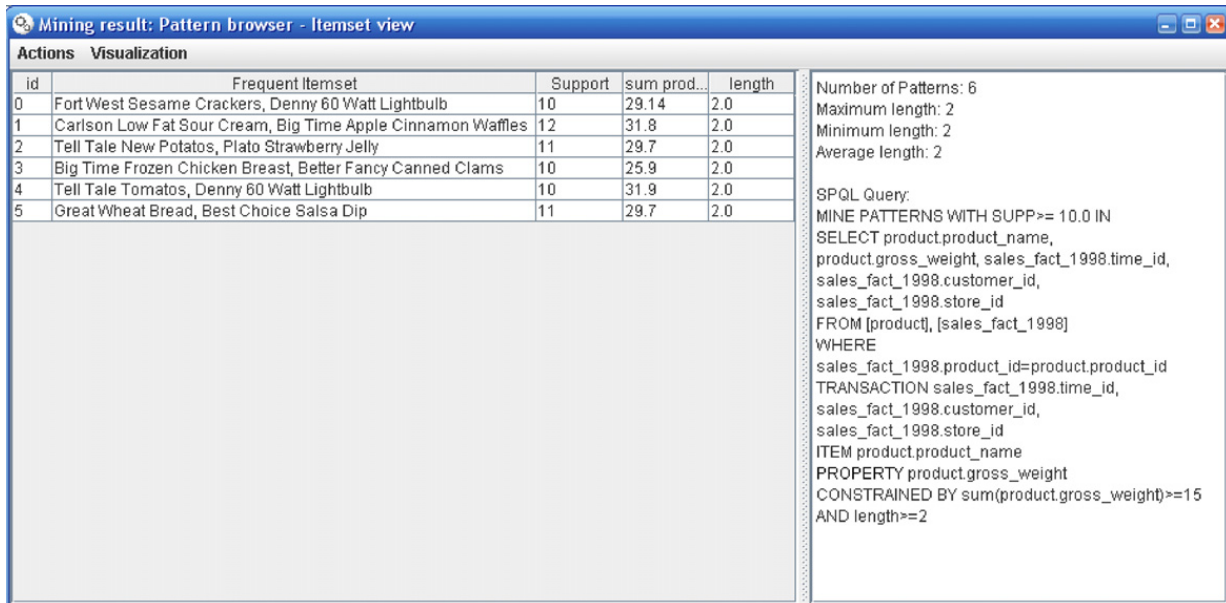
After the selection of the data source, and the needed data transformation and preparation, the actual mining process can start once that the suitable constraints have been defined.

- *Minimum support constraint and frequency constraints*: Constraints may be set by either left-clicking on the desk area or using the constraints pane (Fig. 15 [7]), or also writing them into the SPQL query. Selecting the add new function in the constraints pane, the system shows a window (Fig. 17) where the user can select one of the item attributes  $i \in I$ , the desired constraint and thresh-

old. Additionally, the user can choose the soft option, and, in this case, the alpha value must be specified. In the constraints pane, the user must also provide a frequency value to define the classical minimum support constraint. In case the user specified a query with soft constraints, in this same pane he can choose between  $\lambda$ -interesting and top-k query and he can provide the desired values for  $\lambda$  or k.

### 5.2.3. Interpretation and evaluation

- *Pattern browser*: The result of a mining query is shown in a specialized interface (Fig. 18), which provides additional functionalities to navigate the set of patterns, such as various kinds of pattern sorting. The system also shows the SPQL query that generated the patterns, and it also allows the user to tune the query parameters (e.g., the constraints' thresholds) according to his needs, thus focussing on the interesting patterns. Such tuning could require to re-run the query from scratch (if constraints are relaxed), or simply to filter the previous results (if constraints are tightened): in both cases the pattern browser is updated on-the-fly without the need to reformulate a new query. Finally, two actions can be performed in the pattern browser: (i) materialize the set of extracted patterns as relations into the underlying database, or (ii) extract association rules from such patterns.



id	Frequent Itemset	Support	sum prod.	length
0	FortWest Sesame Crackers, Denny 60 Watt Lightbulb	10	29.14	2.0
1	Carlson Low Fat Sour Cream, Big Time Apple Cinnamon Waffles	12	31.8	2.0
2	Tell Tale New Potatos, Plato Strawberry Jelly	11	29.7	2.0
3	Big Time Frozen Chicken Breast, Better Fancy Canned Clams	10	25.9	2.0
4	Tell Tale Tomatos, Denny 60 Watt Lightbulb	10	31.9	2.0
5	Great Wheat Bread, Best Choice Salsa Dip	11	29.7	2.0

Number of Patterns: 6  
Maximum length: 2  
Minimum length: 2  
Average length: 2

SPQL Query:  
MINE PATTERNS WITH SUPP>= 10.0 IN  
SELECT product.product\_name,  
product.gross\_weight, sales\_fact\_1998.time\_id,  
sales\_fact\_1998.customer\_id,  
sales\_fact\_1998.store\_id  
FROM [product], [sales\_fact\_1998]  
WHERE  
sales\_fact\_1998.product\_id=product.product\_id  
TRANSACTION sales\_fact\_1998.time\_id,  
sales\_fact\_1998.customer\_id,  
sales\_fact\_1998.store\_id  
ITEM product.product\_name  
PROPERTY product.gross\_weight  
CONSTRAINED BY sum(product.gross\_weight)>=15  
AND length>=2

Fig. 18. The pattern browser window.

- **Rule browser:** By specifying the minimum confidence threshold from a menu of pattern browser, the user can extract association rules from the extracted patterns. The rules with their support and confidence are shown in a new window, and similar to frequent patterns, they can be materialized and permanently stored into the DBMS.

### 5.3. Query interpreter

Receiving a SPQL query from the GUI, the QI must perform an analysis of coherency on it. Having a well-formed SPQL query the QI is in charge to accomplish the data preparation phase: it retrieves the source data from the underlying relational database, it transforms it in transactional format according to the mining view definition, it generates the item properties possibly resolving properties conflicts, and finally it passes the mining view together with all constraint and mining parameters to the mining engine.

Since each query is very likely to be a slightly modified version of a previous one, we chose to implement a *cache* module within the QI. Each time a new query is executed, the result of the data transformation step is stored, then for every new incoming query that needs to access a mining view that was already prepared, the QI can avoid the transformation step and directly forward the query to the ME. Also the mining step can be handled directly by the cache reusing previously computed mining results. Obviously, past results can be re-used only if the new query acts on the same relations, performs the same transformation and poses more selective constraints of a past one. For example, suppose that a user submit two similar queries, where the latter differs from the former only for the value of the threshold specifying the frequency constraint. If this threshold is greater than the threshold of the former query, the latter query is answered

by only filtering out the invalid patterns from the results stored in the cache. In this case the QI will not invoke the ME at all, and immediately show the result of the mining process in the pattern browser window, thus greatly improving the responsiveness of the system.

The QI also handles discretization queries and even pure SQL queries.

### 5.4. ME architecture

The last module is the core of the pattern extraction process, where itemsets are actually mined from transactional datasets. The ME fulfills many requirements such as scalability and responsiveness, that have been discussed in Section 4.

The data-reduction optimizations exploited by CONQUE-Sr are nicely separated from the internal pattern miner, also thanks to its *modular* structure. In fact, each class of constraints is coded as a distinct C++ class that implements the corresponding data-reduction techniques, and the core miner invokes them whenever such techniques can be exploited. In other words, the ME is aware of the different classes of constraints and of their properties, but it does not know about the peculiarities of every single constraint.

This internal architecture allows the user to add his own constraints of interest, given that they belong to the aforementioned classes. In this case we refer to an expert user who can edit the source code of the ME. It is sufficient to implement the data-reduction functions corresponding to the properties of the user-defined constraint, and then the miner will take care of exploiting such properties whenever possible. In Fig. 19 we report an example of how constraints classes are implemented. Anti-monotone constraints are implemented as an abstract classes with three methods. Another class is created for the *sumleq* constraint. This implements the three



<pre>class CAntiMonotone : virtual public Constraint { public:     virtual Constraint* parseParameters(char*) = 0;      virtual void PreProcessTransaction(vector&lt;T&gt; &amp;t) = 0;      virtual bool testItemset(vector&lt;T&gt;&amp; X) = 0; };</pre>	<pre>class SUMLEQ : public CAntiMonotone { private:     double threshold; public:     virtual Constraint* parseParameters(char*) {         parse parameters and set threshold     };     virtual void PreProcessTransaction(vector&lt;T&gt; &amp;t) {         remove from t items having attribute value &gt;t     };     virtual bool testItemset(vector&lt;T&gt;&amp; X) {         return true iff the sum of items' attribute value is ≤t     }; };</pre>
---	--

Fig. 19. Inversion of control in CONQUEST mining engine.

methods of the antimonotone abstract class. The first method is invoked at the very beginning, where the constraint threshold has to be extracted from the ME arguments and parsed. The second method is called before the visit of the search space starts, with the goal of reducing the input transactions. In this case, items associated with a high attribute value can be pruned. The third method is called during the visit, in order to check whether an itemset satisfies the constraint or not.

In the design of the ME, we applied the principle of the well-known *inversion of control*. The control is not given to the user, but it is coded inside the ME framework. Only the framework knows when certain methods of certain classes of constraints has to be invoked. The implementation of such methods is left to the user. In our case, the implementation of the constraints data-reduction properties are left to the user, and they are exploited by the framework when needed.

## 6. Conclusions and future work

In this paper we described and motivated the design of CONQUEST, an exploratory pattern discovery system, aimed at assisting the analyst in iteratively and interactively extracting useful and interesting knowledge from data.

CONQUEST is, to the best of our knowledge, the first system exploiting extensively constraint-based pattern mining as a query optimization tool: it supports a very large (and extendible) set of different constraints, and bases the mining process on the effective exploitation of the particular data-reduction properties of each class of constraints used.

We have shown that our system represents a unique example in the framework of Mannila's inductive databases, where the raw data along with the patterns and the knowledge extracted from them coexist elegantly in a single relational database. By using CONQUEST the analyst can perform easily all the tasks involved in a typical *knowledge discovery process* based on constraint patterns mining: (i) source data selection, (ii) data preparation and pre-processing and (iii) pattern discovery and model building.

This is possible thanks to the accurate integration within the CONQUEST system of three key components: (1) a simple, yet powerful, query language for specifying formally data selection and the mining query; (2) a

friendly GUI for accessing the underlying DBMS, for data visualization, preparation and transformation, and for visual query formulation; and (3) an efficient and robust ME exploiting effectively all data-reduction possibilities in order to give to the user the capability of mining interactively also huge datasets.

Behind the design and development of CONQUEST there was a significant theoretical background work that conducted us to the definition of an original and general framework for constrained frequent patterns mining. Within this framework we introduced our definition of mining view, we accommodated soft and crisp constraints, we specified data transformation and discretization methods and we classified constraints on the basis of their properties. All the CONQUEST system was implemented within this framework. As a direct consequence the CONQUEST code itself is particularly modular, extendible, and modifiable: new constraints and data pre-processing methods can be easily added and cleanly integrated in the actual system.

Throughout the paper we tried to give to the reader some flavor of the usability and power of CONQUEST. We described several short examples of usage, and reported the results of performance tests conducted on publicly available large datasets. We invite the reader to visit the CONQUEST website,<sup>3</sup> where it is possible to download the latest version of the software. The site also contains the installation instructions, a user manual and many video tutorials.

Even though CONQUEST is already fruitfully usable on real-world problems, many directions must be explored in the next future: efficient incremental mining, advanced visualization techniques, more complex post-processing, building global models from the interesting patterns, mining patterns from complex data such as sequences and graphs. We are continuously developing new functionalities of CONQUEST.

## References

- [1] R. Srikant, Q. Vu, R. Agrawal, Mining association rules with item constraints, in: Proceedings of the 3rd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'97), 1997.

<sup>3</sup> <http://www-kdd.isti.cnr.it/~conquest/>

- [2] R.T. Ng, L.V.S. Lakshmanan, J. Han, A. Pang, Exploratory mining and pruning optimizations of constrained associations rules, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD'98), 1998.
- [3] J. Han, L.V.S. Lakshmanan, R.T. Ng, Constraint-based, multidimensional data mining, *Computer* 32 (8) (1999) 46–50.
- [4] R.J. Bayardo Jr. R. Agrawal, D. Gunopulos, Constraint-based rule mining in large, dense databases, in: Proceedings of the 15th International Conference on Data Engineering (ICDE'99), Sydney, Australia, 23–26 March 1999.
- [5] J.-F. Boulicaut, B. Jeudy, Constraint-based data mining, in: O. Maimon, L. Rokach (Eds.), *The Data Mining and Knowledge Discovery Handbook*, Springer, Berlin, 2005, pp. 399–416.
- [6] C. Ordonez, L. de Braal, C.A. Santana, Discovering interesting association rules in medical data, in: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'00), 2000.
- [7] C. Ordonez, E. Omiecinski, L. de Braal, C.A. Santana, N. Ezquerro, J.A. Taboada, D. Cooke, E. Krawczynska, E.V. Garcia, Mining constrained association rules to predict heart disease, in: Proceedings of the 1st IEEE International Conference on Data Mining (ICDM'01), 2001.
- [8] A. Lau, S. Ong, A. Mahidadia, A. Hoffmann, J. Westbrook, T. Zrimec, Mining patterns of dyspepsia symptoms across time points using constraint association rules, in: Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'03), 2003.
- [9] J. Besson, C. Robardet, J. Boulicaut, S. Rome, Constraint-based concept mining and its application to microarray data analysis, *Intelligent Data Anal. J.* 9(1) (2005) 59–82.
- [10] H. Mannila, H. Toivonen, Levelwise search and borders of theories in knowledge discovery, *Data Mining and Knowledge Discovery* 1 (3) (1997) 241–258.
- [11] F. Bonchi, F. Giannotti, C. Lucchese, S. Orlando, R. Perego, R. Trasarti, Conquest: a constraint-based querying system for exploratory pattern (demo), in: IEEE International Conference on Data Engineering, 2006.
- [12] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, The kdd process for extracting useful knowledge from volumes of data, *Commun. ACM* 39 (11) (1996) 27–34.
- [13] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proceedings of the 20th International Conference on Very Large Databases (VLDB'94), 1994.
- [14] S. Bistarelli, F. Bonchi, Interestingness is not a dichotomy: introducing softness in constrained pattern mining, in: Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, 2005, pp. 22–33.
- [15] F. Bonchi, F. Giannotti, C. Lucchese, S. Orlando, R. Perego, R. Trasarti, On interactive pattern mining from relational databases, in: International Workshop on Knowledge Discovery in Inductive Databases, 2006.
- [16] F. Bonchi, F. Giannotti, A. Mazzanti, D. Pedreschi, ExAMiner: optimized level-wise frequent pattern mining with monotone constraints, in: Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03), 2003.
- [17] J. Pei, J. Han, L.V.S. Lakshmanan, Mining frequent item sets with convertible constraints, in: 17th IEEE International Conference on Data Engineering (ICDE'01), 2001.
- [18] F. Bonchi, C. Lucchese, Pushing tougher constraints in frequent pattern mining, in: Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05), Hanoi, Vietnam, 2005.
- [19] R. Meo, G. Psaila, S. Ceri, A new SQL-like operator for mining association rules, in: T.M. Vijayarman, A.P. Buchmann, C. Mohan, N.L. Sarda (Eds.), Proceedings of 22th International Conference on Very Large Data Bases (VLDB'96), Mumbai (Bombay), India, 3–6 September 1996, pp. 122–133.
- [20] R. Meo, G. Psaila, S. Ceri, A tightly-coupled architecture for data mining, in: International Conference on Data Engineering (ICDE98), 1998, pp. 316–323.
- [21] J. Han, Y. Fu, K. Koperski, W. Wang, O. Zaiane, DMQL: a data mining query language for relational databases, in: SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96), 1996.
- [22] J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufman, Los Altos, CA, 2000.
- [23] T. Imielinski, A. Virmani, MSQL: a query language for database mining, *Data Min. Knowl. Discovery* 3 (4) (1999) 373–408.
- [24] T. Calders, B. Goetals, A. Prado, Integrating pattern mining in relational databases, in: Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'06), 2006, pp. 454–461.
- [25] S. Orlando, P. Palmerini, R. Perego, F. Silvestri, Adaptive and resource-aware mining of frequent sets, in: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02), Maebashi City, Japan, December 2002, pp. 338–345.
- [26] F. Bonchi, F. Giannotti, A. Mazzanti, D. Pedreschi, ExAnte: anticipated data reduction in constrained pattern mining, in: Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03), 2003.
- [27] F. Bonchi, C. Lucchese, On closed constrained frequent pattern mining, in: Proceedings of the 4th IEEE International Conference on Data Mining (ICDM'04), 2004.
- [28] L.V.S. Lakshmanan, R.T. Ng, J. Han, A. Pang, Optimization of constrained frequent set queries with 2-variable constraints, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD'99), 1999.
- [29] S. Kramer, L.D. Raedt, C. Helma, Molecular feature mining in hiv data, in: Proceedings of the 7th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'01), 2001.
- [30] C. Bucila, J. Gehrke, D. Kifer, W. White, DualMiner: a dual-pruning algorithm for itemsets with constraints, in: Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'02), 2002.
- [31] J. Pei, J. Han, Can we push more constraints into frequent pattern mining? in: Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'00), 2000.
- [32] F. Bonchi, C. Lucchese, Extending the state-of-the-art of constraint-based pattern discovery, *Data Knowl. Eng. (DKE)* 60 (2) (2007) 377–399.
- [33] L.D. Raedt, S. Kramer, The levelwise version space algorithm and its application to molecular fragment finding, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01), 2001.
- [34] B. Jeudy, J.-F. Boulicaut, Optimization of association rule mining queries, *Intelligent Data Anal. J.* 6 (4) (2002) 341–357.
- [35] F. Bonchi, F. Giannotti, A. Mazzanti, D. Pedreschi, Adaptive constraint pushing in frequent pattern mining, in: Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03), 2003.