Available online at www.sciencedirect.com

ScienceDirect

Data & Knowledge Engineering 60 (2007) 377–399

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Extending the state-of-the-art of constraint-based pattern discovery

Francesco Bonchi [a,*], Claudio Lucchese [b]

[a] Pisa KDD Laboratory, ISTI – CNR, Area della Ricerca di Pisa, Via Giuseppe Moruzzi, 1, 56124 Pisa, Italy
[b] Department of Computer Science, Universitá Ca' Foscari, Via Torino 155, 30172 Venice, Italy

Received 3 November 2005; accepted 7 February 2006
Available online 29 March 2006

Abstract

In the last years, in the context of the constraint-based pattern discovery paradigm, properties of constraints have been studied comprehensively and on the basis of this properties, efficient constraint-pushing techniques have been defined. In this paper we review and extend the state-of-the-art of the constraints that can be pushed in a frequent pattern computation. We introduce novel data reduction techniques which are able to exploit convertible anti-monotone constraints (e.g., constraints on *average* or *median*) as well as tougher constraints (e.g., constraints on *variance* or *standard deviation*). A thorough experimental study is performed and it confirms that our framework outperforms previous algorithms for convertible constraints, and exploit the tougher ones with the same effectiveness.

Finally, we highlight that the main advantage of our approach, i.e., pushing constraints by means of data reduction in a level-wise framework, is that different properties of different constraints can be exploited all together, and the total benefit is always greater than the sum of the individual benefits. This consideration leads to the definition of a general Apriori-like algorithm which is able to exploit all possible kinds of constraints studied so far.

Keywords: Frequent pattern mining; Constraint pushing techniques; Data reduction

1. Introduction

Devising fast and scalable algorithms, able to crunch huge amount of data, has been so far one of the main goals of data mining research. But now we realize that this is not enough. It does not matter how much efficient such algorithms can be, the results we obtain are often of limited use in practice. Typically, the knowledge we seek is in a small pool of local patterns hidden within an ocean of irrelevant patterns generated from a sea of data. Therefore, it is the volume of the results itself that creates a second order mining problem for the human expert. This is, typically, the case of association rules and frequent pattern mining [1], to which, during

[*] Corresponding author.
E-mail addresses: francesco.bonchi@isti.cnr.it (F. Bonchi), clucches@dsi.unive.it (C. Lucchese).
URLs: http://www-kdd.isti.cnr.it/~bonchi/ (F. Bonchi), http://hpc.isti.cnr.it/~claudio (C. Lucchese).

0169-023X/$ - see front matter © 2006 Elsevier B.V.. All rights reserved.
doi:10.1016/j.datak.2006.02.006

the last decade a lot of researchers have dedicated their (mainly algorithmic) investigations. The computational problem is that of efficiently mining patterns which satisfy a user-defined constraint of minimum frequency. The simplest form of a frequent pattern is the frequent itemset.

Definition 1 (*Frequent Itemset Mining*). Let $\mathscr{I} = \{x_1, \ldots, x_n\}$ be a set of distinct literals, usually called *items*, where an item is an object with some predefined attributes (e.g., price, type, etc.). An *itemset* X is a non-empty subset of $\mathscr{I}$. If $|X| = k$ then X is called a *k-itemset. A transaction database* $\mathscr{D}$ is a bag of itemsets $t \in 2^{\mathscr{I}}$, usually called *transactions*. The *support* of an itemset X in database $\mathscr{D}$, denoted $supp_{\mathscr{D}}(X)$, is the number of transactions which are superset of X. Given a user-defined *minimum support* σ, an itemset X is called *frequent* in $\mathscr{D}$ if $supp_{\mathscr{D}}(X) \geqslant \sigma$. This defines the minimum frequency constraint: $\mathscr{C}_{\text{freq}[\mathscr{D},\sigma]}(X) \Longleftrightarrow supp_{\mathscr{D}}(X) \geqslant \sigma$. When the dataset and the minimum support are clear from the context, we indicate the frequency constraint simply $\mathscr{C}_{\text{freq}}$.

Recently the research community has turned its attention to more complex kinds of frequent patterns extracted from more structured data: *sequences*, *trees*, and *graphs*. All these different kinds of pattern have different peculiarities and application fields, but they all share the same computational aspects: a usually very large input, an exponential search space, and a too large solution set. This situation – too many data yielding too many patterns – is harmful for two reasons. First, performance degrades: mining generally becomes inefficient or, often, simply unfeasible. Second, the identification of the fragments of interesting knowledge, blurred within a huge quantity of mostly useless patterns, is difficult. The paradigm of *constraint-based pattern mining* was introduced as a solution to both these problems. In such paradigm, it is the user which specifies to the system what is *interesting* for the current application: constraints are a tool to drive the mining process towards potentially interesting patterns, moreover they can be pushed deep inside the mining algorithm in order to fight the exponential search space curse, and to achieve better performance [22,18,14,13].

When instantiated to the pattern class of itemsets, the constraint-based pattern mining problem is defined as follows.

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

Example 1. The following is an example mining query:

$$\mathscr{Q} : supp_{\mathscr{D}}(X) \geqslant 1500 \wedge avg(X.weight) \leqslant 5 \wedge sum(X.price) \geqslant 22$$

It requires to mine, from database $\mathscr{D}$, all patterns which are frequent (have a support at least 1500), have average weight at most 5 and a sum of prices at least 22.

The constraint-based mining paradigm has been successfully applied in medical domain Ordonez et al. [19], and in biological domain Besson et al. [2]. According to the constraint-based mining paradigm, the data analyst must have a high-level vision of the pattern discovery system, without worrying about the details of the computational engine, in the very same way a database designer has not to worry about query optimization: she must be provided with a set of primitives to declaratively specify to the pattern discovery system how the interesting patterns should look like, i.e., which conditions they should obey. Indeed, the task of composing all constraints and producing the most efficient mining strategy (execution plan) for the given data mining query should be left to an underlying *query optimizer*. Therefore, constraint-based frequent pattern mining has been seen as a query optimization problem [17], i.e., developing efficient, sound and complete evaluation strategies for constraint-based mining queries. Or in other terms, designing efficient algorithms to mine all and only the patterns in $Th(\mathscr{C}_{\text{freq}}) \cap Th(\mathscr{C})$. A naïve solution to such a problem is to first mine all frequent patterns ($Th(\mathscr{C}_{\text{freq}})$) and then test them for constraints satisfaction. However more efficient solutions can be found by analyzing the properties of constraints comprehensively, and exploiting such properties in order to push constraints in the frequent pattern computation. Following this methodology, some classes of constraints which exhibit nice properties have been individuated [18] (e.g. monotonicity, anti-monotonicity, succinctness).

One of the toughest classes of constraints studied so far, is the class of *convertible* constraints [20,21]: they are constraints for which there is no clear interplay between subset relationship and constraint satisfiability, but an interplay can be found by arranging the items in some order. Consider for instance the constraint defined on the *average* aggregate (e.g., $avg(X.price) \leqslant v$): subsets (or supersets) of a valid itemset could well be invalid and vice versa. But, if we arrange the items in *price-descending-order* we can observe an interesting property: the average of an itemset is no more than the average of its prefix itemset, according to this order. In Pei and Han [20]; Pei et al. [21] it is shown that, since the FP-growth approach [15] to frequent itemset mining is based on the concept of prefix-itemsets, it is quite easy to push convertible constraints in such an algorithmic framework. The authors also state that pushing this kind of tough constraints *directly* into the level-wise breadth-first exploration of the search space, performed by Apriori-like algorithms, is not possible.

On the contrary, we have recently shown [5] how it is possible to push convertible constraints within a level-wise computation by means of *data reduction techniques*, and to use the same techniques to push much tougher constraints. Since frequent patterns are usually extracted from huge datasets, data-reduction techniques have been proven [8,9] to be very effective in this kind of computation: by reducing the input dataset they implicitly reduce also the search space of the computational problem, sometimes making otherwise intractable computations, feasible.

1.1. Paper contribution and organization

The contribution of this paper is threefold. First, we extend the actual state-of-the-art of constraints that can be pushed in a frequent pattern computation, by introducing a class of tough constraints, i.e., those ones based on *variance* or *standard deviation*, and by showing how to push them into an Apriori-like computation by means of a data reduction technique. We characterize such class showing that it is a superclass of convertible anti-monotone constraints. Therefore, our technique can be used also to push convertible constraints into an Apriori-like computation. Second, we show that, in the case of convertible constraints, other ad-hoc pruning strategies can be adopted in order to improve the efficiency of our method, outperforming previously proposed FP-growth based algorithms Pei and Han [20]. Third, we define a general Apriori-like framework, based on data reduction techniques, which is able to push all possible kinds of constraints studied so far. Note that all the previously proposed constraint pushing techniques were designed to work on their own. Conversely, we show that all of these constraints can be pushed in a unique, general framework. Our framework is unifying not only because it fits every constraint, but also because it can cope with any conjunction of constraints, thus giving even more expressive power to user's queries.

(1) In Section 2, as a side contribution, we provide an exhaustive state-of-the-art of constraint pushing techniques. We provide examples of interesting and meaningful constraints do not fall in any of the previously identified classes of constraints, and neither can be pushed by previous algorithms.
(2) In Section 3, we introduce the class of *loose anti-monotone* constraints and we deeply characterize it by showing that it is a superclass of convertible anti-monotone constraints (e.g. constraints on *average* or *median*) and that it contains tougher constraints (e.g. *variance* or *standard deviation*). We identify an interesting property of loose anti-monotone constraints which allows input data reduction. Exploiting this property, we extend *ExAMiner* [9], which is a level-wise Apriori-like algorithmic framework based on data-reduction techniques, in order to make it cope with loose anti-monotonicity. The resulting algorithm is named *ExAMiner*$^{\mathscr{LAM}}$.
(3) A thorough experimental study is performed. It confirms that by exploiting loose anti-monotonicity, *ExAMiner*$^{\mathscr{LAM}}$ is able to outperform previous algorithms for convertible constraints, and to treat much tougher constraints with the same effectiveness as easier ones.
(4) In Section 4, we develop novel advanced pruning techniques which can be adopted in the case of convertible constraints. The resulting algorithm is named *ExAMiner*$^{\mathscr{CAM}}$, and it further improves the performance of our framework.
(5) In Section 5, we introduce *ExAMiner*$^{\mathscr{GEN}}$, a general framework for constrained pattern mining, able to push into the mining process every conjunction of constraints that have been studied so far.

2. Related work and constraints classification

In this section, by reviewing all fundamental works on constrained frequent itemsets mining, we recall a classification of constraints and their properties.

2.1. Anti-monotone and succinct constraints

A first work defining classes of constraints which exhibit nice properties is Ng et al. [18]. In that paper is introduced an Apriori-like algorithm, named CAP, which exploits two properties of constraints, namely *anti-monotonicity* and *succinctness*, in order to reduce the frequent itemsets computation. Four classes of constraints, each one with its own associated computational strategy, are identified:

(1) constraints that are anti-monotone but not succinct;
(2) constraints that are both anti-monotone and succinct;
(3) constraints that are succinct but not anti-monotone;
(4) constraints that are neither.

Definition 3 (*Anti-monotone constraint*). Given an itemset *X*, a constraint $\mathcal{C}_{\mathrm{AM}}$ is *anti-monotone* if $\forall Y \subseteq X : \mathcal{C}_{\mathrm{AM}}(X) \Rightarrow \mathcal{C}_{\mathrm{AM}}(Y)$.

The frequency constraint is the most known example of a $\mathcal{C}_{\mathrm{AM}}$ constraint. This property, *the anti-monotonicity of frequency*, is used by the Apriori [1] algorithm with the following heuristic: if an itemset *X* does not satisfy $\mathcal{C}_{\mathrm{freq}}$, then no superset of *X* can satisfy $\mathcal{C}_{\mathrm{freq}}$, and hence they can be pruned. This pruning can affect a large part of the search space, since itemsets form a lattice. Therefore the Apriori algorithm (see Algorithm 1) operates in a level-wise fashion moving bottom-up, level-wise, on the itemset lattice, from small to large itemsets, generating the set of candidate itemsets at iteration k (the set C_k) from the set of frequent itemsets at the previous iteration (the set L_{k-1}). This way, each time it finds an infrequent itemset it implicitly prunes away all its supersets, since they will not be generated as candidate itemsets.

Algorithm 1. Apriori

```
Input: $\mathcal{D}, \sigma$
Output: $Th(\mathcal{C}_{\mathrm{freq}[\mathcal{D},\sigma]})$
$C_1 \leftarrow \{\{i\} | i \in \mathcal{I}\}$; $k \leftarrow 1$
while $C_k \neq \emptyset$ do
    $L_k \leftarrow count(\mathcal{D}, C_k)$
    $C_{k+1} \leftarrow generate\_apriori(L_k)$
    $k++$
$Th(\mathcal{C}_{\mathrm{freq}[\mathcal{D},\sigma]}) \leftarrow \bigcup_k L_k$
```

Other $\mathcal{C}_{\mathrm{AM}}$ constraints can easily be pushed deeply down into the frequent itemsets mining computation since they behave exactly as $\mathcal{C}_{\mathrm{freq}}$: if they are not satisfiable at an early level (small itemsets), they have no hope of becoming satisfiable later (larger itemsets). Conjoining other $\mathcal{C}_{\mathrm{AM}}$ constraints to $\mathcal{C}_{\mathrm{freq}}$ we just obtain a more selective anti-monotone constraint.

Example 2. If "price" has values in $\mathbb{R}^+$, then the constraint $sum(X.price) \leqslant 500$ is anti-monotone. Trivially, if an itemset *X* satisfies such constraints, then any of its subsets will satisfy the constraint as well. On the other hand, if *X* does not satisfy the constraint, then it can be pruned since none of its supersets will satisfy the constraint.

Informally, a succinct constraint $\mathcal{C}_{\mathrm{S}}$ is such that, whether an itemset *X* satisfies it or not, can be determined based on the singletons which are in *X*.

Definition 4 (*Succinct constraint*). An itemset $\mathscr{I}_s \subseteq \mathscr{I}$ is a succinct set, if it can be expressed as $\sigma_p(\mathscr{I})$ for some selection predicate p, where σ is the selection operator. $SP \subseteq 2^{\mathscr{I}}$ is a succinct powerset, if there is a fixed number of succinct sets $\mathscr{I}_1, \mathscr{I}_2, \ldots, \mathscr{I}_k \subseteq \mathscr{I}$, such that SP can be expressed in terms of the strict powersets of $\mathscr{I}_1, \mathscr{I}_2, \ldots, \mathscr{I}_k$ using union and minus. Finally, a constraint $\mathscr{C}_s$ is succinct provided that $Th(\mathscr{C}_{\mathrm{S}})$ is a succinct powerset.

Example 3. Consider constraint $\mathscr{C} \equiv S.type \supseteq \{food, toys\}$, the pruned search space consists of all those sets that contain at least one item of type *food* and at least one item of type *toys*. Let $\mathscr{I}_2, \mathscr{I}_3, \mathscr{I}_4$ be the sets $\sigma_{type='food'}(\mathscr{I})$, $\sigma_{type='toys'}(\mathscr{I})$, and $\sigma_{type\neq'food'\wedge type\neq'toys'}(\mathscr{I})$ respectively. Then, $\mathscr{C}_2$ is succinct because $Th(\mathscr{C})$ can be expressed as: $2^{\mathscr{I}} - 2^{\mathscr{I}_2} - 2^{\mathscr{I}_3} - 2^{\mathscr{I}_4} - 2^{\mathscr{I}_2 \cup \mathscr{I}_4} - 2^{\mathscr{I}_3 \cup \mathscr{I}_4}$.

A $\mathscr{C}_{\mathrm{S}}$ constraint is *pre-counting pushable*, i.e., it can be satisfied at candidate-generation time just taking into account the itemset and the single items satisfying the constraint. These constraints are pushed in the level-wise computation by substituting the usual *generate_apriori* procedure (Algorithm 1, line 4), with the proper (w.r.t. $\mathscr{C}_{\mathrm{S}}$) candidate generation procedure, which prunes every itemset which does not satisfy the constraint and that it is not a subset of any further valid itemset.

Constraints that are both anti-monotone and succinct can be pushed completely in the level-wise computation before it starts (at pre-processing time).

Example 4. For instance, consider the constraint $min(X.price) \geqslant v$. It is straightforward to see that it is both anti-monotone and succinct. Thus, if we start with the first set of candidates formed by all singleton items having price greater than v, during the computation we will generate all those itemsets satisfying the given constraint.

Constraints that are neither succinct nor anti-monotone are pushed in the CAP [18] computation by inducing weaker constraints which are either anti-monotone and/or succinct.

Example 5. Consider the constraint $avg(X.price) \leqslant v$ which is neither succinct nor anti-monotone. We can push the weaker constraint $min(X.price) \leqslant v$ with the advantage of reducing the search space and the guarantee that at least all the valid itemsets will be generated.

2.2. Monotone constraints

Monotone constraints work the opposite way of anti-monotone constraints.

Definition 5 (*Monotone constraint*). Given an itemset X, a constraint $\mathscr{C}_{\mathrm{M}}$ is monotone if: $\forall Y \supseteq X : \mathscr{C}_{\mathrm{M}}(X) \Rightarrow \mathscr{C}_{\mathrm{M}}(Y)$.

Example 6. The constraint $sum(X.price) \geqslant 500$ is monotone, since all prices are not negative. Trivially, if an itemset X satisfies such constraint, then any of its supersets will satisfy the constraint as well.

Since the frequent itemset computation is geared on $\mathscr{C}_{\mathrm{freq}}$, which is anti-monotone, $\mathscr{C}_{\mathrm{M}}$ constraints have been considered more hard to be pushed in the computation and less effective in pruning the search space. Many works [12,11,10,7] have studied the computational problem $Th(\mathscr{C}_{\mathrm{freq}}) \cap Th(\mathscr{C}_{\mathrm{M}})$ focussing on its search space, and trying some smart exploration of it. For example, Bucila et al. [11] try to explore the search space from the top and from the bottom of the lattice in order to exploit at the same time the symmetric behavior of monotone and anti-monotone constraints. Anyway, all of these approaches face the inherent difficulty of the computational problem: the $\mathscr{C}_{\mathrm{AM}}$-$\mathscr{C}_{\mathrm{M}}$ *tradeoff* that can be described as follows. Suppose that an itemset has been removed from the search space because it does not satisfy a monotone constraint. This pruning avoids the expensive support count for this itemset, but on the other hand, if we check its support and find it smaller than the frequency threshold, we may prune away all the supersets of this itemset, thus saving the support count for all of them. In other words, by monotone pruning we risk to lose anti-monotone pruning opportunities given by the pruned itemset. The tradeoff is clear: pushing monotone constraint can save frequency tests, however the results of these tests could have lead to more effective anti-monotone pruning. In Bonchi et al. [8] a completely new approach to exploit monotone constraints by means of data-reduction is introduced. The

ExAnte Property is obtained by shifting attention from the pattern search space to the input data. Indeed, the $\mathscr{C}_{\mathrm{AM}}$-$\mathscr{C}_M$ *tradeoff* exists only if we focus exclusively on the search space of the problem, while if exploited properly, monotone constraints can reduce dramatically the data in input, in turn strengthening the anti-monotonicity pruning power. With data reduction techniques we exploit the effectiveness of a $\mathscr{C}_{\mathrm{AM}}$-$\mathscr{C}_{\mathrm{M}}$ *synergy*. The ExAnte property states that a transaction which does not satisfy the given monotone constraint can be deleted from the input database since it will never contribute to the support of any itemset satisfying the constraint.

Proposition 1 (ExAnte property). *Given a transaction database* $\mathscr{D}$ *and a conjunction of monotone constraints* $\mathscr{C}_{\mathrm{M}}$, *we define the* μ*-reduction of* $\mathscr{D}$ *as the dataset resulting from pruning the transactions that do not satisfy* $\mathscr{C}_{\mathrm{M}}$: $\mu_{\mathscr{C}_{\mathrm{M}}}(\mathscr{D}) = \{t \in \mathscr{D} | t \in Th(\mathscr{C}_{\mathrm{M}})\}$. *This data reduction does not affect the support of solution itemsets*: $\forall X \in Th(\mathscr{C}_{\mathrm{M}}) : supp_{\mathscr{D}}(X) = supp_{\mu_{\mathscr{C}_{\mathrm{M}}}(\mathscr{D})}(X)$.

A major consequence of removing transactions from input database in this way, is that it implicitly reduces the support of a large amount of itemsets that do not satisfy $\mathscr{C}_{\mathrm{M}}$ as well, resulting in a reduced number of candidate itemsets generated during the mining algorithm. Even a small reduction in the database can cause a huge cut in the search space, because all supersets of infrequent itemsets are pruned from the search space as well. In other words, monotonicity-based data-reduction of transactions strengthens the anti-monotonicity-based pruning of the search space. This is not the whole story, in fact, singleton items may happen to be infrequent after the pruning and they cannot only be removed from the search space together with all their supersets, but for the same anti-monotonicity property they can be deleted also from all transactions in the input database (this anti-monotonicity-based data-reduction is named α-*reduction*). Removing items from transactions brings another positive effect: reducing the size of a transaction which satisfies $\mathscr{C}_{\mathrm{M}}$ can make the transaction violate it. Therefore a growing number of transactions which do not satisfy $\mathscr{C}_{\mathrm{M}}$ can be found. Obviously, we are inside a loop where two different kinds of pruning (α and μ) cooperate to reduce the search space and the input dataset, strengthening each other step by step until no more pruning is possible (a fix-point has been reached). This is the key idea of the ExAnte pre-processing method. In the end, the reduced dataset resulting from this fix-point computation is usually much smaller than the initial dataset, and it can feed any frequent itemset mining algorithm for a much smaller (but complete) computation. This simple yet very effective idea has been generalized from pre-processing to effective mining in two main directions: in an Apriori-like breadth-first computation in *ExAMiner* [9], and in a FP-growth based depth-first computation in *FP-Bonsai* [3].

2.2.1. The ExAMiner algorithm

ExAMiner [9], generalizes the ExAnte idea to reduce the problem dimensions at all levels of a level-wise Apriori-like computation. In this way, the $\mathscr{C}_{\mathrm{AM}}$-$\mathscr{C}_{\mathrm{M}}$ *synergy* is effectively exploited at each iteration of the mining algorithm, and not only at pre-processing as done by ExAnte, resulting in significant performance improvements. To this purpose, the following set of data reduction techniques, which are based on the anti-monotonicity of $\mathscr{C}_{\mathrm{freq}}$ (see Bonchi et al. [9] for the proof of correctness) are coupled with the μ-reduction for $\mathscr{C}_{\mathrm{M}}$ constraints as described in Proposition 1.

Proposition 2 (Anti-monotonicity based data reductions). *At the generic level k of the level-wise computation*:

$\mathscr{G}_k(i)$: *an item which is not subset of at least k frequent k-itemsets can be pruned away from all transactions in* $\mathscr{D}$.
$\mathscr{T}_k(t)$: *a transaction which is not superset of at least* $k+1$ *frequent k-itemsets can be removed from* $\mathscr{D}$.
$\mathscr{L}_k(i)$: *given an item i and a transaction t, if the number of frequent k-itemsets which are superset of i and subset of t is less than k, then i can be pruned away from transaction t.*

Essentially ExAMiner is an Apriori-like algorithm, which at each iteration $k-1$ produces a reduced dataset $\mathscr{D}_k$ to be used at the subsequent iteration k. Each transaction in $\mathscr{D}_k$, before participating to the support count of candidate itemsets, is reduced as much as possible by means of $\mathscr{C}_{\mathrm{freq}}$-based data reduction, and only if it survives to this phase, it is effectively used in the counting phase. Each transaction which arrives to the counting phase, is then tested against the $\mathscr{C}_{\mathrm{M}}$ (μ-reduction) , and reduced again as much as possible, and only if it survives to this second set of reductions, it is written to the transaction database for the next iteration

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

Algorithm 2. *count&reduce*

```
Input: $\mathscr{D}_k, \sigma, \mathscr{C}_{\mathrm{M}}, C_k, V_{k-1}$
1: for all $i \in \mathscr{I}$ do $V_k[i] \leftarrow 0$
2: for all tuples $t$ in $\mathscr{D}_k$ do
3:    for all $i \in t$ do if $V_{k-1}[i] < k-1$
4:       then $t \leftarrow t \backslash i$
5:       else $i.count \leftarrow 0$
6:    if $|t| \geqslant k$ and $\mathscr{C}_{\mathrm{M}}(t)$ then
7:       for all $X \in C_k$, $X \subseteq t$ do
8:          $X.count{+}{+}$; $t.count{+}{+}$
9:          for all $i \in X$ do $i.count{+}{+}$
10:         if $X.count = \sigma$ then
11:            $L_k \leftarrow L_k \cup \{X\}$
12:            for all $i \in X$ do $V_k[i]{+}{+}$
13:      if $|t| \geqslant k+1$ and $t.count \geqslant k+1$ then
14:         for all $i \in t$ if $i.count < k$
15:            then $t \leftarrow t \backslash i$
16:            if $|t| \geqslant k+1$ and $\mathscr{C}_{\mathrm{M}}(t)$ then
17:               write $t$ in $\mathscr{D}_{k+1}$
```

2.3. Convertible constraints

In Pei and Han [20]; Pei et al. [21] the class of convertible constraints is introduced, and an FP-growth based methodology to push such constraints is proposed.

Definition 6 (*Convertible constraints*). A constraint $\mathscr{C}_{\mathrm{CAM}}$ is convertible anti-monotone provided there is an order $\mathscr{R}$ on items such that whenever an itemset X satisfies $\mathscr{C}_{\mathrm{CAM}}$, so does any prefix of X. A constraint $\mathscr{C}_{\mathrm{CM}}$ is convertible monotone provided there is an order $\mathscr{R}$ on items such that whenever an itemset X violates $\mathscr{C}_{\mathrm{CM}}$, so does any prefix of X.

In order to be convertible, a constraint must be defined over a *Prefix Increasing* (*resp. Decreasing*) *Function*, i.e. a function $f: 2^{\mathscr{I}} \to \mathbb{R}$ such that for every itemset S and item a, if $\forall x \in S,\ x\mathscr{R}a$ then $f(S) \leqslant$ (resp. $\geqslant$) $f(S \cup \{a\})$. Let f be a prefix increasing (resp. decreasing) function w.r.t. a given order $\mathscr{R}$. Then $f(X) \geqslant v$ is a convertible monotone (resp. anti-monotone) constraint, while $f(X) \leqslant v$ is a convertible anti-monotone (resp. monotone) constraint.

Example 7 (*avg constraint is convertible*). Let $\mathscr{R}$ be the value-descending order. It is straightforward to see that *avg* is a prefix decreasing function w.r.t. $\mathscr{R}$. This means that $avg(X) \geqslant v$ is a $\mathscr{C}_{\mathrm{CAM}}$ constraint and $avg(X) \leqslant v$ is $\mathscr{C}_{\mathrm{CM}}$ w.r.t. the same order.

Interestingly, if the order $\mathcal{R}^{-1}$ (i.e. the reversed order of $\mathcal{R}$) is used, the constraint $avg(S) \geqslant v$ can be shown convertible monotone, and $avg(S) \leqslant v$ convertible anti-monotone. Constraints which exhibit this interesting property of being convertible in both a monotone or an anti-monotone constraint, are called *strongly convertible*.

Clearly, not every convertible constraint is strongly convertible.

Example 8. The constraint $sum(X.price) \geqslant v$, being monotone, is also convertible monotone: just pick any order on items. But there is no order for which we can convert such constraint to an anti-monotone one.

In Pei and Han [20], two FP-growth based algorithms are introduced: $\mathcal{FIC}^{\mathcal{A}}$ to mine $Th(\mathcal{C}_{\text{freq}}) \cap Th(\mathcal{C}_{\text{CAM}})$, and $\mathcal{FIC}^{\mathcal{M}}$ to mine $Th(\mathcal{C}_{\text{freq}}) \cap Th(\mathcal{C}_{\text{CM}})$. A major limitation of any FP-growth based algorithm is that the initial database (internally compressed in the prefix-tree structure) and all intermediate projected databases must fit into main memory. If this requirement cannot be met, these approaches can simply not be applied anymore. This problem is even harder with $\mathcal{FIC}^{\mathcal{A}}$ and $\mathcal{FIC}^{\mathcal{M}}$: in fact, using an order on items different from the frequency-based one, makes the prefix-tree lose its compressing power. Thus we have to manage much greater data structures, requiring a lot more main memory which might not be available. This fact is confirmed by our experimental analysis reported in Section 3.2: sometimes $\mathcal{FIC}^{\mathcal{A}}$ is slower than FP-growth, meaning that having constraints brings no benefit to the computation. Another important drawback of this approach is that it is not possible to take full advantage of a conjunction of different constraints, since each constraint in the conjunction could require a different ordering of items. In our data-reduction based approach we can fully exploit different kind of constraints: the more constraints we have the stronger is the data-reduction effect (see later Example 13). Finally, while in $\mathcal{FIC}^{\mathcal{A}}$ the constraint is effectively exploited to reduce the growing of the tree, thus producing a real pruning of the search space, the same does not happen with $\mathcal{FIC}^{\mathcal{M}}$. Strictly speaking, this algorithm cannot be considered a constraint-pushing technique, since it generates the complete set of frequent itemsets, no matter whether they satisfy or not $\mathcal{C}_{\text{CM}}$. The only advantage of $\mathcal{FIC}^{\mathcal{M}}$ against a pure *generate and test* algorithm is that $\mathcal{FIC}^{\mathcal{M}}$ only tests some of frequent itemsets against $\mathcal{C}_{\text{CM}}$: once a frequent itemset satisfies $\mathcal{C}_{\text{CM}}$, all frequent itemsets having it as a prefix also are guaranteed to satisfy the constraint.

2.4. Non-convertible constraints

Unfortunately, many constraints does not fall in any of the classes we described above. Therefore, the classification of constraints needs to be extended to new interesting constraints, in order to discover new strategy that can help in exploiting such constraints during mining process.

Example 9 (*var constraint is not convertible*). Calculating the variance is an important task of many statistical analysis: it is a measure of how spread out a distribution is. The variance of a set of number X is defined as:

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

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

Following a similar reasoning we can show that other interesting constraints, such as for instance those ones based on *standard deviation* (*std*) or *unbiased variance estimator* (var_{N-1}) or *mean deviation* (*md*), are not convertible as well.

A first work, trying to address the problem of how to push constraints which are not convertible, is Kifer et al. [16]. The framework proposed in that paper is based on the concept of finding a *witness*, i.e. an itemset such that, by testing whether it satisfies the constraint we can deduce information about properties of other itemsets, that can be exploited to prune the search space. This idea is embedded in a depth-first visit of the itemsets search space. The authors instantiate their framework to the constraint based on the *variance* aggregate, which we also study in this paper. Although the authors describe algorithms to efficiently find a witness for both the *avg* and *var* constraints, the work in Kifer et al. [16] is mainly theoretical and the proposed algorithms have not been implemented nor experimented. The main drawback of their proposal is the following: it may require quadratic time in the number of frequent singletons to find a witness. The cost can be amortized if items are reordered, but this leads to the same problems discussed for FP-growth based algorithms. Moreover, even if a nearly linear time search is performed, this is done without any certainty of finding a witness which will help to prune the search space. In fact, if the witness found satisfies the given constraint, no pruning will be possible and the search time will be wasted. Our approach is completely orthogonal: while they try to explore the exponentially large search space in some smart way, we massively reduce the dataset as soon as possible, reducing at the same time the search space and obtaining a progressively easier mining problem.

3. Loose anti-monotone constraints

As we have seen in the previous section, many interesting constraints, e.g., those one based on *var* or *std*, do not fall in any previously defined class of constraints. Recently, we have individuated a new class of constraints sharing a nice property that we have named "*loose anti-monotonicity*" [5]. This class is a proper superclass of convertible anti-monotone constraints, and it can also deal with other tougher constraints. Based on loose anti-monotonicity we can define a data reduction strategy, which makes the mining task feasible and efficient.

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

Definition 7 (*Loose Anti-monotone constraint*). Given an itemset X with $|X| > 2$, a constraint is *loose anti-monotone* (denoted $\mathscr{C}_{\text{LAM}}$) if: $\mathscr{C}_{\text{LAM}}(X) \Rightarrow \exists i \in X : \mathscr{C}_{\text{LAM}}(X \setminus \{i\})$.

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

Proposition 3. *Any convertible anti-monotone constraint is trivially loose anti-monotone: if a k-itemset satisfies the constraint so does its* $(k-1)$*-prefix itemset.*

Example 10 (*var, std, md,* var_{N-1} *constraints are loose anti-monotone*). We show that the constraint $var(X.A) \leqslant v$ is a $\mathscr{C}_{\text{LAM}}$ constraint. Given an itemset X, if it satisfies the constraint so trivially does $X\setminus\{i\}$, where i is the element of X which has associated a value of A which is the most far away from $avg(X.A)$. In fact, we have that $var(\{X\setminus\{i\}\}.A) \leqslant var(X.A) \leqslant v$, until $|X| > 2$. Conversely, taking the element of X which has associated a value of A which is the closest to $avg(X.A)$ we can show that $var(X.A) \geqslant v$ is a $\mathscr{C}_{\text{LAM}}$ constraint. Since the standard deviation *std* is the square root of the variance, it is straightforward to see that $std(X.A) \leqslant v$ and $std(X.A) \geqslant v$ are both $\mathscr{C}_{\text{LAM}}$. The mean deviation is defined as: $md(X) = \left(\sum_{i\in X}|i - avg(X)|\right)/|X|$. Once again, we have that $md(X.A) \leqslant v$ and $md(X.A) \geqslant v$ are $\mathscr{C}_{\text{LAM}}$. It is easy to prove that also constraints defined on the unbiased variance estimator, $var_{N-1} = (\sum_{i\in X}(i - avg(X))^2)/(|X| - 1)$ are loose anti-monotone (Fig. 1).

The next key Theorem indicates how a $\mathscr{C}_{\text{LAM}}$ constraint can be exploited in a level-wise Apriori-like computation by means of data-reduction. It states that if at a certain iteration $k > 2$ a transaction is not superset of at least one frequent k-itemset which satisfy the $\mathscr{C}_{\text{LAM}}$ constraint (i.e. it is a solution), then the transaction can be deleted from the database.

Theorem 1. *Given a transaction database* $\mathscr{D}$*, a minimum support threshold* σ*, and a* $\mathscr{C}_{\text{LAM}}$ *constraint, at the iteration* $k \geqslant 2$ *of the level-wise computation, a transaction* $t \in \mathscr{D}$ *such that:* $\nexists X \subseteq t, |X| = k, X \in Th(\mathscr{C}_{\text{freq}[\mathscr{D},\sigma]}) \cap Th(\mathscr{C}_{\text{LAM}})$ *can be pruned away from* $\mathscr{D}$*, since it will never be superset of any solution itemsets of cardinality* $> k$*.*

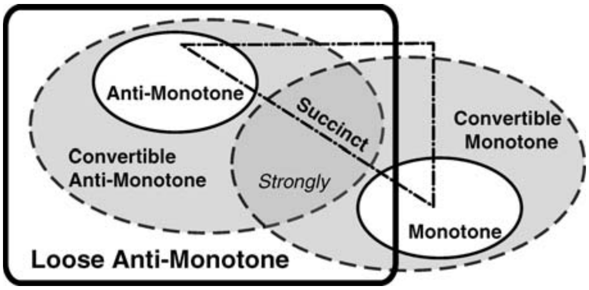

Fig. 1. Characterization of the classes of commonly used constraints.

Proof. Suppose that exists $Y \subseteq t, |Y| = k + j, Y \in Th(\mathscr{C}_{\text{freq}[\mathscr{D},\sigma]}) \cap Th(\mathscr{C}_{\text{LAM}})$. For loose anti-monotonicity this implies that exists $Z \subseteq Y$, $|Z| = k + j - 1$ such that $\mathscr{C}_{\text{LAM}}(Z)$. Moreover, for anti-monotonicity of frequency we have that $\mathscr{C}_{\text{freq}[\mathscr{D},\sigma]}(Z)$. The reasoning can be repeated iteratively downward to obtain that must exist $X \subseteq t,\ |X| = k,\ X \in Th(\mathscr{C}_{\text{freq}[\mathscr{D},\sigma]}) \cap Th(\mathscr{C}_{\text{LAM}})$. □

Note that a conjunction of loose anti-monotone constraints is not a loose anti-monotone constraint anymore, and therefore each constraint in a conjunction must be treated separately. However, a transaction can be pruned whenever Theorem 1 holds for even only one of the constraints, because every itemset in the transaction will not satisfy such constraint and, consequently, any conjunction including it.

Example 11. Given the two constraints $\mathscr{C}^1_{\text{LAM}} \equiv avg(X.A_1) \geqslant 140$ and $\mathscr{C}^2_{\text{LAM}} \equiv avg(X.A_2) \leqslant 320$, where the values of the attributes A_1 and A_2 are respectively $A_1 = \langle a{:}125, b{:}145, c{:}150\rangle$ and $A_2 = \langle a{:}300, b{:}310, c{:}350\rangle$, the conjunction $\mathscr{C}^1_{\text{LAM}} \wedge \mathscr{C}^2_{\text{LAM}}$ is not a loose anti-monotone constraint. Indeed, the itemset $X = \{abc\}$ satisfies $\mathscr{C}^1_{\text{LAM}} \wedge \mathscr{C}^2_{\text{LAM}}$, but the only subset of X satisfying $\mathscr{C}^1_{\text{LAM}}$ is $\{bc\}$, while the only subset satisfying $\mathscr{C}^2_{\text{LAM}}$ is $\{ab\}$, therefore there is no item $i \in X$ such that $\mathscr{C}^1_{\text{LAM}} \wedge \mathscr{C}^2_{\text{LAM}}(X \setminus \{i\})$ holds.

In the next section we exploit such property of $\mathscr{C}_{\text{LAM}}$ constraints in a level-wise Apriori-like computation by means of data-reduction.

3.1. The ExAMiner$^{\mathscr{LAM}}$ algorithm

As in ExAMiner (Section 2.2.1) the anti-monotonicity based data reductions of Proposition 2, were coupled with the μ-reduction for $\mathscr{C}_{\text{M}}$ constraints of Proposition 1; here, in order to cope with the mining problem $Th(\mathscr{C}_{\text{freq}}) \cap Th(\mathscr{C}_{\text{LAM}})$, we couple the same set of $\mathscr{C}_{\text{freq}}$-based data reduction techniques with the $\mathscr{C}_{\text{LAM}}$-based data reduction technique described in Theorem 1. This is done by extending the *count&reduce* procedure (Algorithm 2) to implement also the $\mathscr{C}_{\text{LAM}}$-based data reduction. The resulting algorithm is named *ExAMiner*$^{\mathscr{LAM}}$.

Our thorough experimental study (reported in Section 3.2) confirms that by exploiting loose anti-monotonicity, *ExAMiner*$^{\mathscr{LAM}}$ is able to outperform previous algorithms for convertible constraints (e.g. constraints on *average* or *median*), and to treat much tougher constraints (e.g. *variance* or *standard deviation*) with the same effectiveness as easier ones.

3.1.1. Run through example

In Fig. 2(b) we have a transactional dataset and an associated *item-price* table in Fig. 2(a). Suppose that we want *ExAMiner*$^{\mathscr{LAM}}$ to mine frequent itemsets (minimum support $\sigma = 3$) having a small ($\leqslant$10) variance of prices. In the following we denote with C_k the candidate k-itemsets, with L_k the candidate k-itemsets that are also frequent, and with R_k the set of k-itemsets that are frequent and that satisfy the constraint.

During the first iteration no pruning is possible. We just count the support of singletones $C_1 = \{a,b,c,d,e,f,g,h,i,j\}$ using all transactions in the dataset. At the end of the first iteration we discover that items f and h are infrequent, and therefore they will be discarded during the next iteration.

All the other singletones are frequent and, since the variance of a singleton is zero they all satisfy the $\mathscr{C}_{\text{LAM}}$ constraint, therefore they are all valid itemsets: $L_1 = R_1 = \{a,b,c,d,e,g,i,j\}$.

prices	
a	50
b	30
c	17
d	40
e	60
f	25
g	15
h	35
i	10
j	20

(a)

tID	Items
1	a, b, e, i, j
2	a, c, g, h, i, j
3	b, c, d, e
4	a, b, c, f
5	c, g, h, i
6	a, d, i
7	a, b, c, g, j
8	c, d, e, g, i
9	c, d, f, g, j
10	a, b, c, d, g

(b)

tID	Items
2	a, c, g, i, j
5	c, g, i
7	a, b, c, g, j
8	c, d, e, g, i
9	c, d, g, j
10	a, b, c, d, g

(c)

tID	Items
2	a, c, g, i, j
7	a, c, g, j

(d)

Fig. 2. Run through example.

During the second iteration we generate the set of candidates C_2 from L_1, but both $\mathcal{T}_k(t)$ and $\mathcal{L}_k(i)$ fail in reducing the input dataset. Luckily we can exploit the loose anti-monotonicity of the *var* constraint to remove some transaction. In fact, transaction 4 is superset of 3 candidate itemsets $\{ab, ac, bc\}$, all having a variance greater than 10. Thus the it can be pruned according to Theorem 1. It is easy to see that in the same way also the transactions 1, 3 and 6 can be pruned. In Fig. 2(c) we have the reduced dataset we obtain at end of this second iteration. Moreover we obtain the set of frequent itemsets $L_2 = \{ab, ac, ag, ai, bc, cd, cg, ci, dg, gi, aj, cj, gj\}$, among which only 4 satisfy the *var* constraint: $R_2 = \{cg, gi, cj, gj\}$.

We start the third iteration and as usual we generate the set of candidates $C_3 = \{abc, acg, aci, acj, agi, agj, cdg, cgi, cgj\}$. At this point, we discover that only 3 itemsets are frequent $L_3 = \{cgi, acg, cdg\}$ and only one is a solution $R_3 = \{cgi\}$. The previous $\mathcal{C}_{\text{LAM}}$ pruning has reduced the dataset in such a way that now we can perform additional $\mathcal{C}_{\text{freq}}$-based pruning until get the dataset in Fig. 2(d), where we have only two transactions and therefore no longer itemset can have support at least 3.

The computation on this toy example would be easily done even without any data-reduction, but we have always to keep in mind that frequent patterns are usually extracted from huge datasets. Therefore, by reducing the input size we also reduce the exponential search space and thus the computational cost, sometimes making feasible computations otherwise intractable.

3.2. Loose anti-monotonicity: experimental analysis

In this section we describe in details the experiments we have conducted in order to assess loose anti-monotonicity effectiveness on both convertible constraints (e.g. $avg(X.A) \geqslant m$) and tougher constraints (e.g. $var(X.A) \leqslant m$). The results are reported in Fig. 3.

All the tests were conducted on a Windows XP PC equipped with a 2.8 GHz Pentium IV and 512 MB of RAM memory, within the *cygwin* environment. The datasets used in our tests are those ones of the FIMI repository[1], and the constraints were applied on attribute values generated randomly with a gaussian distribution within the range [0, 150,000].

[1] http://fimi.cs.helsinki.fi/data/.

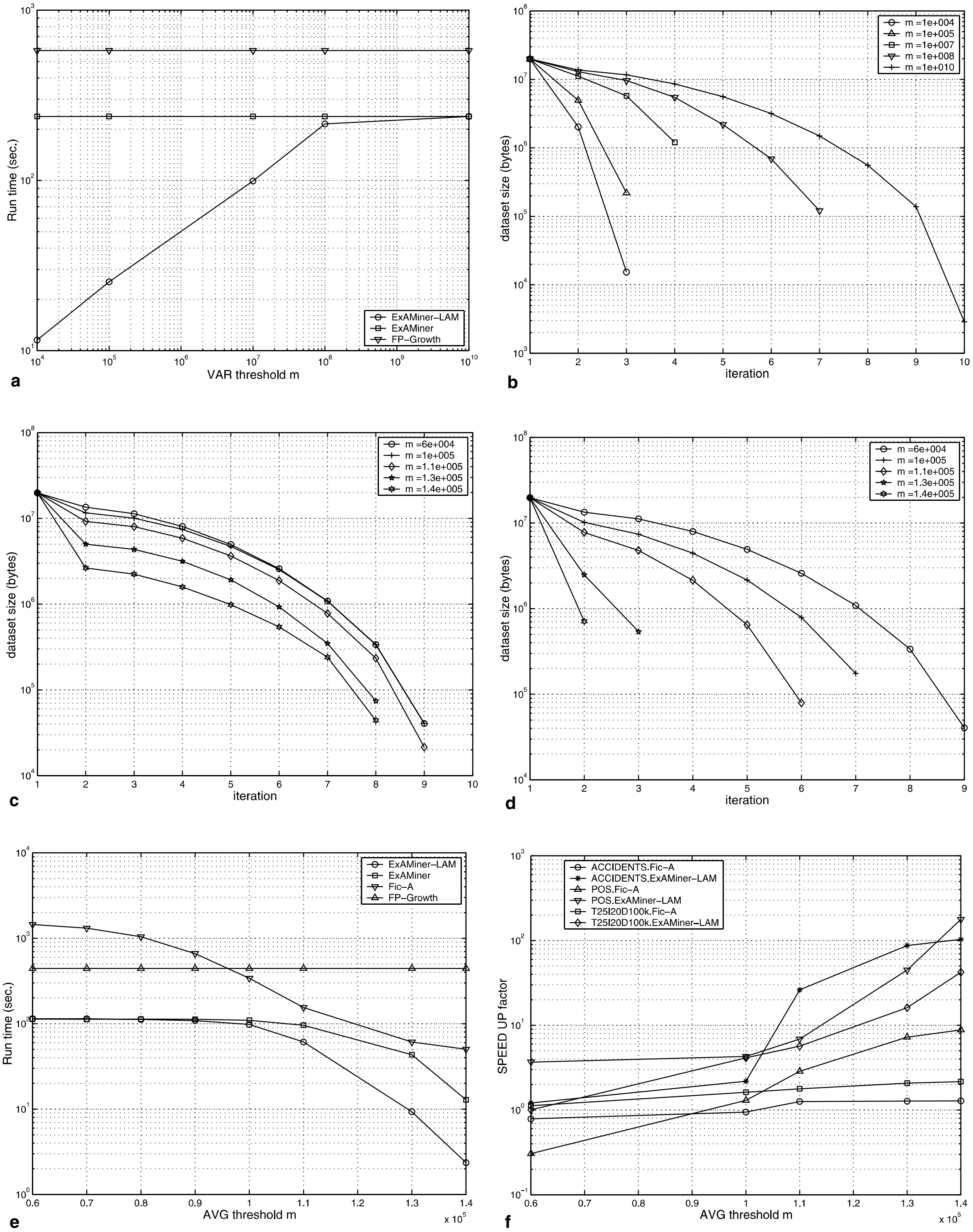

Fig. 3. Loose anti-monotonicity: experimental analysis results. (a) Various algorithms, dataset BMS-POS, $\sigma = 400$, $\mathscr{C}_{\mathrm{LAM}} \equiv var(X.S) \leqslant m$; (b) $ExAMiner^{\mathscr{LAM}}$, dataset BMS-POS, $\sigma = 400$, $\mathscr{C}_{\mathrm{LAM}} \equiv var(X.S) \leqslant m$; (c) ExAMiner, dataset BMS-POS, $\sigma = 300$, $\mathscr{C}_{\mathrm{CAM}} \equiv avg(X.S) \geqslant m$; (d) $ExAMiner^{\mathscr{CAM}}$, dataset BMS-POS, $\sigma = 300$, $\mathscr{C}_{\mathrm{CAM}} \equiv avg(X.S) \geqslant m$; (e) various algorithms, dataset BMS-POS, $\sigma = 300$, $\mathscr{C}_{\mathrm{CAM}} \equiv avg(X.S) \geqslant m$; (f) various algorithms, various datasets, various σ, $\mathscr{C}_{\mathrm{CAM}} \equiv avg(X.S) \geqslant m$.

In Fig. 3(a) and (b) tests over the $\mathscr{C}_{\text{LAM}}$ constraint $var(X.A) \leqslant m$ are reported. Since we are pushing a $\mathscr{C}_{\text{LAM}}$ constraint never studied before, we compare *ExAMiner*$^{\mathscr{LAM}}$ against two unconstrained computation: FP-Growth and ExAMiner without constraints (i.e. it only exploits $\mathscr{C}_{\text{freq}}$-based data reduction). Such tests highlight the effectiveness of loose anti-monotonicity: we have a speed up of much more than one order of magnitude, and a data reduction rate up to four order of magnitude. In Fig. 3(c) and (d) we compared the dataset reduction power of ExAMiner against *ExAMiner*$^{\mathscr{LAM}}$ when mining with the $\mathscr{C}_{\text{CAM}}$ constraint $avg(X.A) \geqslant m$. Since ExAMiner is designed to deal with monotone constraints, and *avg* is not, such constraint is pushed by inducing the weaker but monotone constraint $max(X.A) \geqslant v$ as done in Bonchi et al. [9]. This test is useful to understand how much the new class of constraints is able to prune the input data against a previous state of the art algorithm such as ExAMiner. In Fig. 3(c) we can see that ExAMiner is able to decrease the dataset size up to nearly three orders of magnitude. On the other hand, *ExAMiner*$^{\mathscr{LAM}}$, see Fig. 3(d), behaves much better, since it is able prune the dataset more effectively, in such a way that the dataset is entirely pruned away with the most selective constraints after the first three iterations. This behavior is reflected in run-time performances: *ExAMiner*$^{\mathscr{LAM}}$ is one order of magnitude faster than ExAMiner as reported in Fig. 3(e). Conversely, $\mathscr{FIC}^{\mathscr{A}}$ is not able to bring such improvements. In Fig. 3(f) we report the speed-up of *ExAMiner*$^{\mathscr{LAM}}$ w.r.t. ExAMiner and $\mathscr{FIC}^{\mathscr{A}}$ w.r.t. FP-growth. The tests conducted on various datasets show that exploiting loose anti-monotonicity property brings a higher speed up than exploiting convertibility. In fact, *ExAMiner*$^{\mathscr{LAM}}$ exhibits in average a speed up of factor 100 against its own unconstrained computation, while $\mathscr{FIC}^{\mathscr{A}}$ always provides a speed up w.r.t. FP-growth of a factor lower than 10, and sometimes it is even slower than its unconstrained version. In other words, FP-Growth followed by a filtering of the output in some cases is better that its variant $\mathscr{FIC}^{\mathscr{A}}$, which is explicitly geared on constrained mining. As we have discussed in Section 2.3 this is due to the items ordering based on attribute values and not on frequency.

In the next section we introduce three advanced pruning techniques which can be adopted when mining frequent patterns with convertible constraints. These pruning techniques, conjoined with the loose anti-monotonicity data reduction further improve the performance of our framework.

4. Advanced pruning techniques

In the previous section we have shown that, by exploiting only the property (Theorem 1) for loose anti-monotone constraints, *ExAMiner*$^{\mathscr{LAM}}$ is able to outperform the state-of-the-art algorithms for frequent pattern mining under convertible ($\mathscr{C}_{\text{CAM}}$) constraints (see Fig. 3(e)). However, in the case of convertible constraints, we have further data reduction opportunities then using $\mathscr{C}_{\text{LAM}}$ pruning only. In this Section we focus on the mining problem $Th(\mathscr{C}_{\text{freq}}) \cap Th(\mathscr{C}_{\text{CAM}})$ and we introduce three novel strategies that allow to boost the pruning power of our data reduction based framework. The algorithm resulting by conjoining these three data reduction techniques to the loose anti-monotonicity technique is named *ExAMiner*$^{\mathscr{CAM}}$.

For sake of clarity of presentation, we always refer to $avg(X.A) \geqslant m$ as prototypical $\mathscr{C}_{\text{CAM}}$ constraint without any loss of generality. Similarly to Pei and Han [20]; Pei et al. [21] we require items to be sorted by descending (ascending) order of attribute if $\mathscr{C}_{\text{CAM}}$ is defined over a prefix decreasing (increasing) function f (we denote this order by $\prec$). Transactions in $\mathscr{D}$, frequent itemsets in L_i, as well as candidate itemsets in C_i, must be ordered accordingly. Under this assumption three pruning techniques can be exploited.

4.1. Pre-counting reduction

At the beginning of iteration k of the level-wise framework, the average of the first k items of each transaction $t \in \mathscr{D}$ is calculated, and if it is smaller than m, then it cannot exist any $X \subseteq t$, $|X| \geqslant k$ such that $avg(X.A) \geqslant m$. Therefore transaction t cannot support any solution itemset for the current and future iterations, and thus they can be removed from $\mathscr{D}$. The dataset obtained after such reduction is denoted $\mathscr{D}'$. In Algorithm 3 *prefix*(I,n) denotes the n-prefix of I (i.e. the first n items of I).

Algorithm 3. Pre-counting Reduction

```
Input: $\mathscr{D}, k, \mathscr{C}_{\mathrm{CAM}}$
Output: $\mathscr{D}'$
$\mathscr{D}' \leftarrow \emptyset$
for all $t \in \mathscr{D}$ do
    if $\mathscr{C}_{\mathrm{CAM}}(prefix(t, k))$ then
        $\mathscr{D}' \leftarrow \mathscr{D}' \cup t$
```

Theorem 2 (Pre-counting Reduction). *Given a dataset $\mathscr{D}$ and a convertible anti-monotone constraint $\mathscr{C}_{\mathrm{CAM}}$. Let $\mathscr{D}'$ be the reduced dataset produced by Algorithm* 3. *It holds that*:

$$\forall X \in Th(\mathscr{C}_{\mathrm{CAM}}), \quad |X| \geqslant k : supp_{\mathscr{D}}(X) = supp_{\mathscr{D}'}(X)$$

Proof. For each $X \in Th(\mathscr{C}_{\mathrm{CAM}})$ there exist $supp_{\mathscr{D}}(X)$ transactions $t \supseteq X$ in $\mathscr{D}$. If $\mathscr{C}_{\mathrm{CAM}}$ is defined over a prefix decreasing function f as $f(S.A) \geqslant v$, then items in t are sorted in descending order, and therefore $f(prefix(t,k).A) \geqslant f(X.A) \geqslant v \Rightarrow \mathscr{C}_{\mathrm{CAM}}(prefix(t,k))$. By construction every of such t will be included in $\mathscr{D}'$, i.e. X turns out to be a frequent itemset and with the same support in $\mathscr{D}'$ as in $\mathscr{D}$. Analogously when $\mathscr{C}_{\mathrm{CAM}}$ is defined over a prefix increasing function. □

4.2. Counting early stopping

During the usual counting procedure of Apriori (Algorithm 1, line 3) at the iteration k, for each transaction t, all its k-subsets are generated and matched against the set of candidate itemsets C_k. We would like to stop as soon as possible this costly matching procedure. Such early stopping has an important side-effect because, by reducing the number of intersections between C_k and t, we also reduce the local counts *i.count* for some item $i \in t$, thus boosting the $\mathscr{L}_k(i)$ pruning.

However, we cannot do it straightforwardly, we must be careful in identifying the proper time for the early stopping, guaranteeing that necessary items are not deleted. In particular, as shown in the following Example, we cannot simply stop as soon as we found an itemset $X \subseteq t$, $X \in C_k$ which does not satisfy the constraint.

Example 12. Let $t = \{a,b,c,d,e,f,g,h\}$ be a transaction in $\mathscr{D}$, with associated prices $\langle 100,100,80,40,35,30,20,15\rangle$, and let $\mathscr{C}_{\mathrm{CAM}} \equiv avg(X.A) \geqslant 70$.

We could think to stop as soon as we found an itemset $X \subseteq t$, $X \in C_k$ which does not satisfy the constraint, such as $\{ade\}$. This would not be correct, since we would not discover further valid itemsets like $\{bcf\}$.

Then, we could think to stop when no other interesting itemset $Y \subseteq t$, $Y \in C_k | Y \succ X \wedge \mathscr{C}_{\mathrm{CAM}}(Y)$ exists. This is the case of $X = \{bcg\}$, for which $\neg\exists Y \succ X | \mathscr{C}_{\mathrm{CAM}}(Y)$. But also in this way we would lose some valid itemsets. In fact, stopping after $\{bcg\}$, we would have a local count of 2 for the item h (given by $\{abh\}$ and $\{ach\}$), and therefore h would be deleted because of its low local count, avoiding to discover the valid itemset $\{abch\}$ during the next iteration.

Our goal is to stop as soon as possible the counting procedure and, at the same time, to increase pruning opportunities, guaranteeing that necessary items are not deleted. The stopping criterion we provide works as follows (see Algorithm 4): when an itemsets $X \subseteq t$, $X \in C_k$ which does not satisfies the constraint is met (lines 4–5), its last item *last*(X) is recorded (line 6); afterwards if every other itemset $Y \subseteq t$, $Y \in C_k | first(Y) \preceq last(X)$ does not satisfy the constraint either then the counting procedure is stopped (lines 9–10), otherwise the stopping criterion can be applied to another itemset X such that $\neg\mathscr{C}_{\mathrm{CAM}}(X)$.

To prove the correctness of Algorithm 4, we must assure that, at the iteration k, there is no item i with low *i.count* that will belong to some frequent valid itemset in the following iterations. This is done by the next theorem.

Theorem 3. *The stopping criterion defined by Algorithm* 4 *is such that, after the counting procedure is applied to a transaction* $t \in \mathcal{D}$, *for every item* $i \in t$ *we have that* $i.count < k \Rightarrow \nexists I : i \in I \wedge |I| > k \wedge I \in Th(\mathcal{C}_{\mathrm{freq}}) \cap Th(\mathcal{C}_{\mathrm{CAM}})$.

Proof. We prove by contradiction that no item i, belonging to a valid solution itemset I, will have a local count $i.count < k$. First we note that by construction every item $i \preceq last(X)$ has a correct local count by construction, because every candidate itemset Y subsuming $last(X)$ is evaluated, and therefore we focus on items $i \succ last(X)$. Suppose that at the iteration k we have that $i.count < k$ and that such valid and frequent itemset $I \ni i$ exists. There are two alternatives: either $I \preceq last(X)$ or $I \succ last(X)$. In the former, since I is a frequent l-itemset with $l > k$, there exist at least k frequent k-itemsets $\{Y | i \in Y \wedge first(Y) \preceq last(X)\}$ which are subsets of I, and therefore $i.count \geqslant k$, which is in contradiction with the hypothesis. In the latter, it must hold that $first(I) \succ last(X)$, but since we have that X does not satisfy $\mathcal{C}_{\mathrm{CAM}}$, because of the item ordering I will not either, and therefore $I \notin Th(\mathcal{C}_{\mathrm{CAM}})$ which is again in contradiction with the hypothesis. □

As a special case of the above Theorem we exploit the following Lemma, which allows an immediate detection of a stopping itemset.

Lemma 1. *If exists an itemset* X *such that* $X \subseteq t, X \in C_k, \neg\mathcal{C}_{\mathrm{CAM}}(X)$ *and the items of* X *occur consecutively in* t, *then the counting procedure can stop after the itemset* X.

Proof. It is straightforward to see that since the items of X occur consecutively, then $\neg\exists Y \subseteq t, Y \in C_k | first(Y) \preceq last(X) \wedge \mathcal{C}_{\mathrm{CAM}}(Y)$, and therefore according to Theorem 3 the counting procedure can be stopped after X. □

Algorithm 4. Counting Early Stopping

```
for all $t \in \mathcal{D}$ do
  $invalidFound \leftarrow false$
  for all $X \in C_k | X \subseteq t$ do
    if $\neg\mathcal{C}_{\mathrm{CAM}}(X)$ then
      if $invalidFound = false$ then
        Xlast $\leftarrow last(X)$
      $invalidFound \leftarrow true$
      Yfirst $\leftarrow first(X)$
      if Yfirst > Xlast then
        break {Stopping criterion met}
    else
      $invalidFound \leftarrow false$
    ... perform usual counting ...
```

4.3. Post-counting reduction

At the end of the count and reduce phase, before writing the reduced dataset for the next iteration, we try to repeatedly reduce every transaction t, pulling out singleton items that will never participate to a valid solution itemset. The last item of a transaction t is the best candidate for deletion both when $\mathcal{C}_{\mathrm{CAM}}$ is defined over a prefix decreasing or increasing function. Consider the usual transaction $t = \{a,b,c,d,e,f,g,h\}$ and suppose to be at the end of iteration 3. Before writing t in the dataset for the next iteration we wonder whether h will be useful from now on. If such item is not useful when used together with the items with the best attribute values, then it will be of no use within any other itemset. So we check if $\{abch\}$ satisfy or not $\mathcal{C}_{\mathrm{CAM}}$. If not, we are sure that no 4-itemset containing h and supported by t will satisfy $\mathcal{C}_{\mathrm{CAM}}$ as well. However this is yet not enough to remove h from t. In fact, it could be possible that a larger itemset, for instance $\{abcdh\}$, satisfies $\mathcal{C}_{\mathrm{CAM}}$. Therefore, if k is the current iteration, what we have to check is that h cannot participate to any valid itemset of any size larger than k. This process can be stopped when we rich the size $l = h.count + 1$ which is the maximum possible size of a frequent itemset containing h and supported by t (line 12 of Algorithm 5). In our example suppose that $h.count = 5$: we have only to check $\{abch\}$, $\{abcdh\}$ and $\{abcdeh\}$.

We can tighten the above process, by joining to h only those items which have a sufficient local count. In our example suppose that $c.count = 3$. We can be sure that c will not participate to any solution itemset of size 5 supported by t. Therefore we can skip $\{abcdh\}$ and check $\{abdeh\}$ (if $d.count \geqslant 4$ otherwise also d would be skipped). This process can be early stopped. Suppose f is prefix decreasing, if it happens that an itemset $\{X \cup h\}$ of length l has a value $f(X \cup h)$ smaller than the previous one with length $l - 1$, we are assured that any other itemset $\{X \cup h\}$ with length $> l$ will have a lower value of f, and therefore if no valid itemset subsuming i has not yet been found, we can stop the process. Symmetrically if f is prefix increasing (line 15 of Algorithm 5).

Theorem 4 (Post-counting Reduction). *Given a dataset $\mathcal{D}$ and a convertible anti-monotone constraint $\mathcal{C}_{\mathrm{CAM}}$. Let $\mathcal{D}'$ be the reduced dataset produced by Algorithm 5. It holds that:*

$$\forall X \in Th(\mathcal{C}_{\mathrm{CAM}}), |X| \geqslant k : supp_{\mathcal{D}}(X) = supp_{\mathcal{D}'}(X)$$

Proof. The proof is related to the above three paragraphs explaining the algorithm. Suppose $\mathcal{C}_{\mathrm{CAM}}$ is defined over a prefix decreasing function f (the proof is similar if f is a prefix increasing function), and z is the last element the transaction t sorted by decreasing order of the interesting attribute, and let us denote with t_l the l-prefix of t.

Algorithm 5. Post-counting Reduction

```
Input: $\mathcal{D}, k, \mathcal{C}_{\mathrm{CAM}}$
Output: $\mathcal{D}'$
1:  for all $t \in \mathcal{D}$ do
2:    repeat
3:      $delete \leftarrow false$
4:      $z \leftarrow last(t)$
5:      $l \leftarrow k$
6:      $X \leftarrow take(t,l)$
7:      {takes first $l$ items in $t$ with count $\geqslant l$}
8:      while ($\neg delete$ and $\neg\ \mathcal{C}_{\mathrm{CAM}}(\{X \cup z\})$) do
9:        $old\_f\_val \leftarrow f(\{X \cup z\})$
10:       $l \leftarrow l + 1$
11:       $X \leftarrow take(t,l)$
12:       if $\neg(k \leqslant |X| \leqslant z.\mathrm{count})$ then
13:         $delete \leftarrow true$
14:       else
15:         if $f(\{X \cup z\}) < (>) old\_f\_val$ then
16:           $delete \leftarrow true$
17:     if $delete = true$ then
18:       $t \leftarrow t \backslash z$
19:   until $delete = false$
20:   $\mathcal{D}' = \mathcal{D}' \cup t$
```

Regarding the first part, it is clear that since $X = t_l$ is the l-itemset with the highest value of f, if $f(X \cup z) < v$ then any other $l + 1$-itemset included in t and subsuming z will have an even lower value of f and will not satisfy $\mathcal{C}_{\mathrm{CAM}}$. Thus if $\neg\exists X | f(X \cup z) \geqslant v$ with $|X \cup z| \geqslant l$ where $X = t_i$, then $\neg\exists Y \subseteq t, |Y| > l, z \in Y | f(Y) \geqslant v$ and therefore z can be removed. We can improve the above, by recalling that an item $i \in t$ cannot participate to any itemset of length l if it has a local count less than l. Therefore, for each prefix X of length l, we can take into consideration only those items $i \in t | i.count \geqslant l$.

Finally, if $f(t_i \cup z) < m$ and $f(t_i \cup z) > f(t_{i+1} \cup z)$ then every other itemset $\{t_{j>i+1} \cup z\}$ will have a value of f smaller than m, because following items in t_j have decreasing values. Again, since each $\{t_i \cup z\}$ is the best possible itemset, then there is no itemset $I \in t,\ z \in I | \mathscr{C}_{\mathrm{CAM}}(I)$ and therefore z can be removed. □

4.4. Advanced pruning techniques: experimental analysis

Experimental results presented in this Section confirm that the three proposed advanced pruning strategies bring an additional speed-up when dealing with convertible constraints. In Fig. 4(a) we compared the pruning power of the three proposed strategies with the Loose Anti-Monotone strategy. Only one of the three always performs better than $\mathit{ExAMiner}^{\mathscr{LAM}}$, i.e. *Post-counting Reduction*, and the improvement is of about one order of magnitude. As predictable, the four strategies all together ($\mathit{ExAMiner}^{\mathscr{CAM}}$) perform better than any single one. Such increased pruning power leads to a lower computation time, as shown in Fig. 4(b). $\mathit{ExAMiner}^{\mathscr{LAM}}$ is one order of magnitude faster than $\mathscr{FIC}^{\mathscr{A}}$, while our advanced pruning techniques bring $\mathit{ExAMiner}^{\mathscr{CAM}}$ to be

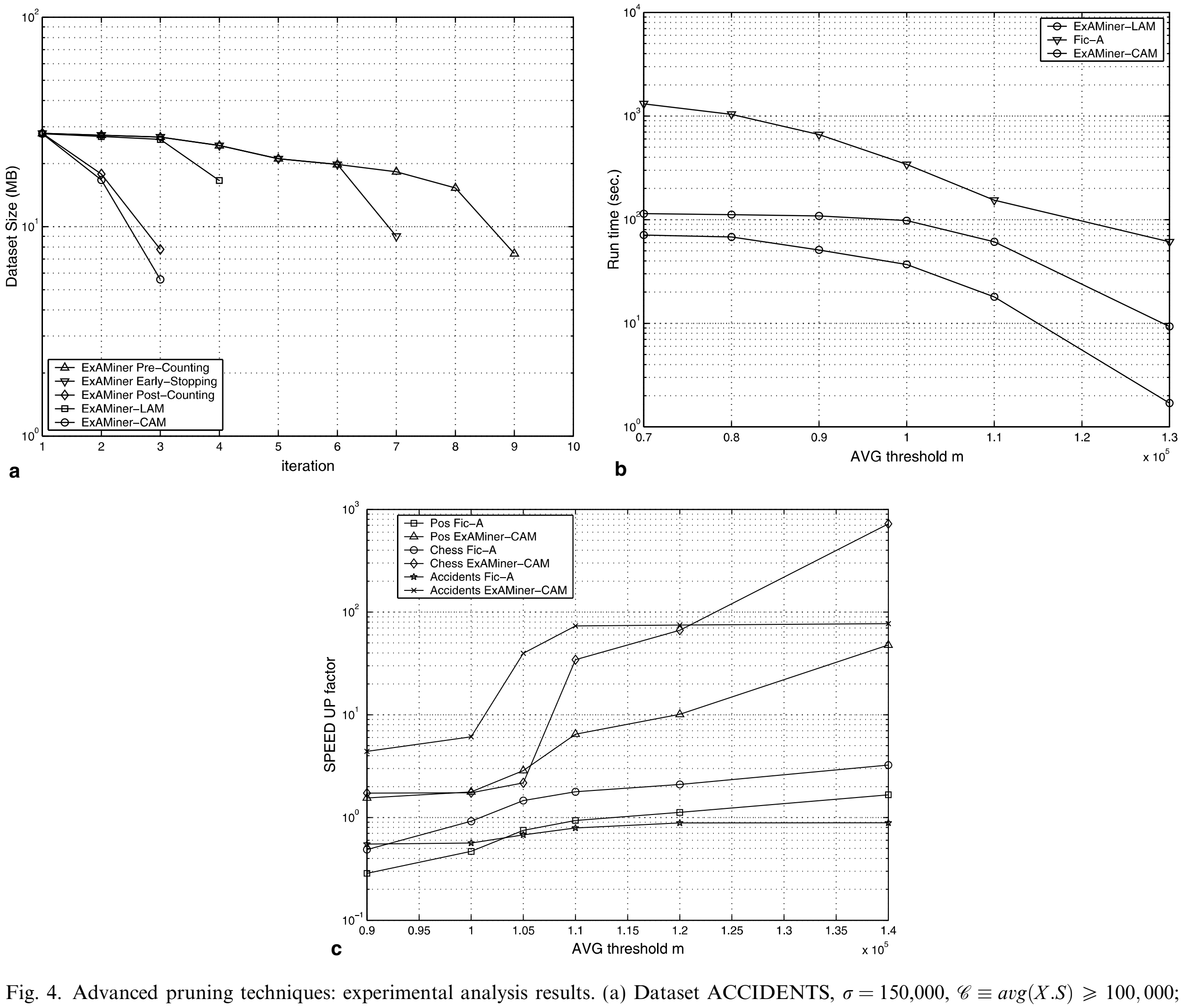

Fig. 4. Advanced pruning techniques: experimental analysis results. (a) Dataset ACCIDENTS, $\sigma = 150{,}000$, $\mathscr{C} \equiv avg(X.S) \geqslant 100{,}000$; (b) dataset BMS-POS, $\sigma = 300$, $\mathscr{C} \equiv avg(X.S) \geqslant m$; (c) various datasets, various σ, $\mathscr{C} \equiv avg(X.S) \geqslant m$.

two orders of magnitude faster than $\mathscr{F}\mathscr{I}\mathscr{C}^{\mathscr{A}}$. Finally, in Fig. 4(c) we plot the speed up factor of every algorithm w.r.t. its own unconstrained version on different kinds of datasets. The figure shows that exploiting data-reduction is fruitful on sparse datasets as well as dense datasets.

5. *ExAMiner*$^{\mathscr{GEN}}$: a generalized unifying framework

The objective of this section is to design a general algorithmic framework, which acts as computational engine of an exploratory pattern discovery system, where the human analyst can impose her own focus and guidance on the discovery process. Of course we want to let the analyst use any of the constraints we have described, but also, any possible conjunction of them: simple constraints are basic building blocks of a powerful and expressive query language. In this paper we have reviewed and characterized five main classes of constraints: *anti-monotone*, *monotone*, *succinct*, *convertible* and *loose anti-monotone*. In Fig. 5 we report some interesting constraints with their properties. Such properties should be used to speed up the underlying mining task and therefore the knowledge extraction process itself. Unluckily, the mining strategies developed by previous works were not compatible to be exploited at the same time. On the contrary, one of the most important advantages of our framework is that, pushing constraints by means of data-reduction in a level-wise framework, we can exploit different properties of constraints all together, and the total benefit is always greater than the sum of the individual benefits. In other words, by means of data-reduction we exploit a real synergy of all constraints that the user defines for the pattern extraction: each constraint does not only play its part in reducing the data, but this reduction in turns strengthens the pruning power of the other constraints. Moreover data-reduction induces a

Constraint	Anti-mon	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

Fig. 5. Classification of commonly used constraints.

pruning of the search space, that in turn strengthens future data reductions. Note that since many constraints fall into more than one class, if we were able to exploit different strategies at the same time, we would gain dramatic performance benefits. In fact, all the properties that we exploit are orthogonal and thus can be combined.

Example 13. The constraint $range(S.A) \geqslant v \equiv max(S.A) - min(S.A) \geqslant v$, is both monotone and loose anti-monotone. Thus, when we mine frequent itemsets which satisfy such constraint we can exploit the benefit of having together, in the same *count&reduce* procedure, the $\mathcal{C}_{\text{freq}}$-based data reductions of Proposition 2, the μ-reduction for monotone constraints (Proposition 1), and the reduction based on $\mathcal{C}_{\text{LAM}}$ (Theorem 1).

Example 14. The constraint $max(S.A) \geqslant v$ is monotone, succinct and loose anti-monotone. This means that we can exploit all these properties by using it as a succinct constraint at candidate generation time as done in Ng et al. [18], and using it as a monotone constraint and as a loose anti-monotone constraint by means of data-reduction at counting time.

In the following we review how the various properties of constraints are exploited within our generalized Apriori-like framework, whose pseudo-code is provided in Algorithm 6. Recall constraints can exhibit more than one property, i.e., they can be in more than one class.

Anti-monotone constraints ($\mathcal{C}_{\text{AM}}$) are exploited in conjunction with the frequency constraint, by not generating as candidate itemsets that have an infrequent subset (line 13);

Succinct constraints ($\mathcal{C}_{\text{S}}$) can be pushed into the computation at generation time, by removing all those invalid itemsets which will not be a subset of a valid itemset (line 13).

Succinct anti-monotone constraints ($\mathcal{C}_{\text{AMS}}$) are exploited at preprocessing time by removing the itemsets which does not satisfy them (line 2). After that the mining process can start without keeping into account that the input dataset was modified, and however we are guaranteed the it will produce all and only valid itemsets.

Monotone constraints ($\mathcal{C}_{\text{M}}$) are exploited directly on the dataset. At any level of the level-wise visit, we can remove transactions that do not satisfy monotone constraints (Proposition 1). This data reduction is implemented by line 3 of Algorithm 7 and lines 11–12 of Algorithm 8.

Algorithm 6. *ExAMiner*$^{\mathcal{GEN}}$

```
Input: $\mathcal{D}, \sigma, \mathcal{C}$ /* where $\mathcal{C} = \mathcal{C}_{\text{AM}} \cup \mathcal{C}_{\text{M}} \cup \mathcal{C}_{\text{S}} \cup \mathcal{C}_{\text{AMS}} \cup \mathcal{C}_{\text{CAM}} \cup \mathcal{C}_{\text{LAM}}$ */
Output: $Th(\mathcal{C}_{\text{freq}[\mathcal{D},\sigma]}) \cap Th(\mathcal{C})$
1: $L_1 \leftarrow \mathcal{I}$
2: $C_1 \leftarrow \{\{i\} | i \in \mathcal{I} \wedge \mathcal{C}_{\text{AMS}}(\{i\}) \wedge \mathcal{C}_{\text{AM}}(\{i\})\}$
3: $\mathcal{D}_1 \leftarrow \pi_{C_1}(\mathcal{D})$
4: $L_1, \mathcal{D}_1 \leftarrow count\_first\_iteration(\mathcal{D}_1, \sigma, C_1, \mathcal{C}_{\text{M}})$
5: while $L_1 \neq C_1$ do
6:    $C_1 \leftarrow L_1$;
7:    $L_1, \mathcal{D}_1 \leftarrow count\_first\_iteration(\mathcal{D}_1, \sigma, C_1, \mathcal{C}_{\text{M}})$
8: $C_2 \leftarrow generate(L_1, \mathcal{C}_{\text{AM}}, \mathcal{C}_{\text{S}})$
9: for all $i \in L_1$ do $V_1[i] \leftarrow 0$
10: $k \leftarrow 2$
11: while $C_k \neq \emptyset$ do
12:    $L_k, \mathcal{D}_{k+1}, V_k \leftarrow count\&reduce^*(\mathcal{D}_k, \sigma, \mathcal{C}_{\text{M}}, \mathcal{C}_{\text{CAM}}, \mathcal{C}_{\text{LAM}}, C_k, V_{k-1})$
13:    $C_{k+1} \leftarrow generate(L_k, \mathcal{C}_{\text{AM}}, \mathcal{C}_{\text{S}})$
14:    $k$++
15: for ($i = 0$; $i \leqslant k$; $i$++) do
16:    for all $X \in L_i$ do
17:      if $\mathcal{C}(X)$ then return $X$
```

Convertible constraints ($\mathscr{C}_{\text{CAM}}$) are pushed by exploiting the loose anti-monotonicity property (Theorem 1). When we are in presence of just one convertible constraint in the given conjunction, we can exploit the advanced pruning techniques described in Section 4, which require a particular reordering of the transactions, and therefore they not always can be applied in presence of multiple convertible constraints.
Loose anti-monotone constraints ($\mathscr{C}_{\text{LAM}}$) are pushed by exploiting the property in Theorem 1. The corresponding data reduction is implemented by lines 33–34 of Algorithm 8.

Algorithm 7. *count_first_iteration*

```
Input: $\mathscr{D}, \sigma, C, \mathscr{C}_{\text{M}}$
Output: $\mathscr{D}_1, L_1$
1: $L_1 \leftarrow \emptyset; \mathscr{D}_1 \leftarrow \emptyset$
2: for all $t \in \mathscr{D}$ do
3:    if $\mathscr{C}_{\text{M}}(t)$ then
4:       for all $i \in t$ do $i.count++$; if $i.count++ = \sigma$ then $L_1 \leftarrow L_1 \cup \{i\}$
5:       $\mathscr{D}_1 \leftarrow \mathscr{D}_1 \cup t$
6: $\mathscr{D}_1 \leftarrow \pi_{L_1}(\mathscr{D}_1)$
```

Let us briefly describe the pseudo-code in Algorithm 6. Lines from 3 to 7 together with procedure *count_first_iteration* (Algorithm 7), implement the ExAnte pre-processing. Lines from 11 to 14 implements the typical central loop of the Apriori algorithm, where the *generate* procedure exploits succinctness and anti-monotonicity to reduce the set of candidates, and the *count&reduce** procedure exploits monotonicity, convertibility and loose anti-monotonicity. Note that the *count&reduce** procedure described in Algorithm 8 is obtained by embedding the loose antimonotonicity based data reduction, and the advanced pruning techniques based on convertibility, into the *count&reduce* procedure described in Algorithm 2. Finally, lines from 15 to 17 implement the post-processing, where possible solution itemsets are check for satisfaction of those constraints for which satisfaction is not already guaranteed.

Algorithm 8. *count&reduce**

```
Input: $\mathscr{D}_k, \sigma, \mathscr{C}_{\text{AM}}, \mathscr{C}_{\text{M}}, \mathscr{C}_{\text{CAM}}, \mathscr{C}_{\text{LAM}}, C_k, V_{k-1}$
Output: $L_k, \mathscr{D}_{k+1}, V_k$
1: for all $i \in \mathscr{I}$ do
2:    $V_k[i] \leftarrow 0$
3: for all tuples $t$ in $\mathscr{D}_k$ do
4:    for all $C \in \mathscr{C}_{\text{LAM}}$ do
5:       $t.lam[C] \leftarrow false$
6:    for all $i \in t$ do
7:       if $V_{k-1}[i] < k - 1$ then
8:          $t \leftarrow t \setminus i$ /* antimonotone pruning */
9:       else
10:         $i.count \leftarrow 0$
11:   if $\neg(\forall C \in \mathscr{C}_{\text{M}} : C(t))$ then
12:      break; /* monotone pruning */
13:   if $\neg\ \mathscr{C}_{\text{CAM}}(prefix(t, k))$ then
14:      break; /* convertible a-m pruning */
15:   for all $X \in C_k$, $X \subseteq t$ do
16:      if Counting Early Stopping Criterion holds then
17:         break;
18:      $X.count$++; $t.count$++
19:      for all $i \in X$ do
```

```
    $i.count{+}{+}$
  **for all** $C \in \mathscr{C}_{\mathrm{LAM}}$ **do**
    $t.lam[C] \leftarrow t.lam[C]$ **OR** $C(X)$
  **if** $X.count = = \sigma$ **then**
    $L_k \leftarrow L_k \cup \{X\}$
    **for all** $i \in X$ **do**
      $V_k[i]{+}{+}$
$t =$ Post Counting Reduction($t$) /* convertible a-m pruning */
**for all** $i \in t$ **do**
  **if** $i.count < k$ **then**
    $t \leftarrow t \backslash i$ /* anti-monotone pruning */
**if** $t.count < k + 1$ **then**
  break; /* anti-monotone pruning */
**if** $\neg$ ($\forall C \in \mathscr{C}_{\mathrm{LAM}} : t.lam[C]$) **then**
  break; /* loose a-m pruning */
**if** $|t| \geqslant k + 1$ **then**
  write $t$ in $\mathscr{D}_{k+1}$
```

6. Conclusion and future work

Constraints in frequent pattern mining play a twofold essential role: they provide the user with guidance on the knowledge discovery process, thus helping in focussing the search on interesting patterns; additionally, they can be pushed in the computation in order to reduce the input data and the search space.

In this paper we have reviewed and extended the state-of-the-art of the constraints that can be pushed in a frequent pattern computation. Many different kinds of constraints are pushed within a general level-wise Apriori-like computation by means of data reduction techniques. The efficiency of the proposed techniques is witnessed by excellent experimental results.

Our framework, by means of data-reduction, exploits a real synergy of all constraints that the user defines for the pattern extraction: each constraint does not only play its part in reducing the data, but this reduction in turns strengthens the pruning power of the other constraints. Moreover data-reduction induces a pruning of the search space, and the pruning of the search space in turn strengthens future data reductions. The orthogonality of the exploited constraint pushing techniques has a twofold benefit: on one hand all the techniques can be amalgamated together achieving a very efficient computation; on the other hand the framework can be easily extended to handle other constraints. Another positive effect of adopting an Apriori-like algorithm, is that in the implementation we can exploit all coding tricks and smart data structures that have been developed in the last decade for the Apriori algorithm.

We believe that this efficient computational framework is a step forward in the road to a realistic pattern discovery systems. We are also aware that many issues remain open, and deserve further research.

- Pattern discovery is usually a highly iterative 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 important to develop techniques for *incremental mining*; i.e., 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 from scratch.
- The exploratory nature of pattern discovery imposes to the system not only to return frequent feedbacks to the user (which is achieved thanks to the efficient computational engine), but also to provide *pattern visualization and navigation tools*. These tools should help the user in visualizing the continuous feedbacks form the systems, allowing an easier and human-based identification of the fragments of interesting knowledge. Such tools should also play the role of graphical querying interface: the action of browsing pattern visualization should be tightly integrated (both by a conceptual and engineering point of view) with the action of iteratively querying.

- The user should be allowed to define her own application-dependent constraints.
- Another important issue is how to integrate *condensed representations* of patterns in the constraint-based mining framework [4].
- Embedding our computational framework within a relational DBMS deserves a great effort: this issue is strictly connected with many other open problems, for instance, how to store and index pattern discovery queries results;
- Finally, we must develop a constraint-based mining framework for more complex kinds of patterns such as sequences and graphs.

Our objective at Pisa KDD Laboratory, is to integrate the results of these investigations in a unified system for exploratory constraint-based pattern discovery. A first prototype of such a system, named ConQueSt [6], has been developed around the efficient mining engine described in this article.

References

[1] 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 1994, Santiago de Chile, Chile, 1994.

[2] J. Besson, C. Robardet, J.F. Boulicaut, S. Rome, Constraint-based concept mining and its application to microarray data analysis, Intelligent Data Analysis Journal 9 (1) (2005) 59–82.

[3] F. Bonchi, B. Goethals, FP-Bonsai: the art of growing and pruning small FP-trees, in: Proceedings of Advances in Knowledge Discovery and Data Mining, 8th Pacific-Asia Conference, PAKDD 2004, Sydney, Australia, 2004.

[4] F. Bonchi, C. Lucchese, On closed constrained frequent pattern mining, in: Proceedings of the Third IEEE International Conference on Data Mining, ICDM 2004, Brighton, UK, 2004.

[5] F. Bonchi, C. Lucchese, Pushing tougher constraints in frequent pattern mining, in: Proceedings of Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, 2005.

[6] F. Bonchi, F.Giannotti, C. Lucchese, S. Orlando, R. Perego, R. Trasarti, ConQueSt: a constraint-based querying system for exploratory pattern discovery, in: Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, Atlanta, Georgia, USA (demo paper), 2006.

[7] 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 2003, Cavtat-Dubrovnik, Croatia, 2003.

[8] 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 2003, Cavtat-Dubrovnik, Croatia, 2003.

[9] 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 2003, Melbourne, Florida, USA, 2003.

[10] J.F. Boulicaut, B. Jeudy, Optimization of association rule mining queries, Intelligent Data Analysis Journal 6 (4) (2002) 341–357.

[11] C. Bucila, J. Gehrke, D. Kifer, W. White, DualMiner: A dual-pruning algorithm for itemsets with constraints, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2002, Edmonton, Alberta, Canada, 2002.

[12] L. De Raedt, S. Kramer, The levelwise version space algorithm and its application to molecular fragment finding, in: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, 2001.

[13] G. Grahne, L. Lakshmanan, X. Wang, Efficient mining of constrained correlated sets, in: Proceedings of the 16th IEEE International Conference on Data Engineering, ICDE 2000, San Diego, California, USA, 2000.

[14] J. Han, L. Lakshmanan, R. Ng, Constraint-based, multidimensional data mining, Computer 32 (8) (1999) 46–50.

[15] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceedings of 2000 ACM International Conference on Management of Data, SIGMOD 2000, Dallas, Texas, USA, 2000.

[16] D. Kifer, J. Gehrke, C. Bucila, Q. White, How to quickly find a witness, in: Proceedings of 2003 ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2003, San Diego, CA, USA, 2003.

[17] L. Lakshmanan, R. Ng, J. Han, A. Pang, Optimization of constrained frequent set queries with 2-variable constraints, in: Proceedings of 1999 ACM International Conference on Management of Data, SIGMOD 1999, Philadelphia, Pennsylvania, USA, 1999, pp. 157–168.

[18] R. Ng, L. Lakshmanan, J. Han, A. Pang, Exploratory mining and pruning optimizations of constrained associations rules, in: Proceedings of 1998 ACM International Conference on Management of Data, SIGMOD 1998, Seattle, Washington, USA, 1998.

[19] C. Ordonez et al., Mining constrained association rules to predict heart disease, in: Proceedings of the First IEEE International Conference on Data Mining, December 2001, San Jose, California, USA, 2001, pp. 433–440.

[20] J, Pei, J. Han, Can we push more constraints into frequent pattern mining? in: Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2000, Boston, MA, USA, 2000.

[21] J. Pei, J. Han, L. Lakshmanan, Mining frequent item sets with convertible constraints, in: Proceedings of the 17th IEEE International Conference on Data Engineering, ICDE 2001, Heidelberg, Germany, 2001.

[22] R. Srikant, Q. Vu, R. Agrawal, Mining association rules with item constraints, in: Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 1997, Newport Beach, California, USA, 1997.

Francesco Bonchi received his Ph.D. in computer science from University of Pisa in December 2003, with a thesis entitled "Frequent Pattern Queries: Language and Optimizations". Currently he is a post-doc at Institute of Information Science and Technologies (ISTI) of the Italian National Research Council in Pisa where he is a member of the Knowledge Discovery and Delivery Laboratory. He has been a visiting fellow at the Kanwal Rekhi School of Information Technology, Indian Institute of Technology, Bombay (2000, 2001). His current research interests are data mining query language and optimization, frequent pattern mining, privacy-preserving data mining, bioinformatics. He is one of the teachers for a course on data mining held at the faculty of Economics at the University of Pisa. He served in the Program Committee of major data mining conferences such as ICDM and PKDD, and he has been co-chair of The Fourth International Workshop on Knowledge Discovery in Inductive Databases (KDID'05).

Claudio Lucchese received summa cum laude the master degree in Computer Science from the Ca' Foscari University of Venice in October 2003. In 2004 he held a fellowship at CNR in Pisa, the italian Council of National Research. He currently is a Ph.D. student in Computer Science at the Ca' Foscari University of Venice, and a research associate at CNR. He is mainly interested in data mining, privacy-preserving data mining and data mining techniques for information retrieval.