Pushing Tougher Constraints in Frequent Pattern Mining

Francesco Bonchi* Claudio Lucchese† Roberto Trasarti*

Abstract

In this paper we extend the state-of-art of the constraints that can be pushed in a frequent pattern computation. We introduce a new class of tough constraints, namely *Loose Anti-monotone* constraints, and we deeply characterize them by showing that they are a superclass of convertible anti-monotone constraints (e.g. constraints on *average* or *median*) and that they model tougher constraints (e.g. *variance* or *standard deviation*) which have never been studied before. Then we show how these constraints can be exploited in a level-wise Apriori-like computation by means of new data-reduction strategies, outperforming previous algorithms for tough constraints, and exploiting much tougher ones with the same effectiveness.

Keywords: Frequent Pattern Mining, Constraints Pushing, Data Reduction.

1 Introduction

Frequent itemsets play an essential role in many data mining tasks that try to find interesting patterns from databases, such as association rules, correlations, sequences, episodes, classifiers, clusters and many more. Although the collection of all frequent itemsets is typically very large, the subset that is really interesting for the user usually contains only a small number of itemsets. This situation is harmful for two reasons. First, performance degrades: mining generally becomes inefficient or, sometimes, simply unfeasible. Second, the identification of the fragments of interesting knowledge, blurred within a huge quantity of mostly useless patterns, is difficult.

Therefore, the paradigm of constraint-based mining was introduced. Constraints provide focus on the interesting knowledge, thus reducing the number of patterns extracted to those of potential interest. Additionally, they can be pushed deep inside the pattern discovery algorithm in order to achieve better performance [6, 7, 10, 11, 12, 13, 14].

Constrained frequent pattern mining is defined as follows. Let $\mathcal{I} = \{x_1, ..., x_n\}$ be a set of distinct literals, usually called *items*, where an item is an object with some predefined attributes (e.g., price, type, etc.). An *itemset* X is a non-empty subset of $\mathcal{I}$. If $|X| = k$ then X is called a *k-itemset*.

A constraint on itemsets is a function $\mathcal{C} : 2^{\mathcal{I}} \rightarrow \{true, false\}$. We say that an itemset I satisfies a constraint if and only if $\mathcal{C}(I) = true$. We define the *theory* of a constraint as the set of itemsets which satisfy the constraint: $Th(\mathcal{C}) = \{X \in 2^{\mathcal{I}} \mid \mathcal{C}(X)\}$. A *transaction database* $\mathcal{D}$ is a bag of itemsets $t \in 2^{\mathcal{I}}$, usually called *transactions*. The *support* of an itemset X in database $\mathcal{D}$, denoted $supp_{\mathcal{D}}(X)$, is the number of transactions which are superset of X. Given a user-defined *minimum support* σ, an itemset X is called *frequent* in $\mathcal{D}$ if $supp_{\mathcal{D}}(X) \geq \sigma$. This defines the minimum frequency constraint: $\mathcal{C}_{freq[\mathcal{D},\sigma]}(X) \Leftrightarrow supp_{\mathcal{D}}(X) \geq \sigma$. When the dataset and the minimum support threshold are clear from the context, we indicate the frequency constraint simply $\mathcal{C}_{freq}$. Thus with this notation, the *frequent itemsets mining problem* requires to compute the set of all frequent itemsets $Th(\mathcal{C}_{freq})$.

In general, given a conjunction of constraints $\mathcal{C}$ the *constrained frequent itemsets mining problem* requires to compute $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$.

Constrained frequent pattern mining can be seen as a query optimization problem: given a mining query $\mathcal{Q}$ containing a set of constraints $\mathcal{C}$, provide an efficient evaluation strategy for $\mathcal{Q}$ which is sound and complete (i.e. it finds all and only itemsets in $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$). A naïve solution to such a problem is to first find all frequent patterns ($Th(\mathcal{C}_{freq})$) and then test them for constraints satisfaction. However more efficient solutions can be found by analyzing the property of constraints comprehensively, and exploiting such properties in order to push constraints in the frequent pattern computation.

Following this methodology, some classes of constraints which exhibit nice properties have been individuated [11] (e.g. monotonicity, anti-monotonicity, succinctness). The toughest class of constraints studied so far, is the class of *convertible* constraints [12, 13]: 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. $avg(X.price) \leq v$): subsets (or supersets) of a valid itemset could well be invalid and vice versa. But, if we arrange the items

*Pisa KDD Laboratory, ISTI-C.N.R. Italy.
†Pisa HPC Laboratory, ISTI-C.N.R. Italy.
e-mail: firstname.familyname@isti.cnr.it

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 [12, 13] it is shown that, since the FP-growth [8] approach 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 of [12, 13] 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, in this paper we show how it is possible to push tough constraints within a level-wise computation by means of data reduction techniques. The proposed algorithmic framework outperforms the previous proposals for pushing tough constraints, and it is also able to push tougher constraints never studied before.

Paper Contribution and Organization The contribution of this paper is twofold. On one hand we extend the actual state-of-art classification of constraints that can be pushed in a frequent pattern computation, by showing how to push tough constraints as those ones based on *variance* or *standard deviation*. On the other hand we show how it is possible to push convertible constraints in a level-wise Apriori-like computation, outperforming previously proposed FP-growth based algorithms [12, 13].

1. In Section 2, as a side contribution, we provide an exhaustive state-of-art of constraint pushing techniques.
2. In Section 3 we show that interesting and meaningful constraints do not fall in any of the previously identified classes of constraints, and neither can be pushed by previous algorithms.
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*) which have never been studied before.
4. We identify an interesting property of Loose Anti-monotone constraints which allows input data reduction.
5. In Section 4 we extend *ExAMiner* [2], which is a level-wise Apriori-like algorithmic framework based on data-reduction techniques, in order to exploit loose anti-monotonicity. The resulting algorithm is named $ExAMiner^{\mathcal{LAM}}$.
6. A thorough experimental study is performed in Section 5. It confirms that by exploiting loose anti-monotonicity, $ExAMiner^{\mathcal{LAM}}$ is able to outperform previous algorithms for convertible constraints, and to treat much tougher constraints with the same effectiveness of easier ones.
7. Finally, in Section 6 we develop advanced pruning techniques which can be adopted in the case of convertible constraints. The proposed techniques further improve our algorithm's performance.

2 Related Work and Constraints Classification

In this Section, by reviewing all basic works on the constrained frequent itemsets mining problem, we recall a classification of constraints and their properties.

Anti-monotone and Succinct Constraints A first work defining classes of constraints which exhibit nice properties is [11]. In that paper is introduced an Apriori-like algorithm, named CAP, which exploits two properties of constraints, namely *anti-monotonicity* and *succinctness*, in order to reduce the frequent itemsets computation. Four classes of constraints, each one with its own associated computational strategy, are defined:

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

Definition 1. (Anti-monotone constraint) *Given an itemset X, a constraint $\mathcal{C}_{AM}$ is anti-monotone if* $\forall Y \subseteq X : \mathcal{C}_{AM}(X) \Rightarrow \mathcal{C}_{AM}(Y)$.

The frequency constraint is the most known example of a $\mathcal{C}_{AM}$ constraint. This property, *the anti-monotonicity of frequency*, is used by the Apriori [1] algorithm with the following heuristic: if an itemset X does not satisfy $\mathcal{C}_{freq}$, then no superset of X can satisfy $\mathcal{C}_{freq}$, and hence they can be pruned. This pruning can affect a large part of the search space, since itemsets form a lattice. Therefore the Apriori algorithm operates in a level-wise fashion moving bottom-up on the itemset lattice, and each time it finds an infrequent itemset it prunes away all its supersets. Other $\mathcal{C}_{AM}$ constraints can easily be pushed deeply down into the frequent itemsets mining computation since they behave exactly as $\mathcal{C}_{freq}$: if they are not satisfiable at an early level (small itemsets), they have no hope of becoming satisfiable later (larger itemsets). Conjoining other $\mathcal{C}_{AM}$ constraints to $\mathcal{C}_{freq}$ we just obtain a more selective anti-monotone constraint.

A succinct constraint $\mathcal{C}_S$ is such that, whether an itemset X satisfies it or not, can be determined based on the singleton items which are in X. Informally, given A_1, the set of singleton items satisfying a succinct constraint $\mathcal{C}_S$, then any set X satisfying $\mathcal{C}_S$ is based on A_1 , i.e. X contains a subset belonging to A_1 (for the formal definition of succinct constraints see [11]). A $\mathcal{C}_S$ constraint is *pre-counting pushable*, i.e. it can be satisfied at candidate-generation time: these constraints are pushed in the level-wise computation by substituting the usual *generate_apriori* procedure, with the proper (w.r.t. $\mathcal{C}_S$) candidate generation procedure.

Constraints that are both anti-monotone and succinct can be pushed completely in the level-wise computation before it starts (at pre-processing time). For instance, consider the constraint $min(S.price) \geq v$: if we start with the first set of candidates formed by all singleton items having price greater than v, during the computation we will generate only itemsets satisfying the given constraint.

Constraints that are neither succinct nor anti-monotone are pushed in the CAP [11] computation by inducing weaker constraints which are either anti-monotone and/or succinct.

Monotone Constraints and the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ Tradeoff
Monotone constraints work the opposite way of anti-monotone constraints.

Definition 2. (Monotone constraint) *Given an itemset X, a constraint $\mathcal{C}_M$ is monotone if: $\forall Y \supseteq X : \mathcal{C}_M(X) \Rightarrow \mathcal{C}_M(Y)$.*

Since the frequent itemset computation is geared on $\mathcal{C}_{freq}$, which is anti-monotone, $\mathcal{C}_M$ constraints have been considered more hard to be pushed in the computation and less effective in pruning the search space. Many works [5, 4, 9] have studied the computational problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$ focussing on its search space, and facing the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *tradeoff*. such tradeoff can be described as follows. Suppose that an itemset has been removed from the search space because it does not satisfy a monotone constraint. This pruning avoids checking support for this itemset, but on the other hand, if we check its support and find it smaller than the frequency threshold, we may prune away all the supersets of this itemset. In other words, by monotone pruning we risk to lose anti-monotone pruning opportunities given by the pruned itemset. The tradeoff is clear: pushing monotone constraint can save frequency tests, however the results of these tests could have lead to more effective anti-monotone pruning.

The ExAnte Property and the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ Synergy
In [3] a completely new approach to exploit monotone constraints by means of data-reduction is introduced. The *ExAnte Property* [3] is obtained by shifting attention from the pattern search space to the input data. Indeed, the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *tradeoff* exists only if we focus exclusively on the search space of the problem, while if exploited properly, monotone constraints can reduce dramatically the data in input, in turn strengthening the anti-monotonicity pruning power. With data reduction techniques we exploit the effectiveness of a $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *synergy*.

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

Proposition 2.1. (ExAnte property [3]) *Given a transaction database $\mathcal{D}$ and a conjunction of monotone constraints $\mathcal{C}_M$, we define the μ-reduction of $\mathcal{D}$ as the dataset resulting from pruning the transactions that do not satisfy $\mathcal{C}_M$: $\mu_{\mathcal{C}_M}(\mathcal{D}) = \{t \in \mathcal{D} \mid t \in Th(\mathcal{C}_M)\}$.*
It holds that this data reduction does not affect the support of solution itemsets:

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

A major consequence of reducing the input database in this way is that it implicitly reduces the support of a large amount of itemsets that do not satisfy $\mathcal{C}_M$ as well, resulting in a reduced number of candidate itemsets generated during the mining algorithm. Even a small reduction in the database can cause a huge cut in the search space, because all supersets of infrequent itemsets are pruned from the search space as well. In other words, monotonicity-based data-reduction of transactions strengthens the anti-monotonicity-based pruning of the search space.

This is not the whole story, in fact, infrequent singleton items can not only be removed from the search space together with all their supersets, for the same anti-monotonicity property they 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 has got another positive effect: reducing the size of a transaction which satisfies $\mathcal{C}_M$ can make the transaction violate it. Therefore a growing number of transactions which do not satisfy $\mathcal{C}_M$ can be found. Obviously, we are inside a loop where two different kinds of pruning (α and μ) cooperate to reduce the search space and the input dataset, strengthening each other step by step until no more pruning is possible (a fix-point has been reached). This is the key idea of the ExAnte pre-processing method [3]. 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.

Convertible Constraints In [12, 13] the class of convertible constraints is introduced, and an FP-growth based methodology to push such constraints is proposed.

DEFINITION 3. (CONVERTIBLE CONSTRAINTS) *A constraint $\mathcal{C}_{CAM}$ is convertible anti-monotone provided there is an order $\mathcal{R}$ on items such that whenever an itemset X satisfies $\mathcal{C}_{CAM}$, so does any prefix of X. A constraint $\mathcal{C}_{CM}$ is convertible monotone provided there is an order $\mathcal{R}$ on items such that whenever an itemset X violates $\mathcal{C}_{CM}$, so does any prefix of X.*

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

EXAMPLE 1. (*avg* CONSTRAINT IS CONVERTIBLE) *Let $\mathcal{R}$ be the value-descending order. It is straightforward to see that avg is a prefix decreasing function w.r.t. $\mathcal{R}$. This means that $avg(X) \geq v$ is a $\mathcal{C}_{CAM}$ constraint and $avg(X) \leq v$ is $\mathcal{C}_{CM}$ 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 $avg(S) \geq v$ can be shown convertible monotone, and $avg(S) \leq v$ convertible anti-monotone. Constraints which exhibit this interesting property of being convertible in both a monotone or an anti-monotone constraints, are called *strongly convertible*.

In [12, 13], two FP-growth based algorithms are introduced: $\mathcal{FIC}^{\mathcal{A}}$ to mine $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_{CAM})$, and $\mathcal{FIC}^{\mathcal{M}}$ to mine $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_{CM})$.

A major limitation of any FP-growth based algorithm is that the initial database (internally compressed in the prefix-tree structure) and all intermediate projected databases must fit into main memory. If this requirement cannot be met, these approaches can simply not be applied anymore. This problem is even harder with $\mathcal{FIC}^{\mathcal{A}}$ and $\mathcal{FIC}^{\mathcal{M}}$: in fact, using an order on items different from the frequency-based one, makes the prefix-tree lose its compressing power. Thus we have to manage much greater data structures, requiring a lot more main memory which might not be available. This fact is confirmed by our experimental analysis reported in Section 5: 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 Example 4).

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 can not be considered a constraint-pushing technique, since it generates the complete set of frequent itemsets, no matter whether they satisfy or not $\mathcal{C}_{CM}$. 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 $\mathcal{C}_{CM}$: once a frequent itemset satisfies $\mathcal{C}_{CM}$, all frequent itemsets having it as a prefix also are guaranteed to satisfy the constraint.

3 Loose Anti-monotone Constraints

In this Section we introduce a new class of tougher constraints, which is a proper superclass of convertible anti-monotone.

EXAMPLE 2. (*var* CONSTRAINT IS NOT CONVERTIBLE) *Calculating the variance is an important task of many statistical analysis: it is a measure of how spread out a distribution is. The variance of a set of number X is defined as:*

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

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

Following a similar reasoning we can show that other interesting constraints, such as for instance those ones based on *standard deviation* (*std*) or *unbiased*

Constraint	Anti-monotone	Monotone	Succinct	Convertible	$\mathcal{C}^l_{LAM}$
$min(S.A) \geq v$	yes	no	yes	strongly	$l = 1$
$min(S.A) \leq v$	no	yes	yes	strongly	$l = 1$
$max(S.A) \geq v$	no	yes	yes	strongly	$l = 1$
$max(S.A) \leq v$	yes	no	yes	strongly	$l = 1$
$count(S) \leq v$	yes	no	weakly	$\mathcal{A}$	$l = 1$
$count(S) \geq v$	no	yes	weakly	$\mathcal{M}$	$l = v$
$sum(S.A) \leq v\ (\forall i \in S, i.A \geq 0)$	yes	no	no	$\mathcal{A}$	$l = 1$
$sum(S.A) \geq v\ (\forall i \in S, i.A \geq 0)$	no	yes	no	$\mathcal{M}$	no
$sum(S.A) \leq v\ (v \geq 0, \forall i \in S, i.A\theta 0)$	no	no	no	$\mathcal{A}$	$l = 1$
$sum(S.A) \geq v\ (v \geq 0, \forall i \in S, i.A\theta 0)$	no	no	no	$\mathcal{M}$	no
$sum(S.A) \leq v\ (v \leq 0, \forall i \in S, i.A\theta 0)$	no	no	no	$\mathcal{M}$	no
$sum(S.A) \geq v\ (v \leq 0, \forall i \in S, i.A\theta 0)$	no	no	no	$\mathcal{A}$	$l = 1$
$range(S.A) \leq v$	yes	no	no	strongly	$l = 1$
$range(S.A) \geq v$	no	yes	no	strongly	$l = 2$
$avg(S.A)\theta v$	no	no	no	strongly	$l = 1$
$median(S.A)\theta v$	no	no	no	strongly	$l = 1$
$var(S.A) \geq v$	no	no	no	no	$l = 2$
$var(S.A) \leq v$	no	no	no	no	$l = 1$
$std(S.A) \geq v$	no	no	no	no	$l = 2$
$std(S.A) \leq v$	no	no	no	no	$l = 1$
$var_{N-1}(S.A)\theta v$	no	no	no	no	$l = 2$
$md(S.A) \geq v$	no	no	no	no	$l = 2$
$md(S.A) \leq v$	no	no	no	no	$l = 1$

Table 1: Classification of commonly used constraints (where $\theta \in \{\geq, \leq\}$, k denotes itemsets cardinality).

variance estimator (var_{N-1}) or *mean deviation* (md), are not convertible as well.

The above example shows that such interesting constraints cannot be exploited within a prefix pattern framework. Luckily, as we show in the following, all these constraints share a nice property that we name *"Loose Anti-monotonicity"*.

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 $k-1$.

Moreover, since some of these interesting constraints are not defined or not interesting on some itemsets in $2^{\mathcal{I}}$, we add to our definition the minimum size of the itemset from which the constraint makes sense. Doing so, we rid the definition of details and special cases without any loss of generality.

Definition 4. (Loose Anti-monotone constraint) *Given an itemset X, a constraint is loose anti-monotone from size $l > 0$ (denoted $\mathcal{C}^l_{LAM}$) if:*

$$(|X| > l \ \wedge \ \mathcal{C}_{LAM}(X)) \Rightarrow \exists i \in X \ : \ \mathcal{C}_{LAM}(X \setminus \{i\})$$

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

Proposition 3.1. *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 3. *We show that the constraint $var(X.A) \leq v$ is a $\mathcal{C}^1_{LAM}$ constraint. Given an itemset X, if it satisfies the constraint so trivially does $X \setminus \{i\}$, where i is the element of X which has associated a value of A which is the most far away from $avg(X.A)$. In fact, we have that $var(\{X \setminus \{i\}\}.A) \leq var(X.A) \leq v$, until $|X| > 1$, i.e. until $var(X \setminus \{i\})$ is defined.*

Taking the element of X which has associated a value of A which is the closest to $avg(X.A)$ we can show that $var(X.A) \geq v$ is a $\mathcal{C}^2_{LAM}$ constraint. In this case we have that the constraint is loose anti-monotone from size 2 because the variance of a singleton item is zero.

Since the standard deviation std is the square root of the variance, it is straightforward to see that $std(X.A) \leq v$ is $\mathcal{C}^1_{LAM}$, and $std(X.A) \geq v$ is $\mathcal{C}^2_{LAM}$. The mean deviation is defined as: $md(X) = (\sum_{i \in X} |i - avg(X)|) \ / \ |X|$. Once again, we have that $md(X.A) \leq v$ is $\mathcal{C}^1_{LAM}$, and $md(X.A) \geq v$ is $\mathcal{C}^2_{LAM}$. 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. In particular, they are $\mathcal{C}^2_{LAM}$ since they are not defined for singleton items.

In Table 1 and Figure 1 we update the state-of-art classification of commonly used constraints.

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

Theorem 3.1. *Given a transaction database $\mathcal{D}$, a minimum support threshold σ, and a $\mathcal{C}^l_{LAM}$ constraint, at the iteration $k > l$ of the level-wise computation, a transaction $t \in \mathcal{D}$ such that:*

$$\nexists X \subseteq t, |X| = k, X \in \mathit{Th}(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \cap \mathit{Th}(\mathcal{C}^l_{LAM})$$

can be pruned away from $\mathcal{D}$, since it will never be superset of any solution itemsets of cardinality $> k$.

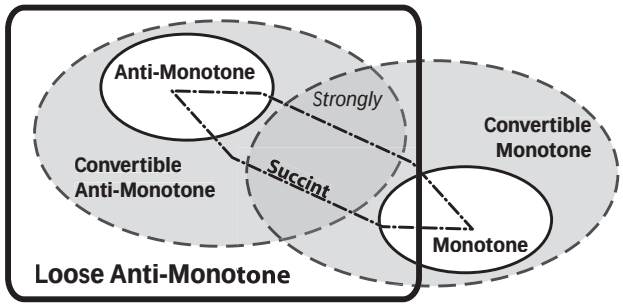


Figure 1: Characterization of the classes of commonly used constraints.

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

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

4 The $ExAMiner^{\mathcal{LAM}}$ Algorithm

The recently introduced algorithm ExAMiner [2], aimed at solving the problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$ (conjunction of anti-monotonicity and monotonicity), generalizes the ExAnte idea to reduce the problem dimensions at all levels of a level-wise Apriori-like computation. In this way, the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *synergy* is effectively exploited at each iteration of the mining algorithm, and not only at pre-processing as done by ExAnte, resulting in significant performance improvements.

The idea is to generalize ExAnte's α-reduction from singletons level to the generic level k. This generalization results in the following set of data reduction techniques, which are based on the anti-monotonicity of $\mathcal{C}_{freq}$ (see [2] for the proof of correctness).

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

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

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

In ExAMiner [2] these data reductions were coupled with the μ-reduction for $\mathcal{C}_M$ constraints as described in Proposition 2.1. Here, in order to cope with the mining problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_{LAM})$, we couple the same set of $\mathcal{C}_{freq}$-based data reduction techniques with the $\mathcal{C}_{LAM}$-based data reduction technique described in Theorem 3.1. The resulting algorithm is named $ExAMiner^{\mathcal{LAM}}$.

Essentially $ExAMiner^{\mathcal{LAM}}$ is an Apriori-like algorithm, which at each iteration $k-1$ produces a reduced dataset $\mathcal{D}_k$ to be used at the subsequent iteration k. Each transaction in $\mathcal{D}_k$, before participating to the support count of candidate itemsets, is reduced as much as possible by means of $\mathcal{C}_{freq}$-based data reduction, and only if it survives to this phase, it is effectively used in the counting phase. Each transaction which arrives to the counting phase, is then tested against the $\mathcal{C}_{LAM}$ property of Theorem 3.1, and reduced again as much as possible, and only if it survives to this second set of reductions, it is written to the transaction database for the next iteration $\mathcal{D}_{k+1}$.

The procedure we have just described, is named $count\&reduce^{\mathcal{LAM}}$, and substitutes the usual support counting procedure of the Apriori algorithm. Therefore to illustrate the $ExAMiner^{\mathcal{LAM}}$ algorithm we just provide the pseudo-code of the $count\&reduce^{\mathcal{LAM}}$ procedure (Algorithm 1), avoiding to provide the well-known Apriori algorithm pseudo-code [1].

We just highlight the we adopt the usual notation of the Apriori pseudo-code:

- C_k: to denote the set of *candidate* itemsets at iteration k;
- L_k: to denote the set of *frequent* (or large) itemsets at iteration k;

In the pseudo-code of $ExAMiner^{\mathcal{LAM}}$, in order to implement the data-reduction $\mathcal{G}_k(i)$ we use an array of integers V_k (of the size of $Items$), which records for each item the number of frequent k-itemsets in which it appears. This information is then exploited during the subsequent iteration $k + 1$ for the global pruning of items from all transaction in $\mathcal{D}_{k+1}$ (lines 5 and 6 of the pseudo-code). On the contrary, data reductions $\mathcal{T}_k(t)$ and $\mathcal{L}_k(i)$ are put into effect during the same iteration in which the information is collected. Unfortunately, they require information (the frequent itemsets of cardinality k) that is available only at the end of the actual counting (when all transactions have been used). However, since the set of frequent k-itemsets is a subset of the set of candidates C_k, we can use such data reductions in a relaxed version: we just check the number of candidate itemsets X which are subset of t ($t.count$ in the pseudo-code, lines 10 and 18)

Algorithm 1 *count&reduce*$^{\mathcal{LAM}}$

Input: $\mathcal{D}_k, \sigma, \mathcal{C}^l_{LAM}, \mathcal{C}_M, C_k, V_{k-1}$

```
1: forall i ∈ I do V_k[i] ← 0
2: forall tuples t in D_k do
3:    if k < l then t.lam ← true
4:      else t.lam ← false
5:    forall i ∈ t do if V_{k-1}[i] < k − 1
6:      then t ← t \ i
7:      else i.count ← 0
8:    if |t| ≥ k and C_M(t) then
9:      forall X ∈ C_k, X ⊆ t do
10:       X.count++; t.count++
11:       if ¬t.lam and C_LAM(X) then
12:         t.lam ← true
13:       forall i ∈ X do i.count++
14:       if X.count = σ then
15:         L_k ← L_k ∪ {X}
16:         forall i ∈ X do V_k[i] + +
17:     if t.lam then
18:       if |t| ≥ k + 1 and t.count ≥ k + 1 then
19:         forall i ∈ t if i.count < k
20:           then t ← t \ i
21:         if |t| ≥ k + 1 and C_M(t) then
22:           write t in D_{k+1}
```

and which are superset of i (*i.count* in the pseudo-code, lines 13 and 19). Analogously, the data reduction based on loose anti-monotonicity described in Theorem 3.1, is exploited in the same relaxed version with candidates instead of frequent itemsets. In the pseudo-code we have a flag *t.lam* which is set to *true* as soon as an itemset $X \in C_k$, such that $X \subseteq t, X \in Th(\mathcal{C}_{LAM})$, is found (lines 12 and 17). A transaction which has the flag *t.lam* set to false after being used in the support counting, will not enter in the database for the next iteration $\mathcal{D}_{k+1}$ (line 17 of the pseudo-code). In fact, such a transaction has not covered any candidate itemset which satisfy $\mathcal{C}_{LAM}$, and thus according to Theorem 3.1 can be removed from the database.

Discussion Note that in the pseudo-code of Algorithm 1 we pass to the procedure both a set of $\mathcal{C}_{LAM}$ and a set of $\mathcal{C}_M$ constraints: obviously if the set of $\mathcal{C}_{LAM}$ constraints is empty we obtain the standard ExAMiner *count&reduce* (no $\mathcal{C}_{LAM}$ data reduction); while if we have an empty set of $\mathcal{C}_M$ constraints, the $\mathcal{C}_M$ testing (lines 8 and 21 of the pseudo code) always succeed and thus the μ-reduction is never applied. Whenever we have both $\mathcal{C}_M$ and $\mathcal{C}_{LAM}$ constraints (i.e. a query corresponding to the mining problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M) \cap Th(\mathcal{C}_{LAM})$) we can benefit of all the data-reduction techniques together, obtaining a stronger synergy.

This is the main advantage of our methodology: 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.

EXAMPLE 4. *The constraint* $range(S.A) \geq v \equiv max(S.A) - min(S.A) \geq v$, *is both* $\mathcal{C}_M$ *and* $\mathcal{C}^2_{LAM}$. *Thus, when we mine frequent itemsets which satisfy such constraint we can exploit the benefit of having together, in the same count&reduce*$^{\mathcal{LAM}}$ *procedure, the* $\mathcal{C}_{freq}$*-based data reductions,* μ*-reduction, and reduction based on* $\mathcal{C}_{LAM}$. *Consider now the constraint* $max(S.A) \geq v$. *This constraint is* $\mathcal{C}_M$, $\mathcal{C}_S$ *and* $\mathcal{C}^1_{LAM}$. *This means that we can exploit all these properties by using it as a succinct constraint at candidate generation time as done in [11], and using it as a monotone constraint and as a loose anti-monotone constraint by means of data-reduction at counting time.*

Run Through Example In Figure 2(b) we have a transactional dataset and an associated *item-price* table in Figure 2(a). Suppose that we want $ExAMiner^{\mathcal{LAM}}$ to mine frequent itemsets (minimum support $\sigma = 3$) having a small (≤ 10) variance of prices. In the following with R_k we denote the set of solution itemsets of size k.

During the first iteration no pruning is possible. We just count the support of singleton items using all transactions in the dataset. At the end of the first iteration we discover that items f and h are infrequent: this information will be used during the second iteration, in particular during the *count&reduce*$^{\mathcal{LAM}}$ phase when this two items will be deleted transaction by transaction (lines 5 and 6 of the pseudo-code in Algorithm 1). This data reduction is the ExAMiner's $\mathcal{G}_k(i)$ anti-monotone reduction, which at level $k = 1$ (singleton items) corresponds to the ExAnte's α-reduction. All the other singleton items are frequent and, since the variance of a singleton is zero they all satisfy the $\mathcal{C}_{LAM}$ constraint, and thus they are all solutions to the given mining problem: $R_1 = \{a, b, c, d, e, g, i, j\}$.

During the second iteration no other $\mathcal{C}_{freq}$-based pruning can be performed. But luckily we can exploit the loose anti-monotonicity of the *var* constraint to remove completely some transactions. Consider, for instance, transaction 4: it is superset of 3 candidate itemsets $\{ab, ac, bc\}$, but all have a variance greater than 10. Thus the transaction can be pruned according to Theorem 3.1. Note that transaction 4 is superset of at least 2 candidate itemsets and every of its items is contained in at least 2 candidate itemsets, i.e. it would not have been pruned by the $\mathcal{T}_k(t)$ and $\mathcal{L}_k(i)$ anti-monotone data reductions. It is easy to see that the same way also the transactions 1, 3 and 6 can be pruned. In Figure 2(c) we have the reduced dataset we obtain at end of the second

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)

Figure 2: Run Through Example

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 from L_2 obtaining $C_3 = \{abc, acg, aci, acj, agi, agj, cdg, cgi, cgj\}$. At this point, we start the $count\&reduce^{\mathcal{LAM}}$ phase where we found that only 3 itemsets are frequent $L_3 = \{cgi, acg, cdg\}$ and only one is a solution $R_3 = \{cgi\}$. The previous $\mathcal{C}_{LAM}$ pruning has reduced the dataset in such a way that now we can perform additional $\mathcal{C}_{freq}$-based pruning until get the dataset in Figure 2(d).

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 untractable.

5 $ExAMiner^{\mathcal{LAM}}$: 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. $avg(X.A) \geq m$) and tougher constraints (e.g. $var(X.A) \leq m$). The results are reported in Figure 3.

All the tests were conducted on a Windows XP PC equipped with a 2.8GHz Pentium IV and 512MB 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 $[0, 150000]$.

In Figure 3(a) and (b) are reported the tests with the $\mathcal{C}_{LAM}$ constraint $var(X.A) \leq m$. Since we are pushing a $\mathcal{C}_{LAM}$ constraint never studied before, we compare $ExAMiner^{\mathcal{LAM}}$ against two unconstrained computation: FP-Growth and ExAMiner without constraint (i.e. it only exploits $\mathcal{C}_{freq}$-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 Figure 3(c) and (d) we compared the dataset reduction power of ExAMiner against $ExAMiner^{\mathcal{LAM}}$ when mining with the $\mathcal{C}_{CAM}$ constraint $avg(X.A) \geq m$. Since ExAMiner is designed to deal with monotone constraints, and *avg* is not, such constraint is pushed by inducing the weaker but monotone constraint $max(X.A) \geq v$ as done in [2]. 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 Figure 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, $ExAMiner^{\mathcal{LAM}}$, see Figure 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: $ExAMiner^{\mathcal{LAM}}$ is one order of magnitude faster than ExAMiner as reported in Figure 3(e). Conversely, $\mathcal{FIC}^{\mathcal{A}}$ is not able to bring such improvements. In Figure 3(f) we report the speed-up of $ExAMiner^{\mathcal{LAM}}$ w.r.t. ExAMiner and $\mathcal{FIC}^{\mathcal{A}}$ 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, $ExAMiner^{\mathcal{LAM}}$ exhibits in average a speed up of factor 100 against its own unconstrained computation, while $\mathcal{FIC}^{\mathcal{A}}$ 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 with a filtering of the output in some cases is better that its variant $\mathcal{FIC}^{\mathcal{A}}$, which is explicitly geared on constrained mining. As we have discussed in Section 2 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 our algorithm's performance.

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

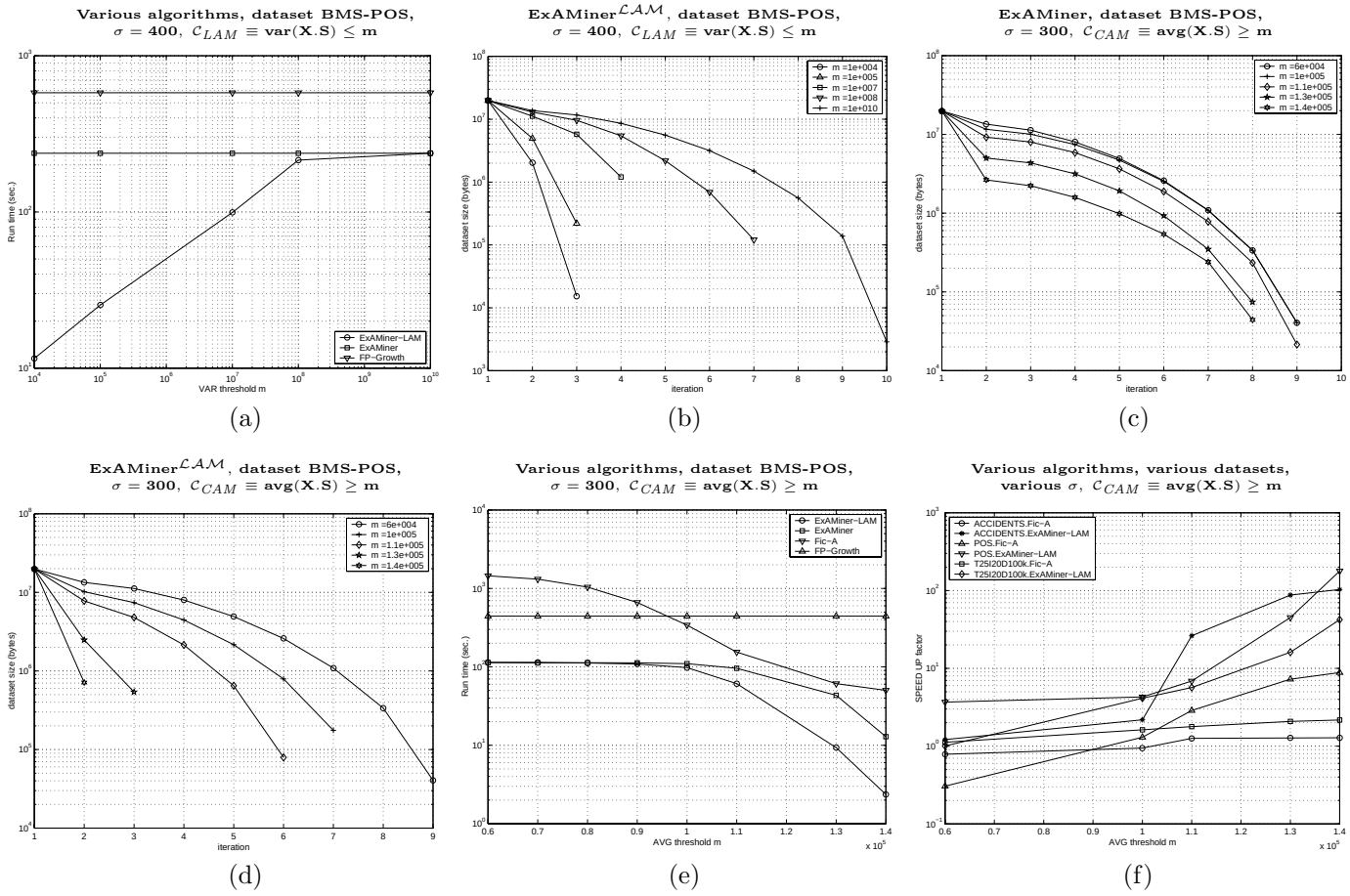


Figure 3: Loose anti-monotonicity: experimental analysis results.

6 Advanced Pruning Techniques

As stated by Proposition 3.1 $\mathcal{C}_{CAM}$ constraints are a subclass of $\mathcal{C}_{LAM}$ constraints. In the previous Section we have shown that, exploiting only loose anti-monotonicity, $ExAMiner^{\mathcal{LAM}}$ is able to outperform the state-of-art algorithms for frequent pattern mining under $\mathcal{C}_{CAM}$ constraints (see Figure 3(e)). However, in the case of convertible constraints, we have further data reduction opportunities.

In this Section we focus on the mining problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_{CAM})$ 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 $ExAMiner^{\mathcal{CAM}}$.

Similarly to [12, 13] we require items to be sorted by descending (ascending) order of attribute if $\mathcal{C}_{CAM}$ is defined over a prefix decreasing (increasing) function f (we denote this order by $\prec$). Transactions in $\mathcal{D}$, 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.

Pre-counting Reduction At the beginning of iteration k, transactions whose k-prefix (w.r.t. order $\prec$) does not satisfy the given $\mathcal{C}_{CAM}$ constraint, will not support any solution itemset for the current and future iterations, and thus they can be removed from $\mathcal{D}$. The dataset obtained after such reduction is denoted $\mathcal{D}'$. In Algorithm 2 $prefix(I, n)$ denotes the n-prefix of I (i.e. the first n items of I).

Algorithm 2 Pre-counting Reduction

Input: $\mathcal{D}, k, \mathcal{C}_{CAM}$
Output: $\mathcal{D}'$
1: $\mathcal{D}' \leftarrow \emptyset$
2: **for all** $t \in \mathcal{D}$ **do**
3: **if** $\mathcal{C}_{CAM}(prefix(t, k))$ **then**
4: $\mathcal{D}' \leftarrow \mathcal{D}' \cup t$

THEOREM 6.1. (PRE-COUNTING REDUCTION) *Given a dataset $\mathcal{D}$ and a convertible anti-monotone constraint $\mathcal{C}_{CAM}$. Let $\mathcal{D}'$ be the reduced dataset produced by Algorithm 2. It holds that:*

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

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

Counting Early Stopping During the usual counting procedure (line 9 of Algorithm 1) of iteration k, for each transaction t, all its k-subsets are generated and matched against C_k. We would like to stop as soon as possible this costly matching procedure.

Let $t = \{a, b, c, d, e, f, g, h\}$ be a transaction in $\mathcal{D}$, with associated prices $\langle 100, 100, 80, 40, 35, 30, 20, 15\rangle$, and let $\mathcal{C}_{CAM} \equiv avg(X.A) \geq 70$. Suppose that we stop as soon as we found an itemset $X \subseteq t, X \in C_k$ which does not satisfy the constraint such as $\{ade\}$. In such case we will not discover further valid itemsets like $\{bcf\}$. But even if we stop when no other interesting itemset $Y \subseteq t, Y \in C_k \mid Y \succ X \wedge \mathcal{C}_{CAM}(Y)$ exists, we could lose some valid itemset. This is the case of $X = \{bcg\}$, in fact $\neg\exists Y \succ X \mid \mathcal{C}_{CAM}(Y)$. Anyway, we must point out that in our framework, stopping the counting procedure has an important side-effect because, by reducing the number of intersections between C_k and t, we reduce the local counts $i.count$ of some item $i \in t$, thus boosting the $\mathcal{L}_k(i)$ pruning (see Alg. 1 lines 19-20). In fact 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 3 which substitutes line 9 of Algorithm 1): 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 $Y \subseteq t, Y \in C_k \mid first(Y) \preceq last(X)$ 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 $X \mid \neg\mathcal{C}_{CAM}(X)$.

To prove the correctness of Algorithm 3, 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 successive iterations. More formally:

Algorithm 3 Counting Early Stopping

```
1:  for all t ∈ D do
2:    invalidFound ← false
3:    for all X ∈ C_k | X ⊆ t do
4:      if ¬ C_CAM(X) then
5:        if invalidFound = false then
6:          Xlast ← last(X)
7:        invalidFound ← true
8:        Yfirst ← first(X)
9:        if Yfirst > Xlast then
10:         break {Stopping criterion met}
11:     else
12:       invalidFound ← false
13:     ... perform usual counting ...
```

THEOREM 6.2. *The stopping criterion defined by Algorithm 3 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 $i.count < k \Rightarrow$*

$$\neg\exists I \mid i \in I \wedge |I| > k \wedge I \in Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_{CAM}).$$

Proof. We prove by contradiction that no item will have a local count too low. 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 $\{Y \mid i \in Y \wedge first(Y) \preceq last(X)\}$ which are subsets of I, and therefore $i.count \geq k$, 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 $\mathcal{C}_{CAM}$, because of the item ordering I will not either, and therefore $I \notin Th(\mathcal{C}_{CAM})$ 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 6.1. *If exists an itemset X such that $X \subseteq t, X \in C_k, \neg\mathcal{C}_{CAM}(X)$ 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 $\neg\exists Y \subseteq t, Y \in C_k \mid first(Y) \preceq last(X) \wedge \mathcal{C}_{CAM}(Y)$, and therefore according to Theorem 6.2 the counting procedure can be stopped after X.

Post-counting Reduction At the end of the counting phase (before line 21 in Algorithm 1), 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 itemset.

The last item of a transaction t is the best candidate for deletion both when $\mathcal{C}_{CAM}$ 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 $\mathcal{C}_{CAM}$. If not, we are sure that no 4-itemset containing h and supported by t will satisfy $\mathcal{C}_{CAM}$ 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 $\mathcal{C}_{CAM}$.

Therefore, if k is the current iteration, what we have to check is that h can not 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 Alg.4). 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 $d.count \geq 4$ 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 Alg.4).

THEOREM 6.3. (POST-COUNTING REDUCTION) *Given a dataset $\mathcal{D}$ and a convertible anti-monotone constraint $\mathcal{C}_{CAM}$. Let $\mathcal{D}'$ be the reduced dataset produced by Algorithm 4. It holds that:*

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

Advanced Pruning Techniques: Experiments Experimental results presented in this Section confirm that the three proposed advanced pruning strategies bring an additional speed-up when dealing with convertible constraints.

Algorithm 4 Post-counting Reduction

Input: $\mathcal{D}, k, \mathcal{C}_{CAM}$
Output: $\mathcal{D}'$

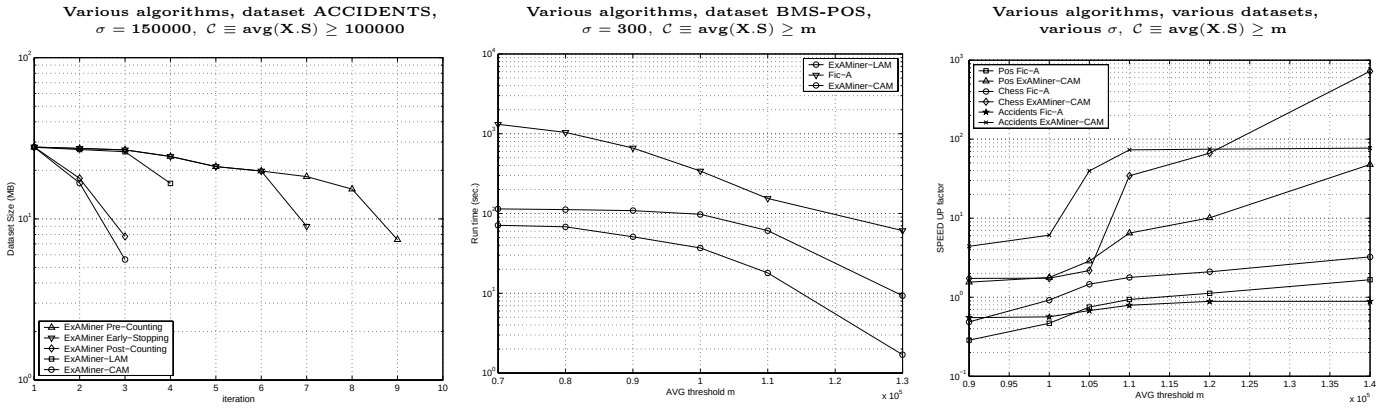
```
1:  for all t ∈ D do
2:    repeat
3:      delete ← false
4:      z ← last(t)
5:      l ← k
6:      X ← take(t, l)
7:      {takes first l items in t with count≥ l }
8:      while (¬delete and ¬ C_CAM({X ∪ z})) do
9:        old_f_val ← f({X ∪ z})
10:       l ← l + 1
11:       X ← take(t, l)
12:       if ¬(k ≤ |X| ≤ z.count) then
13:         delete ← true
14:       else
15:         if f({X ∪ z}) < (>) old_f_val then
16:           delete ← true
17:     if delete = true then
18:       t ← t \ z
19:   until delete = false
```

In Figure 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 $ExAMiner^{\mathcal{LAM}}$, i.e. *Post-counting Reduction*, and the improvement is of about one order of magnitude. As predictable, the four strategies all together ($ExAMiner^{\mathcal{CAM}}$) perform better than any single one. Such increased pruning power leads to a lower computation time, as shown in Figure 4(b). $ExAMiner^{\mathcal{LAM}}$ is one order of magnitude faster than $\mathcal{FIC}^{\mathcal{A}}$, while our advanced pruning techniques bring $ExAMiner^{\mathcal{CAM}}$ to be two orders of magnitude faster than $\mathcal{FIC}^{\mathcal{A}}$.

Finally, in Figure 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.

7 Conclusion

In this paper we have extended the state-of-art of the constraints that can be pushed in a frequent pattern computation. Tough constraints are pushed within a level-wise computation by means of data reduction techniques. The proposed computational framework is not only able to exploit constraints which have never been studied before, but it is also able to outperform the state-of-art algorithms for frequent pattern mining with the toughest class of constraint studied by previous works (i.e. convertible constraints).



(a) (b) (c)

Figure 4: Advanced Pruning Techniques: experimental analysis results.

Being a level-wise Apriori-like computation, our framework can exploit all different properties of constraints all together, and the total benefit is always greater than the sum of the individual benefits. In other words, our contribution can be easily integrated with previous works (e.g. [11, 2]), in a unique Apriori-like computational framework able to take full advantage by any conjunction of possible constraints. In particular:

- Anti-monotone ($\mathcal{C}_{AM}$) constraints are exploited to prune the level-wise exploration of the search space together with the frequency constraint ($\mathcal{C}_{freq}$);
- Succinct ($\mathcal{C}_S$) constraints are exploited at candidate generation time as done in [11];
- Monotone ($\mathcal{C}_M$) constraints are exploited by means of data reduction as done in [2];
- Convertible anti-monotone ($\mathcal{C}_{CAM}$) and Loose anti-monotone ($\mathcal{C}_{LAM}$) constraints are exploited by means of data reduction as described in this paper.

At Pisa KDD Laboratory we are currently developing such unified computational framework (within the P^3D project[2]) which will be soon made available to the community.

References

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of VLDB'94*.

[2] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In *Proceedings of ICDM'03*.

[3] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Exante: Anticipated data reduction in constrained pattern mining. In *Proceedings of PKDD'03*.

[4] C. Bucila, J. Gehrke, D. Kifer, and W. White. DualMiner: A dual-pruning algorithm for itemsets with constraints. In *Proceedings of ACM SIGKDD'02*.

[5] L. DeRaedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *Proceedings of IJCAI'01*.

[6] G. Grahne, L. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In *16th International Conference on Data Engineering (ICDE' 00)*, pages 512–524. IEEE, 2000.

[7] J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-based, multidimensional data mining. *Computer*, 32(8):46–50, 1999.

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD*.

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

[10] L. V. S. Lakshmanan, R. T. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 28(2), 1999.

[11] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the ACM SIGMOD'98*.

[12] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proceedings of the 6th ACM SIGKDD'00*.

[13] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *(ICDE'01)*, pages 433–442, 2001.

[14] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proceedings of KDD'97*.

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