Pushing Tougher Constraints in Frequent Pattern Mining

Francesco Bonchi[1] and Claudio Lucchese[2]

[1] Pisa KDD Laboratory, ISTI - C.N.R., Area della Ricerca di Pisa, Italy
[2] Department of Computer Science, University Ca' Foscari, Venezia, Italy

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. constraints on *variance* or *standard deviation*). Then we show how these constraints can be exploited in a level-wise Apriori-like computation by means of a new data-reduction technique: the resulting algorithm outperforms previous proposals for convertible constraints, and it is to treat much tougher constraints with the same effectiveness of easier ones.

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 [9, 10, 14–18].

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 $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$.

Related Work and Constraints Classification. A first work defining classes of constraints which exhibit nice properties is [15]. 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. 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. 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).

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. 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 preprocessing 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 [15] computation by inducing weaker constraints which are either anti-monotone and/or succinct.

Monotone constraints work the opposite way of anti-monotone constraints. Given an itemset X, a constraint $\mathcal{C}_M$ is monotone if: $\forall Y \supseteq X : \mathcal{C}_M(X) \Rightarrow \mathcal{C}_M(Y)$. Since the frequent itemset computation is geared on $\mathcal{C}_{freq}$, which is anti-monotone, $\mathcal{C}_M$ constraints have been considered more hard to be pushed in the computation and less effective in pruning the search space [2, 8, 7, 12]: while anti-monotone constraints can be used to effectively prune the search space to a small downward closed collection, the upward closed collection of the search space satisfying the monotone constraints cannot be pruned at the same time. Recently, it has has been shown that a real synergy of these two opposite types of constraints exists and can be exploited by reasoning on both the itemset search space and the input database *together*, using the ExAnte data-reduction technique [4]. Using data reduction techniques, anti-monotone and monotone pruning strengthen each other recursively [3, 5].

In [16, 17] the class of **convertible** constraints is introduced, and an FP-growth based methodology to push such constraints is proposed. A constraint $\mathcal{C}_{CAM}$ is convertible anti-monotone provided there is an order $\mathcal{R}$ on items such that whenever an itemset X satisfies $\mathcal{C}_{CAM}$, so does any prefix of X. A constraint $\mathcal{C}_{CM}$ is convertible monotone provided there is an order $\mathcal{R}$ on items such that whenever an itemset X violates $\mathcal{C}_{CM}$, so does any prefix of X. In [16, 17], 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 4: 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.

The first (and to our knowledge unique) work, trying to address the problem of how to push constraints which are **not convertible**, is [13]. 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 main drawback of the proposal is the following: it may require quadratic time in the number of frequent singleton items 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.

Paper Contribution. The contribution of this paper is threefold. First, 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*. Second, we show that it is possible to push convertible constraints in a level-wise Apriori-like computation, outperforming previously proposed FP-growth based algorithms [16, 17]. Third, we propose a general Apriori-like algorithm, based on data-reduction techniques, which is able to push all possible kinds of constraint studied so far.

2 Loose Anti-monotone Constraints

In this Section we introduce a new class of tougher constraints, which is a proper superclass of convertible anti-monotone. The following example shows that exist interesting constraints which are not convertible, and thus cannot be exploited within a prefix pattern framework.

Example 1 (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 variance estimator* (var_{N-1}) or *mean deviation* (*md*), are not convertible as well. 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 $k-1$. Since some of these interesting constraints make sense only on sets of cardinality at least 2, in order to get rid of such details, we shift the definition of loose anti-monotone constraint to avoid considering singleton items.

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

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

Proposition 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.*

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

Table 1. Classification of commonly used constraints.

Example 4. We show that the constraint $var(X.A) \leq v$ is a $\mathcal{C}_{LAM}$ constraint. Given an itemset X, if it satisfies the constraint so trivially does $X \setminus \{i\}$, where i is the element of X which has associated a value of A which is the most far away from $avg(X.A)$. In fact, we have that $var(\{X \setminus \{i\}\}.A) \leq var(X.A) \leq v$, until $|X| > 2$.Taking the element of X which has associated a value of A which is the closest to $avg(X.A)$ we can show that also $var(X.A) \geq v$ is a $\mathcal{C}_{LAM}$ constraint. Since the standard deviation *std* is the square root of the variance, it is straightforward to see that $std(X.A) \leq v$ and $std(X.A) \geq v$ are $\mathcal{C}_{LAM}$. The mean deviation is defined as: $md(X) = (\sum_{i \in X} |i - avg(X)|) \ / \ |X|$. Once again, we have that $md(X.A) \leq v$ and $md(X.A) \geq v$ are loose anti-monotone. It is easy to prove that also constraints defined on the unbiased variance estimator, $var_{N-1} = (\sum_{i \in X}(i - avg(X))^2) \ / \ (|X| - 1)$ are loose anti-monotone.

In Table 1 we update the state-of-art classification of commonly used constraints. The next key Theorem indicates how a $\mathcal{C}_{LAM}$ constraint can be exploited in a level-wise Apriori-like computation by means of data-reduction. It states that if at any iteration $k \geq 2$ a transaction is not superset of at least one frequent k-itemset which satisfy the $\mathcal{C}_{LAM}$ constraint (a solution), then the transaction can be deleted from the database.

Theorem 5. *Given a transaction database $\mathcal{D}$, a minimum support threshold σ, and a $\mathcal{C}_{LAM}$ constraint, at the iteration $k \geq 2$ of the level-wise computation, a transaction $t \in \mathcal{D}$ such that: $\nexists X \subseteq t, |X| = k, X \in Th(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \cap Th(\mathcal{C}_{LAM})$ can be pruned away from $\mathcal{D}$, since it will never be superset of any solution itemsets of cardinality $> k$.*

Proof. Suppose that exists $Y \subseteq t, |Y| = k + j, Y \in Th(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \cap Th(\mathcal{C}_{LAM})$. For loose anti-monotonicity this implies that exists $Z \subseteq Y, |Z| = k + j - 1$ such that $\mathcal{C}_{LAM}(Z)$. 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 $X \subseteq t, |X| = k, X \in Th(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \cap Th(\mathcal{C}_{LAM})$.

Note that a conjunction of loose anti-monotone constraint 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 5 does not hold for even only one constraint in the conjunction (this is implemented by line 14 of the pseudo-code in Figure 1).

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.

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

The recently introduced algorithm ExAMiner [3], 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. This is obtained by coupling the set of data reduction techniques in Table 2 (see [3] for the proof of correctness), which are based on the anti-monotonicity of $\mathcal{C}_{freq}$, with the data reduction based on the $\mathcal{C}_M$ constraint. 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 5. 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 5, 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 from the second iteration on ($k \geq 2$). Therefore to illustrate the $ExAMiner^{\mathcal{LAM}}$ algorithm we just provide the pseudo-code of the $count\&reduce^{\mathcal{LAM}}$ procedure (Figure 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, and L_k to denote the set of *frequent* (or large) itemsets at iteration k.
In the pseudo-code in Figure 1, the $count\&reduce^{\mathcal{LAM}}$ procedure, at iteration k takes in input the actual database $\mathcal{D}_k$, the minimum support threshold σ, a

$\mathcal{G}_k(i)$	*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.*

Table 2. Data-reduction techniques based on the anti-monotonicity of $\mathcal{C}_{freq}$

Procedure $count\&reduce^{\mathcal{LAM}}$

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

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

Fig. 1. Pseudo-code of procedure $count\&reduce^{\mathcal{LAM}}$

user-defined conjunction of loose anti-monotone constraints $\mathcal{C}_{LAM}$, a user-defined conjunction of monotone constraints $\mathcal{C}_M$, the actual set of candidate itemsets C_k, and an array of integers V_{k-1} of the size of $\mathcal{I}$. Such array is used in order to implement the data-reduction $\mathcal{G}_k(i)$. The array V_k records, for each singleton 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 4 and 5 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 8 and 15) and which are superset of i ($i.count$ in the pseudo-code, lines 6, 11 and 16). Analogously, the data reduction based on loose anti-monotonicity described in Theorem 5, is exploited in the same relaxed version with candidates instead of frequent itemsets. In the pseudo-code, for each constraint C in the given conjunction of loose anti-monotone constraints $\mathcal{C}_{LAM}$, we have a flag $t.lam[C]$ which is set to $true$ as soon as an itemset $X \in C_k$, such that $X \subseteq t, X \in Th(C)$, is found (line 10). A transaction which has even only one of the $t.lam[C]$ flags set to false after the counting phase, will not enter in the database for the next iteration $\mathcal{D}_{k+1}$ (line 14 of the pseudo-code). In fact, such a transaction has not covered any candidate itemset which satisfies the constraint C, for some C in the conjunction $\mathcal{C}_{LAM}$, therefore it will not support any itemset satisfying such constraint, and thus any solution itemset.

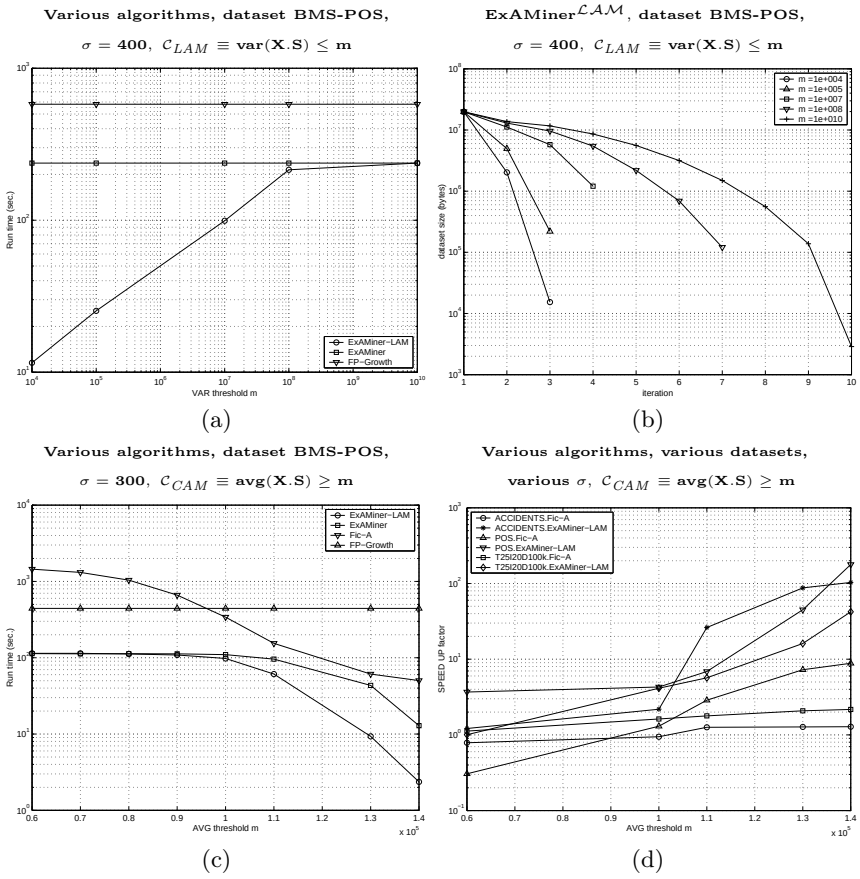

Fig. 2. Loose anti-monotonicity: experimental analysis results.

4 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 2. 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 2(a) and (b) are reported the tests with the $\mathcal{C}_{LAM}$ constraint $var(X.A) \leq m$. 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.

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

This behavior is reflected in run-time performances: $ExAMiner^{\mathcal{LAM}}$ is one order of magnitude faster than ExAMiner as reported in Figure 2(c). Conversely, $\mathcal{FIC}^{\mathcal{A}}$ is not able to bring such improvements. In Figure 2(d) 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 discussed before, this is due to the items ordering based on attribute values and not on frequency.

5 Pushing Multiple Constraints

As already stated, one of the most important advantage of our methodology 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 pruning of the search space, and the pruning of the search space in turn strengthens future data reductions.

Note that in the pseudo-code in Figure 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* [3] (no $\mathcal{C}_{LAM}$ data reduction); while if we have an empty set of $\mathcal{C}_M$ constraints, the $\mathcal{C}_M$ testing (lines 7 and 17 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.

Example 6. The constraint $range(S.A) \geq v \equiv max(S.A) - min(S.A) \geq v$, is both monotone and loose anti-monotone. Thus, when we mine frequent itemsets which satisfy such constraint we can exploit the benefit of having together, in the same $count\&reduce^{\mathcal{LAM}}$ procedure, the $\mathcal{C}_{freq}$-based data reductions of Table 2, the μ-reduction for monotone constraints, and the reduction based on $\mathcal{C}_{LAM}$.

Being a level-wise Apriori-like computation, our framework can exploit all different properties of constraints all together. In other words, our contribution can be easily integrated with previous works (e.g. [15, 3]), 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 [15]; monotone ($\mathcal{C}_M$) constraints are exploited by means of data reduction as done in [3]; 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. Adaptive Constraint Pushing in frequent pattern mining. In *Proceedings of PKDD'03*.
3. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In *Proceedings of ICDM'03*.
4. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAnte: Anticipated data reduction in constrained pattern mining. In *Proceedings of PKDD'03*.
5. F. Bonchi and B. Goethals. FP-Bonsai: the art of growing and pruning small fp-trees. In *Proceedings PAKDD'04*.
6. F. Bonchi and C. Lucchese. On closed constrained frequent pattern mining. In *Proceedings of ICDM'04*.
7. C. Bucila, J. Gehrke, D. Kifer, and W. White. DualMiner: A dual-pruning algorithm for itemsets with constraints. In *Proceedings of ACM SIGKDD'02*.
8. L. DeRaedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *Proceedings of IJCAI'01*.
9. 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.
10. J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-based, multidimensional data mining. *Computer*, 32(8):46–50, 1999.
11. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM SIGMOD'00*.
12. B. Jeudy and J.-F. Boulicaut. Optimization of association rule mining queries. *Intelligent Data Analysis Journal*, 6(4):341–357, 2002.
13. D. Kifer, J. Gehrke, C. Bucila, and W. White. How to quicklyfind a witness. In *Proceedings of PODS'03*.
14. L. V. S. Lakshmanan, R. T. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. *SIGMOD Record*, 28(2), 1999.
15. 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*.
16. J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proceedings of ACM SIGKDD'00*.
17. J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *(Proceedings of ICDE'01)*.
18. R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proceedings of ACM SIGKDD'97*.

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