Under consideration for publication in Knowledge and Information Systems

Efficient Breadth-first Mining of Frequent Pattern with Monotone Constraints

Francesco Bonchi[1], Fosca Giannotti[1], Alessio Mazzanti[2] and Dino Pedreschi[3]

[1] KDD Laboratory, ISTI Area della Ricerca C.N.R. Pisa, Italy; [2] LIST S.p.A. Pisa, Italy;
[3] KDD Laboratory, Department of Computer Science, University of Pisa, Italy

Abstract. The key point of this article is that, in frequent pattern mining, the most appropriate way of exploiting monotone constraints in conjunction with frequency is to use them in order to reduce the input data, this reduction in turn induces a stronger pruning of the search space of the problem. Following this intuition, we introduce ExAMiner, a breadth-first algorithm which exploits the real synergy of antimonotone and monotone constraints: the total benefit is greater than the sum of the two individual benefits. ExAMiner generalizes the basic idea of the preprocessing algorithm ExAnte (Bonchi et al, 2003(b)), embedding such ideas at all levels of an Apriori-like computation. The resulting algorithm is the generalization of the Apriori algorithm when a conjunction of monotone constraints is conjoined to the frequency antimonotone constraint. Experimental results confirm that this is, so far, the most efficient way of attacking the computational problem in analysis.

Keywords: Frequent Itemsets Mining; Constraints; Data Reduction.

1. Introduction

Constrained itemsets mining i.e., finding all itemsets included in a transaction database that satisfy a given set of constraints, is an active research theme in data mining (Srikant et al, 1997; Ng et al, 1998; Han et al, 1999; Lakshmanan et al, 1999; Grahne et al, 2000; Pei et al, 2001; Boulicaut and Jeudy, 2002). The most studied constraint is the frequency constraint, whose anti-monotonicity is used to reduce the exponential search space of the problem. Exploiting the anti-monotonicity of the frequency constraint is known as the *Apriori pruning*

Received Nov. 19
Revised Jan. 9
Accepted Feb. 16

method (Agrawal et al, 1993; Agrawal and Srikant, 1994): it dramatically reduces the search space, thus making the computation feasible. Frequency is not only computationally effective, it is also semantically important since frequency provides "support" to any discovered knowledge. For these reasons frequency is the base constraint of what is generally referred to as *frequent itemsets mining*. However, many other constraints can facilitate user-focussed exploration and control, as well as reduce the computation. For instance, a user could be interested in mining all frequently purchased itemsets having a total price greater than a given threshold and containing at least two products of a given brand. Among these constraints, classes which exhibit nice properties have been characterized. The class of antimonotone constraints is the most effective and easy to use in order to prune the search space. Since any conjunction of antimonotone constraints is in turn antimonotone, we can use the *Apriori pruning method*: the more antimonotone constraints are available, the more selective the *Apriori pruning method* will be. The dual class, monotone constraints, has been considered more complicated to exploit and less effective in pruning the search space. The problem of mining itemsets which satisfy a conjunction of antimonotone and monotone constraints has been studied for a long time (Ng et al, 1998; Pei et al, 2001; Boulicaut and Jeudy, 2002), but all these studies have failed in finding the real synergy between the two opposite pruning opportunities. All the authors have stated that this is the inherent difficulty of the computational problem: when dealing with a conjunction of monotone and antimonotone constraints we face a tradeoff between antimonotone and monotone pruning.

Our observation is that this prejudice holds only if we focus exclusively on the search space of the itemsets, which is the approach followed by the work done so far. In Bonchi et al (2003(b)) we have shown that a real synergy of the two opposite pruning exists, and can be exploited by reasoning on both the itemsets search space and the transactions input database *together*. In this way, pushing monotone constraints does not reduce antimonotone pruning opportunities, on the contrary, such opportunities are boosted. Dually, pushing antimonotone constraints boosts monotone pruning opportunities: the two components strengthen each other. On the basis of these considerations we have introduced ExAnte, a preprocessing data reduction algorithm which reduces dramatically both the search space and the input dataset in constrained frequent patterns mining. The experimentation of the algorithm has pointed out how effective the reduction is, and which potential benefits it offers to the subsequent frequent pattern computation.

In this article we show how the basic ideas of ExAnte can be generalized in a level-wise, Apriori-like computation. The resulting algorithm can be seen as the real generalization of Apriori, able to exploit both kind of constraints to reduce the search space. We named our algorithm **ExAMiner** (**ExA**nte **Miner** in contrast with ExAnte preprocessor, but also **Miner** which **Ex**ploits **A**ntimonotone and **M**onotone constraints together). Since the most appropriate way of exploiting monotone constraints in conjunction with frequency is to reduce the problem input, which in turn induces a reduction of the search space, the mining philosophy of ExAMiner is to reduce as much as possible the problem dimensions at all levels of the computation. Therefore, instead of trying to explore the exponentially large search space in some smart way, we massively reduce such search space as soon as possible, obtaining a progressively easier mining problem. Experimental results confirm that this is, at this moment, the most efficient way of attacking the computational problem in analysis. Moreover, ExAMiner makes

it feasible the computation of *extreme patterns*, i.e. extremely long or extremely costly patterns, which can be found only at very low support levels where all the other known mining approaches can not always complete the computation. Even if the support threshold is very low, ExAMiner, exploiting the extremely strong selectivity of the monotone constraint, reduces drastically the problem dimensions and makes the computation affordable.

1.1. Problem Definition

Let $\mathcal{I} = \{x_1, ..., x_n\}$ be a set of distinct literals, usually called *items*. An *itemset* X is a non-empty subset of $\mathcal{I}$. If $|X| = k$ then X is called a *k-itemset*. A *transaction* is a couple $\langle tid, X\rangle$ where *tid* is the transaction identifier and X is the content of the transaction (an itemset). A *transaction database* $\mathcal{D}$ is a set of transactions. An itemset X is contained in a transaction $\langle tid, Y\rangle$ if $X \subseteq Y$. A constraint on itemsets is a function $\mathcal{C} : 2^{\mathcal{I}} \rightarrow \{true, false\}$. We say that an itemset $I \subseteq \mathcal{I}$ satisfies a constraint if and only if $\mathcal{C}(I) = true$. The *support* of an itemset X in database $\mathcal{D}$, denoted by $supp_{\mathcal{D}}(X)$ is the number of transactions in $\mathcal{D}$ that contain X. Given a user-defined *minimum support* σ, an itemset X is called *frequent* in $\mathcal{D}$ if $supp_{\mathcal{D}}(X) \geq \sigma$. Let $Th(\mathcal{C}) = \{X \mid \mathcal{C}(X) = true\}$ denote the set all itemsets X that satisfy constraint $\mathcal{C}$. The *frequent itemset 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 itemset mining problem* requires to compute $Th(\mathcal{C})$; the *constrained frequent itemset mining problem* requires to compute $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$.

We now formally define the notion of antimonotone and monotone constraints.

Definition 1.1. Given an itemset X, a constraint $\mathcal{C}_{AM}$ is *antimonotone* if

$$\forall Y \subseteq X : \mathcal{C}_{AM}(X) \Rightarrow \mathcal{C}_{AM}(Y).$$

If $\mathcal{C}_{AM}$ holds for X then it holds for any subset of X.

The frequency constraint is clearly antimonotone. This property is used by the Apriori 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.

Definition 1.2. Given an itemset X, a constraint $\mathcal{C}_M$ is *monotone* if:

$$\forall Y \supseteq X : \mathcal{C}_M(X) \Rightarrow \mathcal{C}_M(Y).$$

If $\mathcal{C}_M$ holds for X then it holds for any superset of X.

The general problem that we consider in this article is the mining of itemsets which satisfy a conjunction of monotone and antimonotone constraints:

$$Th(\mathcal{C}_{AM}) \cap Th(\mathcal{C}_M)$$

Since any conjunction of antimonotone constraints is an antimonotone constraint, and any conjunction of monotone constraints is a monotone constraint,

Monotone constraint	$\mathcal{C}_M \equiv$
cardinality	$card(X) \geq n$
sum of prices	$sum(X.prices) \geq n$
maximum price	$max(X.prices) \geq n$
minimum price	$min(X.prices) \leq n$
range of prices	$range(X.prices) \geq n$

Table 1. Strongly monotone constraints.

we just consider the general constraints $\mathcal{C}_{AM}$ and $\mathcal{C}_M$, where $\mathcal{C}_{freq}$ is always present in the conjunction $\mathcal{C}_{AM}$ unless explicitly stated otherwise.

A constraint is called *strong* if it can be evaluated for any given itemset independently of the given transaction database. In this article we will mainly focus on those strong constraints, together with the minimum frequency constraint. This is necessary since we want to distinguish between simple monotone constraints and global constraints such as the *"infrequency constraint"*: $supp_{\mathcal{D}}(X) \leq \sigma$. This constraint is still monotone but has different properties since it is dataset dependent and it requires dataset scans in order to be computed. Obviously, since our algorithm reduces the transaction dataset, we want to exclude such data dependent constraints from our study. Thus, our study focuses on strongly monotone constraints, in the sense that they depend exclusively on the properties of the itemset and not on the underlying transaction database. For example, Table1) shows some data independent monotone constraints. In the rest of this article, we write, for the sake of brevity, monotone and antimonotone instead of strongly monotone and strongly antimonotone respectively. The antimonotone constraints that are not strong will be represented by the minimum frequency constraint, without loss of generality.

1.2. Related Work

The first proposal of an algorithm for mining frequent itemset with monotone constraints is $\mathcal{FIC}^{\mathcal{M}}$, an FP-growth based algorithm, introduced in (Pei et al, 2001). This algorithm is particularly studied for those *"hard"* constraints, named *convertible*, which are neither antimonotone nor monotone, but that can be converted in these kinds of *"simple"* constraints by sorting the items in some specific order in the header table of the FP-tree. Consider for instance the constraint $avg(X.price) \geq n$: there is no stable relation between subsets, supersets and satisfaction of such a constraint. But if we arrange the items in *price-ascending-order*, the average of prices of an itemset is always more than the average of its prefix itemset. Therefore w.r.t. this order the constraint is monotone. The authors state that since a monotone constraint it is also a convertible monotone constraint, this technique can be used also to push simple monotone constraints in the frequent itemsets computation.

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 the monotone constraint. 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 the monotone constraint. Once a frequent itemset satisfies the monotone constraint, all frequent itemsets having it as a prefix also are guaranteed to satisfy the constraint. The main drawback of this proposal is

that sorting items by "prices" makes us lose the compacting effect of the ordering by frequency, and thus we have to manage much greater FP-trees, requiring a lot more main memory which might not be available.

In (Boulicaut and Jeudy, 2002) it is shown that pushing monotone constraints can lead to a reduction of antimonotone pruning opportunities. Therefore, when dealing with a conjunction of monotone and antimonotone constraints we face a tradeoff between antimonotone and monotone pruning. Suppose that a pattern has been removed from the search space because it does not satisfy a monotone constraint. This pruning avoids checking support for this pattern, 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 antimonotone 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 antimonotone pruning.

The first work trying to find a real amalgam between antimonotone and monotone pruning is *DualMiner* (Bucila et al, 2002). However, DualMiner does not compute all solution itemsets with their support. It just finds the family of solution sub-algebras, representing them with their top and bottom elements. This is an analog with maximal frequent pattern mining. In maximal frequent pattern mining one is interested in mining only frequent itemsets which are maximal w.r.t. set inclusion. DualMiner does not compute the whole $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$, but it computes the set of all couples $\langle T, B \rangle$ where T and B are itemsets which represent the top and the bottom, respectively, of a solution sub-algebra. Therefore, DualMiner can not be used for constrained frequent pattern mining in contexts where the support of each solution is required, for instance when we want to compute association rules. Moreover, the dual top-down bottom-up exploration of the search space, performed by DualMiner, faces many practical limitations. First of all, DualMiner uses multiple scans of the database, even to compute supports of itemsets of the same size. This is due to inherent *divide-et-impera* strategy characterizing the DualMiner computation. Moreover the dual top-down bottom-up computation performs very poorly on real-world problems. In real-world problems, even on dense datasets, or at very low support levels, frequency is always a very selective constraint: it is quite hard to find frequent patterns of large size. The DualMiner top-down computation, exploiting $\mathcal{C}_M$ pruning, will usually perform a huge number of useless tests on very large itemsets, inducing a degradation of the performance. The same reasoning, i.e. frequency is often much more selective than any other reasonable constraint, represents the main drawback also for the *Version Space* (De Raedt and Kramer, 2001) and *ACP* (Bonchi et al, 2003(a)) algorithms.

1.3. Our Contribution

In this article we introduce ExAMiner, a breadth-first algorithm which exploits the real synergy of antimonotone and monotone constraints: the total benefit is greater than the sum of the two individual benefits. ExAMiner generalizes the basic idea of the preprocessing algorithm ExAnte, embedding such idea at all levels of a level-wise Apriori-like computation. The resulting algorithm brings to the research field the following interesting contributions:

- ExAMiner is the generalization of the Apriori algorithm, when a conjunction of monotone constraints is conjoined to the frequency antimonotone constraint.
- There are some data reduction techniques known in literature (Park et al, 1995; Orlando et al, 2002), which rely exclusively on the antimonotonicity of frequency and which, if exploited alone, bring little benefit to the computation. Introducing ExAMiner, we show how this techniques can be coupled with data reduction based on a monotone constraint, obtaining dramatically effective optimizations.
- In presence of a cardinality monotone constraint, ExAMiner exploits enhanced data reduction techniques which further amalgamate antimonotonicity and monotonicity. Moreover, we show that this improvement can be induced, even if in weaker form, also for the other forms of monotone constraint.
- We introduce a pruning technique that reduces directly the generators itemsets at each iteration i.e., the frequent itemsets that are not going to produce solutions. This is more effective than reducing the candidate itemsets, as in ordinary Apriori-like algorithms.
- ExAMiner can be used with any constraint which has a monotone component: therefore also succinct monotone constraints (Ng et al, 1998) and convertible monotone constraints (Pei et al, 2001) can be exploited.
- ExAMiner maintains the exact support of each solution itemsets: a necessary condition if we want to compute Association Rules.
- ExAMiner can be used to make feasible the discovery of extreme patterns which can be discovered only at very low support level, for which the computation is unfeasible for traditional algorithms.
- A thorough experimental study has been performed with different monotone constraints on various datasets (both real world and synthetic datasets): ExAMiner performs as the most efficient algorithm so far for constrained frequent pattern mining.

The rest of the article is organized as follows. In the next section we review the ExAnte property and the ExAnte preprocessing algorithm. In Section 3 we introduce some data reduction techniques which rely exclusively on the antimonotonicity of frequency, then we argue that, coupling such techniques with the ExAnte monotone data reduction, we can obtain dramatically effective optimizations. In Section 4 we show that, in presence of a cardinality monotone constraint, antimonotone and monotone data reduction techniques can be further amalgamated and their pruning power enhanced. Moreover we show that this improvement can be induced, even if in weaker form, also for the other kinds of monotone constraint. In Section 5 we describe in detail the ExAMiner algorithm and we provide detailed pseudo-code, then we explain its computation by means of an example, and we discuss the cost of dataset rewriting. In Section 6 we report experimental results. In Section 7 we argue that the general breadth-first data-reduction approach of ExAMiner can be exploited to mine more complex kinds of pattern from more structured data: an example describing the mining of frequent connected subgraphs from a dataset of graphs is provided. Finally in Section 8 we conclude with a discussion on further improvements to the proposed algorithm.

2. The ExAnte Property

As already stated, if we focus only on the itemsets lattice, pushing monotone constraint can lead to a less effective antimonotone pruning. Suppose that an itemset has been removed from the search space because it does not satisfy some monotone constraints $\mathcal{C}_M$. This pruning avoids checking support for it, but it may be that if we check support, the itemset could result to be infrequent, and thus all its supersets could be pruned away. By monotone pruning an itemset we risk to lose antimonotone pruning opportunities given from the itemset itself. The tradeoff is clear (Boulicaut and Jeudy, 2002): pushing monotone constraint can save tests on antimonotone constraints, however the results of these tests could have lead to more effective pruning.

In order to obtain a real amalgam of the two opposite pruning strategies we have to consider the constrained frequent patterns problem in its whole: not focussing only on the itemsets lattice but considering it together with the input database of transactions.

The recently introduced ExAnte property (Bonchi et al, 2003(b)) allows to "clean" the database, retaining only that portion of the data that is absolutely necessary in order to produce all the itemsets satisfying all the given constraints. Hence, it really doesn't matter which mining algorithm is used, the ExAnte property can always be used as a preprocessing step before mining.

In order to explain the ExAnte property, we first need to introduce the μ-*reduction* operator.

Definition 2.1 (μ-reduction). 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 $\mathcal{C}_M$.

$$\mu_{\mathcal{C}_M}(\mathcal{D}) = \{\langle tid, X\rangle \mid \langle tid, X\rangle \in \mathcal{D} \wedge X \in Th(\mathcal{C}_M)\}.$$

In other words, using the ExAnte property, we put the focus on the input data in stead of on the search space. Indeed, the tradeoff between antimonotone and monotone pruning exists only if we focus exclusively on the search space of the problem, which is the approach followed in the current state-of-the-art algorithms. But if we take a step back and look at the overall problem, reasoning on *both* the search space and the input database *together* we can find the real synergy of antimonotonicity and monotonicity constraints, as explained in the rest of this section. Instead of applying monotone constraints to the patterns, we apply them to the input database (recall that a transaction is nothing more than an itemset). 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, and thus of any solution itemset. This is stated more formally in the following Theorem.

Theorem 2.1 (ExAnte property (Bonchi et al, 2003(b))). Given a transaction database $\mathcal{D}$, and a conjunction of monotone constraints $\mathcal{C}_M$, we have

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

Proof. Since $X \in Th(\mathcal{C}_M)$, all transactions containing X will also satisfy $\mathcal{C}_M$ for monotonicity. In other words no transaction containing X will be pruned. This implies the thesis. □

2.1. The $\mathcal{C}_{AM}$-$\mathcal{C}_M$ Synergy

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 the monotone constraint 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 antimonotonicity-based pruning of the search space.

This is not the whole story, in fact, infrequent singleton items can not only be removed from the computation: for the same antimonotonicity property they can be deleted also from all transactions in the input database. This antimonotonicity-based data-reduction has been named *α-reduction*.

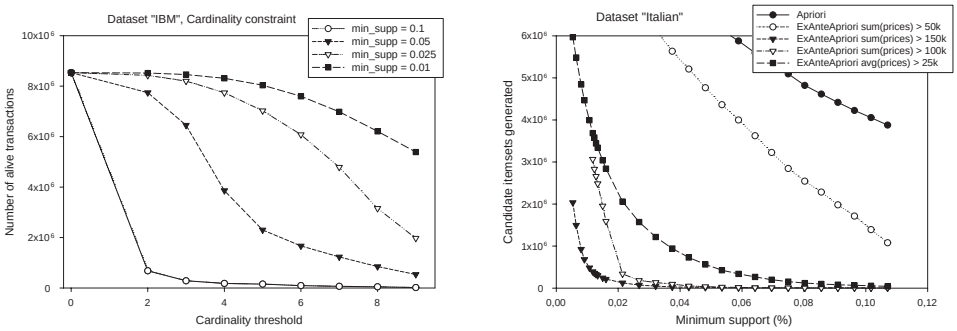
Removing items from transactions has got another positive effect: reducing the size of a transaction which satisfies a monotone constraint can make the transaction violate the monotone constraint. Consider for instance the monotone constraint based on the minimum sum of prices: a transaction which satisfies the constraint having a total sum of prices greater than the given threshold, might end up not satisfying the constraint anymore after its infrequent items are removed, since its total sum of prices might go below the threshold. Therefore a growing number of transactions which do not satisfy the monotone constraint 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 preprocessing method. In the end, the reduced dataset resulting from this fix-point computation is usually much smaller than the initial dataset (obviously depending on the selectivity of the antimonotone and monotone constraints).

Definition 2.2. Given a transaction database $\mathcal{D}$, a conjunction of monotone constraints $\mathcal{C}_M$, and a conjunction of antimonotone constraints $\mathcal{C}_{AM}$, we define the reduced dataset obtained by the fix-point application of μ and α pruning as:

$$\mu^{+}_{\mathcal{C}_{AM},\mathcal{C}_M}(\mathcal{D}) = \{\langle tid, X\rangle \mid \langle tid, Y\rangle \in \mathcal{D} \wedge \\ \wedge\, X \subseteq Y \wedge X \in Th(\mathcal{C}_M) \wedge \forall i \in X : \{i\} \in Th(\mathcal{C}_{AM})\}.$$

2.2. ExAnte Algorithm

ExAnte starts the first iteration as any frequent patterns mining algorithm: counting the support of singleton items. Items that are not frequent are thrown away once and for all. But during this first count only transactions that satisfy C_M are considered. The other transactions are signed to be pruned from the dataset (μ-reduction). Doing so we reduce the number of interesting 1-itemsets. Even a small reduction of this number represents a huge pruning of the search space. At this point ExAnte deletes from alive transactions all infrequent items (α-reduction). This pruning can reduce the monotone value (for instance, the total sum of prices) of some alive transactions, possibly resulting in a violation of the monotone constraints. Therefore we have another opportunity of μ-reducing the dataset. But μ-reducing the dataset we create new opportunities



(a) (b)

Fig. 1. ExAnte preprocessing: transactions reduction (a), and search space reduction on dataset "Italian" (b).

for α-reduction, which can turn in new opportunities for μ-reduction, and so on, until a fix-point is reached.

Algorithm 1 ExAnte

Input: $\mathcal{D}, \sigma, \mathcal{C}_M$
Output: $\mu^+_{\mathcal{C}_{freq},\mathcal{C}_M}(\mathcal{D})$

```
1:  L := ∅
2:  for all tuples t in D do
3:     if C_M(t) then
4:        for all items i in t do
5:           i.count++;
6:           if i.count ≥ σ then
7:              L := L ∪ i
8:  old_number_interesting_items = |I|
9:  while |L| < old_number_interesting_items do
10:    D = α[D]_{C_freq}
11:    D = μ[D]_{C_M}
12:    old_number_interesting_items = |L|
13:    I = ∅
14:    for all tuples t in D do
15:       for all items i in t do
16:          i.count++;
17:          if i.count ≥ σ then
18:             L := L ∪ i
```

Clearly, a fix-point is eventually reached after a finite number of iterations, as at each step the number of alive items strictly decreases.

In Figure 1 we show experimental results of ExAnte: input transactions reduction and search space reduction (measured in term of candidate itemsets considered).

3. Level-wise Antimonotone Data Reduction

In ExAnte, pruning away infrequent 1-itemsets gives us the opportunity of μ-reducing the transaction database in input, which in turn creates more infrequent 1-itemsets. In a level-wise computation, we collect new information level by level. If we can exploit such information to prune items from the transactions, we obtain new opportunities to μ-reduce again the transaction database.

After the introduction of Apriori (Agrawal and Srikant, 1994), a lot of other algorithms, sharing the same level-wise structure, have been proposed. Even if usually proposed as new algorithm with their own names, they can essentially be considered optimizations to the basic Apriori schema. Some of these proposed optimizations are data-reduction techniques, which, if exploited alone, bring little benefit to the computation. But if we couple such anti-monotonicity based optimizations with ExAnte's μ-reduction we can obtain dramatically effective optimizations.

In the rest of this section we review all such data reduction strategies which are based on the antimonotonicity of frequency. In the next section, we couple them with a monotone constraint, and thus with the μ-reduction, obtaining the ExAMiner algorithm.

In the following propositions, we indicate with k the actual iteration, where frequent k-itemsets are computed.

Proposition 3.1 (Antimonotone global pruning of an item). At the iteration k, a singleton item which is not subset of at least k frequent k-itemsets, will not be subset of any frequent $(k+1)$-itemset, and thus it can be pruned away from all transactions in the input database.

Proof. Every frequent $(k+1)$-itemset X has got $k+1$ frequent subsets of size k, which can be obtained by removing a singleton item $i \in X$. Thus, any singleton item $i \in X$ is contained in exactly k subsets of X having size k. □

Note that the ExAnte's α-reduction, i.e. the removal of infrequent singleton items from all transactions, is just the instantiation of this property for $k = 1$. In the pseudo-code of the algorithm in Section 5, we use an array of integers V_k, which records for each item the number of frequent k-itemsets in which it appears.

While the last proposition induce a pruning which is global to the whole database, the next two propositions (Park et al, 1995) induce pruning which are local to a transaction. In particular the first one removes the entire transaction (similarly to the μ-reduction), while the second one removes items from a single transaction.

Proposition 3.2 (Antimonotone pruning of a transaction). Given a transaction $\langle tID, X\rangle$, if X is not superset of at least $k+1$ frequent k-itemsets, then the transaction can be pruned away, since it will never be superset of any frequent $(k+1)$-itemset.

Proof. If a transaction $\langle tID, X\rangle$ contains a frequent $(k+1)$-itemset Y, then it must contain all $k+1$ frequent k-itemsets which are subsets of Y. □

Unfortunately, this property can not be exploited transaction by transaction, since it requires information that is available only at the end of the actual iteration, when all frequent itemsets have been counted. However, since the set of

frequent k-itemsets is a subset of the set of candidates k-itemsets, we can exploit a relaxed version of the previous property, transaction by transaction.

Corollary 3.1 (Weak antimonotone pruning of a transaction). Given a transaction $\langle tID, X\rangle$, if X is not superset of at least $k+1$ candidate k-itemsets, then the transaction can be pruned away since it will never be superset of any frequent $(k+1)$-itemset.

Proof. Trivially from Proposition 3.2, since the set of frequent itemsets at level k is a subsets of the set of candidate itemsets. □

This property can be checked locally for each transaction, when the transaction is used to count support for candidate itemsets. If a transaction does not participate to the count of at least $k+1$ candidate k-itemsets, it is pruned away, i.e. it will not be part of the transaction database at the next iteration. In the pseudo-code of the algorithm, for each transaction t, we use a counter $t.count$ for the number of candidate k-itemsets covered by t.

Actually, we can perform more local pruning in a transaction $\langle tID, X\rangle$. Suppose, we know for each item $i \in X$, the number of candidate k-itemsets which are superset of $\{i\}$ and subset of X.

Definition 3.1 (Multiplicity of an item w.r.t. a transaction). Given a transaction $\langle tID, X\rangle$ and an item i we define the multiplicity of i w.r.t. X at the iteration k:

$$M(i, X)_k = |\{Y \in C_k | Y \subseteq X, i \in Y\}|$$

as the number of candidate k-itemsets which are superset of $\{i\}$ and subset of X.

Proposition 3.3 (Antimonotone local pruning of an item). Given a transaction $\langle tID, X\rangle$, for each $i \in X$, if $M(i, X)_k < k$ then i can be removed from X after the support counting phase of iteration k.

Proof. If $M(i, X)_k < k$ then i will not be contained in any candidate $(k+1)$-itemset which is subset of X. Thus i can be removed from X after the support counting phase of iteration k. □

In the pseudo-code of the algorithm, for each transaction t at the iteration k, and for each item in t, we use a counter $i.count$ for keeping track of $M(i, X)_k$.

Clearly, every time we reduce the size of a transaction we create a new opportunity of pruning the transaction (μ-reduction of the dataset). The basic philosophy of ExAMiner is to write a new reduced transaction database for each iteration, exploiting all possible antimonotone and monotone pruning opportunities.

4. Further Pruning Opportunities

In the previous section we have introduced a set of possible pruning strategies, known in literature, which rely on anti-monotonicity of frequency. Then, we have indicated that the basic idea of ExAMiner is to couple this pruning with the μ-reduction of the database, induced by the monotone constraint, providing strong data reduction, since the two components strengthen each other. In this section we introduce novel powerful pruning strategies, which couple more tightly the antimonotone and monotone pruning.

When the monotone constraint is a cardinality constraint $\mathcal{C}_M \equiv card(S) \geq n$, the following proposition can be used to obtain a stronger pruning at a very low computational price. The following is a generalization of Proposition 3.1.

Proposition 4.1 (Enhanced global pruning of an item). At the iteration k, a singleton item which is not subset of at least $\binom{n-1}{k-1}$ frequent k-itemsets (where $k < n$), will not be subset of any frequent n-itemset.

Proof. Let $X = i_1 \dots i_n$ be a frequent n-itemset. For the anti-monotonicity of frequency all its subsets are frequent too. In particular, are frequent all its $\binom{n}{k}$ subset of size k. A generic item $i_p \in X$ is subset of exactly $\binom{n-1}{k-1}$ since this is the number of possible subsets of $X \setminus \{i_p\}$ of size $k-1$. Adding i_p to all this subsets we obtain the thesis. □

The same condition can be further exploited. Consider a generic iteration k of a generic level-wise computation. At the end of the *count&reduce* phase, we have generated the set of frequent k-itemsets L_k, which normally would be used to generate the set of candidate itemsets for the next iteration. For each item i, which does not satisfy the condition in the above proposition, we can prune from the set of generators L_k, each k-itemset containing it, before the generation phase starts. The following corollary of Proposition 4.1, represents the first proposal of a strategy for pruning the generators, since usually they are defined for pruning the candidate itemsets (the generated ones).

Corollary 4.1 (Generators Pruning). Consider the computational problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$ where $\mathcal{C}_M \equiv card(S) \geq n$. At the iteration $k < n$ of the level-wise computation, consider the set S_k of itemsets in L_k which contain at least a singleton item which do not appear in at least $\binom{n-1}{k-1}$ itemset of L_k.

$$S_k = \{X \in L_k | \exists i \in X, V_k[i] < m\} \; where \; m = \binom{n-1}{k-1}$$

In order to generate the set of candidates for the next iteration C_{k+1}, we can use as generators the itemsets in $L_k \setminus S_k$ without losing solutions to the given problem.

Analogously, the same kind of enhancement can be applied to the local pruning of an item from a transaction. The next corollary of Proposition 4.1, enhances Proposition 3.3, when we have to deal with a cardinality constraint.

Corollary 4.2 (Enhanced local pruning of an item). Consider the computational problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$ where $\mathcal{C}_M \equiv card(S) \geq n$. At the iteration k of the level-wise computation, consider a transaction $\langle tID, X \rangle$, for each $i \in X$, if

$$M(i, X)_k < \binom{n-1}{k-1}$$

then i can be pruned from X.

Similar pruning enhancements can be obtained also for all monotone constraints, inducing weaker conditions from the above propositions on cardinality.

Consider, for instance, the computational problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$ where $\mathcal{C}_M \equiv sum(S.price) \geq m$, and suppose that we are at the end of iteration k of the level-wise computation. As usual, we have recorded in V_k for each item the

number of frequent k-itemsets in which it appears. Consider an item i which, so far, is not in any solution. It is always possible to compute the maximum value of n for which continues to hold:

$$V_k[i] \geq \left(\begin{array}{c} n-1 \\ k-1 \end{array} \right)$$

This value of n represents an upper-bound for the maximum size of a frequent itemsets containing i. In the pseudo-code in the next Section we use a function **determine_max_n**$(V_k[i], k)$ to determine such upper-bound.

Therefore, if we sum the *price* of i with the prices of the $n-1$ most expansive items which are still alive, we can obtain an upper-bound for the total sum of prices of a frequent itemset containing i. In the pseudo-code we use a function **optimistic_monotone_value**$(i, \mathcal{I}, n, \mathcal{C}_m)$ to compute such upper-bound. If this sum is less then the monotone constraint threshold m, the item i can be globally pruned away, with all its supersets which are in L_k, since they can not be solutions.

This generators pruning techniques is twofold benefic. In fact, the proposed technique, not only reduces the generators, and hence the number of candidates at the next iteration, but it also reduces the number of checking of $\mathcal{C}_M$: in fact we prune itemsets from L_k which can not be solution, before the checking of $\mathcal{C}_M$.

A similar reasoning can be done to enhance the local pruning of items from a transaction for all kinds of monotone constraint.

Algorithm 2 *enhanced_local_pruning*

Input: $\mathcal{I}, \mathcal{C}_M, t$
Output: $drop_L[]$
1: **for all** $i \in t$ **do**
2: $drop_L[i] := false$
3: **if** $\mathcal{C}_M \equiv card(X) \geq n$ **then**
4: **for all** $i \in t$ **do**
5: **if** $i.count < \left(\begin{array}{c} n-1 \\ k-1 \end{array} \right)$ **then**
6: $drop_L[i] := true$
7: **else**
8: **for all** $i \in t$ **do**
9: $n = determine\_max\_n(i.count, k)$
10: **if** $optimistic\_monotone\_value(i, t, n, \mathcal{C}_M) \not\models \mathcal{C}_M$ **then**
11: $drop_L[i] := true$

5. ExAMiner Algorithm

Essentially ExAMiner is a breadth-first Apriori-like algorithm, which exploits antimonotone and monotone constraints to reduce the problem dimensions levelwise. Each transaction, before participating to the support count, is reduced as much as possible, and only if it survives to this phase, it is used to count the support of candidate itemsets. Each transaction which arrives to the counting phase at iteration k, is then reduced again as much as possible, and only of it survives to this second set of reductions, it is written to the transaction database for the

Algorithm 3 *enhanced_global_pruning*

Input: $\mathcal{I}, \mathcal{C}_M, V_k$
Output: $drop_G[]$
1: **for all** $i \in \mathcal{I}$ **do**
2: $drop_G[i] := false$
3: **if** $\mathcal{C}_M \equiv card(X) \geq n$ **then**
4: **for all** $i \in \mathcal{I}$ **do**
5: **if** $V_k[i] < \binom{n-1}{k-1}$ **then**
6: $drop_G[i] := true$
7: **else**
8: **for all** $i \in \mathcal{I}$ **do**
9: $n = determine\_max\_n(V_k[i], k)$
10: **if** $optimistic\_monotone\_value(i, \mathcal{I}, n, \mathcal{C}_M) \not\models \mathcal{C}_M$ **then**
11: $drop_G[i] := true$

next iteration. Therefore, in order to describe the proposed algorithm, is sufficient to provide the pseudo-code for the procedure which substitutes the counting phase of the Apriori algorithm. This new procedure is named *count&reduce*. In the following with $\mathcal{D}_k$ we indicate the transaction database at the iteration k.

The *count&reduce* procedure takes as parameters the actual transaction database $\mathcal{D}_k$, the set of candidates C_k, the minimum support threshold σ, the monotone constraint $\mathcal{C}_M$ and the array V_{k-1}. As already stated, each transaction t in the database passes through two series of reductions and tests. The first one, brings the transaction, if still alive, to be exploited in the supports count. The second one, brings the transaction, if still alive, to be written in the transaction database from the next iteration. The first reduction (lines from 4 to 6 of Algorithm 4) is the global pruning of items (Proposition 3.1 and 4.1) which exploits the information in the array V_{k-1}. After this reduction the transaction is first tested for its cardinality (line 9), and then it's tested against the monotone constraint (line 10, μ-reduction). Only if both tests are passed the transaction t is matched against candidate itemsets to increase their support counts (lines from 11 to 18). During this phase we collect other information regarding t and each item i in t: the number of candidates itemset contained in t (line 12), the multiplicity of i w.r.t. t at the current iteration (line 12, Definition 3.1), and the number of frequent itemsets containing i (line 18).

If the transaction t has arrived alive to this point, it has still to pass some tests in order to enter into the database for the next iteration. First of all we check if its cardinality is at least $k+1$ (line 19). Then we check if during the counting phase it has participated to the count of at least $k+1$ candidate k-itemsets (line 20, Proposition 3.1). After this, if the transaction is still alive, it is further reduced by enhanced local pruning of items (lines 22, 23 and 24). After this reduction we check again the size of the transaction t and if it satisfies the monotone constraint. If also these last two tests are positive the transaction t enters in the database for the next iteration. Finally, we collect information for the enhanced global pruning (line 28), and we perform the pruning of generators (lines from 29 to 32).

Algorithm 4 *count&reduce*

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

```
1: for all i ∈ I do
2:    V_k[i] := 0
3: for all tuples t in D_k do
4:    for all i ∈ t do
5:       if V_{k-1}[i] < k − 1 or drop_G[i] then
6:          remove i from t and from I
7:       else
8:          i.count := 0
9:    if |t| ≥ k then
10:      if C_M(t) then
11:         for all X ∈ C_k, X ⊆ t do
12:            X.count + +; t.count + +
13:            for all i ∈ X do
14:               i.coun + +
15:            if X.count == σ then
16:               L_k := L_k ∪ {X}
17:               for all i ∈ X do
18:                  V_k[i] + +
19:         if |t| ≥ k + 1 then
20:            if t.count ≥ k + 1 then
21:               drop_L[] := enhanced_local_pruning(I, C_M, t)
22:               for all i ∈ t do
23:                  if i.count < k or drop_L[i] then
24:                     remove i from t
25:               if |t| ≥ k + 1 then
26:                  if C_M(t) then
27:                     write t in D_{k+1}
28: drop_G[] := enhanced_global_pruning(I, C_M, V_k)
29: G_k := L_k
30: for all X ∈ G_k do
31:    if ∃i ∈ X : drop_G[i] then
32:       remove X from G_k
```

5.1. Moving Through the Levels

The proposed data reduction strategy, when instantiated at the first iteration ($k = 1$) corresponds to the ExAnte algorithm, with the unique difference that, in ExAnte, the computation starts again and again from the same level until there are pruning opportunities.

This approach can be exploited also at the other levels of the ExAMiner level-wise computation. In fact, since during a *count&reduce* round, we reduce the input dataset, as well as the search space, we could start again with another *count&reduce* round at the same level with the reduced transaction dataset. Therefore, we face a choice between different strategies.

On one hand, we can have a strictly level-wise computation, in the style of Apriori. On the other hand, we can stand for more than one *count&reduce*

round on each level. Between these two extremes we can have a whole range of computational strategies.

Essentially we have two dimensions:

- how many *count&reduce* rounds we admit,
- and for which levels of the level-wise computation.

There is something more. At the end of any *count&reduce* round of any iteration, we could also decide to go back to some previous level. For instance after the counting phase of the third level, before generating candidates for the fourth level, we could go back to the first level, and start again from the beginning but with a much smaller problem. We can go on this way until the input dataset is so reduced, that it only contains the data essential for the solutions. However, this opportunity does not seems appealing by the point of view of efficiency.

We have implemented and tested (see Section 6) three different version of ExAMiner:

- **ExAMiner$_0$**: it strictly moves on level-wise, allowing only one *count&reduce* round for each level. Also at level one, we do not have the ExAnte fix-point computation. This is the real generalization of the Apriori algorithm which uses the monotone constraint to reduce the input data and the search space.
- **ExAMiner$_1$**: it allows an undefined number of rounds, until a fix point is reached, only at the first level. This corresponds to an ExAnte preprocessing followed by a strictly level-wise ExAMiner computation (only one *count&reduce* round per level).
- **ExAMiner$_2$**: it allows an undefined number of *count&reduce* rounds, until a fix point is reached, only at the first two levels. Then, from the third iteration, the computation becomes strictly level-wise.

The rationale behind these implementation choices is that the first two iterations of the usual level-wise algorithm, are the most efficient, since the count can be performed directly, without using complex data structures. From the third iteration, the count become very expensive. Therefore, we have decided to reduce as much as possible the problem during the first two iterations, and then go on directly to compute the solution of the problem.

5.2. Run-through Example

We now show an example of execution of $ExAMiner_2$. Suppose that the transaction and price dataset in Table 2 are given. Suppose that we want to compute frequent itemsets ($\sigma = 3$) with a sum of prices ≥ 30. During the first iteration the total price of each transaction is checked to avoid using transactions which do not satisfy the monotone constraint. All transaction with a sum of prices ≥ 30 are used to count the support for the singleton items. Only the first transaction is discarded.

At the end of the count we find items b, d and h to be infrequent. Note that, if the first transaction had not been discarded, item h would have been counted as frequent. At this point we perform an α-reduction of the dataset: this means removing b, d and h from all transactions in the dataset.

After the α-reduction we have more opportunities to μ-reduce the dataset. In fact, transaction 3, which at the beginning has a total price of 34, now has its

item	price
a	10
b	20
c	8
d	22
e	15
f	10
g	6
h	5
i	10
j	5
k	18
l	14

(a)

tID	Itemset	Total price
1	g,h,i	21
2	a,d,i,k	60
3	a,c,g,h,j	34
4	i,l,j,k	47
5	f,h,k	33
6	c,e,j,k	46
7	a,c,g,l,j,k	61
8	c,e,g,i,j	44
9	f,g,i,j	31
10	c,f,g,i,j	39
11	b,c,e,g,i	59
12	a,d,g,k	56
13	e,g,i	31
14	a,b,i,l,j	59

(b)

tID	Itemset	Total price
2	a,i,k	38
4	i,l,j,k	47
6	c,e,j,k	46
7	a,c,g,l,j,k	61
8	c,e,g,i,j	44
11	c,e,g,i	39
12	a,g,k	34
13	e,g,i	31
14	a,i,l,j	39

(c)

tID	Itemset	Total price
8	e,g,i	31
11	e,g,i	31
13	e,g,i	31

(d)

Fig. 2. Run-through Example: price table (a), input transaction database (b), reduced transaction database at the end of the first level (c), reduced transaction database at the end of the second level (d).

total price reduced to 29 due to the pruning of h. This transaction can now be pruned away. The same reasoning holds for the transactions number 3 and 5.

At this point we count once again the support of alive items with the reduced dataset. The item f which initially has got a support of 3 now has become infrequent due to the pruning of transaction 5. We can α-reduce again the dataset, removing f from all transactions. Then we can μ-reduce again. In fact, after the removal of f, transactions 9 and 10 have a total price which is less than 30, and thus they are pruned too. We count once again the support of all itemsets and no one has become infrequent. We have reached a fix-point for level one. At this point the input transaction database is as in Figure 2(c). Moreover we have $L_1 = \{a, c, e, g, i, j, k, l\}$, and C_2 which contains all possible couples of items from L_1.

We start with the second level of the level-wise computation. The first set of reductions and tests (lines from 4 to 6 of Algorithm 4) produce no pruning at the beginning of the second level. The support counting phase starts. At the end of this phase we have $L_2 = \{ak, ce, cg, cj, eg, ei, gi, ij, jk, jl\}$.

The second part of reductions and test (lines from 11 to 14) produces no pruning since we are at the second level. We start another *count&reduce* round at the second level. Items a and l are globally pruned from all transactions because they appear in only one frequent 2-itemset: $V_2[a] = 1, V_2[l] = 1$. This pruning gives us more opportunities of μ-reducing the dataset. Transactions number 2, 12 and 14 can be pruned away since they no longer satisfy the monotone constraint. Thanks to this data reduction, at the end of another counting phase, we have a smaller set of frequent 2-itemsets: $L_2 = \{ce, cg, cj, eg, ei, gi, jk\}$. At this point item k appears in only one frequent 2-itemset, and hence is globally pruned from the transactions. Due to this pruning, transactions number 4, 6 and 7 satisfy no longer the monotone constraint and they are pruned too. The input transaction database has reduced from 14 to 3 transactions. We count using only these 3 transactions and find that $L_2 = \{eg, ei, gi\}$. Items c and j do not appear in any frequent 2-itemsets and they can be pruned away. The resulting transaction

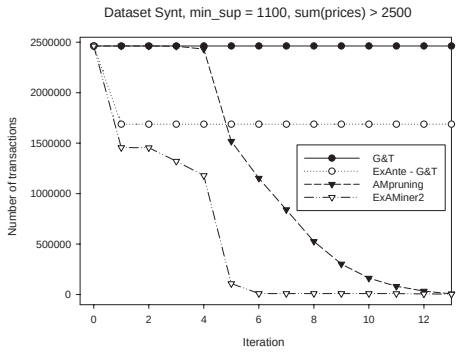


Fig. 3. Transaction reduction on dataset *Synt*.

database is as in Figure 2(d). Essentially, only the solution to the problem (egi) is alive in the database.

5.3. Dataset Rewriting

ExAMiner at each iteration rewrites a dataset of smaller size. The next iterations has thus to cope with a smaller input dataset than the previous one. The benefits are not only in term of I/O costs, but also of reduced work in subset counting due to the reduced number and size of transactions.

It is worth to observe that, as shown in (Orlando et al, 2002), when a dataset is sequentially scanned by reading fixed blocks of data, one can take advantage of the OS *prefetching*. Overlapping between computation and I/O activity can occur, provided that the computation granularity is large enough. Moreover, OS buffer caching and pipelining techniques virtualize I/O disk accesses, and, more importantly, small datasets (that we can obtain after few iterations from the initial dataset) can be completely contained in buffer cache or in main memory.

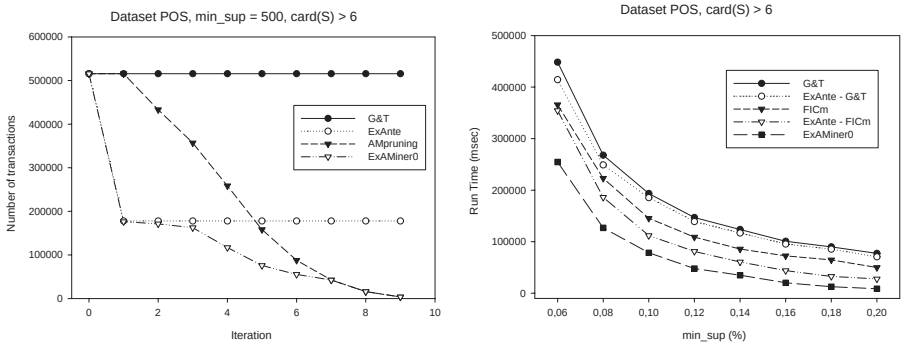
In summary, if granularity of computation is large enough and datasets are sequentially scanned, prefetching and buffer cache are able to hide I/O time.

In our first implementation we have chosen to rewrite the dataset at each iteration, no matter whether the benefit of the data reduction is worth the I/O cost. This choice was done to concentrate our study on data reduction. However, we can improve our algorithm by forcing it to rewrite the dataset only when the reduction is substantial. For instance, the data reduction rate shown in Figure 3 for the $ExAMiner_2$ algorithm suggest to quit rewriting the dataset after the big drop at the fifth iteration.

6. Experimental Results

The test bed architecture used in our experiments was a Windows2000 PC with a Pentium III processor running at 1000MHz and 320MB RAM.

In our experiment we used two different datasets with different characteristics. The first dataset, named "POS", was used in the KDD-Cup 2000 competition and it is described in (Zheng et al, 2001). The dataset is available from the



(a) (b)

Fig. 4. Dataset POS with cardinality constraint: transactions reduction (a), and run time comparison (b).

Dataset	Transactions	Items	Max Trans Size	Avg Trans Size
POS	515,597	1657	164	6.5
Synt	2,463,146	1000	43	14.8

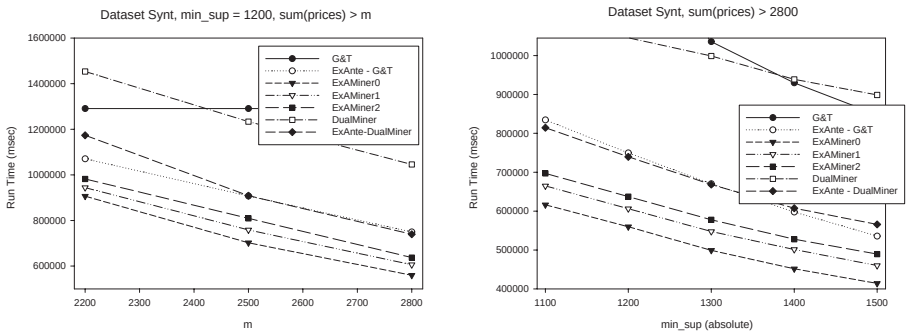
Table 2. Characteristics of datasets used in the experiments.

KDD-Cup 2000 home page[1]. "POS" is a real world dataset containing several years worth of point-of-sale data from a large electronic retailer, aggregated at the product category level. The second dataset, named *"Synt"* is a synthetic dataset obtained with the most commonly adopted dataset generator, available from IBM Almaden[2]. We associated a price to each item using a uniform distribution.

Figures 3 and 4(a) report the number of transactions considered, iteration by iteration, by different algorithms. *G&T* (generate and test) is a usual Apriori computation, followed by the filtering based on the monotone constraint; *G&T* does not exploit any data reduction technique, as it uses the whole initial dataset during the level-wise computation. *ExAnte-G&T* is a computation composed by an ExAnte preprocessing followed by a *G&T* computation: it reduces the dataset as much as possible at the first iteration. With *AMpruning* we denote a level-wise computation which uses the antimonotone pruning only: it corresponds to $ExAMiner$ with a trivial monotone constraint, e.g. $sum(prices) \geq 0$. The difference behavior of this computation and $ExAMiner_2$ indicates how the pruning is boosted when a monotone constraints is coupled with the frequency constraints.

A run-time comparison between different algorithms on dataset POS, with a monotone cardinality constraint, is reported in Figure 4(b). We have experimented also $\mathcal{FIC}^{\mathcal{M}}$ and *ExAnte-*$\mathcal{FIC}^{\mathcal{M}}$ (i.e. ExAnte preprocessing followed by $\mathcal{FIC}^{\mathcal{M}}$). We have avoided experimenting DualMiner on this dataset, since its dual pruning strategy performs very poorly with the cardinality constraint. It is interesting that, even with a scarcely selective monotone constraint, such as the cardinality constraint, $ExAMiner$ outperforms significantly all other competitors.

[1] http://www.ecn.purdue.edu/KDDCUP/
[2] http://www.almaden.ibm.com/cs/quest/syndata.html#assocSynData



(a) (b)

Fig. 5. Runtime comparison between different algorithms.

The performance improvements become clearer as the monotone constraints become more selective and/or the transaction database larger. In Figure 5(a) and (b) run-time comparisons on the *Synt* dataset with a $sum(prices) \geq m$ constraint are reported. On this dataset, in our test-bed, $\mathcal{FIC}^{\mathcal{M}}$ was not able to complete the computation due to the excessive memory request to allocate the FP-tree. The *ExAMiner* computation cut times by the half, and performance improves as the monotone constraint gets more selective (Figure 5(a)) or the minimum support threshold gets more selective (Figure 5(b)).

7. Different Kinds of Pattern

In this article we have described the ExAMiner algorithm for the mining of frequent itemsets with monotone constraints. However, the same methodology works as well for the mining of other kinds of more complex pattern from more structured data. Indeed, in any mining domain where the antimonotone frequency constraint is conjoined with some monotone constraints, we can exploit the same breadth-first data-reduction-based approach.

Consider for instance the problem of mining *frequent subgraphs* from a database of graphs. In (Inokuchi et al, 2000; Kuramochi and Karypis, 2001) have been proposed breadth-first Apriori-like algorithms to solve this problem. Analogously to what done for frequent itemsets we can conjoin the antimonotone pruning to the μ-reduction given by some monotone constraints.

Suppose that we want to mine frequent *connected* subgraphs containing at least 4 vertices from a database of graphs. The constraint of containing at least 4 vertices is clearly monotone. Suppose that the graph in Figure 6(a) is one of the graphs in the input database (that is the corresponding of a transaction in the frequent itemsets problem). This graph contains 18 vertices and its well far from not satisfying the monotone constraints. Suppose now that vertices H and L are found infrequent. We can α-reduce this graphs by pruning the two infrequent vertices. After this α-reduction we obtain the graph in Figure 6(b). The graph now contains 16 nodes but there is no connected substructure of at least 4 vertices: this graphs will never participate to the support count of any interesting (solution) subgraph. Therefore we have discovered that our graph,

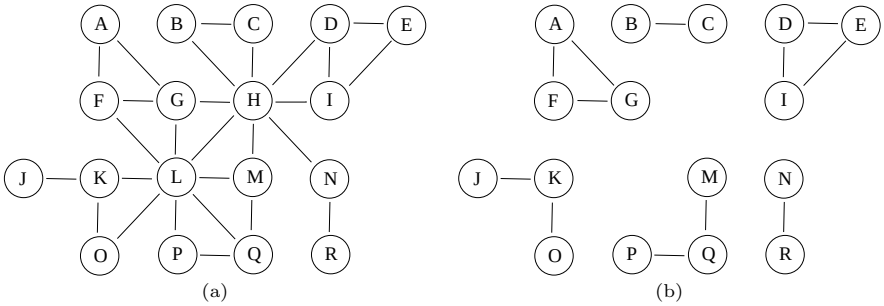


Fig. 6. Example of data-reduction for the mining of frequent connected subgraphs.

even if it was quite large, does not contain any interesting pattern and can be deleted (μ-reduced) from the input database.

In this last example we had a stronger synergy between antimonotonicity and monotonicity, due to the requirement of being a *connected* subgraph.

8. Conclusion and Future Improvements

This article introduced ExAMiner, a fast algorithm for frequent patterns mining that exploits antimonotone and monotone constraints to optimize a level-wise, Apriori-like computation. The strength of the algorithm lies in the fact that the two kinds of constraints are used synergistically to reduce both the itemset search space and the input database as the computation proceeds. As observed in the suite of experiments, ExAMiner exhibits a sensible improvement with respect to existing algorithms, from the different points of view of performance, reduction of the search space, and the ability to cope with extremely large transaction databases and extremely low support thresholds. Also, in its distinctive features of dynamically reducing the transaction database, the new algorithm exhibits such dramatic reduction to make it convenient to pay for the extra I/O overhead. We found this results striking, especially in view of the fact that the current implementation of ExAMiner is rather naive, and can be engineered, in principle, to further improve its performance. One possible improvement would be to avoid to dynamically change the transaction database if its reduction rate falls below some given threshold. Another improvement could be obtained by dynamically shifting to the vertical representation of the transaction database (TID lists) as soon as it fits into main memory, as done in Orlando et al (2002). Also, the new algorithm is in principle well-suited to parallel or distributed implementations, which are worth considering in the future. The achieved results and the possibility for further improvements bring us to the conclusion that ExAMiner may enable to deal with frequent patterns queries that are, to date, considered untractable.

Acknowledgements. We thank anonymous reviewers for their very useful comments and suggestions. We are indebted with Laks V.S. Lakshmanan, who first suggested the problem to the first author.

References

Agrawal R, Srikant R (1994) Fast Algorithms for Mining Association Rules in Large Databases. In Proceedings of the Twentieth International Conference on Very Large Databases, Santiago de Chile, Chile, September 1994, pp 487–499

Agrawal R, Imielinski T, Swami A (1993) Mining associations between sets of items in massive databases. In Proceedings of the ACM SIGMOD international conference on management of data, Washington DC, May 1993, pp 207–216

Bonchi F, Giannotti F, Mazzanti A, Pedreschi D (2003) Adaptive Constraint Pushing in Frequent Pattern Mining. In Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Cavtat-Dubrovnik, Croatia, September 2003. Lecture Notes in Computer Science 2838, Springer, Berlin, pp 47–58

Bonchi F, Giannotti F, Mazzanti A, Pedreschi D (2003) ExAnte: Anticipated Data Reduction in Constrained Pattern Mining. In Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Cavtat-Dubrovnik, Croatia, September 2003. Lecture Notes in Computer Science 2838, Springer, Berlin, pp 59–70

Boulicaut JF, Jeudy B (2002) Optimization of Association Rule Mining Queries. Intelligent Data Analysis Journal 6(4):341–357

Bucila C, Gehrke J, Kifer D, White W (2002) DualMiner: A Dual-Pruning Algorithm for Itemsets with Constraints. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 2002, Edmonton, Alberta, Canada, pp 42–51

De Raedt L, Kramer S (2001) The Levelwise Version Space Algorithm and its Application to Molecular Fragment Finding. In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, USA, August 2001, pp 853–862

Grahne G, Lakshmanan L, Wang X (2000) Efficient Mining of Constrained Correlated Sets. In Proceedings of the 16th IEEE International Conference on Data Engineering, March, 2000, San Diego, California, USA, pp 512–524

Han J, Lakshmanan L, Ng R (1999) Constraint-Based, Multidimensional Data Mining. Computer, 32(8): pp 46–50

Inokuchi A, Washio T, Motoda H (2000) An Apriori-based algorithm for mining frequent substructures from graph data. In Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases, Lyon, France, September 2000. Lecture Notes in Computer Science 1910, Springer, Berlin, pp 13–23

Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In Proceedings of the 2001 IEEE International Conference on Data Mining, December 2001,San Jose, California, USA pp 313–320

Lakshmanan L, Ng R, Han J, Pang A (1999) Optimization of Constrained Frequent Set Queries with 2-variable Constraints. In Proceedings ACM SIGMOD International Conference on Management of Data, June 1999, Philadelphia, Pennsylvania, USA, pp 157–168

Ng R, Lakshmanan L, Han J, Pang A (1998) Exploratory Mining and Pruning Optimizations of Constrained Associations Rules. In Proceedings ACM SIGMOD International Conference on Management of Data, June 1998, Seattle, Washington, USA, pp 13–24

Orlando S, Palmerini P, Perego R, Silvestri F (2002) Adaptive and Resource-Aware Mining of Frequent Sets. In Proceedings of the 2002 IEEE International Conference on Data Mining, December 2002, Maebashi City, Japan, pp 338–345

Park JS, Chen MS, Yu PS (1995) An Effective Hash Based Algorithm for Mining Association Rules. In Proceedings ACM SIGMOD International Conference on Management of Data, May 1995, San Jose, California, USA, pp 175–186

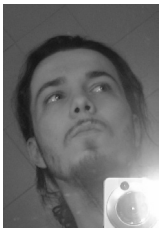
Pei J, Han J (2000) Can we push more constraints into frequent pattern mining? In Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 2000, Boston, MA, USA, pp 350–354

Pei J, Han J, Lakshmanan L (2001) Mining Frequent Item Sets with Convertible Constraints. In Proceedings of the 17th IEEE International Conference on Data Engineering, April 2001, Heidelberg, Germany, pp 433-442

Srikant R, Vu Q, Agrawal R (1997) Mining Association Rules with Item Constraints. In Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 1997, Newport Beach, California, USA, pp 67–73

Zheng Z, Kohavi R, Mason L (2001) Real World Performance of Association Rule Algorithms. In Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 2001, San Francisco, California, USA, pp 401–406

Author Biographies



Francesco Bonchi just received his Ph.D. in computer science from University of Pisa, with the thesis "Frequent Pattern Queries: Language and Optimizations". Actually he is a post-doc at Institute of Information Science and Technologies (ISTI) of the Italian National Research Council in Pisa where he is a member of the Knowledge Discovery and Delivery Laboratory. He has been a visiting fellow at the Kanwal Rekhi School of Information Technology, Indian Institute of Technology, Bombay (2000, 2001). His current research interests are data mining query language and optimization, frequent pattern mining, privacy-preserving data mining, web mining. He is one of the teachers for a course on data mining held at the faculty of Economics at the University of Pisa. He served as a referee at various national and international conferences on databases, data mining, logic programming and artificial intelligence. He is in the organizing committee for ECML-PKDD '04 conference. Contact him at francesco.bonchi@isti.cnr.it; http://www-kdd.isti.cnr.it/∼bonchi/

Fosca Giannotti is a senior researcher at Institute of Information Science and Technologies (ISTI) of the Italian National Research Council in Pisa where she leads the Knowledge Discovery and Delivery Laboratory. From 1982 to 1985 she was a research assistant, Dip. Informatica, Univ. Pisa. From 1985 to 1989 she was a Senior Researcher at R&D Lab. of Sipe Optimization, Pisa and at R&D Lab. of Systems and Management, Pisa. In 1989/90 she was a visiting researcher of MCC, Austin, Texas, USA, involved in the LDL (Logic Database Language) project. Her current research interests include data mining query languages, knowledge discovery support environment, web-mining, spatio-temporal reasoning, and spatio-temporal data mining. She has taught classes on databases and data mining at universities in Italy and abroad, and has coordinated a master course on data mining in collaboration with the faculty of Economics at the University of Pisa. She served in the organization and in the scientific committee of various conferences in the area of logic programming, databases and data mining. She is one of the program chairs for the ECML-PKDD '04 conference. Contact her at fosca.giannotti@isti.cnr.it

Dino Pedreschi is a full professor at the Computer Science Department of the University of Pisa. He has been a visiting scientist and professor at the University of Texas at Austin (1989/90), at CWI Amsterdam (1993) and at UCLA (1995). His current research interests are in logic in databases, and particularly in data analysis, in deductive databases, in the integration of data mining and database querying, in spatio-temporal reasoning, and in formal methods for deductive computing. He has taught classes on programming languages and databases in universities in Italy and abroad, and is collaborating with F. Giannotti in course on data mining at the faculty of Economics at the University of Pisa. He participated in the scientific committee of various conferences in the area of Logic Programming and Databases, including the LID'96 Workshop on Logic in Databases, where he was program co-chair, and the LPNMR'99 Workshop on Logic Programming and Non Monotonic Reasoning, 1999. He is one of the program chairs for ECML-PKDD '04 conference. Contact him at pedre@di.unipi.it

Alessio Mazzanti just received a Laurea degree from University of Pisa, with a thesis on constrained frequent pattern mining. He is a software architect at LIST S.p.A., Pisa. Contact him at mazzanta@cli.di.unipi.it; http://www.cli.di.unipi.it/∼mazzanta/

Correspondence and offprint requests to: Francesco Bonchi, KDD Laboratory, ISTI Area della Ricerca C.N.R. di Pisa, Via Giuseppe Moruzzi, 1 - 56124 Pisa, Italy. Email: bonchi@di.unipi.it