ExAMiner:
Optimized Level-wise Frequent Pattern Mining with Monotone Constraints

Francesco Bonchi, Fosca Giannotti
Pisa KDD Laboratory
ISTI - C.N.R. Area della Ricerca di Pisa
Via Giuseppe Moruzzi 1, 56124 Pisa, Italy
bonchi@di.unipi.it, f.giannotti@cnuce.cnr.it

Alessio Mazzanti, Dino Pedreschi
Pisa KDD Laboratory
Dipartimento di Informatica, Università di Pisa
Via F. Buonarroti 2, 56127 Pisa, Italy
mazzanta@cli.di.unipi.it, pedre@di.unipi.it

Abstract

The key point of this paper 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 problem input together with the search space. Following this intuition, we introduce ExAMiner, a level-wise algorithm which exploits the real synergy of anti-monotone 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 [4], 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 anti-monotone constraint. Experimental results confirm that this is, so far, the most efficient way of attacking the computational problem in analysis.

1. Introduction

Constrained itemset 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 [6, 8, 11, 12]. The class of anti-monotone constraints is the most effective and easy to use in order to prune the search space. Since any conjunction of anti-monotone constraints is in turn anti-monotone, we can use the *Apriori pruning method*: the more anti-monotone 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 anti-monotone and monotone constraints has been studied for a long time [8, 11], 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 anti-monotone constraints we face a tradeoff between anti-monotone 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 [4] 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 anti-monotone pruning opportunities, on the contrary, such opportunities are boosted. Dually, pushing anti-monotone constraints boosts monotone pruning opportunities: the two components strengthen each other. On the basis of these considerations we have introduced ExAnte [4], a pre-processing data reduction algorithm which reduces dramatically both the search space and the input dataset in constrained frequent pattern mining.

In this paper 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**nti-monotone 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.

2. Problem Definition

Let $Items = \{x_1, ..., x_n\}$ be a set of distinct literals, usually called **items**. An **itemset** X is a non-empty subset of $Items$. 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** TDB is a set of transactions. An itemset X is **contained** in a transaction $\langle tID, Y\rangle$ if $X \subseteq Y$. Given a transaction database TDB the subset of transaction which contain an itemset X is named $TDB[X]$. The **support** of an itemset X, written $supp_{TDB}(X)$ is the cardinality of $TDB[X]$. Given a user-defined **minimum support** δ, an itemset X is called **frequent** in TDB if $supp_{TDB}(X) \geq \delta$. This defines the frequency constraint $\mathcal{C}_{freq}[TDB]$: if X is frequent we write $\mathcal{C}_{freq}[TDB](X)$ or simply $\mathcal{C}_{freq}(X)$. Let $Th(\mathcal{C}) = \{X|\mathcal{C}(X)\}$ denotes 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 itemsets mining problem* requires to compute $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$.

We now formally define the notion of anti-monotone and monotone constraints.

Definition 1. *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 clearly anti-monotone. 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. 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 2. *Given an itemset X, a constraint $\mathcal{C}_M$ is strongly monotone if: $\forall Y \supseteq X : \mathcal{C}_M(X) \Rightarrow \mathcal{C}_M(Y)$ independently from the given input transaction database.*

Note that in the last definition we have required a strongly monotone constraint to be satisfied independently from the given input transaction database. This is necessary since we want to distinguish between simple monotone constraints and global constraints such as the *"infrequency constraint"* ($supp_{TDB}(X) \leq \delta$). This constraint is still monotone but has different properties since it is dataset dependent and it requires dataset scans in order to be computed. Since our algorithm reduces the transaction dataset, we want to exclude the infrequency constraint from our study. In the rest of this paper, we write, for the sake of brevity, monotone instead of strongly monotone.

The general problem that we consider in this paper is the mining of itemsets which satisfy a conjunction of monotone and anti-monotone constraints: $Th(\mathcal{C}_{AM}) \cap Th(\mathcal{C}_M)$. Since any conjunction of anti-monotone constraints is an anti-monotone constraint, and any conjunction of monotone constraints is a monotone constraint, we just consider two constraints: one per class. In particular, we choose frequency ($\mathcal{C}_{AM} \equiv supp_{TDB}(X) \geq \delta$) as anti-monotone constraint, in conjunction with various simple monotone constraints: $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$.

3. Related Work

In [11] a FP-tree based algorithm for mining frequent itemset with monotone constraints, is introduced. Strictly speaking, this algorithm, named $\mathcal{FIC}^{\mathcal{M}}$, 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 against a pure *generate & test* algorithm is that it 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.

In [7] it is shown that pushing monotone constraints can lead to a reduction of anti-monotone pruning. Therefore, when dealing with a conjunction of monotone and anti-monotone constraints we face a tradeoff between anti-monotone and monotone pruning. This work has a pure theoretical value, as no practical algorithm is proposed.

The first work trying to find an amalgam between anti-monotone and monotone pruning is **DualMiner** [5]. However, DualMiner does not compute all solution itemsets with their support and thus 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 it uses multiple scan 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. For the same reason, it can not exploit data reduction techniques as those ones introduced in this paper. Finally, in real-world problems, even on dense datasets, or at very low support levels, frequency remains always a very selective constraint: 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.

In [3] we have introduced a general adaptive constraint pushing (***ACP***) strategy. Both monotone and anti-monotone pruning are exploited in a level-wise computation. Level by level, while acquiring new knowledge about the dataset characteristics and selectivity of constraints, *ACP* adapts its behavior giving more power to one pruning over the other in order to maximize efficiency. The proposed algorithm exhibits interesting performances when attacking very dense datasets, on which data reduction strategies are less effective. Thus, *ACP* more than a concurrent of ExAMiner can be seen as an alternative when the input dataset is very dense.

4. Search Space and Input Data Reduction

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. In fact, as proved by the following key theorem [4], monotone constraints can prune away transactions from the input dataset *without losing solutions*.

Proposition 1 (μ-reduction). *Given a transaction database TDB and a monotone constraint $\mathcal{C}_M$, we can prune from TDB the transactions that do not satisfy $\mathcal{C}_M$, without changing the supports to solutions itemsets.*

This monotone pruning of transactions has got another positive effect: while reducing the number of transactions in input it reduces the support of items too, hence the total number of frequent 1-itemsets. In other words, the monotone pruning of transactions strengthens the anti-monotone pruning. Moreover, infrequent items can be deleted by the computation and hence pruned away from the transactions in the input dataset (we named this pruning α-reduction). This anti-monotone pruning has got another positive effect: reducing the size of a transaction which satisfies a monotone constraint can make the transaction violates the monotone constraint. Therefore a growing number of transactions which do not satisfy the monotone constraint can be found. We are clearly inside a loop where two different kinds of pruning cooperates 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 ExAnte preprocessing [4] which is generalized to a level-wise computation in this paper.

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 [1], 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 possible data reduction strategies which are based on the anti-monotonicity of frequency. Coupling them with a monotone constraint, and thus with the μ-reduction, we obtaini the ExAMiner algorithm.

In the following propositions [9], we indicate with k the actual iteration, where frequent k-itemsets are computed.

Proposition 2 (Anti-monotone 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.*

In the pseudo-code of the algorithm, we use an array of integers V_k (of the size of $Items$), which records for each item the number of frequent k-itemsets in which it appears. While the last proposition induce a pruning which is global to the whole database, the next two propositions [10] induce pruning which are local to a transaction.

Proposition 3 (Anti-monotone local 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.*

This property can be checked locally for each transaction t, during the support counting phase, by keeping track of the number of candidate k-itemsets covered by t ($t.count$ in the pseudo-code).

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. We call this number multiplicity of i w.r.t. X at the iteration k and indicate it as $M(i,X)_k$.

Proposition 4 (Anti-monotone 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 pruned from X.*

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 computing the multiplicity of i w.r.t. t at the iteration 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).

5. Further Pruning Opportunities

In this section we introduce novel powerful pruning strategies, which couple more tightly the anti-monotone 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 2.

Proposition 5 (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 $1 < k < n$), will not be subset of any frequent n-itemset.*

The same condition can be further exploited. Consider a generic iteration k of a generic level-wise computation. At the end of the count 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 5, represents the first proposal of a strategy for pruning the generators, since usually pruning strategies are defined for pruning the candidate itemsets (the generated ones).

Corollary 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 ($1 < k < n$) of the level-wise computation, consider the set S_k of itemsets in L_k which contain at least a singleton item which does not appear in at least $\binom{n-1}{k-1}$ itemsets 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 5, enhances Proposition 4, when we have to deal with a cardinality constraint.

Corollary 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 ($1 < k < n$) 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, $\mathcal{C}_M \equiv sum(S.price) \geq m$, and suppose that we are at the end of iteration k of the level-wise computation. As usual, we have recorded in $V_k[i]$ the number of frequent k-itemsets in which i appears. It is always possible to compute the maximum value of n for which continues to hold: $V_k[i] \geq \binom{n-1}{k-1}$. This value of n represents an upper-bound for the maximum size of a frequent itemset containing i. In the pseudo-code we use a function $\mathbf{determine\_max\_n}(V_k[i], k)$ 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 $\mathbf{optimistic\_monotone\_value}(i, Items, n, \mathcal{C}_M)$ 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.

6. ExAMiner Algorithm

Essentially ExAMiner is an Apriori-like algorithm, which exploits anti-monotone and monotone constraints to reduce the problem dimensions level-wise. 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 if it survives to this second set of reductions, it is written to the transaction database for the 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 TDB_k we indicate the transaction database at the iteration k.

As already stated, each transaction t in the database passes through two series of reductions and tests. The first reduction (lines 3 and 4 of the pseudo-code) is the global pruning of items (Proposition 2 and 5) which exploits the in-

Procedure: **enhanced_local_pruning**$(Items, \mathcal{C}_M, t)$

1. **forall** $i \in t$ **do** $drop_L[i] = false$;
2. **if** $\mathcal{C}_M \equiv card(X) \geq n$ **then**
3.   **forall** $i \in t$ **do**
4.   **if** $i.count < \binom{n-1}{k-1}$ **then** $drop_L[i] = true$;
5. **else forall** $i \in t$ **do**
6. $n = determine\_max\_n(i.count, k)$;
7.   **if** $optimistic\_monotone\_value(i, t, n, \mathcal{C}_M) \not\vDash \mathcal{C}_M$
8.   **then** $drop_L[i] = true$
9. **return** $drop_L[\,]$

Procedure: **enhanced_global_pruning**$(Items, \mathcal{C}_M, V_k)$

1. **forall** $i \in Items$ **do** $drop_G[i] = false$;
2. **if** $\mathcal{C}_M \equiv card(X) \geq n$ **then**
3.   **forall** $i \in Items$ **do**
4.   **if** $V_k[i] < \binom{n-1}{k-1}$ **then** $drop_G[i] = true$;
5. **else forall** $i \in Items$ **do**
6. $n = determine\_max\_n(V_k[i], k)$;
7.   **if** $optimistic\_monotone\_value(i, Items, n, \mathcal{C}_M) \not\vDash \mathcal{C}_M$
8.   **then** $drop_G[i] = true$
9. **return** $drop_G[\,]$

Procedure: **count&reduce**$(min\_supp, C_k, TDB_k, \mathcal{C}_M, V_{k-1})$

1. **forall** $i \in Items$ **do** $V_k[i] = 0$;
2. **forall** tuples t in TDB_k **do**
3.   **forall** $i \in t$ **do if** $((V_{k-1}[i] < k-1)$ **or** $drop_G[i])$
4.   **then remove** i **from** t; **remove** i **from** $Items$;
5.   **else** $i.count = 0$;
6.   **if** $|t| \geq k$ **then**
7.   **if** $\mathcal{C}_M(t)$ **then**
8.   **forall** $X \in C_k, X \subseteq t$ **do**
9. $X.count$++; $t.count$++;
10.   **forall** $i \in X$ **do** $i.count$++;
11.   **if** $X.count == min\_supp$ **then**
12. $L_k = L_k \cup \{X\}$;
13.   **forall** $i \in X$ **do** $V_k[i] + +$;
14.   **if** $|t| \geq k+1$ **then**
15.   **if** $t.count \geq k+1$ **then**
16. $drop_L$[]=**enhanced_local_pruning**$(Items, \mathcal{C}_M, t)$;
17.   **forall** $i \in t$ **if** $((i.count < k)$ **or** $drop_L[i])$
18.   **then remove** i **from** t;
19.   **if** $|t| > k$ **then**
20.   **if** $\mathcal{C}_M(t)$ **then** $write(t, TDB_{k+1})$;
21. $drop_G[\,]$=**enhanced_global_pruning**$(Items, \mathcal{C}_M, V_k)$;
22. **forall** $X \in L_k$ **do**
23.   **if** $\exists i \in X \; : \; drop_G[i]$ **then remove** X **from** L_k;

formation in the array V_{k-1}. After this reduction the transaction is first tested for its cardinality (line 6), and then it's tested against the monotone constraint (line 7, μ-reduction). Only if both tests are passed the transaction t is matched against candidate itemsets to increase their support counts (lines from 8 to 12). During this phase we collect other information regarding t and each item i in t: the number of candidates itemset contained in t (line 9), the multiplicity of i w.r.t. t at the current iteration (line 10), and the number of frequent itemsets containing i (line 13).

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 14). Then we check if during the counting phase it has participated to the count of at least $k+1$ candidate k-itemsets (line 15, Proposition 3). After this, if the transaction is still alive, it is further reduced by enhanced local pruning of items (lines 16, 17 and 18). 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 21), and we perform the pruning of generators (line 23).

6.1. Moving Through the Levels

The proposed data reduction strategy, when instantiated at the first iteration, 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: (i) how many *count&reduce* rounds we admit, (ii) and for which levels of the level-wise computation. We have implemented and tested (see Section 7) three different versions of ExAMiner:

- **ExAMiner$_0$**: it strictly moves on level-wise, allowing only one *count&reduce* round for each level. 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.

- **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.

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
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
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
8	e,g,i	31
11	e,g,i	31
13	e,g,i	31

(d)

Figure 1. Run-through Example

6.2. Run-through Example

We now show an example of execution of $ExAMiner_2$. Suppose that the transaction and price dataset in Figure 1 (a) and (b) are given. Suppose that we want to compute frequent itemsets ($min\_supp = 3$) 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 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 1(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.

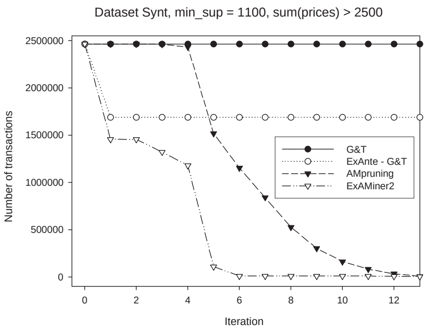
We start with the second level of the level-wise computation. The first set of reductions and tests (lines from 3 to 5 of the pseudo-code) produce no pruning at the beginning of the second level. The support counting phase starts. At the end of this phase we have $L_2 = \{ak, ce, cg, cj, eg, ei, gi, ij, jk, jl\}$.

The second part of reductions and test 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 database is as in Figure 1(d). Essentially, only the solution to the problem (egi) is alive in the database.

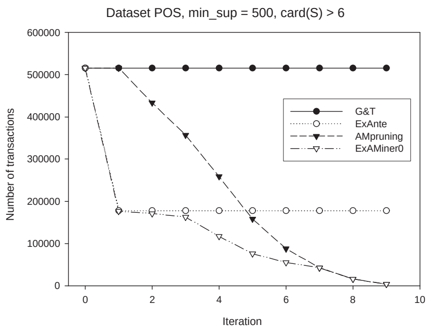
6.3. Dataset Rewriting

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 [2], 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



(a)



(b)

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

(c)

Figure 2. Data reduction rate (a) (b), and characteristics of the datasets (c).

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(a) for the $ExAMiner_2$ algorithm suggests to quit rewriting the dataset after the big drop at the fifth iteration.

7. 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 [13]. The dataset is available from the 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 2 (a) and (b) 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 anti-monotone pruning only: it corresponds to $ExAMiner$ with a trivial monotone constraint, e.g. $sum(prices) \geq 0$. The different behavior of this computation and $ExAMiner_2$ indicates how the pruning is boosted when a monotone constraint is coupled with the frequency constraint.

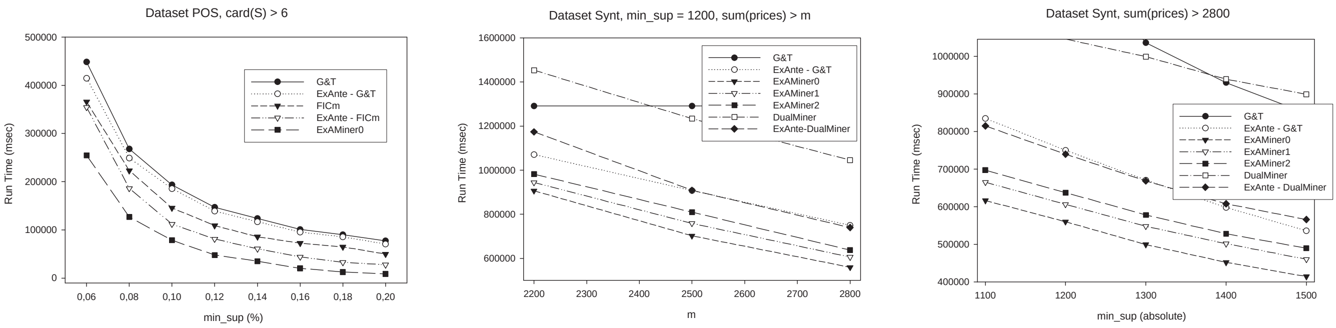
A run-time comparison between different algorithms on dataset POS, with a monotone cardinality constraint, is reported in Figure 3(a). We have experimented also $\mathcal{FIC}^{\mathcal{M}}$ and *ExAnte*-$\mathcal{FIC}^{\mathcal{M}}$ (i.e. ExAnte preprocessing followed by $\mathcal{FIC}^{\mathcal{M}}$). 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.

The performance improvements become clearer as the monotone constraints become more selective and/or the transaction database larger. In Figure 3(b) and (c) 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 3(b)) or the minimum support threshold gets more selective (Figure 3(c)).

8. Conclusion and Future Work

This paper introduced ExAMiner, a fast algorithm for frequent pattern mining that exploits anti-monotone and monotone constraints to optimize a level-wise, Apriori-like computation. The strength of the algorithm lies in the

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



(a) (b) (c)

Figure 3. Runtime comparison between different algorithms.

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 feature 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 [9]. 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 pattern queries that are, to date, considered untractable.

Acknowledgements. We are indebted with Laks V.S. Lakshmanan, who first suggested the problem to the first author. This work was supported by "Fondazione Cassa di Risparmio di Pisa" under the "WebDigger Project".

References

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 487–499, Santiago, Chile, 1994.

[2] R. Baraglia, D. Laforenza, S. Orlando, P. Palmerini, and R. Perego. Implementation issues in the design of I/O intensive data mining applications on clusters of workstations. In *Proc. of the 3rd Workshop on High Performance Data Mining, LNCS 1800*, pages 350–357, 2000.

[3] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Adaptive Constraint Pushing in frequent pattern mining. In *Proc. of the 7th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD03)*, 2003.

[4] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAnte: Anticipated data reduction in constrained pattern mining. In *Proc. of the 7th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD03)*, 2003.

[5] C. Bucila, J. Gehrke, D. Kifer, and W. White. DualMiner: A dual-pruning algorithm for itemsets with constraints. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

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

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

[8] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *(SIGMOD-98)*, pages 13–24.

[9] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri. Adaptive and Resource-Aware Mining of Frequent Sets. In *(ICDM'02)*, pages 338–345, Maebashi City, Japan, 2002.

[10] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash based algorithm for mining association rules. In *SIGMOD95*, pages 175–186, 1995.

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

[12] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining, KDD*, pages 67–73, 1997.

[13] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.