Adaptive Constraint Pushing in Frequent Pattern Mining

Francesco Bonchi[†‡], Fosca Giannotti[†], Alessio Mazzanti,[‡] and Dino Pedreschi[‡]

Pisa KDD Laboratory
http://www-kdd.cnuce.cnr.it

[†]ISTI - CNR Area della Ricerca di Pisa, Via Giuseppe Moruzzi, 1 - 56124 Pisa, Italy
e-mail `Giannotti@cnuce.cnr.it`

[‡]Department of Computer Science, University of Pisa
Via F. Buonarroti 2, 56127 Pisa, Italy
e-mail {`bonchi,mazzanti,pedre`}`@di.unipi.it`

Abstract. Pushing monotone constraints in frequent pattern mining can help pruning the search space, but at the same time it can also reduce the effectiveness of anti-monotone pruning. There is a clear tradeoff. Is it better to exploit more monotone pruning at the cost of less anti-monotone pruning, or viceversa? The answer depends on characteristics of the dataset and the selectivity of constraints. In this paper, we deeply characterize this trade-off and its related computational problem. As a result of this characterization, we introduce an adaptive strategy, named *ACP* (Adaptive Constraint Pushing) which exploits any conjunction of monotone and anti-monotone constraints to prune the search space, and level by level adapts the pruning to the input dataset and constraints, in order to maximize efficiency.

1 Introduction

Constrained itemsets mining is a hot research theme in data mining [3–6]. 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 also known as the *apriori trick* [1]: this is a valuable heuristic that drastically reduces the search space making the computation feasible in many cases. 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 constraints and in general we talk about *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. Classes of constraints sharing nice properties have been individuated. 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 an anti-monotone constraint, we can use all the constraints in a conjunction to make the *apriori trick* more selective.

The dual class, monotone constraints, has been considered more complicated to exploit and less effective in pruning the search space. As highlighted by Boulicaut and Jeudy in [3], 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.

In [2] we have shown that the above consideration holds only if we focus completely on the search space of all itemsets, which is the approach followed so far. With the novel algorithm ExAnte we have shown that an effective way of attacking the problem is to reason 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 recursively. ExAnte is a pre-processing data reduction algorithm which reduces dramatically both the search space and the input dataset. It can be coupled with any constrained patterns mining algorithm, and it is always profitable to start any constrained patterns computation with an ExAnte preprocess. Anyway, after the ExAnte preprocessing, when computing frequent patterns we face again the tradeoff between anti-monotone and monotone pruning.

The Tradeoff

Suppose that an itemset has been removed from the search space because it does not satisfy a monotone constraint. This pruning avoids checking support for this itemset, but however if we check its support and find it smaller than the threshold, we may prune away all the supersets of this itemset. In other words, by monotone pruning we risk to loose anti-monotone pruning opportunities given by the removed itemset. The tradeoff is clear [3]: pushing monotone constraint can save tests on anti-monotone constraints, however the results of these tests could have lead to more effective pruning.

On one hand we can exploit all the anti-monotone pruning with an *apriori* computation, checking the monotone constraint at the end, and thus not performing any monotone constraint pushing. We call this strategy *g&t* (generate and test). On the other hand, we can exploit completely any monotone pruning opportunity, but the price to pay is less anti-monotone pruning. We call this strategy *mcp* (monotone constraint pushing).

No one of the two extremes outperforms the other on every input dataset and conjunction of constraints. The best strategy depends of the characteristics of the input and the optimum is usually in between the two extremes.

In this paper, we introduce a general strategy, *ACP*, that balances the two extremes adaptively. Both monotone and anti-monotone pruning are exploited in a level-wise computation. Level by level, while acquiring new knowledge about the dataset and selectivity of constraints, ACP adapts its behavior giving more power to one pruning over the other in order to maximize efficiency.

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 $k = |X|$ 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 the definition of the frequency constraint $\mathcal{C}_{freq}[TDB]$: if X is frequent we write $\mathcal{C}_{freq}[TDB](X)$ or simply $\mathcal{C}_{freq}(X)$ when the dataset is clear from the context. 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 $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$.

Definition 1. Given an itemset X, a constraint $\mathcal{C}_{AM}$ is anti-monotone if:

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

Definition 2. Given an itemset X, a constraint $\mathcal{C}_{M}$ is monotone if:

$$\forall Y \supseteq X : \mathcal{C}_{M}(X) \Rightarrow \mathcal{C}_{M}(Y)$$

independently from the given input transaction database.

Observe that the independence from the input transaction database 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 it is dataset dependent and it requires dataset scans in order to be computed. Since any conjunction of monotone constraints is a monotone constraint, in this paper we consider the problem:

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

The concept of border is useful to characterize the solution space of the problem.

Definition 3. Given an anti-monotone constraint $\mathcal{C}_{AM}$ and a monotone constraint $\mathcal{C}_{M}$ we define their borders as:

$$B(\mathcal{C}_{AM}) = \{X|\forall Y \subset X : \mathcal{C}_{AM}(Y) \wedge \forall Z \supset X : \neg\, \mathcal{C}_{AM}(Z)\}$$
$$B(\mathcal{C}_{M}) = \{X|\forall Y \supset X : \mathcal{C}_{M}(Y) \wedge \forall Z \subset X : \neg\, \mathcal{C}_{M}(Z)\}$$

Moreover, we distinguish between positive and negative borders. Given a general constraint $\mathcal{C}$ we define:

$$B^{+}(\mathcal{C}) = B(\mathcal{C}) \cap Th(\mathcal{C}) \quad B^{-}(\mathcal{C}) = B(\mathcal{C}) \cap Th(\neg\, \mathcal{C})$$

tID	itemset
1	b,d
2	b,d,e
3	b,c,d,e
4	c,e
5	c,d,e
6	c,d
7	a,d,e

item	price
a	4
b	3
c	5
d	6
e	15

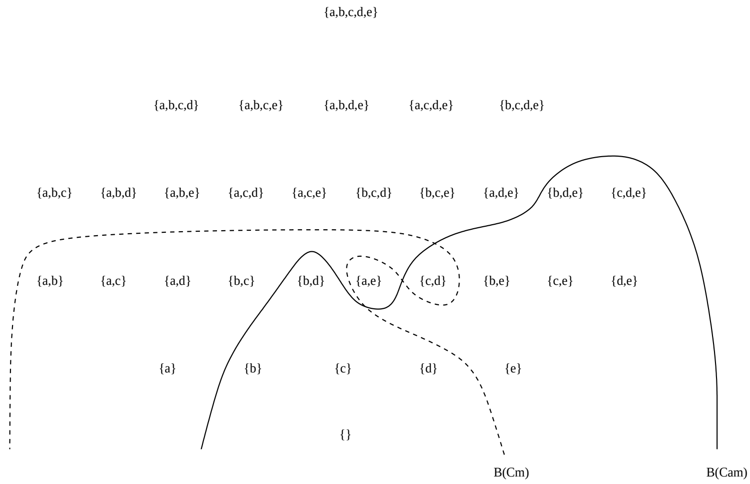

Fig. 1. The borders $B(\mathcal{C}_M)$ and $B(\mathcal{C}_{freq})$ for the transaction database and the price table on the left with $\mathcal{C}_M \equiv sum(X.prices) \geq 12$ and $\mathcal{C}_{AM} \equiv supp_{TDB}(X) \geq 2$.

In Figure 1 we show the borders of two constraints: the anti-monotone constraint $supp(X) \geq 2$, and the monotone one $sum(X.prices) \geq 12$. In the given situation the borders are:

$$B^+(\mathcal{C}_M) = \{e, abc, abd, acd, bcd\} \quad B^+(\mathcal{C}_{freq}) = \{bde, cde\}$$
$$B^-(\mathcal{C}_M) = \{ab, ac, ad, bc, bd, cd\} \quad B^-(\mathcal{C}_{freq}) = \{a, bc\}$$

The solutions to our problem are the itemsets that lie under the anti-monotone border and over the monotone one: $R = \{e, be, ce, de, bde, cde\}$.

Our Contributions:

In the next section we provide a through characterization of the addressed computational problem, and we compare the two opposite extreme strategies. Then we introduce a general adaptive strategy, named ACP, which manages the two extremes in order to adapt its behavior to the given instance of the problem. The proposed strategy has the following interesting features:

- It exploits both monotone and anti-monotone constraints in order to prune the search space.
- It is able to adapt its behavior to the given input in order to maximize efficiency. This is the very first adaptive algorithm in literature on the problem.
- It computes the support of every solution itemset, which is necessary when we want to compute association rules.
- Being a level-wise solution it can be implemented exploiting the many optimization techniques and smart data structure studied for the apriori algorithm.

2 Level-wise Solutions

Strategy *g&t* performs an *apriori* computation and then tests among frequent itemsets which ones satisfy also the monotone constraint. Strategy *mcp*, introduced by Boulicaut and Jeudy [3] works the opposite. The border $B^+(\mathcal{C}_M)$ is considered already computed and is given in input. Only itemsets over that border, and hence in $Th(\mathcal{C}_M)$, are generated by a special generation procedure $\mathbf{generate_m}$. Therefore we just need to check frequency for these candidates. The procedure $\mathbf{generate_m}$ takes in input the set of the solutions at the last iteration R_k, the border $B^+(\mathcal{C}_M)$, and the maximal cardinality of an element in $B^-(\mathcal{C}_M)$, which we denote $maxb$. In the rest of this paper we use $Items_k$ to denote the set of all k-itemsets.

Procedure: $\mathbf{generate_m}(k, R_k, B^+(\mathcal{C}_M))$

---

if $k = 0$ **then return** $B^+(\mathcal{C}_M) \cap Items$
else if $k \leq maxb$ **then**
return $generate_1(R_k, Items) \cup (B^+(\mathcal{C}_M) \cap Items_{k+1})$
else if $k > maxb$ **then**
return $generate_{apriori}(R_k)$

Where:

- $generate_1(R_k, X) = \{A \cup B | A \in R_k \wedge B \in X\}$
- $generate_{apriori}(R_k) = \{X | X \in Items_{k+1} \wedge \forall Y \in Items_k : Y \subseteq X \,.\, Y \in R_k\}$

The procedure $\mathbf{generate_m}$ creates as candidates only supersets of itemsets which are solution at the last iteration. Thus these candidates only need to be checked against $\mathcal{C}_{freq}$ since they surely satisfy $\mathcal{C}_M$. These candidates are generated adding to a solution a 1-itemset. Unluckily we can not use the apriori trick completely with this strategy. In fact a candidate itemset can not be pruned away simply because all its subsets are not solution, since some of them could have not been considered at all. What we can do is prune whenever we know that at least one subset of the candidate itemset is not a solution because it does not satisfy $\mathcal{C}_{freq}$. This pruning is performed on the set of candidates C_k by the following procedure.

Strategy: ***mcp***

$C_1 := generate_m(0, \emptyset, B^+(\mathcal{C}_M)); R_0 := \emptyset;\ k := 1$
while $C_k \neq \emptyset$ **or** $k \leq maxb$ **do**
$C_k := prune_m(R_{k-1}, C_k)$
test $\mathcal{C}_{\mathbf{freq}}(\mathbf{C_k})$; $R_k := Th(\mathcal{C}_{freq}) \cap (C_k)$
$C_{k+1} := generate_m(k, R_k, B^+(\mathcal{C}_M));\ k := k + 1$
end while
return $\bigcup_{i=1}^{k-1} R_i$

Procedure: $\mathbf{prune_m}(R_{k-1}, C_k)$

$C' := C_k$
for all $S \in C_k$ **do for** all $S' \subseteq S\ :\ S' \in Items_{k-1}$
do if $S' \notin R_{k-1} \wedge \mathcal{C}_m(S')$ **then** remove S from C'
return C'

Example 4. Consider the executions of strategy *mcp* and strategy *g&t* on the dataset and the constraints in Figure 1, focussing on the numbers of checking of $\mathcal{C}_{freq}$. At the first iteration strategy *mcp* produces a unique candidate $C_1 = \{e\}$ which is the only 1-itemset in $B^+(\mathcal{C}_M)$. This candidate is checked for the anti-monotone constraint and it results to be a solution $R_1 = \{e\}$. At the second iteration 4 candidates are produced $C_2 = \{ae, be, ce, de\}$. Only ae does not satisfy $\mathcal{C}_{AM}$, hence $R_2 = \{be, ce, de\}$. At the third iteration 7 candidates are produced $C_3 = \{abc, abd, acd, bcd, bce, bde, cde\}$. Only two of these pass the anti-monotone checking: $R_3 = \{bde, cde\}$. Finally $C_4 = \emptyset$. Therefore, with the given dataset and constraints, strategy *mcp* performs $1+4+7 = 12$ checking of $\mathcal{C}_{freq}$. Strategy *g&t* uses a normal *apriori* computation in order to find $Th(\mathcal{C}_{freq})$ and then check the satisfaction of $\mathcal{C}_M$. It performs 13 checking of $\mathcal{C}_{freq}$.

Example 5. This example is borrowed by [3]. Suppose we want to compute frequent itemsets $\{X | supp(X) \geq 100 \wedge |X| \geq 10\}$. This is a conjunction of an anti-monotone constraint (frequency) with a monotone one (cardinality of the itemset ≥ 10). Strategy *mcp* generates no candidate of size lower than 10. Every itemset of size 10 is generated as candidate and tested for frequency in one database scan. This leads to at least $\binom{n}{10}$ where $n = |Items|$ candidates and, as soon as n is large this turns to be intractable. On the other hand strategy *g&t* generates candidates that will never be solutions, but this strategy remains tractable ever for large n.

The two examples show that no one of the two strategies outperforms the other on every input dataset and conjunction of constraints.

2.1 Strategies Analysis

We formally analyze the search space explored by the two extreme level-wise strategies. To this purpose we focus on the number of frequency tests, since the monotone constraint is cheaper to test.

Definition 6. Given a strategy $\mathcal{S}$ the number of frequency test performed by $\mathcal{S}$ is indicated with $|\mathcal{C}_{freq}|_{\mathcal{S}}$.

Generally, a strategy $\mathcal{S}$ checks for frequency a portion of $Th(\mathcal{C}_M)$ (which can produce solutions) and a portion of $Th(\neg\mathcal{C}_M)$ (which can not produce solutions):

$$|\mathcal{C}_{freq}|_{\mathcal{S}} = \gamma|Th(\neg\mathcal{C}_M)| + \beta|Th(\mathcal{C}_M)| \qquad \gamma, \beta \in [0,1] \tag{1}$$

The *mcp* strategy has $\gamma = 0$, but evidently it has a β much larger than strategy *g&t*, since it can not benefit from the pruning of infrequent itemsets in $Th(\neg\mathcal{C}_M)$, as we formalize later. Let us further characterize the portion of $Th(\neg\mathcal{C}_M)$ explored by strategy *g&t* as:

$$\gamma|Th(\neg\mathcal{C}_M)| = \gamma_1|Th(\neg\mathcal{C}_M) \cap Th(\mathcal{C}_{freq})| + \gamma_2|Th(\neg\mathcal{C}_M) \cap B^-(\mathcal{C}_{freq})| \tag{2}$$

For *g&t* strategy $\gamma_1 = \gamma_2 = 1$: it explores all frequent itemsets ($Th(\mathcal{C}_{freq})$) and candidate frequent itemsets that results to be infrequent ($B^-(\mathcal{C}_{freq})$) even if they are in $Th(\neg\mathcal{C}_M)$ and thus can not produce solutions. Let us examine what happens over the monotone border. We can further characterize the explored portion of $Th(\mathcal{C}_M)$ as:

$$\beta|Th(\mathcal{C}_M)| = \beta_1|B^+(\mathcal{C}_M) \cap (Th(\neg\mathcal{C}_{freq}) \setminus B^-(\mathcal{C}_{freq}))| + \beta_2|R| + \beta_3|B^-(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)| \tag{3}$$

Trivially $\beta_2 = 1$ for any strategy which computes all the solutions of the problem. Moreover, also β_3 is always 1 since we can not prune these border itemsets in any way. The only interesting variable is β_1, which depends from γ_2. Since the only infrequent itemsets checked by strategy *g&t* are itemsets in $B^-(\mathcal{C}_{freq})$, it follows that for this strategy $\beta_1 = 0$. On the other hand, strategy *mcp* generates as candidates all itemsets in $B^+(\mathcal{C}_M)$ (see line 3 of $generate_m$ procedure), thus for this strategy $\beta_1 = 1$. the following proposition summarizes the number of frequency tests computed by the two strategies.

Proposition 7.

$$|\mathcal{C}_{freq}|_{g\&t} = |Th(\neg\mathcal{C}_M) \cap Th(\mathcal{C}_{freq})| + |Th(\neg\mathcal{C}_M) \cap B^-(\mathcal{C}_{freq})| + |R| + |B^-(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)|$$

$$|\mathcal{C}_{freq}|_{mcp} = |B^+(\mathcal{C}_M) \cap (Th(\neg\mathcal{C}_{freq}) \setminus B^-(\mathcal{C}_{freq}))| + |R| + |B^-(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)|$$

In the next section we introduce an adaptive algorithm which manages this tradeoff, balancing anti-monotone and monotone pruning w.r.t. the given input.

3 Adaptive Constraint Pushing

The main drawback of strategy *g&t* is that it explores portions of search space in $Th(\neg\mathcal{C}_M)$ which will never produce solutions ($\gamma_1 = 1$); while the main drawback of strategy *mcp* is that it can generate candidate itemsets in $Th(\neg\,\mathcal{C}_{freq}) \setminus B^-(\mathcal{C}_{freq})$ that would have been already pruned by a simple *apriori* computation ($\beta_1 = 1$). This is due to the fact that strategy *mcp* starts computation bottom-up from the monotone border and has no knowledge about the portion of search space below such a border($Th(\neg\,\mathcal{C}_M)$). However, some knowledge about small itemsets which do not satisfy $\mathcal{C}_M$ could be useful to have a smaller number of candidates over the border. But on the other hand, we need some additional computation below the monotone border in order to have some knowledge. Once again we face the tradeoff. The basic idea of a general adaptive pushing strategy (ACP) is to explore only a *portion* of $Th(\neg\,\mathcal{C}_M)$: this computation will never create solutions, but if well chosen it can prune heavily the computation in $Th(\mathcal{C}_M)$. In other terms, it tries to balance γ and β_1. To better understand we must further characterize the search space $Th(\neg\mathcal{C}_M)$:

$$\gamma|Th(\neg\mathcal{C}_M)| = \gamma_1|Th(\neg\mathcal{C}_M) \cap Th(\mathcal{C}_{freq})| + \gamma_2|Th(\neg\mathcal{C}_M) \cap B^-(\mathcal{C}_{freq})| + \gamma_3|Th(\neg\mathcal{C}_M) \cap (Th(\neg\mathcal{C}_{freq}) \setminus B^-(\mathcal{C}_{freq}))| \tag{4}$$

Strategy ACP tries to reduce γ_1 but the price to pay is a possible reduction of γ_2. Since the portion of search space $Th(\neg\mathcal{C}_M) \cap B^-(\mathcal{C}_{freq})$ is helpful to prune, a reduction of γ_2 yields a reduction of pruning opportunities. This can lead to the exploration of a portion of search space ($\gamma_3 > 0$) that would have not been explored by a *g&t* strategy. This phenomenon can be seen as a virtual raising of the frequency border: suppose we loose an itemset of the frequency border, we will later explore some of its supersets and obviously find them infrequent. By the point of view of the strategy these are frequency border itemsets, even if they are not really in $B^-(\mathcal{C}_{freq})$. The optimal *ideal* strategy would have $\gamma_1 = \gamma_3 = 0$ and $\gamma_2 = 1$ since we are under the monotone border and we are just looking for infrequent itemsets in order to have pruning in $Th(\mathcal{C}_M)$. Therefore, a general ACP strategy should explore a portion of $Th(\neg\mathcal{C}_M)$ in order to find infrequent itemsets, trying not to loose pieces of $Th(\neg\mathcal{C}_M) \cap B^-(\mathcal{C}_{freq})$.

Proposition 8.

$$|\mathcal{C}_{freq}|_{ideal} = |Th(\neg\mathcal{C}_M) \cap B^-(\mathcal{C}_{freq})| + |R| + |B^-(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)|$$

$$|\mathcal{C}_{freq}|_{acp} = \gamma|Th(\neg\mathcal{C}_M)| + \beta|Th(\mathcal{C}_M)| \text{ (as defined in equations (3) and (4)).}$$

Two questions arise:

1. *What is a "good" portion of candidates?*
2. *How large this set of candidates should be?*

The answer to the first question is simply: *"itemsets which have higher probabilities to be found infrequent"*. The answer to the second question is what the adaptivity of ACP is about. We define a parameter $\alpha \in [0, 1]$ which represents the fraction of candidates to be chosen among all possible candidates. This parameter is initialized after the first scan of the dataset using all information available, and it is updated level by level with the newly collected knowledge.

Let us now introduce some notation useful for the description of the algorithm.

- $L_k \subseteq \{I | \, I \in (Items_k \cap Th(\mathcal{C}_{freq}) \cap Th(\neg\, \mathcal{C}_M))\}$
- $N_k \subseteq \{I | \, I \in (Items_k \cap Th(\neg\, \mathcal{C}_{freq}) \cap Th(\neg\, \mathcal{C}_M))\}$
- $R_k = \{I | \, I \in (Items_k \cap Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M))\}$
- $P_k = \{I | \, I \in Items_k \wedge \forall n, m < k.(\nexists L \subset I.L \in B_m) \wedge (\nexists J \subset I.J \in N_n)\}$
- $B_k = \{I | \, I \in (B^+(\mathcal{C}_M) \cap P_k)\}$
- $E_k = \{I | \, I \in (Th(\neg\, \mathcal{C}_M) \cap P_k)\}$

L_k is the set of frequent k-itemsets which are under the monotone border: these have been checked for frequency even if they do not satisfy the monotone constraint hoping to find them infrequent; N_k is the set of infrequent k-itemsets under the monotone border: these are itemsets used to prune over the monotone border; R_k is the set of solutions k-itemsets; P_k is the set of itemsets potentially frequent (none of their subsets have been found infrequent) and potentially in $B^+(\mathcal{C}_M)$ (none of their subsets have been found satisfying $\mathcal{C}_M$). B_k is the subset of elements in P_k which satisfy $\mathcal{C}_M$ and hence are in $B^+(\mathcal{C}_M)$ since all their subsets are in $Th(\neg\, \mathcal{C}_M)$. E_k is the subset of elements in P_k which still do not satisfy $\mathcal{C}_M$. From this set is chosen an α-portion of elements to be checked against frequency constraint named $C_k^{\mathcal{U}}$ (candidates $\mathcal{U}$nder). This selection is indicated as $\alpha \otimes E_k$. Finally we have the set of candidates in which we can find solutions $C_k^{\mathcal{O}}$ (candidates $\mathcal{O}$ver) which is the set of candidates over the monotone border. The frequency test for these two candidates sets is performed with a unique database scan and data structure. Itemsets in $C_k^{\mathcal{O}}$ which satisfy $\mathcal{C}_{freq}$ will be solutions; itemsets in $C_k^{\mathcal{U}}$ which do not satisfy $\mathcal{C}_{freq}$ go in N_k and will prune itemsets in $C_j^{\mathcal{O}}$ for some $j > k$.

We now introduce the pseudo-code for the generic adaptive strategy. In the following with the sub-routine $\mathbf{generate_{over}}$, we mean $\mathbf{generate_1}$ followed by the pruning of itemsets which are superset of itemsets in N (we call it $\mathbf{prune_{am}}$), followed by $\mathbf{prune_m}$ (decribed in Section 2).

Strategy: generic ***ACP***

$R_0, N := \emptyset; \quad C_1 := Items$
test $\mathcal{C}_{\mathbf{freq}}(\mathbf{C_1}) \Rightarrow C_1 := Th(\mathcal{C}_{freq}) \cap (C_1)$
test $\mathcal{C}_{\mathbf{M}}(\mathbf{C_1}) \Rightarrow R_1 := Th(\mathcal{C}_M) \cap (C_1);\ L_1 := Th(\neg\,\mathcal{C}_M) \cap (C_1)$
$P_2 := generate_{apriori}(L_1)$
$k := 2$
while $P_k \neq \emptyset$ **do**
    test $\mathcal{C}_{\mathbf{M}}(\mathbf{P_k}) \Rightarrow B_k := Th(\mathcal{C}_M) \cap (P_k); E_k := Th(\neg\mathcal{C}_M) \cap (P_k)$
    $C_k^{\mathcal{O}} := generate_{over}(R_{k-1}, C_1, N) \cup B_k$
    $initialize/update(\alpha)$
    $C_k^{\mathcal{U}} := \alpha \otimes E_k$
    test $\mathcal{C}_{\mathbf{freq}}(\mathbf{C_k^{\mathcal{U}}} \cup \mathbf{C_k^{\mathcal{O}}}) \Rightarrow$
    $R_k := Th(\mathcal{C}_{freq}) \cap C_k^{\mathcal{O}};\ L_k := Th(\mathcal{C}_{freq}) \cap C_k^{\mathcal{U}};\ N_k := Th(\neg\mathcal{C}_{freq}) \cap C_k^{\mathcal{U}}$
    $N := N \cup N_k$
    $P_{k+1} := generate_{apriori}(E_k \setminus N_k)$
    k := k+1
end while
$C_k := generate_{over}(R_{k-1}, C_1, N)$
while $C_k \neq \emptyset$ **do**
    test $\mathcal{C}_{\mathbf{freq}}(\mathbf{C_k})$; $R_k := Th(\mathcal{C}_{freq}) \cap (C_k)$
    $C_{k+1} := generate_{apriori}(R_k)$
    $k := k + 1$
end while
return $\bigcup_{i=1}^{k-1} R_i$

It is worthwhile to highlight that the pseudo-code given in Section 2 for strategy *mcp*, which is a theoretical strategy, does not perform the complete first anti-monotone test, while strategies *ACP* and *g&t* perform it. This results to be a reasonable choice on our toy-example, but it turns to be a suicide choice on every reasonably large dataset. Anyway, we can imagine that any practical implementation of *mcp* would perform at least this first anti-monotone test. We call this practical implementation strategy mcp^*. Moreover, strategy mcp^* does not take the monotone border in input but it discovers it level-wise as *ACP* does. In our experiments we will use mcp^* instead of *mcp*.

Note that:

- if $\alpha = 0$ constantly, then $ACP \equiv$ strategymcp^*;
- if $\alpha = 1$ constantly, then $ACP \equiv$ strategy *g&t*;

Our adaptivity parameter can be seen as a setting knob which ranges from 0 to 1, from an extreme to the other. To better understand how *ACP* works, we show the execution given the input in Figure 1.

3.1 Run-through Example

Strategy ACP starts with $C_1 = \{a, b, c, d, e\}$, tests the frequency constraint, tests the monotone constraint and finds the first solution, $R_1 = \{e\}$ and $L_1 = \{b, c, d\}$. Now (Line 4) we generate the set of 2-itemsets potentially frequent and potentially in $B^+(\mathcal{C}_M)$: $P_2 = \{bc, bd, cd\}$. At this point we enter in the loop from line 6 to 15. The set P_2 is checked for $\mathcal{C}_M$, and it turns out that no element in P_2 satisfies the monotone constraint: thus $B_2 = \emptyset$, $E_2 = P_2$. At line 8 ACP generates candidates for the computation over the monotone border $C_2^{\mathcal{O}} = \{be, ce, de\}$ and performs the two pruning procedure that in this case have no effects. At this point ACP initializes our adaptivity parameter $\alpha \in [0, 1]$. The procedure *initialize*(α) can exploit all the information collected so far, such as number of transactions, total number of 1-itemsets, support threshold, number of frequent 1-itemsets and their support, number of solutions at the first iteration. For this example suppose that α is initialized to 0.33. Line 10 assigns to $C_k^{\mathcal{U}}$, a portion equals to α of E_k. Intuitively, since we want to find infrequent itemsets in order to prune over the monotone border, the best third is the 2-itemset which has the subsets with the lowest support, therefore $C_2^{\mathcal{U}} = \{bc\}$. Line 11 performs the frequency test for both set of candidates sharing a unique dataset scan. The count of support gives back four solutions $R_2 = \{be, ce, de\}$, moreover we have $L_2 = \emptyset$ and $N_2 = \{bc\}$. Then ACP generates $P_3 = \emptyset$ (line 13) and exits the loop (line 6). At line 16 we generate $C_3 = \{bde, cde\}$, we check their support (line 18) and obtain that $R_3 = C_3$; finally we obtain $C_4 = \emptyset$ and we exit the second loop. Algorithm ACP performs $5 + 4 + 2 = 11$ tests of frequency.

4 Adaptivity Strategies and Optimization Issues

In the previous section we have introduced a generic strategy for adaptive constraint pushing. This can not really be considered an algorithm since we have left not instantiated the *initialize/update* function for the adaptivity parameter α (line 9), as well as the α-selection (line 10). In this section we propose a very simple adaptivity strategy for α and our first experimental results. We believe that many other different adaptivity strategies can be defined and compared.

Since we want to select itemsets which are most likely infrequent, the simplest idea is to estimate on the fly, using all information available at the moment, a support measure for all candidates itemsets below the monotone border. Then the α-selection (line 10) will simply choose among all itemsets in E_k the α-portion with lowest estimated support.

In our first set of experiments, we have chosen to estimate the support for an itemset using only the real support value of items belonging to the given itemset, and balancing two extreme conditions of the correlation measure among the items: complete independence and maximal correlation. In the former the estimated itemset support is obtained as the product, and in the latter as the minimum, of relative support of the items belonging to the itemset. Also for the α-adaptivity we have chosen a very simple strategy. The parameter α is initialized w.r.t. the number of items which satisfy frequency and monotone constraint at the first iteration. Then at every new iteration it adapts its value according to the results of the α-selection at the previous iteration. Let us define the α-focus as the ratio of itemsets found infrequent among α-selected itemsets. An α-focus very close to 1 (greater than 0.98) suggests that we have selected and counted too few candidates and thus we raise the α parameter for the next iteration accordingly. An α-focus less then 0.95 suggests that we are selecting and counting too much candidates and thus produces a shrink of α.

These two proposed strategies for estimating candidates support and for the adaptivity of α do not exploit all available information, but they allow an efficient implementation, and they experimentally exhibit very good candidates-selection capabilities.

Experimental Results

Since ACP balances the tradeoff between frequency and a monotone constraint, it gives the best performance when the two components are equally strong, i.e. no constraint is much more selective than the other. On sparse datasets frequency is always very selective even at very low support levels: joining it with an equally strong monotone constraint would result in an empty set of solutions. Therefore, ACP is particularly interesting in applications involving dense datasets.

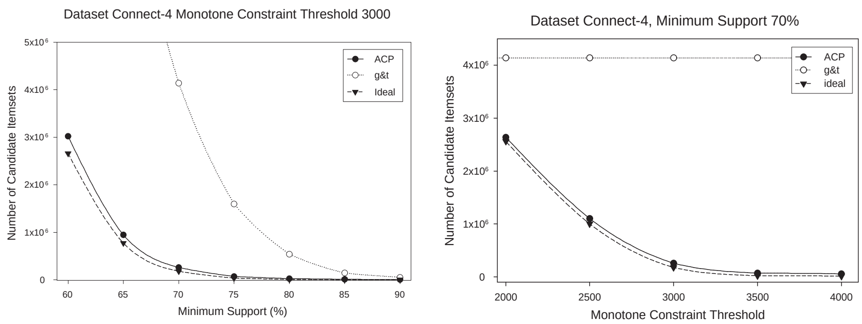

Fig. 2. Number of candidate itemsets tested against $\mathcal{C}_{freq}$.

In Figure 2, we show a comparison of the 4 strategies *g&t*, mcp^*, *ideal* and ACP, based on the portion of search space explored, i.e. the number of $\mathcal{C}_{freq}$ tests performed, on the well-known dense dataset *connect-4* [1], for different support thresholds and monotone constraints. In order to create a monotone constraint we have attached to each item a value v selected using a normal distribution. Then we have chosen as monotone constraint the sum of values v in an itemset to be greater than a given threshold.

Strategy mcp^* always performs very poorly and its results could not be reported in the graph in Figure 2. Strategy *g&t* explores a portion of search space that obviously does not depend by the monotone constraint. On this dense

[1] http://www.ics.uci.edu/∼mlearn/MLRepository.html

dataset it performs poorly and becomes hard to compute for low supports (less than 55%). Our simple strategy for selecting candidates under the monotone border provides a very good performance: during the first 3-4 iterations (where is more important not to miss infrequent itemsets) we catch all the infrequent itemsets with an $\alpha \approx 0.2$; i.e. checking only a fifth of all possible candidates. Thanks to this capability, our ACP strategy does not loose low-cardinality itemsets in $B^-(\mathcal{C}_{freq})$ and thus approximates very well the *ideal* strategy, as showed by Figure 2, performing a number of $\mathcal{C}_{freq}$ tests one order of magnitude smaller than strategy *g&t*.

5 Conclusions

In this paper, we have deeply characterized the problem of the computation of a conjunction of monotone and anti-monotone constraints. As a result of this characterization, we introduce a generic adaptive strategy, named ACP (Adaptive Constraint Pushing) which exploits any conjunction of monotone and anti-monotone constraints to prune the search space. We have introduce an adaptivity parameter, called α which can be seen as a setting knob which ranges from 0 (favorite monotone pruning) to 1 (favorite anti-monotone pruning); and level by level adapts itself to the input dataset and constraints, giving more power to one pruning over the other in order to maximize efficiency. The generic algorithmic architecture presented can be instantiated with different adaptivity strategy for α. In this paper we have presented a very simple strategy which does not do not exploit all available information, but it still provides very good selection capability and it allows an efficient implementation.

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. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAnte: Anticipated data reduction in constrained pattern mining. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD03)*, 2003.
3. J.-F. Boulicaut and B. Jeudy. Using constraints during set mining: Should we prune or not? In *Actes des Seizime Journes Bases de Donnes Avances BDA'00, Blois (F)*, pages 221–237, 2000.
4. G. Grahne, L. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In *16th International Conference on Data Engineering (ICDE' 00)*, pages 512–524. IEEE, 2000.
5. R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, volume 27,2 of *ACM SIGMOD Record*, pages 13–24, New York, June 1–4 1998. ACM Press.
6. J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *ICDE'01*, pages 433–442, 2001.