

Incremental Mining of Constrained Association Rules

Ahmed Ayad^{*}, *Nagwa El-Makky*^{**} and *Yousry Taha*^{***}

Abstract

The problem of mining association rules has attracted a lot of attention in the research community. Several techniques for efficient discovery of association rules have appeared. However, it is nontrivial to perform incremental mining or efficient mining of constrained association rules, in spite of their practical benefits. The research community has recently focused on providing separate solutions for these two problems.

Since many believe that constrained mining will be the standard, incremental mining of constrained rules will be necessary. In this paper, a new algorithm, *ICAP*, for incremental mining of constrained association rules is introduced. The concept of constrained negative border is also introduced and its usage for the maintenance of constrained association rules when new transaction data is added to a transactional database is proposed. This fast incremental mining technique is applied to the constrained association mining algorithm *CAP*. A key feature of the proposed algorithm is that it requires at most one scan of the original database only if the database insertions cause the constrained negative border to expand. Thus the speed-up of incremental mining is combined with the flexibility of the framework of constrained frequent-set queries, *CFQs*, used for specifying user constraints on the

^{*} Computer Sciences Department, University of Wisconsin – Madison, ahmed@cs.wisc.edu. Work done when author was in graduate school, Faculty of Engineering, Alexandria University, Alexandria Egypt.

^{**} Computer Science Dept., Faculty of Engineering, Alexandria University, Alex., Egypt, nagwamakky@hotmail.com.

^{***} Computer Science Dept., Faculty of Engineering, Alexandria University, Alex., Egypt, yousry_taha@hotmail.com.

produced association rules. The correctness of the proposed algorithm is studied and a proof of it is given.

Several experiments were conducted to measure the relative performance of the new algorithm compared to the single alternative approach available so far, which is rerunning the *CAP* algorithm on the whole updated database. The results of the experiments show a significant improvement over rerunning *CAP* in almost all of the cases.

Key words: Data mining, association rules, incremental mining, constrained frequent-set queries.

1. Introduction

Since its first introduction in [2], mining association rules has been a hot topic in the data mining research community. Several algorithms were introduced to solve the problem [3, 4, 7, 10, 18, 22, 24, 27, 28]. However, it was not until its first introduction in [11] that the problem of incrementally mining associations began to attract attention. Since then, several proposals to solve the problem appeared [6, 12, 17, 21, 26]. Those proposals serve to optimize the discovery of the new frequent itemsets after a non-trivial update of the database. Parallel to those efforts, constrained mining of association rules was also investigated on several levels. Such investigations can be found in [8, 15, 16, 20, 25]. In an interactive mining environment, it becomes a necessity to enable the user to express his interests through constraints on the discovered rules, and to change these interests interactively. In [20], Ng et al. presented a new model for exploratory mining of association rules to enable true interactive mining through ad-hoc queries that specify several constraints of different types. The problem of constrained rule mining is how to integrate the constraints into the core of rule generation to avoid generation of unnecessary rules earlier. In the same paper, Ng. et al. introduced the notion of anti-monotonicity and succinctness of a constraint by which the constraint can be *pushed deeply* into the algorithm of mining association rules. They also introduced the concept of constrained frequent-set queries (*CFQs*).

It can be seen that the research community has proposed separate solutions for the incremental mining and the constrained mining problems. However, incremental mining of constrained association rules is motivated by the fact that soon, constrained mining of associations will be the standard and methods will be needed to maintain the discovered rules after database updates [6]. In this paper, we present a new algorithm called *ICAP* (Incremental Constrained APriori) for incremental mining of constrained association rules. This algorithm incorporates the benefits of fast incremental mining into the framework of constrained frequent-set queries (*CFQs*). We introduce the concept of constrained negative border and apply it in the context of *CFQs*. The proposed algorithm requires a full scan of the whole database only if the constrained negative border expands. The new algorithm is based on the fast incremental mining algorithm in [26] and the constrained mining algorithm *CAP* in [20]. Thus, it combines the speed-up of fast incremental mining with the flexibility of the framework of *CFQs*.

The rest of the paper is organized as follows: section 2 presents the bases on which the proposed algorithm is built. Section 3 gives a formal description of the problem. It defines the constrained negative border and proves that this border is an indicator for the necessity of checking the original database. Section 4 presents the new algorithm in detail and gives a descriptive example. Section 5 deals with the method by which the constrained negative boarder is calculated. Section 6 gives a

detailed performance study for the proposed algorithm. Finally, section 7 concludes the paper.

2. Foundations of The Proposed Algorithm

This section presents the previous theoretical foundations which form the bases on which our proposed algorithm is built. These bases include the theoretical foundations of the two problems of constrained and incremental mining of association rules. We begin by formally defining association rules. Then we move on to a brief discussion of constrained mining of association rules in which we present the concept of constrained frequent-set queries (*CFQs*), the anti-monotonicity and succinctness properties of constraints and the optimizations using these properties. Finally, a description of the incremental mining problem and a formal definition of the concept of the *negative border* are given.

2.1 Association rules

Let $Item = \{i_1, i_2, \dots, i_m\}$ be a set of literals called *items*, DB be a database of transactions where each transaction $T \subseteq Item$ and has a unique identifier, TID . Given an itemset $X \subseteq Item$, X is contained in T iff $X \subseteq T$. An association rule is an implication of the form $S_a \Rightarrow S_c$, where both S_a (*rule antecedent*) and S_c (*rule consequent*) are itemsets and $S_a \cap S_c = \Phi$. A rule has support s iff $s\%$ of the transactions in DB contain $S_a \cup S_c$. A rule has confidence c iff $c\%$ of the transactions containing S_a also contain $S_a \cup S_c$. An itemset is *frequent* iff its support exceeds a certain support threshold *minsup*.

2.2 Constrained mining of association rules

2.2.1 Constrained Frequent-set Queries (CFQs)

Following the definition in [20], a *CFQ* is defined to be a query of the form: $\{(S_a, S_c) | C\}$, where C is a set of constraints on S_a and S_c (*rule antecedent* and *rule consequent* respectively). The syntax of constraint constructs is detailed in [20].

Assuming that the minable view consists of the relations **trans**(TID , $Itemset$) and **itemInfo**($Item$, $Type$, $Price$), some examples of *CFQs* are as follows: The query $\{(S_1, S_2) | count(S_1)=1 \& count(S_2)=1\}$ asks for all pairs of single items satisfying frequency constraints. The query $\{(S_1, S_2) | S_1.Type \supseteq \{snacks, sodas\} \& S_2.price \geq 30\}$ asks for itemsets in the antecedent of the rule, which contain at least one snack item and one soda item, that are associated with itemsets that contain items with a price of at least \$30. Finally, $\{(S_1, S_2) | S_1.Type \subseteq \{snacks\} \& S_2.Type \subseteq \{beverages\} \& \max(S_1.price) \leq \min(S_2.price)\}$ finds pairs of sets of cheaper snack items and sets of more expensive beverage items. In all the examples, the domain and frequency constraints (those stating that itemsets should belong to the universal set *Item* and that they should be frequent) are omitted for brevity and are assumed to exist by default.

Constraints in [20] were divided into 1-variable constraints (those constraints concerning only one *set variable*, as in the first and second examples) and 2-variable constraints (those relating two *set variables*, as in the third example). In this paper, however, focus is only on 1-variable constraints.

2.2.2 Anti-monotone and succinct constraints

The constraints were also classified according to two orthogonal properties; anti-monotonicity and succinctness. Informally, an anti-monotone constraint is a *frequency-like* constraint, meaning that it satisfies the same closure property of the support constraint, which states that if an itemset satisfies the constraint, then all its subsets will do. An example would be to require that the minimum price of any item in the itemset exceeds \$30. A succinct constraint, on the other hand, is a constraint having the property that all itemsets following it can be generated, using some *member generating function (MGF)*, once and for all before any iterations take place. An example of a succinct constraint would be to require that the itemset should contain at least one ‘soda’ item. Clearly, one can generate all such itemsets without any counting of the transactional database. The importance of this classification lies in that one can exploit both properties while trying to discover frequent sets that follow the constraints. This gives a great advantage over the naïve approach of counting the frequent sets first and then filtering out those which do not follow the constraints.

The optimization strategy for anti-monotone constraints in *CAP* [20] is similar to that used in *Apriori* [4] for the support constraint (which is also an anti-monotone constraint). This means that the candidate itemsets are generated in the same way as in *Apriori*; i.e., if a candidate set does not satisfy the anti-monotone constraint it is discarded from further counting and generation of candidates. For succinct constraints, *CAP* used the member generating functions to avoid the generate-and-test environment of other types of constraints. The *MGF* ensures that all the itemsets generated are known to satisfy the constraint.

For a complete description of the *CAP* algorithm and its optimization strategies the reader is referred to [20].

2.3 Incremental Mining of Association Rules

In their paper [11], Cheung et al. introduced the problem of incrementally mining association rules according to the support-confidence framework. Cheung et al. stated that maintenance of frequent itemsets involves searching for two kinds of itemsets:

- a- Losers: frequent itemsets that became infrequent after adding the increment data to the database.
- b- Winners: infrequent itemsets that became frequent after adding the increment data to the database.

Cheung et al., proposed the *FUP (Fast UPdate)* algorithm to solve the problem. It is based on the *Apriori* algorithm and requires $O(n)$ passes over the database, where n is the size of the maximal frequent itemset. Since the introduction of *FUP*, several proposals to solve the problem appeared, e.g. [6, 12, 17, 21, 23, 26].

We will include, without proof, the important lemmas cited in [11] upon which *FUP* and virtually all other incremental mining algorithms are based. Table 1 summarizes the notations used henceforth.

Lemma 1

An infrequent itemset S in DB can become a *winner* in the updated database (DB^+) only if it is frequent in the increment database db (i.e. only if $S \in L^{db}$). \square

Lemma 2

Let S be an itemset. If $S \in L^{DB}$ and $S \in L^{db}$ then $S \in L^{DB^+}$. \square

DB	The original database
Db	The increment database
DB+	The updated database
A	Number of transactions in database A
minsup	Minimum support threshold
C	A constraint on association rules
C_k^c	The candidate set of size k . The superscript c (if present) represents a constraint C
$C_{freq}, C_{am}, C_{succ}, C_{sam}$	The frequency, anti-monotone, succinct and succinct-anti-monotone components of the constraint C respectively
$C-C_x$	The components of the constraint C excluding the component x , where x stands for <i>succ</i> , <i>am</i> or <i>freq</i> .
L_c^A	The set of frequent itemsets in database A . The subscript c (if present) represents a constraint C
$t_A(s)$	Support count of itemset s in database A

Table 1: List of used notations

The first lemma is the most important one since it greatly restricts the number of the candidates that need to be checked against the original database, which is the most demanding task of the whole process with respect to time and resources.

The *BORDERS* algorithm [13, 26], developed simultaneously by Thomas et al. and Feldman et al., is a novel and creative algorithm that uses the concept of the *negative border* introduced by Toivonen in [27] to indicate whether it is necessary or not to check any candidate against the original database. The contribution is that the algorithm introduced needs at most one scan of the original database to update the frequent itemsets. Since this is closely related to our work, we include the definition of the negative border [26, 27].

Definition 1: Negative border (NBd(L)) of a set of frequent itemsets L

Given a collection $L \subseteq P(R)$ of sets, where $P(R)$ is the power set of R , closed with respect to the set inclusion relation, the negative border $NBd(L)$ of L consists of the set of minimal itemsets $X \subseteq R$ not in L . \square

The algorithm in [13, 26] maintains information about the support of frequent itemsets in the original database along with the support of their negative border. An important result cited in [26] is that if any itemset is to become a winner in the updated database, it follows that some itemset formerly in the negative border will also become a winner. This means that the negative border becomes an indicator for the necessity of looking for winners in the original database. If no expansion happens to the border, then we do not need to scan the original database. This result is formulated in the next theorem [26], which we include without proof.

Theorem 1

Let S be an itemset, $S \in L^{DB+}$ & $S \notin L^{DB} \cup NBd(L^{DB})$, then there exists an itemset t such that $t \subset S, t \in NBd(L^{DB})$ & $t \in L^{DB+}$. That is some itemset has moved from $NBd(L^{DB})$ to L^{DB+} . \square

3. Incremental Mining of Constrained Association Rules

In this section, the required formal description of the tackled problem is set. Section 3.1 defines the problem of incremental mining of constrained association rules in the context of *CFQs*. Next, section 3.2 defines the proposed constrained negative border to be used in this context. Then the proof that the new constrained border is a valid indicator for the necessity of checking the original database after adding transactions is given.

3.1 Formal problem definition

In the context of constrained frequent-set queries, the problem of incremental mining of association rules can be restated as follows: let L_c^{DB} be a set of frequent itemsets in a database DB that satisfies a certain constraint C defined by some *CFQ*. After some update of the database, another set of transactions db is added to DB to get the updated database $DB+$. It is required to discover the new set L_c^{DB+} of frequent itemsets that satisfies the same constraint C in the updated database.

As mentioned before, the concept of the negative border proved useful in the problem of incremental mining of association rules. The challenge is to try to adapt the concept of negative border for *CFQs*. It can be noticed that the negative border as defined before represents the candidate itemsets counted by the *Apriori* algorithm that turned to be not frequent. This may give an insight to use the same sets counted by the *CAP* algorithm but found small, as the new negative border. This leads to the following definition of the new negative border, which is named the *constrained negative border*.

3.2 The constrained negative border

Definition 2: Constrained Negative Border $CNBd(L_c)$ of a set of constrained frequent itemsets L_c

The constrained negative border of a set of frequent itemsets L_c , satisfying a constraint C , henceforth referred to as $CNBd(L_c)$ is defined as follows:

$$CNBd(L_c) = \left\{ S \mid S \text{ is an itemset satisfying only } C - C_{freq} \right. \\ \left. \& S' \subset S \Rightarrow S' \in L_c \text{ or } S' \text{ does not satisfy } C_{succ} \right\} \quad \square$$

Informally, the constrained negative border consists of all the itemsets that are not frequent but have all their subsets (of which their frequencies are known) frequent. The reason for our uncertainty is the generate-only paradigm of succinct constraints which leaves us with no knowledge of the frequency of itemsets which do not satisfy the constraint since they are not generated from the first place. Just like strategy II of the *CAP* algorithm, if an itemset in the border has such a subset, it is given the merit of doubt by assuming it to be frequent.

Given that the normal negative border was an indicator for the necessity of checking the original database for winners, it seems natural for the constrained negative border to play the same role. The original negative border possessed the property of having all its members minimally small, meaning that all their *proper* subsets are frequent. In the context of *CFQs*, the same concept can be generalized to include all anti-monotone constraints. But since succinct constraints do not possess the same closure property, it is not guaranteed that all proper subsets satisfy the constraint. The following lemmas, which we include without proof for the lack of space, help better understand the succinct constraints. They clear the way for the proof of the next theorem, which states that the constrained negative border is still

indeed an indicator for checking the original database. The proof of the following lemmas and Theorem 2 can be found in [5].

Lemma 3

Let C be a succinct constraint with an MGF M defined as follows: $M = \{X_1 \cup \dots \cup X_n \mid X_i \subseteq \sigma_{p_i}(Item), 1 \leq i \leq n \ \& \ \exists k \leq n : X_j \neq \phi, 1 \leq j \leq k\}$. It is assumed, without loss of generality, that the satisfying sets of the selection predicates (σ_{p_i} 's) of M are all disjoint for $1 \leq j \leq k$. Let S be an itemset satisfying C . It follows that $|S| \geq k$ where k is as defined in M . \square

In other words, k is the lower bound of the size of any itemset that satisfies the succinct constraint. If the X_j 's were not disjoint, this means that there might be an item included in the satisfying set of two or more selection predicates. We then have a lower bound than k for the size of an itemset satisfying a succinct constraint. This lower bound, m , is between 1 and k . This will not affect the result of the lemma.

To see the result of this lemma, consider the following simple example. Referring to the example database of Figure 1, let C be the succinct constraint $S.Type \supseteq \{snack, soda\}$. The MGF of this constraint is: $M = \{X_1 \cup X_2 \cup X_3 \mid X_i \subseteq \sigma_{p_i}(Item), X_1 \neq \phi, X_2 \neq \phi\}$ where $p_1 = (Type = 'soda')$, $p_2 = (Type = 'snack')$, and $p_3 = (Type \neq 'soda' \wedge Type \neq 'snack')$. The value of k here is 2. The lemma simply states that any itemset satisfying the constraint should have at least 2 items. This is clear since a satisfying itemset must contain both a 'snack' and a 'soda' item (this is of course assuming no single item is both a 'snack' and a 'soda' item at the same time). For the example database, those minimum sized itemsets satisfying the constraint are the itemsets BC and BF .

itemInfo			trans	
Item	Type	Price	TID	Itemset
A	Dairy	10	1	A B
B	Soda	20	2	A B C
C	Snack	50	3	B C E
D	Juice	30	4	C D F
E	Candy	25	5	B E
F	Snack	45	6	A
			7	B
			8	A B C F
			9	A E
			10	A B C D E

Figure 1: An example database

Lemma 4

Assuming k is as defined in the MGF of Lemma 3 and a general constraint C . All itemsets of size k satisfying $C-C_{freq}$ are in $L_c \cup CNBd(L_c)$. Furthermore, all itemsets of size $k+1$ satisfying $C-C_{freq}$ and having all their k -sized subsets which satisfy C_{succ} in L_c are in $L_c \cup CNBd(L_c)$. \square

To see this by example, consider the same constraint $S.Type \supseteq \{snack, soda\}$. As discussed after Lemma 3 the set of itemsets of size 2 satisfying the constraint is $\{BC, BF\}$. It is clear that both itemsets should be in $L_c \cup CNBd(L_c)$. Now, consider the itemset ABC . It sure satisfies the constraint, however, all its proper subsets except BC do not satisfy the constraint. Assuming BC is in L_c , if ABC is frequent then it is in L_c .

Otherwise, it would be in $CNBd(L_c)$ by definition since its only proper subset satisfying the succinct constraint is in L_c . Hence, ABC is in $L_c \cup CNBd(L_c)$. The previous lemma sets a lower bound on the size of itemsets in the constrained negative border.

Lemma 5

Let C be a succinct constraint with an MGF M similar to that defined in Lemma 3, and let S be an itemset satisfying C . If $|S| > k$, where k is as defined in M , then $\exists t \subset S$ such that t satisfies C . \square

Informally, the previous lemma states that any itemset satisfying the succinct constraint with a greater cardinality than the minimum cardinality set by Lemma 3 will have a *proper* subset of it that also satisfies the constraint.

To see this by example, consider the same constraint and example database. We know that the minimum sized itemsets satisfying C are $\{BC, BF\}$. Consider the itemset ABC , it has exactly one soda item, ‘ B ’, and one snack item, ‘ C ’. Then, it has the itemset set BC as the required proper subset satisfying the constraint. Furthermore, consider the itemset BCF which also satisfies the constraint. Here, there are more than one item of type ‘snack’, namely ‘ C ’ and ‘ F ’. This makes both BC and BF qualify as the required proper subsets.

The previous results clear the road for proving the next theorem.

Theorem 2

Let S be an itemset, $S \in L_c^{DB+}$ and $S \notin L_c^{DB} \cup CNBd(L_c^{DB})$, then there exists an itemset t such that $t \subset S$, $t \in CNBd(L_c^{DB})$ and $t \in L_c^{DB+}$. \square

The previous theorem proved that the constrained negative border is playing the same role played by the normal negative border in [26]. This result will be employed in the proposed algorithm *ICAP*. *ICAP* uses the *CAP* algorithm to discover the initial set of constrained frequent itemsets along with its constrained negative border and assumes that the support count of each itemset in both sets is kept in the database.

4. The Proposed Algorithm ICAP (Incremental Constrained APriori)

Before proceeding with the description of the algorithm it should be noted that only updates of the transactional database is taken into consideration (i.e. no changes occur in the data describing the items). Therefore, it should be clear that adding and removing transactions can only affect the support of an itemset and has no effect on the satisfiability of C - C_{freq} . Formally, if an itemset S satisfied C - C_{freq} , before the database update, S will satisfy C - C_{freq} after the database update.

The algorithm uses as input the set of constrained frequent sets L_c^{DB} of the original database DB and the constrained negative border $CNBd(L_c^{DB})$ discovered using the algorithm *CAP* along with the frequencies of the itemsets in both sets.

The proposed algorithm can be summarized in the following steps:

1. Use algorithm *CAP* to discover all frequent itemsets following the constraint in the increment database db .
2. Count the frequencies of the itemsets in the previously discovered set L_c^{DB} along with the constrained negative border $CNBd(L_c^{DB})$ in the increment database db to discover the losers and those itemsets which will remain frequent.
3. If no new itemsets qualify as candidate frequent itemsets (i.e., the border expanded), then there is no need to rescan the original database. The new

negative border can be recomputed in a similar manner as in [26] only if losers exist (see section 5 for the method of generating and maintaining the constrained negative border).

4. If there are new qualifying candidates, generate the constrained negative border closure in the same manner as in [26] and count the candidates in this closure against the original database.

It is clear from the description of *ICAP* that it requires at most one scan of the original database (done in step 4) only if the database insertions cause the constrained negative border to expand. Expansion of the border means that there are potential itemsets that can become winners other than those in $L_c \cup CNBd(L_c)$. This happens if there exist winners that can be joined with other frequent itemsets or that can allow other itemsets to be candidates for testing.

The fact that *ICAP* can exploit and use the constrained negative border makes it superior compared to the classical level-wise algorithms. A high-level description of the proposed algorithm *ICAP* is shown in Figure 3. The following theorem establishes the correctness of *ICAP*. A detailed proof for it can be found in appendix A.

Theorem 3

Algorithm *ICAP* is sound and complete with respect to counting all constrained frequent itemsets in the updated database DB^+ . \square

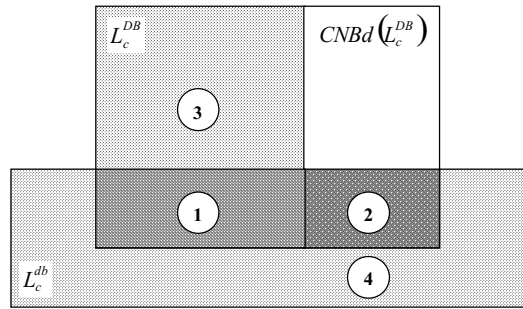


Figure 2: The possible intersections of L_c^{DB} , $CNBd(L_c^{DB})$, and L_c^{db}

Example:

Consider a transactional database as depicted in Figure 1. Let the first 7 transactions be the original database DB and the last 3 transactions be the increment database db . Consider the following *CFQ*:

$$\{X | X.Type \supseteq \{soda, snack\} \& \min(X.price) \geq 20 \& \min sup = 0.2\}$$

The algorithm *ICAP* is now applied on the database to find the updated constrained frequent sets. It is assumed that the *CAP* algorithm has been applied on the original database DB . Thus the set of constrained large frequent set $L_c^{DB} = \{BC\}$ and the constrained negative border $CNBd(L_c^{DB}) = \{BF, BCD, BCE\}$. Initially, *CAP* is run on db to get the set $L_c^{db} = \{BC, BF, BCF, BCD, BCE, BCDE\}$. Referring to Figure 2, the sets of itemsets $\{BC\}$, $\{BF, BCD, BCE\}$, and $\{BCF, BCDE\}$ correspond to those in regions 1, 2, and 4. Region 3 is empty since the only frequent itemset in DB is also frequent in db . Now, BC is frequent in the updated database. Checking items in the negative border, we have BCE only moving from the negative border to the updated set of frequent itemsets. Before checking for the itemsets in region 4, we check the constrained negative border expansion. Since, moving BCE into the frequent sets does not expand the negative border, we conclude that there is no need to count the support of the candidates of region 4 in DB . Thus,

the final updated constrained frequent itemsets $L_c^{DB+} = \{BC, BCE\}$ and the new constrained negative border $CNBd(L_c^{DB+}) = \{BF, BCD\}$. \square

As can be seen from the previous example, *ICAP* did not need a scanning of the original database. This is a great improvement over rerunning the *CAP* algorithm on the original database in terms of the number of transactions read. It would be natural to assume that a more drastic improvement gain will be obtained for typical sizes of databases and database increments or updates.

The operations of computing and maintaining the constrained negative border are not as straightforward as those for the negative border in [26]. The next section describes the problems faced when trying to compute the constrained negative border and explains how it can be generated.

Function $ICAP(L_c^{DB}, CNBd(L_c^{DB}), db)$

- 1 Compute L_c^{db} using *CAP*
- 2 Count support of all items of L_c^{DB} and $CNBd(L_c^{DB})$ in db .
- 3 $L_c^{DB+} = \Phi$
- 4 **For each** itemset s in $L_c^{DB} \cap L_c^{db}$ **do** //itemsets still frequent
- 5 $t_{DB+}(s) = t_{DB}(s) + t_{db}(s)$ //calculate new support count
- 6 $L_c^{DB+} = L_c^{DB+} \cup \{s\}$
- 7 **For each** itemset s in $CNBd(L_c^{DB}) \cap L_c^{db}$ **do** //cand. of the border
- 8 $t_{DB+}(s) = t_{DB}(s) + t_{db}(s)$ //calculate new support count
- 9 **If** $t_{DB+}(s) \geq \text{minsup} * |DB+|$ **then**
- 10 $L_c^{DB+} = L_c^{DB+} \cup \{s\}$
- 11 **For each** itemset s in $(L_c^{DB} - L_c^{db})$ **do**
- 12 **If** $t_{db}(s) + t_{DB}(s) \geq \text{minsup} * |DB+|$ **then**
- 13 $L_c^{DB+} = L_c^{DB+} \cup \{s\}$
- 14 **If** $L_c^{DB+} \neq L_c^{DB}$ **then**
- 15 $CNBd(L_c^{DB+}) = \text{constrained - negativeborder - gen}(L_c^{DB+}, C)$
- 16 Else $CNBd(L_c^{DB+}) = CNBd(L_c^{DB})$
- 17 **If** $L_c^{DB} \cup CNBd(L_c^{DB}) \neq L_c^{DB+} \cup CNBd(L_c^{DB+})$ **then** //scan necessary?
- 18 $S = L_c^{DB+}$
- 19 **Repeat**
- 20 $S = S \cup \text{constrained - negativeborder - gen}(S, C)$
- 21 **Until** S does not grow
- 22 **For each** itemset s in $S - L_c^{DB+}$
- 23 //count candidate itemsets against the updated database
- 24 **If** $t_{DB+}(s) \geq \text{minsup} * |DB+|$ **then**
- 25 $L_c^{DB+} = L_c^{DB+} \cup \{s\}$
- 26 $CNBd(L_c^{DB+}) = \text{constrained - negativeborder - gen}(L_c^{DB+}, C)$
- 27 **Return** $L_c^{DB+} \cup CNBd(L_c^{DB+})$

Figure 3: A high level description of the algorithm *ICAP*

5. Computing The Constrained Negative Border

The original border in [26] was easily produced and maintained as a by product of the *Apriori* algorithm since it represents all candidate itemsets that did not pass the counting test, which means that those itemsets are counted by the algorithm anyway.

The constrained negative border, on the other hand, does not possess the same property. According to *Lemma 4*, all itemsets of minimum size that satisfy $C-C_{freq}$ should be in the constrained border. *CAP*, however, does not produce and hence does not count all those itemsets since it guarantees that all generated itemsets satisfying $C-C_{freq}$ of minimum size will have all their subsets satisfying $C-C_{succ}$ [20]. It can also be noticed that, assuming the minimum size of itemsets satisfying $C-C_{freq}$ is k , all itemsets of size $k+1$, having all their k -size subsets satisfying C_{succ} , in L_c should also be in the constrained border. *CAP*, on the other hand, only counts a subset of such itemsets.

To see this, consider the example database of Figure 1 and the succinct constraint $S.Type \supseteq \{snack, soda\}$. From the previous discussions it is known that any generated itemset should contain at least one item from the set $\{B\}$ and at least one item from the set $\{C, F\}$ and optionally one or more other items from the whole universal set of items. Assume that the set of frequent items was $\{A, B, C\}$. According to strategy II of *CAP*, only the itemsets BC and ABC will be generated and counted. However, BF , BCD , BCE and BCF should also be counted, since they are by definition in the constrained border (*Lemma 4*).

To work around this problem, *CAP* should be modified to generate and count all the candidates belonging to the constrained negative border. This can be done as follows:

Assuming C_{succ} is as defined in *strategy II* of *CAP* [20] and observing that such definition makes the minimum size of an itemset, satisfying $C-C_{freq}$, equal one (i.e. $k=1$), strategy II should be as follows:

- a. Define $C_1^c = \sigma_{p_1}(Item)$ and $C_1^{-c} = \sigma_{p_2}(Item)$. Define corresponding sets of frequent sets of size 1: $L_1^c = \{e \mid e \in C_1^c \ \& \ freq(e)\}$, and $L_1^{-c} = \{e \mid e \in C_1^{-c} \ \& \ freq(e)\}$.
- b. Add $C_1^c - L_1^c$ to $CNBd(L_c)$.
- c. Define $C_2 = L_1^c \times (L_1^c \cup C_1^{-c})$, and L_2 the set of frequent itemsets in C_2 .
- d. Add $C_2 - L_2$ to $CNBd(L_c)$.
- e. Do step 3 of strategy II with no modification.

Function $Constrained-negativeborder-gen(L_c, C)$

- 1 Split L_c into L_m, L_{m+1}, \dots, L_n where L_k is the set of constrained frequent itemsets in L of size k , m and n are the size of the smallest and the largest sets of frequent itemsets in L_c respectively.
 - 2 Use the modified strategy II in section 5 to generate all candidates of size m , C_m (using step a) and all candidates of size $m+1$, C_{m+1} (using step c), according to the constraint C .
 - 3 $CNBd(L_c) = (L_m - C_m) \cup (L_{m+1} - C_{m+1})$
 - 4 **For** $k = m+1$ to $n-1$ **do**
 - 5 Use *CAP* to generate C_{k+1} from L_k
 - 6 $CNBd(L_c) = CNBd(L_c) \cup (C_{k+1} - L_{k+1})$
 - 7 **Return** $CNBd(L_c)$
-

Figure 4: A high level description of the function *constrained-negativeborder-gen*

The process of re-computing the constrained negative border, given a set of constrained frequent-sets L_c and a constraint C , is done by the function *constrained-negativeborder-gen*(L_c, C). Figure 4 gives a high-level description of this function.

6. Performance Study

6.1 The experimental environment and performance parameters

The algorithm was implemented and tested on an IBM compatible PC with a Pentium II[®] 300 processor and 96 MB of main memory running the Microsoft Windows 95[®] operating system. The program was the only major job running on the machine throughout all the experiments to achieve a fair environment for comparison. As for the test data, the program developed in IBM Almaden research center was used to generate the test data (available from the IBM QUEST web site <http://www.almaden.ibm.com/cs/quest>).

Number of Transactions in the updated database	400,000 (in the basic set of experiments)
Number of Items	1000
Average Transaction Size	10
Average size of maximal potentially large itemsets	6

Table 2: Parameter Settings

The parameter settings used for the experimental database are listed in Table 2. Such parameters are assumed to mimic a reasonable retailing environment. They also give the chance to explore the relative merits of the incremental algorithm. When generating the increment database, the method in [11, 23, 26] was used. In this method, the increment database is taken as a contiguous block from a generated set representing the whole updated database $DB+$. The increment size $x\%$ means that $(DB+|-x\%)$ of the database is considered as DB and the remaining $x\%$ is considered as db . The increment database was taken from the first $x\%$ transactions of the database.

6.2 Measuring performance for different increment sizes and different query types

As discussed in [20], there are four different categories of constraints; namely succinct only, anti-monotone only, succinct anti-monotone and non-succinct non-anti-monotone constraints. Representative queries of each of the four categories were used in the first set of experiments. The set of figures Figure 5 to Figure 8 represent the speedup of *ICAP* for these different types of constraints, the speedup is measured for different increment sizes (measured as a percentage from the size of the original database). Figure 5 shows the results when using the succinct only constraint $S.Type \supseteq \{soda\}$. Figure 6 shows the results when using the succinct anti-monotone constraint $\min(S.Price) \geq 60$. Figure 7 uses the anti-monotone constraint $\sum(S.Price) \leq 50$. To test the non-succinct non-anti-monotone category of constraints, the constraint $\text{avg}(S.Price) \leq 15$ was used. Finally, Figure 8 tests the speedup when using a hybrid constraint combining both the succinct constraint $S.Type \supseteq \{soda, snack\}$ and the succinct anti-monotone constraint $\min(S.Price) \geq 20$. The selectivity of the *soda* items and *snack* items were 5% each throughout all experiments. The previous constraints mimic those reported in [20] for which the *CAP* algorithm performed well.

Several observations apply generally to all the results of this set of experiments. The results show that *ICAP* is superior to *CAP* in almost all of the cases. The speedup also exhibits a similar pattern as reported in [26] for the *BORDERS* algorithm. The justification is also the same. For high support thresholds, it is less costly to re-run *CAP*. At low support thresholds, however, there is a greater possibility for the border to expand and for *ICAP* to scan the database. This is why the speedup has a global minimum around the support value of 0.6% in all cases. The speedup also increases for smaller increment sizes since there is less data for *ICAP* to process.

Several glitches appear in the figures that disobey the general trend discussed above (e.g. the case of 0.8% *minsup* in Figure 5). Such abnormalities are due to the local characteristics of the transactions in the increment, which cause the border to expand. To eliminate the effect of the local characteristics of the data in the increment database, the experiment for the hybrid constraint was repeated 10 times for 10% and 5% increment sizes taking the increment block from a different part of the database each time. Average speedup is taken for all cases. The results are shown in Figure 9. From the figure, it can be noticed that speedup still obeys the same general trend.

Some particular observations can be deduced for every case. *For the hybrid constraint*, due to the high selective power of the constraint, the performance gain was more drastic than all the other cases. *For the anti-monotone constraint and the succinct-anti-monotone constraint*, *ICAP* exhibits a similar behavior with respect to *CAP* as that reported in [26]. In both cases, the behavior of *CAP* degenerates to *Apriori*. Finally, *for the non-succinct non-anti-monotone constraint*, it should be noted that the constraint used will induce the weaker constraint $\min(S.Price) \leq 15$ which has exactly the same constraining capability as the succinct only constraint of Figure 5 since both constraints have the same selectivity. The extra pass required to filter out the extra itemsets not satisfying the original constraint accounted for a negligible amount of time with respect to the other tasks. This made the results of Figure 5 apply also to this case.

6.3 Measuring scalability with database size

The second experiment intended to test how *ICAP* scales-up when changing the database size. The database size was changed in 100,000 transactions increments starting from 100,000 transactions until 400,000 transactions. The experiment was conducted for the 10% increment database size to test *ICAP* in its worst cases. The results of the experiment can be seen in Figure 10. It was expected that the performance gain should increase drastically with the increase in database size since this increases more the I/O cost of the non-incremental algorithm. The results of the experiment, as seen from the figure, was not quite what was expected. In spite of the general trend which shows that performance gain increases with the database size, the recorded increase was not that significant. The reason for this is that the size of the database file is small (18 MB in the case of 400,000 transactions) compared to the size of the main memory. This probably enables the operating system to cache the whole database file into main memory after the first pass. The increase in the database size in this case did not impose a drastic difference on performance. An interesting question arises here which is, *if the whole database can be cached in memory, from where does the incremental algorithm obtain its edge?* The answer is that, no matter what, the non-incremental algorithm wastes too much time sequentially passing on the database many times. More drastic increase in performance is expected for larger practical databases.

6.4 Measuring sensitivity to item selectivity

In case of succinct constraints, and from Lemma 4, the constrained negative border requires the generation and counting of several itemsets that are known

beforehand to be not frequent. From section 5, to generate the minimum sized itemsets satisfying the succinct constraint, it is required to generate the cross product of the mandatory sets in the *MGF* of the constraint. The number of itemsets in the border can be very large, especially if the cardinality of the sets involved in the cross product is high. This places a burden on *ICAP* that does not exist for *CAP* and raises a natural question, *does the performance gain disappear for higher number of border items?*

The following experiment was designed to answer this question. The algorithms are compared for the same succinct constraint of Figure 5 and for 10% increment size (again to be harsh on *ICAP*) and the different support settings. The item selectivity was increased from 5% to 25% (i.e. the number of soda items in this case was changed from 5% of the whole set of items to 25%). This increased the number of border itemsets.

The results of the experiment can be seen in Figure 11. The interesting discovery is that *ICAP* still has its performance edge. This is due to the fact that, as reported in [20], the performance of *CAP* itself also drops with item selectivity because it has to count more itemsets that satisfy the constraint, and more importantly the probability that the length of the frequent itemsets increases is highly costing it extra precious passes over the database. This balances the deterioration in time in *ICAP* needed to maintain the large number of border itemsets.

7. Conclusion and Future extensions

In this paper, a new algorithm (Incremental Constrained APriori *ICAP*) for incremental mining of constrained association rules is proposed. The algorithm applies the techniques of incremental mining in the *BORDERS* algorithm on the constrained mining algorithm *CAP* to produce the new set of updated constrained frequent-sets after the update of the original transactional database by adding new transactions. In the course of developing the proposed algorithm, the concept of the *constrained negative border* is introduced as the counterpart of the classical concept of the *negative border*. The *constrained border* is proved to have the same property as the normal border, namely being an indicator for the necessity of checking the original database, which is the most costly operation in incremental mining. The proposed algorithm utilizes the *constrained border* efficiently to maintain the set of constrained frequent-sets. As a direct consequence of this usage, *ICAP* performs at most *one* pass on the original database and only when it is *absolutely* necessary.

Several experiments are conducted to measure the relative performance of the new algorithm compared to rerunning the *CAP* algorithm from scratch on the whole updated database. The test results show that *ICAP* exhibits a speedup gain in virtually all situations. The sensitivity of the algorithm is tested for several increment sizes and support thresholds.

A natural extension to the presented problem would be to handle the case of deleted transactions and the exploration of the benefits of incremental mining on 2-variable constraints. Another harder problem is to try to handle the case when the database updates includes updating the characteristic data of the items (e.g. changing the price of some items or changing the item type).

Recently, a number of studies appeared [1, 14, 19] that utilize clever tree structures to compactly represent the transaction database and frequency information. These structure allow very efficient implementations of algorithms for discovering association rules especially for dense databases with very long frequent itemsets. It remains to be seen how such techniques could be utilized for the incremental discovery of constrained association rules.

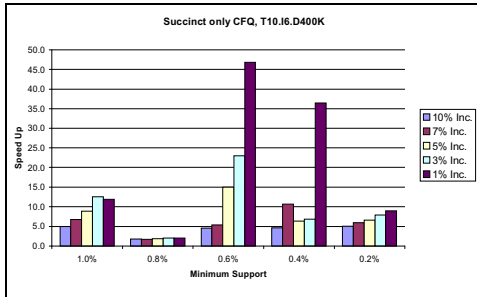


Figure 5: Speed up for succinct only and non-succinct non-anti-monotone constraints

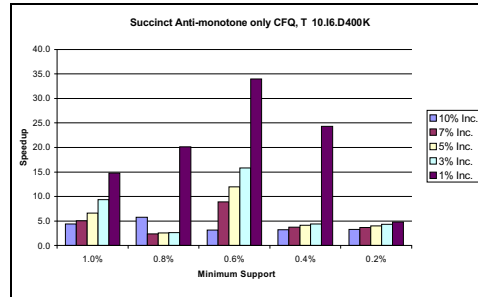


Figure 6: Speedup for succinct anti-monotone constraint

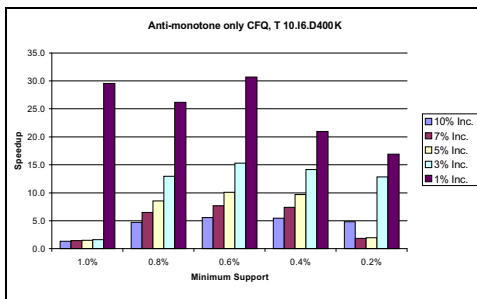


Figure 7: Speedup for anti-monotone constraint

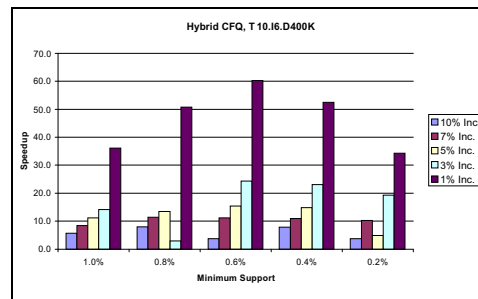


Figure 8: Speedup for a hybrid constraint

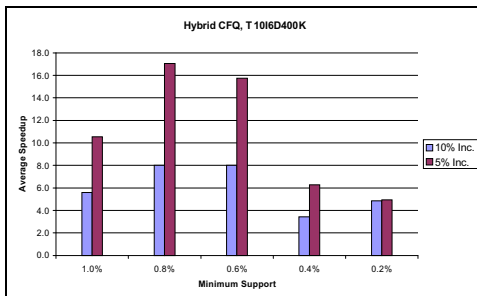


Figure 9: Average Speedup for the hybrid constraint

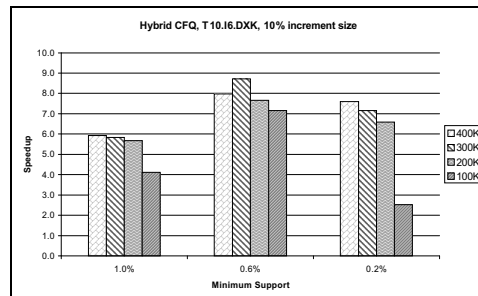


Figure 10: Measuring performance for different database sizes

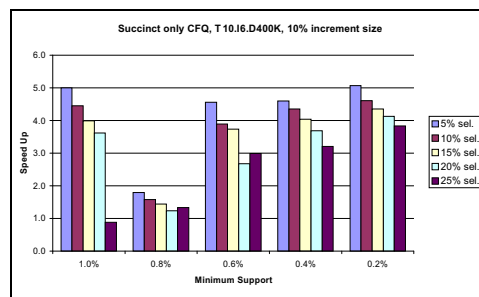


Figure 11: Measuring performance with item selectivity

8. References

- [1] AGARWAL, R.; AGGARWAL, C.; AND PRASAD, V. V. V. *A Tree Projection Algorithm for Generation of Frequent Itemsets*. In Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining), 2000
- [2] AGRAWAL, R.; IMIELINSKI, T. AND SWAMI, A. *Mining Associations between Sets of Items in Massive Databases*. Proc. of the ACM SIGMOD Int'l Conference on Management of Data, pp. 207-216, Washington D.C., May 1993.
- [3] AGRAWAL, R.; SHAFER, J.C. *Parallel Mining of Association Rules*. IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, December 1996.
- [4] AGRAWAL, R. AND SRIKANT, R. *Fast Algorithms for Mining Association Rules*. Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile, Sept. 1994.
- [5] AYAD, M. AHMED. *Incremental Mining of Constrained Association Rules*. Master Thesis, Dept. of Computer Science and Automatic Control, Alexandria University-2000.
- [6] AYAN, NECIP F.; TANSEL, ABDULLAH U.; AND ARKUN, EROL. *An efficient algorithm to update large itemsets with early pruning*. Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'99) August 15 - 18, 1999, San Diego, CA USA pp. 287 – 291.
- [7] BAYARDO, R. J. *Efficiently Mining Long Patterns from Databases*. In Proc. of the 1998 ACM-SIGMOD Int'l Conf. on Management of Data, pp. 85-93.
- [8] BAYARDO, R. J.; AGRAWAL, R.; AND GUNOPULOS, D. *Constraint-Based Rule Mining in Large, Dense Databases*. In Proc. of the 15th Int'l Conf. on Data Engineering , pp. 188-197, 1999.
- [9] BRADLEY, PAUL; FAYYAD, USAMA; AND MANGASARIAN, OLVI. *Data Mining: Overview and Optimization Opportunities*. Microsoft Research Report MSR-TR-98-04, January 1998
- [10] BRIN, S.; MOTWANI, R.; ULLMAN, J.; AND TSUR, S. *Dynamic Itemset Counting and Implication Rules for Market Basket Data*. In Proc. of the 1997 SIGMOD Conf. on the Management of Data, pp. 255-264.
- [11] CHEUNG, D.; HAN, J.; NG, V. AND WONG, C.Y. *Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique*. Proc. of 1996 Int'l Conf. on Data Engineering (ICDE'96), New Orleans, Louisiana, USA, Feb. 1996.
- [12] CHEUNG, DAVID W. L.; LEE S.D., AND BENJAMIN, KAO. *A general incremental technique for maintaining discovered association rules*. In Proceedings of the Fifth International Conference On Database Systems For Advanced Applications, pp. 185-194, Melbourne, Australia, March 1997.
- [13] FELDMAN, R.; AUMANN, Y.; AMIR, A.; AND MANNILA, H. *Efficient Algorithms for Discovering Frequent Sets in Incremental Databases*. In Proceedings of the 1997 SIGMOD Workshop on DMKD, Tucson, Arizona, May 1997.
- [14] HAN, J.; PEI, J.; AND YIN, Y. *Mining Frequent Patterns without Candidate Generation*. Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX, May 2000.
- [15] HAN, J.; LAKSHMANAN, L. V. S.; NG, R. *Constraint-based, Multidimensional data mining*. IEEE Computer, Special issue on data mining, August 1999.
- [16] LAKSHMANAN, L. V. S.; NG, R.; HAN, J. and Pang, A. *Optimization of Constrained Frequent Set Queries with 2-Variable Constraints*. Proc. 1999 ACM-SIGMOD Conf. on Management of Data (SIGMOD'99), Philadelphia, PA, June 1999, pp. 157-168.

- [17] LEE, S.D.; AND CHEUNG, DAVID W. L. *Maintenance of Discovered Association Rules: When to update?*. In Proceedings of the 1997 ACM-SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD-97), Tucson, Arizona, May 1997.
- [18] LIN, D. AND KEDEM, Z. M. *Pincer-Search: A New Algorithm for Discovering the Maximum Frequent Set*. In Proc. of the Sixth European Conf. on Extending Database Technology.
- [19] MEO, ROSA. *A New Approach for the Discovery of Frequent Itemsets*. In Proc. of Data Warehousing and Knowledge Discovery, First International Conference, DaWaK '99, Florence, Italy, 1999.
- [20] NG, R. T.; LAKSHMANAN, V. S.; HAN, J.; AND PANG, A. *Exploratory Mining and Pruning Optimizations of Constrained Association Rules*. In Proc of the 1998 ACM-SIG-MOD Int'l Conf. on the Management of Data, pp. 13-24, 1998.
- [21] OMIECINSKI, E.;AND SAVASERE, A. *Efficient mining of association rules in large dynamic databases*. In Proc. BNCOD'98, pages 49-63, 1998.
- [22] PARK, J.-S.; CHEN, M.-S.; AND YU, P. S. *An Effective Hash Based Algorithm for Mining Association Rules*. Proceedings of ACM SIGMOD, pp. 175-186, May 1995.
- [23] SARDA, N. L. ; AND SRINIVAS, N. V. *An adaptive algorithm for incremental mining of association rules*. In Proceedings of DEXA Workshop'98, pp. 240-245, 1998.
- [24] SAVASERE, A.; OMIECINSKI, E.; AND NAVATHE, S. *An Efficient Algorithm for Mining Association Rules in Large Data-bases*. In Proc. of the 21st Conf. on Very Large DataBases, pp. 432-444.
- [25] SRIKANT, R.; VU, Q.; AND AGRAWAL, R. *Mining Association Rules with Item Constraints*. In Proc. of the Third Int'l Conf. on Knowledge Discovery in Databases and Data Mining, August 1997 pp. 67-73.
- [26] THOMAS, SHIBY; BODAGALA, SREENATH; ALSABTI, KHALED; AND RANKA, SANJAY. *An efficient algorithm for the incremental updation of association rules in large databases*. In Proceedings of the 3rd International conference on Knowledge Discovery and Data Mining (KDD 97), New Port Beach, California, August 1997.
- [27] TOIVONEN, H. *Sampling large databases for association rules*. In 22nd International Conference on Very Large Databases (VLDB'96),pp. 134-145, Mumbai, India, September 1996.
- [28] ZAKI, M. J.; PARTHASARATHY, S.; OGIHARA, M.; AND LI, W. *New Algorithms for Fast Discovery of Association Rules*. In Proc. of the Third Int'l Conf. on Knowledge Discovery in Databases and Data Mining, pp. 283-286, 1997.

Appendix A: Correctness proof of the ICAP algorithm

In this appendix, we will prove *theorem 3* which states that *ICAP* is sound and complete; it discovers all and only those itemsets that are frequent in DB^+ and satisfying the *CFQ*. We rely on the correctness of *CAP* proved in [20].

Theorem 3: Proof

In Figure 2 - which represents all the possible intersections of L_c^{DB} , $CNBd(L_c^{DB})$, and L_c^{db} - we have four shaded regions, representing all the possible candidates for inclusion in $L_c^{DB^+}$ (Lemma 1). Regarding the steps of algorithm *ICAP* shown in Figure 3, step 1 computes L_c^{db} using *CAP* and should be correct by the correctness of

CAP. Steps 4-6 add itemsets in *region 1* of Figure 2 to L_c^{DB+} after updating their support count. By *Lemma 2* we know that all those itemsets are in L_c^{DB+} . Steps 7-10 check candidates in *region 2*, those are candidates of the negative border. If an itemset is found to be frequent in DB^+ , it is directly considered frequent by definition of the support constraint. Steps 11-13 check the candidates in *region 3* by scanning the increment database. Now we have only *region 4* remaining to check. By *Theorem 2* we know that there is no need to do that if the constrained border does not expand. Steps 14-17 check for the negative border expansion by observing the change in $L_c^{DB} \cup CNBd(L_c^{DB})$. If an expansion occurs, then we need to scan DB for candidates in *region 4*. Steps 18-21 generate the constrained negative border closure of items in L_c^{DB+} discovered so far. Steps 22-25 count the candidates against DB to discover winners. Finally, step 26 generates the constrained negative border of the complete L_c^{DB+} . Since the candidates counted by the steps of *ICAP* are all and only those candidates that need counting, it follows that *ICAP* is sound and complete. \square