Mining Frequent Itemsets with Convertible Constraints *

Jian Pei Jiawei Han Laks V.S. Lakshmanan

Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
{peijian, han}@cs.sfu.ca

Concordia University & IIT - Bombay
Montreal, Quebec & Mumbai, India
laks@it.iitb.ernet.in

Abstract

Recent work has highlighted the importance of the constraint-based mining paradigm in the context of frequent itemsets, associations, correlations, sequential patterns, and many other interesting patterns in large databases. In this paper, we study constraints which cannot be handled with existing theory and techniques. For example, $avg(S)\ \theta\ v$, $median(S)\ \theta\ v$, $sum(S)\ \theta\ v$ *(*S *can contain items of arbitrary values)* ($\theta \in \{\geq, \leq\}$)*, are customarily regarded as "tough" constraints in that they cannot be pushed inside an algorithm such as Apriori. We develop a notion of* convertible constraints *and systematically analyze, classify, and characterize this class. We also develop techniques which enable them to be readily pushed deep inside the recently developed* FP-growth *algorithm for frequent itemset mining. Results from our detailed experiments show the effectiveness of the techniques developed.*

1. Introduction

It has been well recognized that frequent pattern mining plays an essential role in many important data mining tasks. However, frequent pattern mining often generates a very large number of frequent itemsets and rules, which reduces not only the efficiency but also the effectiveness of mining since users have to sift through a large number of mined rules to find useful ones.

Recent work has highlighted the importance of the paradigm of constraint-based mining: the user is allowed to express his focus in mining, by means of a rich class of constraints that capture application semantics. Besides allowing user exploration and control, the paradigm allows many of these constraints to be pushed deep inside mining, thus pruning the search space of patterns to those of interest to the user, and achieving superior performance.

Itemset constraints have been incorporated into association mining [10]. A systematic method for the incorporation of two large classes of constraints—*anti-monotone* and *succinct*—in frequent itemset mining is presented in [7, 6]. A method for mining association rules in large, dense databases by incorporation of user-specified constraints that ensure every mined rule offers a predictive advantage over any of its simplifications, is developed in [2]. Constraint-based mining of correlations, by exploration of *anti-monotonicity* and *succinctness*, as well as *monotonicity*, is studied in [4].

While previous studies cover a large class of useful constraints, many other useful and natural constraints remain. For example, consider the constraints $avg(S)\ \theta\ v$, $median(S)\ \theta\ v$, and $sum(S)\ \theta\ v$ ($\theta \in \{\geq, \leq\}$). The first two are neither anti-monotone, nor monotone, nor succinct. The last one is anti-monotone when θ is $\leq$ and *all items have non-negative values.* If S can contain items of arbitrary values, $sum(S) \leq v$ is rather like the first two constraints. Intuitively, this means these constraints are hard to optimize. In this paper, we investigate a whole class of constraints that subsumes these examples. The main idea is that constraints that exhibit no nice properties do so in the presence of certain item orders. We make the following contributions.

- We introduce (Section 3) the concept of *convertible constraints* and classify them into three classes: *convertible anti-monotone*, *convertible monotone*, and *strongly convertible*. This covers a good number of useful constraints which were previously regarded tough, including all the examples above.
- We characterize (Section 3) the class of convertible constraints using the notion of *prefix monotone* functions, and study the arithmetical closure properties of such functions. As a byproduct, we can show that large classes of constraints involving arithmetic are convertible, e.g., $max(S)/avg(S) \leq v$ is convertible anti-monotone and $median(S) - min(S) \geq v$ is convertible monotone.
- We show that convertible constraints cannot be pushed deep into the basic Apriori framework. However, they can be pushed deep into the frequent pattern growth mining. We thus develop (Section 4) algorithms for fast mining of frequent itemsets satisfying the various constraints.
- We report our results from a detailed set of experi-

*The work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, and the Networks of Centres of Excellence of Canada (NCE/IRIS-3).

ments, which show the effectiveness of the algorithms developed(Section 5); and finally, we conclude the study in Section 7.

2. Problem Definition: Frequent Itemset Mining with Constraints

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of all *items*, where an item is an object with some predefined attributes (e.g., price, weight, etc.). A *transaction* $T = \langle tid, I_t \rangle$ is a tuple, where tid is the *identifier* of the transaction and $I_t \subseteq I$. A transaction database $\mathcal{T}$ consists of a set of transactions. An itemset $S \subseteq I$ is a subset of the set of items. A k-itemset is an itemset of size k. We write itemsets as $S = i_{j_1} i_{j_2} \cdots i_{j_k}$, omitting set brackets.

An itemset S is contained in a transaction $T = \langle tid, I_t \rangle$, if and only if $S \subseteq I_t$. The *support* $sup(S)$ of an itemset S in a transaction database $\mathcal{T}$ is the number of transactions in $\mathcal{T}$ containing S. Given a *support threshold* ξ $(1 \leq \xi \leq |\mathcal{T}|)$, an itemset S is *frequent* provided $sup(S) \geq \xi$.

A *constraint* C is a predicate on the powerset of the set of items I, i.e., $C : 2^I \rightarrow \{true, false\}$. An itemset S satisfies a constraint C if and only if $C(S)$ is true. The set of itemsets satisfying a constraint C is $sat_C(I) = \{S \mid S \subseteq I \wedge C(S) = \text{ true}\}$. We call an itemset in $sat_C(I)$ *valid*.

Problem definition. Given a transaction database $\mathcal{T}$, a support threshold ξ, and a set of constraints $\mathcal{C}$, the problem of *mining frequent itemsets with constraints* is to find the complete set of frequent itemsets satisfying $\mathcal{C}$, i.e., find $\mathcal{F}_{\mathcal{C}} = \{S \mid S \in sat_C(I) \wedge sup(S) \geq \xi\}$.

Many kinds of constraints can be associated with frequent itemset mining. Two categories of constraints, *succinctness* and *anti-monotonicity*, were proposed in [7, 6]; whereas the third category, *monotonicity*, was studied in [3, 4, 8] in the contexts of mining correlated sets and frequent itemsets. We briefly recall these notions below.

Definition 2.1 (Anti-monotone, Monotone, and Succinct Constraints) A constraint C_a is *anti-monotone* if and only if whenever an itemset S violates C_a, so does any superset of S. A constraint C_m is *monotone* if and only if whenever an itemset S satisfies C_m, so does any superset of S. Succinctness is defined in steps, as follows.

- An itemset $I_s \subseteq I$ is a succinct set, if it can be expressed as $\sigma_p(I)$ for some selection predicate p, where σ is the selection operator.
- $SP \subseteq 2^I$ is a succinct powerset, if there is a fixed number of succinct sets $I_1, I_2, \ldots, I_k \subseteq I$, such that SP can be expressed in terms of the strict powersets of $I_1, \ldots, I_k$ using union and minus.
- Finally, a constraint C_s is *succinct* provided $sat_{C_s}(I)$ is a succinct powerset.

We can show the following result.

Theorem 2.1 *A constraint C is both anti-monotonic and monotonic if and only if $C(S) \equiv true$ for all itemset S, or $C(S) \equiv false$ for all itemset S.*

Theorem 2.2 *Every succinct constraint involving only aggregate functions can be expressed using conjunction and/or disjunction of monotone and anti-monotone constraints.*

These three categories of constraints cover a large class of popularly encountered constraints. However, there are still many useful constraints, such as $avg(S)\ \theta\ v$ and $sum(S)\ \theta\ v$ where $\theta \in \{\leq, \geq\}$ (shown in the table) that belong to *none* of the three classes.

Example 1 Let Table 1 be our running transaction database $\mathcal{T}$, with a set of items $I = \{a, b, c, d, e, f, g, h\}$. Let the support threshold be $\xi = 2$. Itemset $S = acd$ is frequent since it is in transactions 10 and 30, respectively. The complete set of frequent itemsets are listed in Table 2.

Transaction ID	Items in transaction
10	a, b, c, d, f
20	b, c, d, f, g, h
30	a, c, d, e, f
40	c, e, f, g

Table 1. The transaction database $\mathcal{T}$ in Example 1.

Length l	Frequent l-itemsets
1	a, b, c, d, e, f, g
2	$ac, ad, af, bc, bd, bf, cd, ce, cf, cg, df, ef, fg$
3	$acd, acf, adf, bcd, bcf, bdf, cdf, cef, cfg$
4	$acdf, bcdf$

Table 2. Frequent itemsets with support threshold $\xi = 2$ in transaction database $\mathcal{T}$ in Table 1.

Let each item have an attribute *value* (such as *profit*), with the concrete value shown in Table 3. In all constraints such as $sum(S)\ \theta\ v$, we implicitly refer to this value.

Item	a	b	c	d	e	f	g	h
Value	40	0	-20	10	-30	30	20	-10

Table 3. The values (such as profit) of items in Example 1.

The constraint $range(S) \leq 15$ requires that for an itemset S, the value range of the items in S must be no greater than 15. It is an anti-monotone constraint, in the sense that if an itemset, say ab, violates the constraint, any of its supersets will violate it; and thus ab can be removed safely from the candidate set during an Apriori-like frequent itemset mining process [7]. However, the constraint $C_{avg} \equiv avg(S) \geq 25$ is not anti-monotone (nor monotone, nor succinct, which can be verified by readers). For example, $avg(df) = (10 + 30)/2 < 25$, vio-

lates the constraint. However, upon adding one more item a, $avg(adf) = (40 + 10 + 30)/3 \geq 25$, adf satisfies C_{avg}.

This example scratches the surface of a large class of useful constraints involving avg, $median$, etc. as well as arithmetic. Exploiting them in mining calls for new techniques, which is the subject of this paper.

3. Convertible Constraints and Their Classification

Before introducing the concept of convertible constraint, we motivate it with an example.

Example 2 Suppose we wish to mine frequent itemsets over transaction database $\mathcal{T}$ in Table 1, with the support threshold $\xi = 2$ and with constraint $C \equiv avg(S) \geq 25$.

The complete set of frequent itemsets satisfying C can be obtained by first mining the frequent itemsets without using the constraint (i.e., Table 2) and then filtering out those not satisfying the constraint. Since the constraint is neither anti-monotone, nor monotone, nor succinct, it cannot be directly incorporated into an Apriori-style algorithm. E.g., itemset fg satisfies the constraint, while its subset g and its superset dfg do not.

If we arrange the items in *value-descending* order, $\langle a, f, g, d, b, h, c, e \rangle$, we can observe an interesting property, as follows. Writing itemsets w.r.t. this order leads to a notion of a prefix. E.g., afd has af and a as its prefixes. Interestingly, the average of an itemset is no more than that of its prefix, according to this order.

3.1. Convertible Constraints

The observation made in Example 2 motivates the following definition. We will frequently make use of an order[1] over the set of all items and assume itemsets are written according to this order.

Definition 3.1 (Prefix itemset) Given an order $\mathcal{R}$ over the set of items I, an itemset $S' = i_1 i_2 \cdots i_l$ is called a *prefix* of itemset $S = i_1 i_2 \cdots i_m$ w.r.t. $\mathcal{R}$, where $(l \leq m)$ and items in both itemsets are listed according to order $\mathcal{R}$. S' is called a *proper prefix* of S if $(l < m)$.

We next formalize convertible constraints as follows.

Definition 3.2 (Convertible Constraints) A constraint C is *convertible anti-monotone* provided there is an order $\mathcal{R}$ on items such that whenever an itemset S satisfies C, so does any prefix of S. It is *convertible monotone* provided there is an order $\mathcal{R}$ on items such that whenever an itemset S violates C, so does any prefix of S. A constraint is *convertible* whenever it is convertible anti-monotone or monotone.

[1] Unless otherwise stated, every order used in this paper is assumed to be total over the set of items.

Note that any anti-monotone (resp., monotone) constraint is trivially convertible anti-monotone (resp., convertible monotone): just pick any order on items.

Example 3 We show $avg(S)\ \theta\ v$ where $\theta \in \{\leq, \geq\}$ is a convertible constraint.

Let $\mathcal{R}$ be the value-descending order. Given an itemset $S = a_1 a_2 \cdots a_l$ satisfying the constraint $avg(S) \geq v$, where items in S are listed in the order $\mathcal{R}$. For each prefix $S' = a_1 \cdots a_k$ of S $(1 \leq k \leq l)$, since $a_k \geq a_{k+1} \geq \cdots \geq a_{l-1} \geq a_l$, we have $avg(S') \geq avg(S' \cup \{a_{k+1}\}) \geq \cdots \geq avg(S) \geq v$. This implies S' also satisfies the constraint. So, constraint $avg(S) \geq v$ is convertible anti-monotone. Similarly, it can be shown that constraint $avg(S) \leq v$ is convertible monotone.

Interestingly, if the order $\mathcal{R}^{-1}$ (i.e., the reversed order of $\mathcal{R}$) is used, the constraint $avg(S) \geq v$ can be shown convertible monotone. For lack of space, we leave this as an exercise to the reader.

In summary, constraint $avg(S)\ \theta\ v$ is convertible constraint. Furthermore, there exists an order $\mathcal{R}$ such that the constraint is convertible anti-monotone w.r.t. $\mathcal{R}$ and convertible monotone w.r.t. $\mathcal{R}^{-1}$.

As another example, let us examine the constraints with function $sum(S)$.

Example 4 Constraint $sum(S) \leq v$ is anti-monotone if items are all with non-negative values. However, if items are with negative, zero or positive values, the constraint becomes neither anti-monotone, nor monotone, nor succinct.

Curiously, this constraint exhibits a "piecewise" convertible monotone or anti-montone behavior. If $v \geq 0$ in the constraint, the constraint is convertible anti-monotone w.r.t. item value *ascending* order. Given an itemset $S = a_1 a_2 \cdots a_l$ such that $sum(S) \leq v$, where items are listed in value ascending order. For a prefix $S' = a_1 a_2 \cdots a_j$ $(1 \leq j \leq l)$, if $a_j \leq 0$, that means $a_1 \leq a_2 \leq \cdots \leq a_{j-1} \leq a_j \leq 0$. So, $sum(S') \leq 0 \leq v$. On the other hand, if $a_j > 0$, we have $0 < a_j \leq a_{j+1} \leq \cdots \leq a_l$. Thus, $sum(S') = sum(S) - sum(a_{j+1} \cdots a_l) < v$. Therefore, $sum(S') \leq v$ in both cases, which means S' satisfies the constraint.

If $v \leq 0$ in the constraint, it becomes convertible monotone w.r.t. item value descending order. We leave it to the reader to verify this.

Similarly, we can also show that, if items are with negative, zero or positive values, constraint $sum(S) \geq v$ is convertible monotone w.r.t. value ascending order when $v \geq 0$, and convertible anti-monotone w.r.t. value descending order when $v \leq 0$.

The following lemma can be proved with a straightforward induction.

Lemma 3.1 *Let C be a constraint over a set of items I.*

1. *C is convertible anti-monotone if and only if there exists an order $\mathcal{R}$ over I such that for every itemset S*

and item $a \in I$ such that $\forall x \in S, x \mathcal{R} a$, $C(S \cup \{a\})$ implies $C(S)$.

2. *C is convertible monotone if and only if there exists an order $\mathcal{R}$ over I such that for every itemset S and item $a \in I$ such that $\forall x \in S, x \mathcal{R} a$, $C(S)$ implies $C(S \cup \{a\})$.*

The notion of prefix monotone functions, introduced below, is helpful in determining the class of a constraint. We denote the set of real numbers as $\mathbf{R}$.

Definition 3.3 (Prefix monotone functions) Given an order $\mathcal{R}$ over a set of items I, a function $f : 2^I \rightarrow \mathbf{R}$ is a *prefix (monotonically) increasing function* w.r.t. $\mathcal{R}$ if and only if for every itemset S and its prefix S' w.r.t. $\mathcal{R}$, $f(S') \leq f(S)$. A function $g : 2^I \rightarrow \mathbf{R}$ is called a *prefix (monotonically) decreasing function* w.r.t. $\mathcal{R}$ if and only if for every itemset S and its prefix S' w.r.t. $\mathcal{R}$, $g(S') \geq g(S)$.

We have the following lemma on the determination of prefix monotone functions. The proof is similar to that of Lemma 3.1.

Lemma 3.2 *Given an order $\mathcal{R}$ over a set of items I.*

1. *A function $f : 2^I \rightarrow \mathbf{R}$ is a prefix decreasing function w.r.t. $\mathcal{R}$ if and only if for every itemset S and item a such that $\forall x \in S, x \mathcal{R} a$, $f(S) \geq f(S \cup \{a\})$.*
2. *A function $g : 2^I \rightarrow \mathbf{R}$ is a prefix increasing function w.r.t. $\mathcal{R}$ if and only if for every itemset S and item a such that $\forall x \in S, x \mathcal{R} a$, $g(S) \leq g(S \cup \{a\})$.*

It turns out that prefix monotone functions satisfy interesting closure properties with arithmetic. An understanding of this would shed light on characterizing a whole class of convertible functions involving arithmetic. The following theorem establishes the arithmetical closure properties of prefix monotone functions. We say a function $f : 2^I \rightarrow \mathbf{R}$ is positive, provided $\forall S \subseteq I : f(S) > 0$.

Theorem 3.1 *Let f and f' be prefix decreasing functions, and g and g' be prefix increasing functions w.r.t. an order $\mathcal{R}$, respectively. Let c be a positive real number.*

1. *Functions $-f(S)$, $\frac{1}{f(S)}$, $c \cdot g(S)$ and $g(S) + g'(S)$ are prefix increasing functions. Functions $-g(S)$, $\frac{1}{g(S)}$, $c \cdot f(S)$ and $f(S) + f'(S)$ are prefix decreasing functions.*
2. *If f and g are positive functions, then $f(S) \times f'(S)$ is prefix decreasing, and $g(S) \times g'(S)$ is prefix increasing.*
3. *A constraint $h(S) \geq v$ (resp., $h(S) \leq v$) is convertible anti-monotone (resp., monotone) if and only if h prefix decreasing. Similarly, $h(S) \geq v$ (resp., $h(S) \leq v$) is convertible monotone (resp., anti-monotone) if and only if h is prefix increasing.*

Example 5 As an illustration, notice that $avg(S)$ is a prefix decreasing function w.r.t. value-descending order, and $avg(S) \geq 20$ is convertible anti-monotone w.r.t. the same order. Also, $max(S)$ is a prefix increasing[2] function w.r.t. this order. From Theorem 3.1, it follows that $1/avg(S)$ is prefix increasing and hence $max(S)/avg(S)$ is prefix increasing.[3] Consequently, we immediately deduce that $max(S)/avg(S) \leq v$ is convertible anti-monotone w.r.t. this order.

We know from Theorem 2.2 that a succinct constraint can be expressed in terms of conjunction and/or disjunction of anti-monotone and monotone constraints. By definition, every monotone/anti-monotone is convertibly so. A natural question is, what is the relationship between succinct constraints and convertible constraints? The following theorem settles this question.

Theorem 3.2 *Every succinct constraint is either anti-monotone, or monotone, or convertible.*

Proof Sketch. The proof of the theorem is constructed by induction on the structure of $sat_C(I)$ of a succinct constraint C, according to the definition of succinctness.

3.2. Strongly convertible constraint

Some convertible constraints have the additional desirable property that w.r.t. an order $\mathcal{R}$ they are convertible anti-monotone, while w.r.t. its inverse $\mathcal{R}^{-1}$ they are convertible monotone. E.g., $avg(S) \leq v$ is convertible monotone w.r.t. value ascending order and convertible anti-monotone w.r.t. value descending order (see also Example 3). This property provides great flexibility in data mining query optimization.

Definition 3.4 (Strongly convertible constraint) A constraint C_{sc} is called a *strongly convertible constraint*, provided there exists an order $\mathcal{R}$ over the set of items such that C_{sc} is convertible anti-monotone w.r.t. $\mathcal{R}$ and convertible monotone w.r.t. $\mathcal{R}^{-1}$.

Notice that $median(S) \ \theta \ v$ ($\theta \in \{\leq, \geq\}$) is also strongly convertible. Clearly, not every convertible constraint is strongly convertible. E.g., $max(S)/avg(S) \leq v$ [4] is convertible anti-monotone w.r.t. value descending order, when all items have a non-negative value. However, it is not convertible monotone w.r.t. value ascending order.

The following lemma links strongly convertible constraints to prefix monotone functions.

Lemma 3.3 *Constraint $f(S) \ \theta \ v$ is strongly convertible, if and only if there exists an order $\mathcal{R}$ over the set of items such that f is a prefix decreasing function w.r.t. $\mathcal{R}$ and a prefix increasing function w.r.t. $\mathcal{R}^{-1}$.*

[2] It is also prefix decreasing w.r.t. this order.

[3] Assuming all items have non-negative values.

[4] It says the proportion of the max price of any item in the itemset over the average price of the items in the set cannot go over certain limit.

For example, $avg(S)$ and $median(S)$ are both prefix decreasing w.r.t. value descending order and prefix increasing w.r.t. value ascending order.

There still exist some constraints that cannot be pushed by item ordering. For example, the constraint $avg(S) - median(S) = 0$[5] does not admit any natural ordering on items w.r.t. which it is convertible. We call such constraints *inconvertible*.

3.3. Summary: a classification on constraints

As a general picture, constraints (only involving aggregate functions) can be classified into the following categories according to their interactions with the frequent itemset mining process: *anti-monotone, monotone, succinct* and *convertible*, which in turn can be subdivided into *convertible anti-monotone* and *convertible monotone*. The intersection of the last two categories is precisely the class of *strongly convertible* constraints (which can be treated either as convertible anti-monotone or monotone by ordering the items properly). Figure 1 shows the relationship among the various classes of constraints.

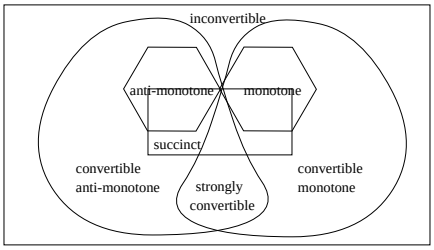

Figure 1. A classification of constraints and their relationships

Some commonly used convertible constraints are listed in Table 4.

4. Mining Algorithms

In this section, we explore how to mine frequent itemsets with convertible constraints efficiently. The general idea is to push the constraint into the mining process as deep as possible, thereby pruning the search space.

In Section 4.1, we first argue that the Apriori algorithm cannot be extended to mining with convertible constraints efficiently. Then, a new method is proposed by examining an example. Section 4.2 presents the algorithm $\mathcal{FIC}^{\mathcal{A}}$ for mining frequent itemsets with convertible anti-monotone constraints. Algorithm $\mathcal{FIC}^{\mathcal{M}}$, which computes the complete set of frequent itemsets with convertible monotone constraint, is given in Section 4.3. Section 4.4 discusses mining frequent itemsets with strongly convertible constraints.

[5]The constraint requires the median item in the itemset is with the average value.

4.1. Mining frequent itemsets with convertible constraints: An example

We first show that convertible constraints cannot be pushed deep into the Apriori-like mining.

Remark 4.1 *A convertible constraint that is neither monotone, nor anti-monotone, nor succinct, cannot be pushed deep into the* Apriori *mining algorithm.*

Rationale. As observed earlier for such a constraint (e.g., $avg(S) \leq v$), subsets (supersets) of a valid itemset could well be invalid and vice versa. Thus, within the levelwise framework, no direct pruning based on such a constraint can be made. In particular, whenever an invalid subset is eliminated without support counting, its supersets that are not suffixes cannot be pruned using frequency.

For example, itemset df in our running example violates the constraint $avg(S) \geq 25$. However, an Apriori-like algorithm cannot prune such itemsets. Otherwise, its superset adf, which satisfies the constraint, cannot be generated.

Before giving our algorithms for mining with convertible constraints, we give an overview in the following example.

Example 6 Let us mine frequent itemsets with constraint $C \equiv avg(S) \geq 25$ over transaction database $\mathcal{T}$ in Table 1, with the support threshold $\xi = 2$. Items in every itemset are listed in value descending order $\mathcal{R}$: $\langle a(40), f(30), g(20), d(10), b(0), h(-10), c(-20), e(-30)\rangle$. It is shown that constraint C is *convertible anti-monotone* w.r.t. $\mathcal{R}$. The mining process is shown in Figure 2.

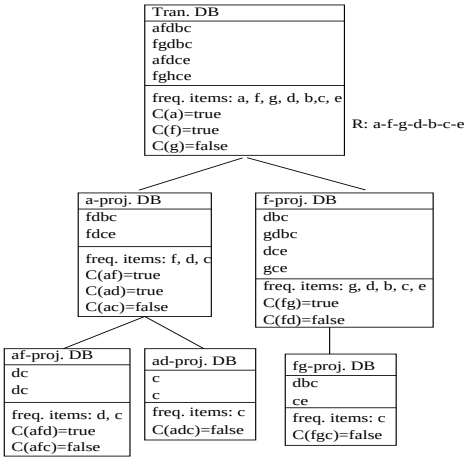

Figure 2. Mining frequent itemsets satisfying constraint $avg(S) \geq 25$.

By scanning $\mathcal{T}$ once, we find the support counts for every item. Since h appears in only one transaction, it is an infrequent items and is thus dropped without further consideration. The set of frequent 1-itemsets are a, f, g, d, b, c and e, listed in order $\mathcal{R}$. Among them, only a and f satisfy

Constraint	Convertible anti-monotone	Convertible monotone	Strongly convertible
$avg(S)\ \theta\ v\ (\theta \in \{\leq, \geq\})$	yes	yes	yes
$median(S)\ \theta\ v\ (\theta \in \{\leq, \geq\})$	yes	yes	yes
$sum(S) \leq v\ (v \geq 0, \forall a \in S, a\vartheta 0, \theta, \vartheta \in \{\leq, \geq\})$	yes	no	no
$sum(S) \leq v\ (v \leq 0, \forall a \in S, a\vartheta 0, \theta, \vartheta \in \{\leq, \geq\})$	no	yes	no
$sum(S) \geq v\ (v \geq 0, \forall a \in S, a\vartheta 0, \theta, \vartheta \in \{\leq, \geq\})$	no	yes	no
$sum(S) \geq v\ (v \leq 0, \forall a \in S, a\vartheta 0, \theta, \vartheta \in \{\leq, \geq\})$	yes	no	no
$f(S) \geq v$ (f is a prefix decreasing function)	yes	$*$	$*$
$f(S) \geq v$ (f is a prefix increasing function)	$*$	yes	$*$
$f(S) \leq v$ (f is a prefix decreasing function)	$*$	yes	$*$
$f(S) \leq v$ (f is a prefix increasing function)	yes	$*$	$*$

Table 4. Characterization of some commonly used, SQL-based convertible constraints. ($*$ means it depends on the specific constraint.)

the constraint[6]. Since C is a convertible anti-monotone constraint, itemsets having g, d, b, c or e as prefix cannot satisfy the constraint. Therefore, the set of frequent itemsets satisfying the constraint can be partitioned into two subsets:

1. The ones having itemset a as a prefix w.r.t. $\mathcal{R}$, i.e., those containing item a; and
2. The ones having itemset f as a prefix w.r.t. $\mathcal{R}$, i.e., those containing item f but no a.

The two subsets form two projected databases [5] which are mined respectively.

1. **Find frequent itemsets satisfying the constraint and having a as a prefix.** First, a is a frequent itemset satisfying the constraint. Then, the frequent itemsets having a as a proper prefix can be found in the subset of transactions containing a, which is called *a-projected database*. Since a appears in every transaction in the a-projected database, it is omitted. The a-projected database contains two transactions: $bcdf$ and $cdef$. Since items b and e is infrequent within this projected database, neither ab nor ae can be frequent. So, they are pruned. The frequent items in the a-projected database is f, d, c, listed in the order $\mathcal{R}$. Since ac does not satisfy the constraint, there is no need to create an ac-projected database.

   To check what can be mined in the a-projected database with af and ad, as prefix, respectively, we need to construct the two projected databases and mine them. This process is similar to the mining of a-projected databases. The *af-projected database* contains two frequent items d and c, and only afd satisfy the constraint. Moreover, since $afdc$ does not satisfies the constraint, the process in this branch is complete. Since afc violates the constraint, there is no need to construct afc-projected database. The *ad-projected database* contains one frequent item c, but adc does not satisfy the constraint. Therefore, the set of frequent itemsets satisfying the constraint and having a as prefix contains a, af, afd, and ad.

2. **Find frequent itemsets satisfying the constraint and having f as a prefix**. Similarly, the *f-projected database* is the subset of transactions containing f, with both a and f removed. It has four transactions: bcd, $bcdg$, cde and ceg. The frequent items in the projected database are g, d, b, c, e, listed in the order of $\mathcal{R}$. Since only itemsets fg and fd satisfy the constraint, we only need to explore if there is any frequent itemset having fg or fd as a proper prefix which satisfies the constraint. The *projected fg-database* contains no frequent itemset with fg as a proper prefix that satisfies the constraint. Since b is the item immediately after d in order $\mathcal{R}$, and fdb violates the constraint, any itemset having fd as a proper prefix cannot satisfy the constraint. Thus, f and fg are the only two frequent itemsets having f as a prefix and satisfying the constraint.

In summary, the complete set of frequent itemsets satisfying the constraint contains 6 itemsets: a, f, af, ad, afd, fg. Our new method generates and tests only a small set of itemsets.

4.2. $\mathcal{FIC}^{\mathcal{A}}$: Mining frequent itemsets with convertible anti-monotone constraint

Now, let us justify the correctness and completeness of the mining process in Example 6.

First, we show that the complete set of frequent itemsets satisfying a given convertible anti-monotone constraint can be partitioned into several non-overlapping subsets. It leads to the soundness of our algorithmic framework.

Lemma 4.1 *Consider a transaction database $\mathcal{T}$, a support threshold ξ and a convertible anti-monotone constraint C w.r.t. an order $\mathcal{R}$ over a set of items I. Let $a_1, a_2, \ldots, a_m$ be the items satisfying C. The complete set of frequent itemsets satisfying C can be partitioned into m disjoint subsets: the j^{th} subset $(1 \leq j \leq m)$ contains frequent itemsets satisfying C and having a_j as a prefix.*

We mine the subsets of frequent itemsets satisfying the constraint by constructing the corresponding *projected database*.

[6]The fact that itemset g does not satisfy the constraint implies none of any 1-itemsets after g in order $\mathcal{R}$ can satisfy the constraint avg.

Definition 4.1 (Projected database) Given a transaction database $\mathcal{T}$, an itemset α and an order $\mathcal{R}$.

1. Itemset β is called the *max-prefix projection* of transaction $\langle tid, I_t \rangle \in \mathcal{T}$ w.r.t. $\mathcal{R}$, if and only if (1) $\alpha \subseteq I_t$ and $\beta \subseteq I_t$; (2) α is a prefix of β w.r.t. $\mathcal{R}$; and (3) there exists no proper superset γ of β such that $\gamma \subseteq I_t$ and γ also has α as a prefix w.r.t. $\mathcal{R}$.
2. The α-*projected database* is the collection of max-prefix projections of transactions containing α, w.r.t. $\mathcal{R}$.

Remark 4.2 *Given a transaction database $\mathcal{T}$, a support threshold ξ and a convertible anti-monotone constraint C. Let α be a frequent itemset satisfying C. The complete set of frequent itemsets satisfying C and having α as a prefix can be mined from the α-projected database.*

The mining process can be further improved by the following lemma.

Definition 4.2 (Ascending and descending orders) An order $\mathcal{R}$ over a set of items I is called an *ascending order* for function $h : 2^I \rightarrow \mathbf{R}$ if and only if (1) for items a and b, $h(a) < h(b)$ implies $a \ \mathcal{R} \ b$, and (2) for itemsets $\alpha \cup \{a\}$ and $\alpha \cup \{b\}$ such that both of them have α as a prefix and $a \ \mathcal{R} \ b$, $f(\alpha \cup \{a\}) \leq f(\alpha \cup \{b\})$. $\mathcal{R}^{-1}$ is called a *descending order* for function h.

For example, it can be verified that the value ascending order is an *ascending* order for function $avg(S)$ and a *descending order* for function $max(S)/avg(S)$.

Lemma 4.2 *Given a convertible anti-monotone constraint $C \equiv f(S) \ \theta \ v \ (\theta \in \{\leq, \geq\})$ w.r.t. ascending/descending order $\mathcal{R}$ over a set of items I, where f is a prefix function. Let α be a frequent itemset satisfying C and $a_1, a_2, \ldots, a_m$ be the set of frequent items in α-projected database, listed in the order of $\mathcal{R}$.*

1. *If itemset $\alpha \cup \{a_i\}$ $(1 \leq i < m)$ violates C, for j such that $i < j \leq m$, itemset $\alpha \cup \{a_j\}$ also violates C.*
2. *If itemset $\alpha \cup \{a_j\}$ $(1 \leq j < m)$ satisfies C, but $\alpha \cup \{a_j, a_{j+1}\}$ violates C, no frequent itemset having $\alpha \cup \{a_j\}$ as a proper prefix satisfies C.*

Based on the above reasoning, we have the algorithm $\mathcal{FIC}^{\mathcal{A}}$ as follows for mining Frequent Itemsets with Convertible Anti-monotone constraints.

Algorithm 1 ($\mathcal{FIC}^{\mathcal{A}}$) Given a transaction database $\mathcal{T}$, a support threshold ξ and a convertible anti-monotone constraint C w.r.t. an order $\mathcal{R}$ over a set of items I, the algorithm computes the complete set of frequent itemsets satisfying the constraint C.

Method: Call $fic_a(\emptyset, \mathcal{T})$;

function $fic_a(\alpha, \mathcal{T}|_\alpha)$[7]

1. Scan $\mathcal{T}|_\alpha$ once, find frequent items in $\mathcal{T}|_\alpha$. Let I_α be the set of frequent items within $\mathcal{T}|_\alpha$ such that $\forall a \in I_\alpha, C(\alpha \cup \{a\}) = true$.
2. If $I_\alpha = \emptyset$ return, else $\forall a \in I_\alpha$, output $\alpha \cup \{a\}$ as a frequent itemset satisfying the constraint.
3. If C is in form of $f(S) \ \theta \ v$ where f is a prefix function and $\theta \in \{\leq, \geq\}$, using Lemma 4.2 to optimize the mining by removing items b from I_α such that there exists no frequent itemset satisfying C and having $\alpha \cup \{b\}$ as a proper prefix.
4. Scan $\mathcal{T}|_\alpha$ once more, $\forall a \in I|_\alpha$, generate $\alpha \cup \{a\}$-projected database $\mathcal{T}|_{\alpha \cup \{a\}}$.
5. For each item a in $I|_\alpha$, call $fic_a(\alpha \cup \{a\}, \mathcal{T}|_{\alpha \cup \{a\}})$.

Rationale. The correctness and completeness of the algorithm has been reasoned step-by-step in this section. The efficiency of the algorithm is that it pushes the constraint deep into the mining process, so that we do not need to generate the complete set of frequent itemsets in most of cases. Only related frequent itemsets are identified and tested. As shown in Example 6 and in the experimental results, the search space is decreased dramatically when the constraint is sharp.

4.3. $\mathcal{FIC}^{\mathcal{M}}$: Mining frequent itemsets with monotone constraints

In the last two subsections, an efficient algorithm for mining frequent itemsets with convertible anti-monotone constraints is developed. Under similar spirit, an algorithm for mining frequent itemsets with convertible monotone constraints can also be developed. Due to lack of space, instead of giving details of formal reasoning, we illustrate the ideas using an example and then present the algorithm.

Example 7 Let us mine frequent itemsets in transaction database $\mathcal{T}$ in Table 1 with constraint $C \equiv avg(S) \leq 20$. Suppose the support threshold $\xi = 2$. In this example, we use the value descending order $\mathcal{R}$ exactly as is used in Example 6. Constraint C is convertible monotone w.r.t. order $\mathcal{R}$.

After one scan of transaction database $\mathcal{T}$, the set of frequent 1-itemsets is found. Among the 7 frequent 1-itemsets, g, d, b, c and e satisfy the constraint C. According to the definition of convertible monotone constraints, frequent itemset having one of these 5 itemsets as a prefix must also satisfy the constraint. That is, the g-, d-, b-, c- and e-projected database can be mined without testing constraint C, because adding smaller items will only decrease the value of avg. But a- and f-projected databases should be mined with constraint C testing. However, as soon as its frequent k-itemsets for any k satisfy the constraint, constraint checking will not be needed for further mining of their projected databases.

We present the algorithm $\mathcal{FIC}^{\mathcal{M}}$ for mining frequent itemsets with convertible monotone constraint as follows.

[7] α is the itemset as prefix and $\mathcal{T}|_\alpha$ is the α-projected database.

Algorithm 2 ($\mathcal{FIC}^{\mathcal{M}}$) Given a transaction database $\mathcal{T}$, a support threshold ξ and a convertible monotone constraint C w.r.t. an order $\mathcal{R}$ over a set of items I, the algorithm computes the complete set of frequent itemsets satisfying the constraint C.

Method: Call $fic_m(\emptyset, \mathcal{T}, 1)$;

function $fic_m(\alpha, \mathcal{T}|_\alpha, check\_flag)$[8]

1. Scan $\mathcal{T}|_\alpha$ once, find frequent items in $\mathcal{T}|_\alpha$. If $check\_flag$ is 1, let I_α^+ be the set of frequent items within $\mathcal{T}|_\alpha$ such that $\forall a \in I_\alpha^+, C(\alpha \cup \{a\}) = true$, and I_α^- be the set of frequent items within $\mathcal{T}|_\alpha$ such that $\forall b \in I_\alpha^-, C(\alpha \cup \{b\}) = false$. If $check\_flag$ is 0, let I_α^+ be the set of frequent items within $\mathcal{T}|_\alpha$ and I_α^- be $\emptyset$.
2. $\forall a \in I_\alpha^+$, output $\alpha \cup \{a\}$ as a frequent itemset satisfying the constraint.
3. Scan $\mathcal{T}|_\alpha$ once more, $\forall a \in I|_\alpha^+ \cup I|_\alpha^-$, generate $\alpha \cup \{a\}$-projected database $\mathcal{T}|_{\alpha\cup\{a\}}$.
4. For each item a in $I|_\alpha^+$, call $fic_m(\alpha \cup \{a\}, \mathcal{T}|_{\alpha\cup\{a\}}, 0)$; For each item a in $I|_\alpha^-$, call $fic_m(\alpha \cup \{a\}, \mathcal{T}|_{\alpha\cup\{a\}}, 1)$;

Rationale. The correctness and completeness of the algorithm can be shown based on the similar reasoning in Section 4.2. Here, we analyze the difference between $\mathcal{FIC}^{\mathcal{M}}$ with an Apriori-like algorithm using constraint-checking as post-processing.

Both $\mathcal{FIC}^{\mathcal{M}}$ and Apriori-like algorithms have to generate the complete set of frequent itemsets, no matter whether the frequent itemsets satisfy the convertible monotone constraint. The frequent itemsets not satisfying the constraint cannot be pruned. That is the inherent difficulty of convertible monotone constraint.

The advantage of $\mathcal{FIC}^{\mathcal{M}}$ against Apriorix-like algorithms lies in the fact that $\mathcal{FIC}^{\mathcal{M}}$ only tests some of frequent itemsets against the constraint. Once a frequent itemset satisfies the constraint, it guarantees all of frequent itemsets having it as a prefix also satisfy the constraint. Therefore, all that testing can be saved. An Apriori-like algorithm has to check every frequent itemset against the constraint. In the situation such that constraint testing is costly, such as spatial constraints, the saving over constraint testing could be non-trivial. Exploration of spatial constraints is beyond the scope of this paper.

4.4. Mining frequent itemsets with strongly convertible constraints

The main value of strong convertibility is that the constraint can be treated either as convertible anti-monotone or monotone by choosing an appropriate order. The main point to note in practice is when the constraint has a high selectivity (fewer itemsets satisfy it), converting it into an anti-monotone constraint will yield maximum benefits by search space pruning. When the constraint selectivity is low (and checking it is reasonably expensive), then converting it into a monotone constraint will save considerable effort in constraint checking. The constraint $avg(S) \leq v$ is a classic example.

[8] α is the itemset as prefix, $\mathcal{T}|_\alpha$ is the α-projected database, and $check\_flag$ is the flag for constraint checking.

5. Experimental Results

To evaluate the effectiveness and efficiency of the algorithms, we performed an extensive experimental evaluation.

In this section, we report the results on a synthetic transaction database with 100K transactions and 10K items. The dataset is generated by the standard procedure described in [1]. In this dataset, the average transaction size and average maximal potentially frequent itemset size are set to 25 and 20, respectively. The dataset contains a lot of frequent itemsets with various length. This dataset is chosen since it is typical in data mining performance study.

The algorithms are implemented in C. All the experiments are performed on a 233MHz Pentium PC with 128MB main memory, running Microsoft Windows/NT.

To evaluate the effect of a constraint on mining frequent itemsets, we make use of constraint selectivity, where the *selectivity* δ of a constraint C on mining frequent itemsets over transaction database $\mathcal{T}$ with support threshold ξ is defined as

$$\delta = \frac{\text{\# of frequent itemsets NOT satisfying } C}{\text{\# of frequent itemsets}}$$

Therefore, a constraint with 0% selectivity means every frequent itemset satisfies the constraint, while a constraint with 100% selectivity is the one cannot be satisfied by any frequent itemset. The selectivity measure defined here is consistent with those used in [7, 6].

To facilitate the mining using projected databases, we employ a data structure called FP-tree in the implementations of $\mathcal{FIC}^{\mathcal{A}}$ and $\mathcal{FIC}^{\mathcal{M}}$. FP-tree is first proposed in [5], and also be adopted by [8, 9]. It is a prefix tree structure to record complete and compact information for frequent itemset mining. A transaction database/projected database can be compressed into an FP-tree, while all the consequent projected databases can be derived from it efficiently. We refer readers to [5] for details about FP-tree and methods for FP-tree-based frequent itemset mining.

Since FP-growth [5] is the FP-tree-based algorithm mining frequent itemsets and is much faster than Apriori, we include it in our experiment. Comparison among $\mathcal{FIC}^{\mathcal{A}}$, $\mathcal{FIC}^{\mathcal{M}}$ and FP-growth makes more sense than using pure Apriori as the only reference method.

5.1. Evaluation of $\mathcal{FIC}^{\mathcal{A}}$

To test the efficiency of $\mathcal{FIC}^{\mathcal{A}}$ w.r.t. constraint selectivity in mining frequent itemsets with convertible anti-monotone constraints, we run a test over the dataset with

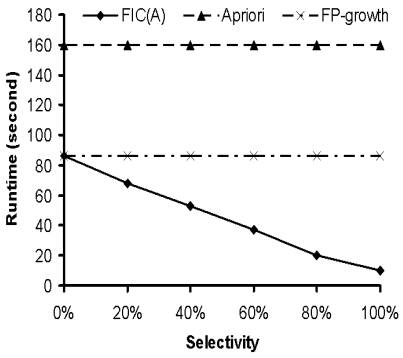

Figure 3. Scalability with constraint selectivity.

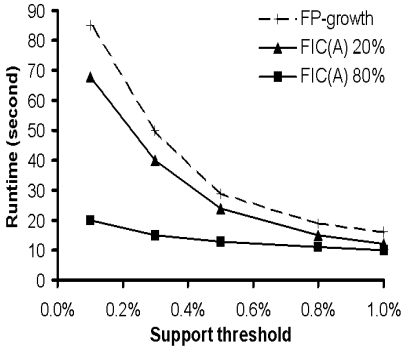

Figure 4. Scalability with support threshold.

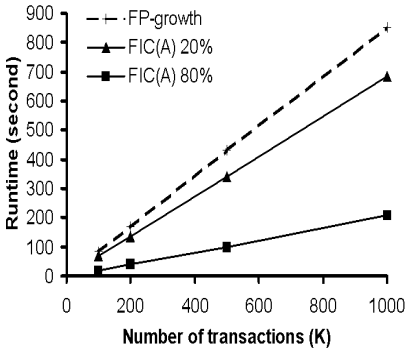

Figure 5. Scalability with number of transactions.

support threshold $\xi = 0.1\%$. The result is shown in Figure 3. Various settings are used in the constraint for various selectivities.

As can be seen from the figure, $\mathcal{FIC}^{\mathcal{A}}$ achieves an almost linear scalability with the constraint selectivity. As the selectivity goes up, i.e., fewer itemsets satisfy the constraint, $\mathcal{FIC}^{\mathcal{A}}$ cuts more search space, since one frequent itemset not satisfying the constraint means all frequent itemsets having it as a prefix can be pruned.

We compare the runtime of both Apriori and FP-growth in the same figure. All these two methods first compute the complete set of frequent itemsets, and then use the constraint as a filter. So, their runtime is constant w.r.t. constraint selectivity. However, only when the constraint selectivity is 0%, i.e., every frequent itemset satisfies the constraint, does $\mathcal{FIC}^{\mathcal{A}}$ need the same runtime as FP-growth. In all other situations, $\mathcal{FIC}^{\mathcal{A}}$ always requires less time.

We also tested the scalability of $\mathcal{FIC}^{\mathcal{A}}$ with support threshold and the number of transactions, respectively. The corresponding results are shown in Figure 4 and Figure 5. From these figures, we can see that $\mathcal{FIC}^{\mathcal{A}}$ is scalable in both cases. Furthermore, the higher the constraint selectivity, the more scalable $\mathcal{FIC}^{\mathcal{A}}$ is. That can be explained by the fact that $\mathcal{FIC}^{\mathcal{A}}$ always cuts more search space using constraints with higher selectivity.

5.2. Evaluation of $\mathcal{FIC}^{\mathcal{M}}$

As analyzed before, convertible monotone constraint can be used to save the cost of constraint checking, but it cannot cut the search space of frequent itemsets. In our experiments, since we use relatively simple constraints, such as those involving avg and sum, the cost of constraint checking is CPU-bound. However, the cost of the whole frequent itemset mining process is I/O-bound. This makes the effect of pushing convertible monotone constraint into the mining process hard to be observed from runtime reduction. In our experiments, $\mathcal{FIC}^{\mathcal{M}}$ achieves less than 3% runtime benefit in most cases.

However, if we look at the number of constraint tests performed, the advantage of $\mathcal{FIC}^{\mathcal{M}}$ can be evaluated objectively. $\mathcal{FIC}^{\mathcal{M}}$ can save a lot of effort on constraint testing. Therefore, in the experiments about $\mathcal{FIC}^{\mathcal{M}}$, the number of constraint tests is used as the performance measure.

We test the scalability of $\mathcal{FIC}^{\mathcal{M}}$ with constraint selectivity in mining frequent itemsets with convertible monotone constraint. The result is shown in Figure 6. The figure shows that $\mathcal{FIC}^{\mathcal{M}}$ has a linear scalability. When the constraint selectivity is low, i.e., most frequent itemsets can pass the constraint checking, most of constraint tests can be saved. This is because once a frequent itemset satisfies a convertible monotone constraint, every subsequent frequent itemset derived from corresponding projected database has that frequent itemset as a prefix and thus satisfies the constraint, too.

We also tested the scalability of $\mathcal{FIC}^{\mathcal{M}}$ with support threshold. The result is shown in Figure 7. The figure shows that $\mathcal{FIC}^{\mathcal{M}}$ is scalable. Furthermore, the lower the constraint selectivity, the better the scalability $\mathcal{FIC}^{\mathcal{M}}$ is.

In summary, our experimental results show that the method proposed in this paper is scalable for mining frequent itemsets with convertible constraints in large transaction databases. The experimental results strongly support our theoretical analysis.

6. Discussions: Mining Frequent Itemsets with Multiple Convertible Constraints

We have studied the push of *single* convertible constraints into frequent itemset mining. "*Can we push multiple constraints deep into the frequent pattern mining process*?"

Multiple constraints in a mining query may belong to the same category (e.g. all are anti-monotone) or to different categories. Moreover, different constraints may be on different properties of items (e.g. some could be on item price,

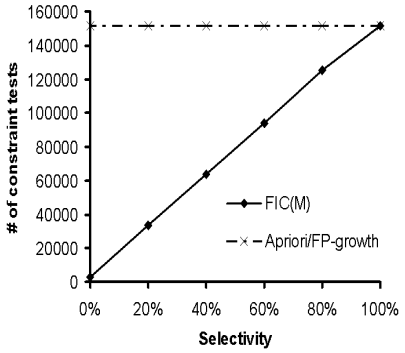

Figure 6. Scalability with constraint selectivity.

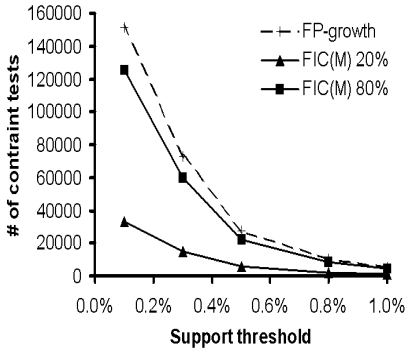

Figure 7. Scalability with support threshold.

others on sales profits, the number of items, etc.).

As shown in our previous analysis, unlike anti-monotone, monotone and succinct constraints, *convertible* constraints can be mined only by ordering items properly. However, different constraints may require different and even conflicting item ordering. Our general philosophy is to conduct a cost analysis to determine how to combine multiple order-consistent *convertible* constraints and how to select a sharper constraint among order-conflicting ones. The details will not be presented here for lack of space.

7. Conclusions

Constraints involving holistic functions such as $median$, algebraic functions such as avg, or even those involving distributive functions like sum over sets with positive and negative item values are difficult to incorporate in an optimization process in frequent itemset mining. The reason is such constraints do not exhibit nice properties like monotonicity, etc. A main contribution of this paper is showing that by imposing an appropriate order on items, such tough constraints can be converted into ones that possess monotone behavior. To this end, we made a detailed analysis and classification of the so-called convertible constraints. We characterized them using prefix monotone functions and established their arithmetical closure properties. As a byproduct, we shed light on the overall picture of various classes of constraints that can be optimized in frequent set mining. While convertible constraints cannot be literally incorporated into an Apriori-style algorithm, they can be readily incorporated into the FP-growth algorithm. Our experiments show the effectiveness of the algorithms developed.

We have been working on a systematic implementation of constraint-based frequent pattern mining in a data mining system. More experiments are needed to understand how best to handle multiple constraints. An open issue is given an arbitrary constraint, how can we quickly check if it is (strongly) convertible. We are also exploring the use of constraints in clustering.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, Sept. 1994.

[2] R. J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining on large, dense data sets. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, Sydney, Australia, Apr. 1999.

[3] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97)*, pages 265–276, Tucson, Arizona, May 1997.

[4] G. Grahne, L. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In *Proc. 2000 Int. Conf. Data Engineering (ICDE'00)*, pages 512–521, San Diego, CA, Feb. 2000.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.

[6] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 157–168, Philadelphia, PA, June 1999.

[7] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 13–24, Seattle, WA, June 1998.

[8] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, pages 350–354, Boston, MA, Aug. 2000.

[9] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00)*, pages 11–20, Dallas, TX, May 2000.

[10] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 67–73, Newport Beach, CA, Aug. 1997.