Can We Push More Constraints into Frequent Pattern Mining?

Jian Pei and Jiawei Han
School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
{peijian, han}@cs.sfu.ca

Abstract

Recent studies show that constraint pushing may substantially improve the performance of frequent pattern mining, and methods have been proposed to incorporate interesting constraints in frequent pattern mining. However, some popularly encountered constraints are still considered as "*tough*" constraints which cannot be pushed deep into the mining process.

In this study, we extend our scope to those tough constraints and identify an interesting class, called *convertible constraints*, which can be pushed deep into frequent pattern mining. Then we categorize all the constraints into five classes and show that four of them can be integrated into the frequent pattern mining process. This covers most of the constraints popularly encountered and composed by SQL primitives. Moreover, a new constraint-based frequent pattern mining method, called *constrained frequent pattern growth*, or simply CFG, which integrates constraint pushing with a recently developed frequent pattern growth method, is developed. We show this integration opens more room on constraint pushing since finer constraint checking can be enforced on each projected database. Our performance study shows that the method is powerful and outperforms substantially the existing constrained frequent pattern mining algorithms.

1 Introduction

As an active research theme in data mining, frequent pattern mining covers a broad spectrum of data mining tasks. Most of the previous studies work on some variants of the Apriori algorithm [1]. However, the Apriori-like method suffers from two major inefficiencies: (1) the candidate sets so generated could be very large, and (2) it is costly to repeatedly scan the database and accumulate the counts for a large set of candidates by pattern matching. Our recent research [3] proposes a *frequent pattern growth* method, FP-growth, which constructs a compressed frequent pattern tree, FP-tree, and applies a divide-and-conquer strategy to grow long patterns from short ones. This method avoids the costly candidate set generation and test, dramatically reduces the cost of frequent pattern mining, and has demonstrated an order of magnitude performance gain over the Apriori algorithm.

There is another research frontier on efficient frequent pattern mining: *the exploration of user-specified constraints*. Constrained mining facilitates user exploration and control, focuses its search on only the subset of patterns of interest to the user, and leads to efficient mining as well as concise, user-desired mining results. A systematic method of the incorporation of two important classes of constraints, *antimonotonicity* and *succinctness*, in frequent pattern mining are presented in [5, 4].

In this study, we propose to integrate the two frontiers, *highly efficient frequent pattern mining method* and *constrained mining*, and develop a constrained frequent pattern mining method which handles "tough" constraints and integrates constraint pushing with the newly developed frequent pattern growth method [3]. Our study has made the following contributions.

1. We extend our scope of constrained mining to the unsolved, "tough" constraints and identify an interesting class, called *convertible constraints*, which can be pushed deep into frequent pattern mining. Then we categorize all the constraints into five classes, i.e., *succinct*, *anti-monotone*, *monotone*, *convertible*, and *inconvertible*, and show that the first four of them can be pushed deep into the frequent pattern mining process. This covers most of the constraints popularly encountered and composed by SQL primitives.
2. We integrate constraint pushing with a recently developed frequent pattern growth mining method, and show this integration opens additional room on constraint pushing since finer constraint checking can be enforced on each projected database.
3. An integrated new constraint-based frequent pattern mining method, called *constrained frequent*

pattern growth, or simply CFG, is proposed for integration of multiple classes of constraints in frequent pattern mining. Our performance study shows that CFG achieves significant performance improvement over previously proposed constraint-based mining algorithms, and is efficient and scalable in large databases.

The remaining of the paper is organized as follows. In Section 2, we define the problem of *constrained pattern mining*, categorize the constraints, identify a new class of constraints, *convertible constraint*, and study the relationships among different classes of constraints. In Section 3, we develop constraint-based frequent pattern mining methods and report performance study results. In Section 4, we summarize the study.

2 Categories of Constraints

Let $I = \{i_1, i_2, \ldots, i_m\}$ be **a set of items**. A **transaction** $T = \langle tid, I_t \rangle$ is a tuple, where tid is the *identifier* of the transaction, and I_t is a subset of the set of items. A **transaction database** TDB is a set of transactions. A **pattern** S is a subset of the set of items. If $|S| = l$, S is also called a **(length)** l**-pattern**. For the sake of simplicity, a pattern $S = \{i_{j_1}, i_{j_2}, \ldots, i_{j_k}\}$ can also be denoted as $S = i_{j_1} i_{j_2} \cdots i_{j_k}$ in short.

A pattern S is **contained in** a transaction $T = \langle tid, I_t \rangle$ if and only if $S \subseteq I_t$. A pattern S' is a **subpattern** of pattern S and S is a **super pattern** of S', if and only if $S' \subseteq S$. If $S' \subset S$, S' is called a **proper subpattern** of S and S a **proper super pattern** of S'. The **support** of a pattern S in a transaction database TDB is the number of transactions in TDB containing S.

Given a **support threshold** ξ, $1 \leq \xi \leq |TDB|$, a pattern S is called a **frequent pattern** if and only if $support(S) \geq \xi$.

Given a transaction database TDB and a support threshold ξ, the problem of **frequent pattern mining** is to *find the complete set of frequent patterns in* TDB.

A **constraint** C is a predicate on the powerset of the set of items I. A pattern S **satisfies** a constraint C if and only if $C(S)$ is true. The complete set of patterns satisfying a constraint C, denoted as $SAT_C(I)$, is called **satisfying pattern set**, where I is the set of items.

Given a transaction database, a support threshold and a set of constraints C, the problem of **mining frequent patterns with constraints** is to *find the complete set of frequent patterns satisfying* C.

Many kinds of constraints can be associated with frequent pattern mining. Two categories of constraints, *succinctness* and *anti-monotonicity*, were proposed in [5, 4]; whereas the third category, *monotonicity*, was introduced in [2] in the context of mining correlated sets. Although the three categories of constraints cover a large portion of the constraints in frequent pattern mining, there still exist some useful constraints, such as $avg(S)\ \theta\ v$ where $\theta \in \{\leq, \geq\}$, which do not fall into either of them and are regarded as *tough* (i.e., hard to push) constraints.

Definition 1 (Anti-monotone and monotone constraints) A constraint C_a is **anti-monotone** if and only if for any pattern S not satisfying C_a, none of the super-patterns of S can satisfy C_a.

A constraint C_m is **monotone** if and only if for any pattern S satisfying the constraint, C_m, every super-pattern of S also satisfies C_m. □

Definition 2 (Succinct constraint)

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

A large portion of constraints in applications belongs to one of the above three categories of constraints. However, there are still many useful constraints which do not fall into any of these categories. For example, $avg(S)\ \theta\ v$ where $\theta \in \{\leq, \geq\}$, which requires that the average value of a pattern is no less (or no greater) than a given value v, is neither anti-monotone, nor monotone, nor succinct.

Fortunately, many of such *tough* constraints belong to a new category of constraints, called *convertible* constraints, which can still be effectively integrated into the frequent pattern mining process.

Given a total order R over the set of items I, a pattern S' is a **suffix** of a pattern S w.r.t. the order R if pattern $S = i_1 i_2 \cdots i_m$, in which items are sorted according to order R, and $S' = i_l i_{l+1} \cdots i_m$, where $1 \leq l \leq m$. If $l > 1$, S' is called a **proper suffix** of S (w.r.t. R).

Definition 3 (Convertible constraint) A constraint $C(S)$ is **convertible anti-monotone** if and only if (i) $C(S)$ is neither monotone, nor anti-monotone, nor succinct, and (ii) there exists an order R over the set of items I such that any pattern S satisfying the constraint implies that each suffix of S w.r.t. order R also satisfies the constraint.

A constraint $C(S)$ is **convertible monotone** if and only if (i) $C(S)$ is neither monotone, nor anti-monotone, nor succinct, and (ii) there exists an order R over the set of items I such that any pattern S satisfying the constraint implies that each pattern of which S is a suffix w.r.t. order R also satisfies the constraint.

A constraint $C(S)$ is **convertible** if and only if $C(S)$ is either convertible anti-monotone or convertible monotone. □

Example 1 (Convertible constraint) We first show $avg(S)\ \theta\ v$ where $\theta \in \{\leq, \geq\}$ is a convertible constraint.

Let R be the value descending order over the set of items I and S' be a suffix of a pattern S. There must exist some S'' such that (i) $S = S''S'$, and (ii) each item in S'' is greater than every item in S'. That follows $avg(S) \geq avg(S')$. That is, if $avg(S') \geq v$, then $avg(S) \geq v$. Thus, $avg(S) \geq v$ is convertible monotone.

Let R^{-1} be the reversed order of R, i.e., R^{-1} be the value ascending order over the set of items, and S' be a suffix of a pattern S. It is easy to see that $avg(S') \geq avg(S)$. That is, if the constraint $avg(S') \geq v$ is violated, $avg(S) \geq v$ cannot be satisfied. Therefore, $avg(S) \geq v$ is convertible anti-monotone.

Similarly, we can show that $avg(S) \leq v$ is also a convertible constraint. □

From this example, we can observe an interesting relationship between convertible monotone and convertible anti-monotone constraints for the constraint $avg(S)\ \theta\ v$ where $\theta \in \{\leq, \geq\}$: "*If a constraint C is convertible anti-monotone w.r.t. an order R over a set of items, C is convertible monotone w.r.t. order R^{-1}, which is the reversed order of R, and vice versa.*" Unfortunately, this property does not hold for all the convertible constraints. For example, $max(S)/avg(S) \leq v\ (\forall a \in S, a > 0)$ is a convertible anti-monotone constraint with respect to value ascending order. However, the constraint is not convertible monotone with respect to the value descending order.

Notice this does not mean that every tough constraint is convertible. For example, $sum(S)\ \theta\ v$, where $\theta \in \{\leq, \geq\}$ and each element in S could be of any real value, is not convertible. This implies that there still exists a distinct category of constraints, called *inconvertible constraints*, which belongs to none of the four categories discussed so far.

As a summary, a representative subset of constraints is listed in Table 1. Limited by space, only existence operators (e.g., $=, \in$, but not $\neq, \notin$) and comparison (or containment) operators with equality (e.g., $\leq$, $\subseteq$) are given.

Based on the above discussion, all the constraints can be classified into the following five categories w.r.t. frequent pattern mining: (1) anti-monotone, (2) monotone, (3) succinct, (4) convertible, and (5) inconvertible. Figure 1 illustrates the relationships among these five categories of constraints.

Although there still exist some tough constraints which are not even convertible, the good news is that most simple SQL expressions with built-in SQL aggregates belong to one of the first four categories to which efficient constraint mining methods can be explored, as shown in the next section.

Constraint	Anti.	Succ.	Mono.
$v \in S$	no	yes	yes
$S \supseteq V$	no	yes	yes
$S \subseteq V$	yes	yes	no
$min(S) \leq v$	no	yes	yes
$min(S) \geq v$	yes	yes	no
$max(S) \leq v$	yes	yes	no
$max(S) \geq v$	no	yes	yes
$count(S) \leq v$	yes	weakly	no
$count(S) \geq v$	no	weakly	yes
$sum(S) \leq v\ (\forall a \in S, a \geq 0)$	yes	no	no
$sum(S) \geq v\ (\forall a \in S, a \geq 0)$	no	no	yes
$range(S) \leq v$	yes	no	no
$range(S) \geq v$	no	no	yes
$avg(S)\ \theta\ v,\ \theta \in \{\leq, \geq\}$	conv.	no	conv.
$support(S) \geq \xi$	yes	no	no
$support(S) \leq \xi$	no	no	yes

Table 1: Characterization of commonly used, SQL-based constraints.

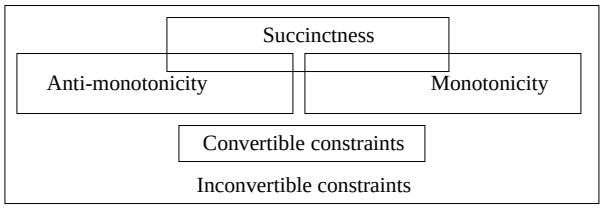

Figure 1: Relationships among different categories of constraints.

3 Pushing Constraints into Frequent Pattern Growth

In [3], we have demonstrated that the frequent pattern growth approach may achieve an order of magnitude performance gain in large databases over the Apriori approach. Thus it is natural to examine how constraints can be pushed into the frequent pattern growth process and whether further improvements can be explored with this new framework. Here, we only introduce the primitives of frequent pattern growth using the following short example, which will help us to pursue our constraint pushing techniques. We refer readers to [3] for details on frequent pattern growth approach.

Example 2 Let the transaction database TDB be Table 2, and the support threshold be $\xi = 3$. The frequent pattern growth mining proceeds as follows. The first scan of the transaction database TDB collects the support count of every item. The *frequent items* are sorted according to the support descending order (in the format of *item: frequency*) as $a : 5, e : 4, b : 3, c : 3, d : 3, f : 3$.

The prefixes of the ordered item list in each transaction can be projected to the corresponding *conditional databases* $TDB|_f$, $TDB|_d$, $TDB|_c$, ...,

Transaction ID	Items in transaction
100	a, c, d, e, f
200	a, b
300	a, c, e, f
400	a, b, c, d, e, f
500	a, b, d, e

Table 2: The transaction database TDB.

and $TDB|_e$, respectively, as shown in Figure 2. For example, the ordered item list $\langle a, e, c, d, f\rangle$ of transaction ID = 100 will insert f's prefix $\langle a, e, c, d\rangle$ into $TDB|_f$, then d's prefix $\langle a, e, c\rangle$ into $TDB|_d$, c's prefix $\langle a, e\rangle$ into $TDB|_c$, and e's prefix $\langle a\rangle$ into $TDB|_e$. By doing so, the set of frequent patterns in transaction database TDB are divided into: the frequent patterns containing f, the ones containing d but no f, the ones containing c but no d or f, etc.

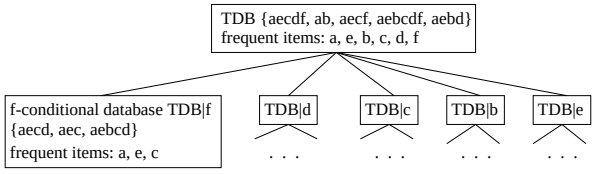

Figure 2: Mining frequent patterns in pattern growth paradigm.

A conditional database has the following property: If a pattern α is frequent in $TDB|_f$, $\alpha \cup f$ must be a frequent pattern in TDB. On the other hand, for every frequent pattern containing f in TDB except for the length-1 pattern f, the subpattern formed by removing f from it must also be frequent in $TDB|_f$. Thus, the problem of mining frequent patterns containing f is reduced to (i) including f as a frequent pattern, and (ii) finding frequent patterns in $TDB|_f$.

This process proceeds recursively: find the set of frequent items, β, in $TDB|_f$, called *conditional frequent itemset* of f, and for each frequent item e in β, find the set of frequent items in $TDB|_{ef}$, and so on. This is the core of frequent pattern growth method. With such frequent pattern growth, all the frequent patterns for TDB can be discovered without generation of any candidate itemsets. □

Now, let us examine how to push constraints into frequent pattern mining using frequent pattern growth approach. To illustrate the principle, we mine frequent patterns with an anti-monotone constraint in the following example.

Example 3 Suppose an anti-monotone constraint $C_a \equiv sum(S) \leq 180$ is enforced at mining the transaction database in Table 2. Let the values (such as price) of items a, b, c, d, e and f be 50, 150, 10, 200, 20 and 80, respectively, and the support threshold $\xi = 3$.

There are several ways that the constraint C_a can be pushed into the mining process.

1. Removing individual items which cannot satisfy the constraint. For example, item d is a single item whose value is $200 > 180$, so it can be removed from consideration.
2. Not generating a conditional itemset α nor α's conditional database $TDB|_\alpha$ if α does not satisfy the constraint. Since $sum(\{a, b\}) = 200 > 180$ cannot satisfy C_a, any super-pattern containing $\{a, b\}$ will violate the constraint. Thus the conditional itemset $\{a, b\}$ and its associated conditional database should not be generated at all.
3. No constraint checking in the remaining conditional database $TDB|_\alpha$ if $\alpha \cup \beta$ satisfies the constraint, where β is the set of frequent items in $TDB|_\alpha$. Suppose an f-conditional database, $TDB|_f$, contains only 3 frequent (single) items, $\{a, c, e\}$, and $sum(\{a, c, e, f\}) = 160 < 180$. Obviously, any subpattern of $\{a, c, e, f\}$ satisfies the constraint C_a. Thus, there is no need to check constraint C_a in the mining of this conditional database, which avoids some unnecessary constraint checking in the mining process.

An appropriate ordering of items[1] may help early pruning of the conditional databases. For example, if a more expensive item is ordered first, and so on (i.e., b (= 150) first, then f (=80), and so on), it may avoid generation of conditional itemset bf and conditional database $TDB|_{bf}$. It may also create a better chance for the sum of the items in the remaining conditional databases to be less than 180, and thus avoid unnecessary constraint checking. □

This example leads to the following mining strategy for pushing an anti-monotone constraint into the frequent pattern growth.

Strategy 1 (Frequent pattern growth with an anti-monotone constraint) When mining frequent patterns with an anti-monotone constraint C_a in the pattern growth paradigm, the constraint can be pushed into the mining process as follows,

1. Removing the individual items which cannot satisfy the constraint.
2. Not generating a conditional itemset α nor α's conditional database $TDB|_\alpha$ if α does not satisfy the constraint.
3. No constraint checking in the remaining conditional database $TDB|_\alpha$ if $\alpha \cup \beta$ satisfies the constraint, where β is the set of frequent items in $TDB|_\alpha$. □

[1]Notice that in the frequent pattern growth method [3], as stated in Section 3.1, the items are ordered in the item occurrence frequency descending order. This is to facilitate item sharing in order to generate a smaller FP-tree. However, based on our performance study, this ordering improves the performance only minorly. Thus, it is reasonable to explore other possible item ordering to facilitate constrained processing.

Notice that one may find similar pruning rules for strategies 1.1 & 1.2 in Apriori-based constraint mining [5]. However, strategy 1.3 works only with frequent pattern growth method. This is because the saving is related to a particular $TDB|_\alpha$ together with a particular set of frequent items in $TDB|_\alpha$.

Similarly, monotone and succinct constraints can be pushed into the frequent pattern growth mining process. Furthermore, since convertible constraints can be treated as anti-monotone/monotone ones using specific order of items, they can also be pushed into the frequent pattern growth mining process. Strategies for the constraints can be developed respectively. Based on the strategies, we develop an efficient algorithm, CFG, for constraint-based pattern growth mining.

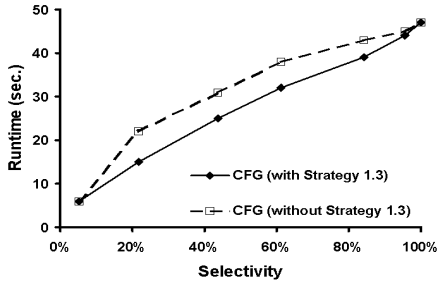

Figure 3: Exploring different strategies with anti-monotone constraints.

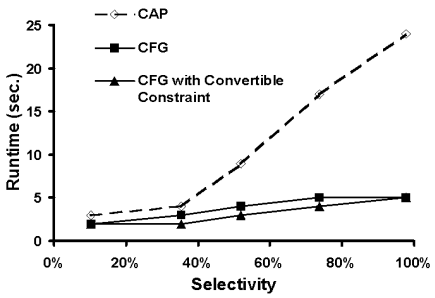

Figure 4: Scalability with constraint selectivity of CAP, CFG and CFG with convertible constraint.

We test CFG and the Apriori-based algorithm CAP [5] on various datasets with various constraints. Limited by space, we only report results here on data set T25I20D100k on a 233MHz Pentium PC with 128M main memory. Figure 3 shows the effect of Strategy 1.3 that cannot be used in Apriori-like methods.

CFG can push convertible constraints deep into the mining process, while CAP can only employ some post-processing to treat convertible constraints. Figure 4 shows that CFG outperforms CAP with a wide margin in mining with convertible constraints.

We believe that the performance improvement is due to two factors: (1) the high performance of the FP-growth mining method which avoids the generation of a large number of candidates, and focuses the subsequent search to projected small databases, and (2) the strategies proposed in this paper achieve good effect in constraint-based mining.

4 Conclusions

We have re-examined the issues on constrained mining of frequent patterns and made the following progress.

First, we have proposed and studied one class of constraints, called *convertible constraints*, which cannot be handled efficiently in previous studies. We show this class covers a good set of useful constraints and can be nicely integrated into the existing framework of constrained frequent pattern mining. Second, we introduced monotone constraint in the framework of frequent pattern mining, and classified constraints into five categories. We show that four categories: *succinct*, *anti-monotone*, *monotone* and *convertible*, can be integrated nicely into the frequent pattern growth mining process. Third, we integrated constraint pushing and frequent pattern growth mining into one unified framework, which leads to further improvement of mining efficiency.

Our study show the integration of two frameworks, *constraint pushing* and *frequent pattern growth*, leads to high performance in data mining.

Acknowledgements. Authors would like to express their thanks to Laks V.S. Lakshmanan for his comments which help improve the quality of the paper.

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, September 1994.

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

[3] 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)*, Dallas, TX, May 2000.

[4] 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.

[5] 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, Washington, June 1998.