Constrained Frequent Pattern Mining: A Pattern-Growth View*

Jian Pei
School of Computing Science
Simon Fraser University
Burnaby, British Columbia, Canada
peijian@cs.sfu.ca

Jiawei Han
Department of Computer Science
Univ. of Illinois at Urbana-Champaign
Urbana, Illinois, U.S.A.
hanj@cs.uiuc.edu

ABSTRACT

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 patterns and rules, which reduces not only the efficiency but also the effectiveness of mining. Recent work has highlighted the importance of the constraint-based mining paradigm in the context of mining frequent itemsets, associations, correlations, sequential patterns, and many other interesting patterns in large databases.

Recently, we developed efficient pattern-growth methods for frequent pattern mining. Interestingly, pattern-growth methods are not only efficient but also effective in mining with various constraints. Many tough constraints which cannot be handled by previous methods can be pushed deep into the pattern-growth mining process. In this paper, we overview the principles of pattern-growth methods for constrained frequent pattern mining and sequential pattern mining. Moreover, we explore the power of pattern-growth methods towards mining with tough constraints and highlight some interesting open problems.

1. INTRODUCTION

It has been well recognized that frequent pattern mining plays an essential role in many important data mining tasks, such as mining association rules [2; 16], correlations [6], causality [31], sequential patterns [3], episodes [20], multi-dimensional patterns [15; 18], max-patterns [4], partial periodicity [11], and emerging patterns [8]. Frequent pattern mining techniques can also be extended to solve many other problems, such as iceberg-cube computation [5] and classification [19]. Thus, effective and efficient frequent pattern mining is an important research problem.

Frequent pattern mining often generates a very large number of frequent patterns 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: User is allowed to express her 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 into mining, thus confining the search space of patterns to those of interest to the user, and achieving superior performance.

Most of the previous studies on frequent pattern mining, such as [2; 9; 16; 18; 21; 22; 29; 30; 32], adopt an Apriori-like approach, which is based on an *anti-monotone* Apriori *property* [2]: *if any length k pattern is not frequent in the database, its length $(k+1)$ super-pattern can never be frequent.* The essential idea is to iteratively generate the set of candidate patterns of length $(k+1)$ from the set of frequent patterns of length k (for $k \geq 1$), and check their corresponding occurrence frequencies in the database. Therefore, an intuitive methodology to push constraints into Apriori-like approaches is to use anti-monotonic constraints to prune candidates. However, many commonly used constraints are not anti-monotonic, like $avg(X) \geq v$, which requires that the average value (price) in pattern X be greater than or equal to v. Thus, the Apriori-like methods meet challenges when mining with such constraints.

Recently, we developed efficient pattern-growth methods [12] for frequent pattern mining [14; 26; 25] and sequential pattern mining [13; 27]. Pattern-growth methods are not Apriori-like. They avoid or dramatically reduce candidate-generation-and-test. Interestingly, our studies [23; 24; 28] show that pattern-growth methods are not only efficient but also effective in constraint-based frequent pattern mining and sequential pattern mining. Many tough constraints that cannot be handled by previous methods, like $avg() \geq v$, can be pushed deep into the pattern-growth mining process.

In this paper, we provide an overview of the principles of the pattern-growth methods for frequent pattern/sequential pattern mining with various constraints. Moreover, we explore the power of pattern-growth methods towards mining with tough constraints and highlight some interesting open problems.

The remaining of this paper is arranged as follows. In Sec-

*The work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, the Networks of Centres of Excellence of Canada, and the Univ. of Illinois at Urbana-Champaign, U.S.A.

tion 2, we recall the problem of constraint-based frequent pattern mining and illustrate some categories of interesting constraints according to their utilization and properties for constraint pushing. In Section 3, we illustrate the ideas of pushing various constraints deep into pattern-growth frequent pattern mining. Pattern-growth sequential pattern mining is exemplified in Section 4. Some implications and extensions of pattern-growth methods for mining various patterns with interesting constraints are discussed in Section 5. This study is concluded in Section 6.

2. WHAT ARE THE INTERESTING CONSTRAINTS?

In this section, we first recall the problem of constrained frequent pattern mining, and then present some categories of interesting constraints.

2.1 Constrained Frequent Pattern Mining

Let $I = \{i_1, \ldots, i_m\}$ be a set of *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 $X \subset I$ is a subset of the set of items. A k-itemset is an itemset of size k. We write itemsets as $S = i_{j_1} \cdots i_{j_k}$, omitting set brackets.

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

EXAMPLE 1. *Let Table 1 be our running transaction database $\mathcal{T}$, with a set of items $I = \{a, b, c, d, e, f, g, h\}$.*

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.

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. □

A *constraint* is a predicate on the powerset of the set of items I, i.e. $C : 2^I \mapsto \{true, false\}$. An itemset X satisfies a constraint C if $C(X) = true$. The set of itemsets satisfying a constraint C is $SAT_C(I) = \{X | X \subseteq I \wedge C(X) = true\}$. We call an itemset in $SAC_C(I)$ *valid*.

Problem definition. Given a transaction database $\mathcal{T}$, a support threshold min_sup, and a constraints C, the problem of *constrained frequent pattern mining* is to find the complete set of frequent itemsets satisfying C, i.e. find $F_C = \{X | X \in SAT_C(I) \wedge sup(X) \geq min\_sup\}$.

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.

2.2 Categories of Constraints

For real-world data miners, it is interesting to examine some interesting constraints from the application point of view. We present the following categories of constraints on the semantics and the forms of the constraints. Although this is by no means complete, it covers most of the interesting constraints in applications.

CONSTRAINT 1 (ITEM CONSTRAINT). *An* item constraint *specifies what are the particular individual or groups of items that should or should not be present in the pattern.* □

For example, a dairy company may be interested in patterns containing only dairy products, when it mines transactions in a grocery store.

CONSTRAINT 2 (LENGTH CONSTRAINT). *A* length constraint *specifies the requirement on the length of the patterns, i.e., the number of items in the patterns.* □

For example, when mining classification rules for documents, a user may be interested in only frequent patterns with at least 5 keywords, a typical length constraint.

CONSTRAINT 3 (MODEL-BASED CONSTRAINT). *A* model-based constraint *looks for patterns which are sub- or super-patterns of some given patterns (models).* □

For example, a travel agent may be interested in what other cities that a visitor is likely to travel if s/he visits both Washington and New York city. That is, they want to find frequent patterns which are super-patterns of {Washington, New York city}.

CONSTRAINT 4 (AGGREGATE CONSTRAINT). *An* aggregate constraint *is on an aggregate of items in a pattern, where the aggregate function can be* SUM, AVG, MAX, MIN, *etc.* □

For example, a marketing analyst may like to find frequent patterns where the average price of all items in each pattern is over $100.

Alternatively, constraints can be categorized according to their properties for constraint pushing. Two categories of constraints, *succinctness* and *anti-monotonicity*, were proposed in [21; 17]; whereas the third category, *monotonicity*, was studied in [6; 9; 23] in the contexts of mining correlated sets and frequent itemsets. We briefly recall these notions below.

DEFINITION 2.1. **(Anti-monotone, Monotone, Succinct, and Convertible 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.* □

A fourth category, *convertibility*, was studied in [23; 24], which covers many tough constraints, like $avg() \geq v$.

DEFINITION 2.2. **(Convertible constraints)** *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 items in both itemsets are listed according to order* $\mathcal{R}$ *and* $(l \leq m)$*.* S' *is called a* proper prefix *of* S *if* $(l < m)$*.*

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.* □

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

EXAMPLE 2. *Let each item in the transaction database in Table 1 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 (e.g., profit) of items in $\mathcal{T}$ in Table 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 [21].*

Constraint $C_{avg} \equiv avg(S) \geq 25$ *is not anti-monotone (nor monotone, nor succinct, which can be verified by readers). For example,* $avg(df) = (10 + 30)/2 < 25$*, violates the constraint. However, upon adding one more item* a*,* $avg(adf) = (40 + 10 + 30)/3 \geq 25$*,* adf *satisfies* C_{avg}*.*

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. For example, itemset* afd *has itemsets* af *and* a *as its prefixes. Interestingly, the average of an itemset is no more than that of its prefix, according to this order. Thus, if a prefix* X *violates the constraint, so does every longer itemset with* X *as prefix. Thus, constraint* C_{avg} *is convertible (anti-monotone).* □

These four categories of constraints cover a large class of popularly encountered constraints. A representative subset of commonly used, SQL-based constraints is listed in Table 4 and 5. (For brevity, we only show a small subset of representative constraints, involving aggregates. See [21; 17] for more details.)

3. CONSTRAINED FREQUENT PATTERN MINING IN PATTERN-GROWTH METHODS

With the above enumerated interesting constraints, one may ask "*how can we push those constraints deep into the frequent pattern mining process?*" In this section, we illustrate the general ideas of constrained frequent pattern mining in pattern-growth methods with some examples.

Although there are many frequent pattern mining algorithms and many interesting constraints, not every constraint can be pushed deep into the mining process of every frequent pattern mining algorithm. A typical such example is that a convertible constraint that is neither monotonic, nor anti-monotonic, nor succinct, cannot be pushed deep into the Apriori mining algorithm.

For example, itemset *df* in our running example violates the constraint $avg(S) \geq 25$. However, an Apriori-like algorithm cannot prune such an itemset. Otherwise, its super-pattern, *adf*, which satisfies the constraint, cannot be generated. In general, sub-patterns or super-patterns of a valid pattern could well be invalid and vice versa. Thus, within the level-wise framework, no direct pruning based on such a constraint can be made.

"*Are there any method that is more powerful at pushing the above categories of constraints deeply into the mining pro-*

Constraint	Anti-monotone	Monotone	Succinct
$min(S) \leq v$	no	yes	yes
$min(S) \geq v$	yes	no	yes
$max(S) \leq v$	yes	no	yes
$max(S) \geq v$	no	yes	yes
$count(S) \leq v$	yes	no	weakly
$count(S) \geq v$	no	yes	weakly
$sum(S) \leq v$ $(\forall a \in S, a \geq 0)$	yes	no	no
$sum(S) \geq v$ $(\forall a \in S, a \geq 0)$	no	yes	no
$sum(S)\ \theta\ v$, $\theta \in \{\leq, \geq\}$ $(\forall a \in S, a\ \theta\ 0)$	no	no	no
$range(S) \leq v$	yes	no	no
$range(S) \geq v$	no	yes	no
$avg(S)\ \theta\ v$, $\theta \in \{\leq, \geq\}$	no	no	no
$sup(S) \geq \xi$	yes	no	no
$sup(S) \leq \xi$	no	yes	no

Table 4: Characterization of commonly used constraints.

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 5: Characterization of some commonly used convertible constraints. ($*$ means it depends on the specific constraint.)

cess?" Our answer is "yes," with a special recommendation of *the pattern-growth method.*

EXAMPLE 3. *Let us mine frequent patterns 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 in Table 5 that constraint C is* convertible anti-monotone *w.r.t. $\mathcal{R}$. The mining process is shown in Figure 1.*

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 the constraint[1]. 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 can be 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 are 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 satisfies the constraint. Moreover, since $afdc$ does not satisfy 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*

[1] 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.

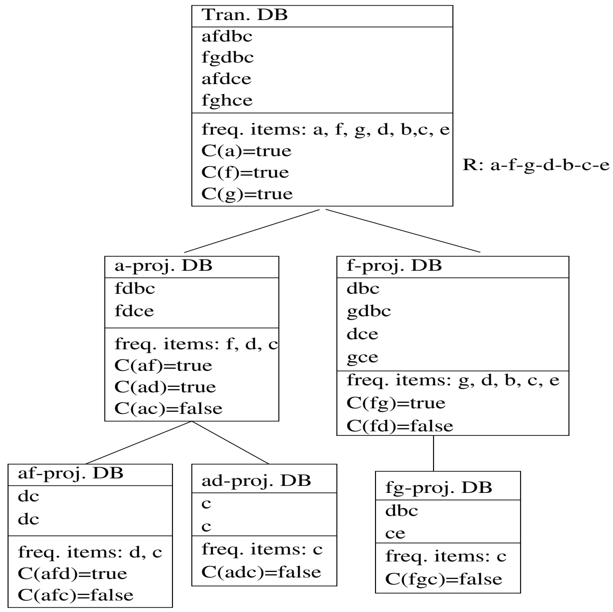

Figure 1: Mining frequent itemsets satisfying constraint $avg(S) \geq 25$.

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 with* fg *or* fd *as a proper prefix that 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 with* 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*. This pattern-growth method generates and tests only a small set of patterns.* □

As shown in the example, a pattern-growth method mines frequent patterns by growing short patterns to long ones. Unlike the Apriori-like methods which proceed level-by-level for candidate-generation-and-test, a pattern growth method found frequent patterns in three steps. First, it finds length-1 frequent patterns as seeds. Then, for each frequent length-1 pattern, it forms a dedicated projected database, which contains only transactions having that pattern. Local frequent items are identified in the projected database and used to grow longer patterns recursively. One distinct advantage of pattern-growth methods is that there is no candidate generation nor candidate test. It has been shown that the pattern-growth method outperforms the Apriori-like methods in mining large databases and databases with many long patterns.

Why can one push some tough constraints deep into the pattern-growth mining process? 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 Apriori-like frequent pattern mining method, because such constraints do not exhibit nice properties like monotonicity, etc. A pattern-growth method can handle such tough constraints by imposing an appropriate order on items and convert them into ones possessing monotonic behavior. By growing frequent patterns in an appropriate order, a pattern-growth method can mine frequent patterns with such constraints effectively.

Interestingly, the ideas for constraint pushing used in pattern-growth methods may also be applied to some recently proposed frequent pattern mining algorithms, such as Depth-First Search [1], MAFIA [7], and CHARM [33]. They adopt a *depth-first manner* to enumerate frequent patterns according to a set enumeration tree, which is essentially dynamic tree-structuring of the search space. Thus, they share with pattern-growth methods the similar idea of growing long patterns from shorter ones. Thus, these methods can also be adapted to handle some tough constraints, like convertible ones.

The efficiency of pattern-growth methods on mining with constraints is that they push the constraints deep into the mining process so that one does not need to generate the complete set of frequent patterns in most cases. Instead, only related frequent patterns are identified and tested. As shown in Example 3, the search space is reduced dramatically when the constraint is sharp.

4. PATTERN-GROWTH SEQUENTIAL PATTERN MINING WITH CONSTRAINTS

In many applications, people want to know not only frequent patterns, but also frequent subsequences as patterns. That leads to the problem of sequential pattern mining [3].

In general, a sequence is an ordered list of transactions. Given a sequence database containing a set of sequences, the problem of sequential pattern mining is to find subsequences that appear frequently in the database.

Sequential pattern mining may return too many patterns from large sequence databases. Thus, it is natural to explore whether constrained frequent pattern mining can be extended to sequential pattern mining. In this section, we exemplify the general idea of pattern-growth sequential pattern mining with constraints.

4.1 Interesting Constraints

For many applications, people may feel interested in some extensions of constraints we have discussed in Section 2.2. That is, people can compose constraints about items appearing in the patterns (i.e. item constraints), the length of the patterns (i.e. length constraints), patterns related to some sequences (i.e. model-based constraints), and aggregates related to patterns (i.e. aggregate constraints). Furthermore, the following three kinds of constraints are also helpful for mining sequences.

CONSTRAINT 5 (REGULAR EXPRESSION CONSTRAINT). A regular expression constraint *is a constraint specified as a regular expression over the set of items using the established set of regular expression operators, such as disjunction and Kleene closure. A sequential pattern satisfies a regular expression constraint if the pattern is accepted by its equivalent deterministic finite automata.* □

For example, to find sequential patterns about a web click stream starting from Yahoo's home page and reaching hotels in New York city, one may use regular expression constraint "*Travel (New York | New York city) (Hotels | Hotels and Motels | Lodging)*", where " | " stands for disjunction. The concept of regular expression constraint was first proposed in [10].

CONSTRAINT 6 (DURATION CONSTRAINT). *A* duration constraint *is defined only in sequence databases where each transaction in every sequence has a time stamp. It requires that the pattern appears frequently in the sequence database such that the time stamp difference between the first and the last transactions in the pattern must be longer or shorter than a given period.* □

For example, when a financial analyst mining for long-term investment patterns, she may set a duration constraint that the first and the last items in the pattern must last at least 1 year.

CONSTRAINT 7 (GAP CONSTRAINT). *A* gap constraint *is also defined only in sequence databases where each transaction in every sequence has a time stamp. It requires that the pattern appears frequently in the sequence database such that the time stamp difference between every two adjacent transactions must be longer or shorter than a given gap.* □

For example, to find the sequential patterns of NBA basketball players, one may only be interested in those on the field regularly, say every week, i.e., the gap being less than 2 weeks.

Among the constraints listed above, duration constraints and gap constraints are *support-related*, i.e., they are applied to confine how a sequence matches a pattern. To find whether a sequential pattern satisfies these constraints, one needs to examine the sequence databases. In other words, they are not succinct. Table 6 shows the anti-monotonic, monotonic and succinct characteristics of some commonly used constraints for sequential pattern mining.

4.2 Pattern-growth Mining Sequential Patterns with Constraints

Sequential patterns can also be mined using level-by-level, candidate-generation-and-test Apriori-like methods. For example, to find a sequential pattern $\langle abc \rangle$, which is an ordered list of three transactions $\{a\}$, $\{b\}$ and $\{c\}$[2], we can first find frequent length-1 patterns $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$. Then, length-2 candidates $\langle aa \rangle$, $\langle ab \rangle$, ..., $\langle cc \rangle$, $\langle (ab) \rangle$[3], $\langle (ac) \rangle$, and $\langle (bc) \rangle$ can be generated and tested. If $\langle ab \rangle$, $\langle ac \rangle$ and $\langle bc \rangle$ are all frequent, length-3 candidate $\langle abc \rangle$ can be generated and tested.

Even Apriori-like sequential pattern mining methods are intuitive extensions, they meet difficulties in handling many constraints. For example, an Apriori-like method cannot push a regular expression constraint $a * c$ deep into its level-by-level candidate-generation-and-test mining procedure. Short patterns like $\langle b \rangle$ and $\langle bc \rangle$ cannot be pruned even they fail the constraint. Otherwise, some longer super-patterns like $\langle abc \rangle$ cannot be assembled later.

One proposal within the Apriori evaluation framework is to use some relaxed constraints having nice properties (like anti-monotonicity) to prune some unpromising patterns and candidates in the early stage. The SPIRIT family algorithms [10] achieve various degrees of constraint enforcement. By doing so, regular expression constraints can be pushed into the Apriori-based mining to some degree. However, there

[2] for brevity, the brackets are omitted if a transaction has only one item.

[3] $\langle ab \rangle$ means a and b are in two different transactions, while $\langle (ab) \rangle$ means both a and b are in the same transaction.

	Constraint	Anti-mono	Mono	Succ
Item	$C_{item}(\alpha) \equiv (\forall i : 1 \leq i \leq len(\alpha),\ \alpha[i]\theta V)\ (\theta \in \{\subseteq, \supseteq\})$	Yes	No	Yes
	$C_{item}(\alpha) \equiv (\forall i : 1 \leq i \leq len(\alpha),\ \alpha[i] \cap V \neq \emptyset)$	Yes	No	Yes
	$C_{item}(\alpha) \equiv (\exists i : 1 \leq i \leq len(\alpha),\ \alpha[i]\theta V)\ (\theta \in \{\subseteq, \supseteq\})$	No	Yes	Yes
	$C_{item}(\alpha) \equiv (\exists i : 1 \leq i \leq len(\alpha),\ \alpha[i] \cap V \neq \emptyset)$	No	Yes	Yes
Length	$len(\alpha) \leq l$	Yes	No	Yes
	$len(\alpha) \geq l$	No	Yes	Yes
Super-pattern	$C_{pat}(\alpha) \equiv (\exists \gamma \in P \text{ s.t. } \gamma \sqsubseteq \alpha)$	No	Yes	Yes
Simple aggregates	$max(\alpha) \leq v),\ min(\alpha) \geq v$	Yes	No	Yes
	$max(\alpha) \geq v),\ min(\alpha) \leq v$	No	Yes	Yes
	$sum(\alpha) \leq v$ (with non-negative values)	Yes	No	No
	$sum(\alpha) \geq v$ (with non-negative values)	No	Yes	No
Tough aggregates	g_sum: $sum(\alpha)\ \theta\ v$, $\theta \in \{\leq, \geq\}$ (with positive and negative values)	No	No	No
	average: $avg(\alpha)\ \theta\ v$	No	No	No
RE	(Regular Expression) ¶	No	No	No
Duration	$Dur(\alpha) \leq \Delta t$	Yes	No	No
	$Dur(\alpha) \geq \Delta t$	No	Yes	No
Gap	$Gap(\alpha)\ \theta\ \Delta t\ (\theta \in \{\leq, \geq\})$	Yes	No	No

Table 6: Characterization of commonly used constraints. (¶ In general, a regular expression (RE) constraint is not necessarily anti-monotonic, monotonic, or succinct, though there are cases that are anti-monotonic, monotonic, or succinct. For example, constraint $*$ is anti-monotonic, monotonic and succinct.)

exist two problems. First, many practical constraints are not covered, like the aggregate constraints. Second, such patches are more ad-hoc: it lacks systematic handling of various kinds of constraints in the same mining process.

"Is it possible to extend the pattern-growth methods to mine sequential patterns with various constraints?" The answer is yes. We illustrate the general idea in the following example.

Example 4. *Let the sequence database SDB be Table 7, and the task be mining sequential patterns with a regular expression constraint $C = \langle a * \{bb|(bc)d|dd\}\rangle$ and support threshold $min\_sup = 2$. The mining can be performed in the following steps.*

Sequence_id	Sequence
10	$\langle a(bc)e\rangle$
20	$\langle e(ab)(bc)dd\rangle$
30	$\langle c(aef)(abc)dd\rangle$
40	$\langle addcb\rangle$

Table 7: Sequence database SDB.

1. Find length-1 patterns and remove irrelevant sequences. *Scanning SDB once, patterns $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle d\rangle$, and $\langle e\rangle$ are identified as length-1 patterns. Infrequent items, such as f, is removed. Also, in the same scan, the sequences that contain no subsequence satisfying the constraint, such as the first sequence, $\langle a(bc)e\rangle$, should be removed.*

2. Divide the set of sequential patterns into subsets without overlap. *The complete set of sequential patterns can be divided into five subsets without overlap according to the set of length-1 sequential patterns: (1) those with prefix $\langle a\rangle$; (2) those with prefix $\langle b\rangle$; ...; and (5) those with prefix $\langle e\rangle$. Since only patterns with prefix $\langle a\rangle$ may satisfy the constraint, i.e. only $\langle a\rangle$ is legal w.r.t. constraint C, the other four subsets of patterns can be pruned.*

3. Construct $\langle a\rangle$-projected database and mine it. *Only the sequences in SDB containing item a and satisfying constraint C should be projected. The $\langle a\rangle$-projected database, $SDB|_{\langle a\rangle} = \{\langle(\_b)(bc)dd\rangle, \langle(\_e)(abc)dd\rangle, \langle ddcb\rangle\}$. Notice that $\langle e(ab)(bc)dd\rangle$ is projected as $\langle(\_b)(bc)dd\rangle$, where symbol "_" in the first transaction indicates that it is in the same transaction with a.*

   During the construction of the $\langle a\rangle$-projected database, we also find locally frequent items: (1) b can be inserted into the same transaction with a to form a longer frequent prefix $\langle(ab)\rangle$, and (2) $\langle b\rangle$, $\langle c\rangle$ and $\langle d\rangle$ can be concatenated to $\langle a\rangle$ to form longer frequent prefixes, i.e., $\langle ab\rangle$, $\langle ac\rangle$ and $\langle ad\rangle$. Locally infrequent items, such as e, should be ignored in the remaining mining of this projected database.

   Then the set of patterns with prefix $\langle a\rangle$ can be further divided into five subsets without overlap: (1) pattern $\langle a\rangle$ itself; (2) those with prefix $\langle(ab)\rangle$; (3) those with prefix $\langle ab\rangle$; (4) those with prefix $\langle ac\rangle$; and (5) those with prefix $\langle ad\rangle$. By examining constraint C, one can see that pattern $\langle a\rangle$ fails C and thus is discarded; and $\langle(ab)\rangle$ is illegal w.r.t. constraint C, so the second subset of patterns is pruned. The remaining subsets of patterns should be explored one by one.

4. Recursive mining. *The mining proceeds recursively. In the mining, if the prefix itself is a pattern satisfying the constraint, it should be an output. The prefixes legal w.r.t. the constraint should be grown and mined recursively. The process terminates when there is no local frequent item or there is no legal prefix. It results in two final patterns: $\{\langle a(bc)d\rangle, \langle add\rangle\}$.* □

As shown in the example, a pattern-growth method mines sequential patterns with constraints in the following steps. First, length-1 sequential patterns are found. Then, a sequential pattern is used to grow longer pattern only if it is a prefix of some patterns potential for the constraint. Only sequences in the database satisfying the constraint and supporting the current pattern as well as its potential extensions are collected and projected into the projected database. Then, recursive mining grows longer patterns.

"*Why can tough constraints be pushed deep into a pattern growth-based sequential pattern mining process?*" If a pattern satisfies a constraint, its prefix must partially satisfy the constraint. A pattern-growth method takes advantage of this property and grows longer patterns from shorter ones that partially satisfy the given constraints. Any prefix, if not promising, should be pruned immediately.

It can be shown that one can push many tough constraints, including those involving aggregates like $avg()$, deep into the pattern-growth mining process. Thus, it is an effective and efficient methodology.

5. DISCUSSION

The above sections show that constraints can be classified into a few categories, such as *monotonic, anti-monotonic, succinct, convertible*, and *inconvertible*, based on their interactions with a frequent pattern mining process. Many commonly used constraints have nice properties, such as monotonic, anti-monotonic, and succinct, and they can be pushed deep into the mining process, no matter which mining approach is adopted: Apriori-*based* or *pattern-growth-based*. However, at handling some tough classes of constraints, such as convertible constraints, and regular expression constraints in the case of sequential pattern mining, there is a sharp distinction between the two mining approaches.

In this section, we discuss (1) why there is such a sharp distinction between the two approaches, and (2) whether the pattern-growth method can be extended to mining more complex types of structured patterns.

As shown in the previous two sections, the pattern-growth-based approach is more powerful than the Apriori-based one at pushing tough constraints deep into the mining process. This can be explained as follows. The Apriori-based method grows its frequent patterns from length k to length $(k+1)$ relying on its downward closure property: *a pattern is frequent only if every of its subpatterns is frequent*. This is perfectly sound for mining frequent patterns without constraints. However, for constraint pushing, the testing of the downward closure property (which is the essence of the Apriori approach) implies that it follows a revised statement: *a pattern is a candidate only if every of its subpatterns (1) is frequent and (2)* satisfies the constraint. Unfortunately, this statement is not always true for all kinds of constraints (e.g., convertible ones) since not every candidate that satisfies a constraint must have all of its subpatterns satisfy the same constraint. For example, for a constraint $avg(S) \geq 25$, generating a candidate pattern "$S = afd$" requires the testing of constraint satisfaction for every of its subpatterns: "af, ad, fd", however, even when "$avg(afd) \geq 25$", there is still a chance that "$avg(fd) < 25$". Therefore, this constraint cannot be pushed deep into Apriori-based mining. On the other hand, the pattern-growth approach can grow a pattern in a desired direction, without checking all the combinations of its subpatterns. Thus it is possible to allow a convertible constraint to be pushed deep if the pattern grows in a controlled manner, such as in value descending order as shown in Example 3.

With the similar reasoning, one can show the strength of pattern-growth approach in constraint-based sequential pattern mining, especially at handling regular expression constraints, as illustrated in Section 4, and aggregate constraints.

Constraint-based mining with the pattern-growth approach can be extended to structured pattern mining as well. For example, constraints associated with structured patterns, such as trees, lattices and graphs, can be categorized similarly as the cases of mining frequent and sequential patterns. By extension of the pattern-growth approach towards mining such patterns, it is obvious that the constraint-based pattern mining framework developed here can be extended accordingly to cover such patterns. The detailed extension will be left to interested readers as an exercise.

6. CONCLUSIONS

Due to the fact that a large number of patterns or rules are often found in frequent pattern mining, it is highly desirable for a user to pose constraints as *mining queries* to make the mining focused on the desired data sets and patterns, and therefore improve the mining *effectiveness*. As a data mining system implementor, especially a mining query optimizer implementor, the critical issue at handling user- or expert-specified constraints is how to push such constraints deep into the mining process to improve the *efficiency* of mining.

In this paper, we have examined the methods for pushing constraints deep into the frequent and sequential pattern mining processes. Our focus is at introducing a pattern-growth method and showing its strength at constraint pushing over the popular Apriori-based constraint-based mining approach.

There are still many interesting research issues to be examined. For example, constraint-based mining of max-patterns and closed patterns, in the context of mining frequent, sequential and structured patterns, and constraint-based classification, clustering, outlier analysis in large data sets are interesting topics for future research.

Acknowledgements

The authors would like to express their thanks to Roberto Bayardo for his constructive comments on the initial version of the paper.

7. REFERENCES

[1] R. Agarwal, C. Aggarwal, and V. V. V. Prasad. Depth-first generation of large itemsets for association rules. In *IBM Technical Report RC21538*, July 1999.

[2] 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.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, Mar. 1995.

[4] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 85–93, Seattle, WA, June 1998.

[5] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 359–370, Philadelphia, PA, June 1999.

[6] 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.

[7] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pages 443–452, Heidelberg, Germany, April 2001.

[8] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, pages 43–52, San Diego, CA, Aug. 1999.

[9] 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.

[10] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, pages 512–521, Sydney, Australia, Mar. 1999.

[11] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, pages 106–115, Sydney, Australia, April 1999.

[12] J. Han and J. Pei. Mining frequent patterns by pattern-growth: Methodology and implications. *SIGKDD Explorations (Special Issue on Scalable Data Mining Algorithms)*, 2:14–20, 2000.

[13] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. FreeSpan: Frequent pattern-projected sequential pattern mining. In *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'00)*, pages 355–359, Boston, MA, Aug. 2000.

[14] 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.

[15] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, Newport Beach, CA, Aug. 1997.

[16] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int. Conf. Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov. 1994.

[17] 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.

[18] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proc. 1997 Int. Conf. Data Engineering (ICDE'97)*, pages 220–231, Birmingham, England, April 1997.

[19] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86, New York, NY, Aug. 1998.

[20] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[21] 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.

[22] J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'95)*, pages 175–186, San Jose, CA, May 1995.

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

[24] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pages 433–332, Heidelberg, Germany, April 2001.

[25] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. H-Mine: Hyper-structure mining of frequent patterns in large databases. In *Proc. 2001 Int. Conf. Data Mining (ICDM'01)*, pages 441–448, San Jose, CA, Nov. 2001.

[26] 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.

[27] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pages 215–224, Heidelberg, Germany, April 2001.

[28] J. Pei, J. Han, and W. Wang. Constraint-based sequential pattern mining in large databases. In *Submitted for publication*, May 2002.

[29] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 343–354, Seattle, WA, June 1998.

[30] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pages 432–443, Zurich, Switzerland, Sept. 1995.

[31] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 594–605, New York, NY, Aug. 1998.

[32] 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.

[33] M. J. Zaki and C. J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *Proc. 2002 SIAM Int. Conf. Data Mining*, pages 457–473, Arlington, VA, April 2002.