J Intell Inf Syst
DOI 10.1007/s10844-006-0006-z

# Constraint-based sequential pattern mining: the pattern-growth methods

**Jian Pei · Jiawei Han · Wei Wang**

Received: 31 January 2003 / Revised: 23 March 2005 /
Accepted: 28 June 2005


**Abstract** Constraints are essential for many sequential pattern mining applications. However, there is no systematic study on *constraint-based sequential pattern mining*. In this paper, we investigate this issue and point out that the framework developed for constrained frequent-pattern mining does not fit our mission well. An extended framework is developed based on a sequential pattern growth methodology. Our study shows that constraints can be effectively and efficiently pushed deep into the sequential pattern mining under this new framework. Moreover, this framework can be extended to constraint-based structured pattern mining as well.

**Keywords** Sequential pattern mining · Frequent pattern mining · Mining with constraints · Pattern-growth methods

## 1 Introduction

Sequential pattern mining (Agrawal & Srikant, 1995) is an important data mining task with broad applications. In the last 10 years, there have been many studies

This research is supported in part by NSERC Grant 312194-05, NSF Grants IIS-0308001, IIS-0513678, BDI-0515813 and National Science Foundation of China (NSFC) grants No. 60303008 and 69933010. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

J. Pei (✉)
School of Computing Science, Simon Fraser University, Canada
e-mail: jpei@cs.sfu.ca

J. Han
University of Illinois at Urbana-Champaign, USA
e-mail: hanj@cs.uiuc.edu

W. Wang
Fudan University, China
e-mail: weiwang1@fudan.edu.cn

on efficient sequential pattern mining and its applications (e.g., Agrawal & Srikant, 1995; Ayres, Flannick, Gehrke, & Yiu, 2002; Chiu, Wu, & Chen, 2004; Kum, Pei, & Wang, 2003; Mannila, Toivonen, & Verkamo, 1997; Pei, Han, Mortazavi-Asl, Pinto, Chen, Dayal, & Hsu, 2001; Pei, Han, & Wang, 2002; Pinto, Han, Pei, Wang, Chen, & Dayal, 2001; Srikant & Agrawal, 1996; Tzvetkov, Yan, & Han, 2003; Wang & Tan, 1996; Yan, Han, & Afshar, 2003; Yang, Yu, Wang, & Han, 2002; Zaki, 2001).

Sequential pattern mining algorithms, in general, can be categorized into three classes:

- Apriori-based, horizontal formatting method, with GSP Srikant and Agrawal (1996) as its representative;
- Apriori-based, vertical formatting method, such as SPADE (Zaki, 2001); and
- Projection-based pattern growth method, such as PSP (Pei et al., 2001) and SPAM (Ayres et al., 2002).

Although efficiency of mining the complete set of sequential patterns has been improved substantially, in many cases, sequential pattern mining still faces tough challenges in both effectiveness and efficiency. On the one hand, there could be a large number of sequential patterns in a large database. A user is often interested in only a small subset of such patterns. Presenting the complete set of sequential patterns may make the mining result hard to understand and hard to use. This brings in the effectiveness concern: "Can we only mine the sequential patterns that are highly interesting to users?"

On the other hand, although efficient algorithms have been proposed, mining a large amount of sequential patterns from large data sequence databases is inherently a computationally expensive task. If we can focus on only those sequential patterns interesting to users, we may be able to save a lot of computation cost by those uninteresting patterns. This opens a new opportunity for performance improvement: "Can we improve the efficiency of sequential pattern mining by focusing only on interesting patterns?"

For effectiveness and efficiency considerations, constraints are essential in many data mining applications. In the context of constraint-based sequential pattern mining, Srikant and Agrawal (1996) generalized the scope of sequential pattern mining (Agrawal & Srikant, 1995) to include time constraints, sliding time windows, and user-defined taxonomy. Mining frequent episodes in a sequence of events studied by Mannila et al. (1997) can also be viewed as a constrained mining problem, since episodes are essentially constraints on events in the form of acyclic graphs. Garofalakis, Rastogi, and Shim (1999) proposed regular expressions as constraints for sequential pattern mining and developed a family of SPIRIT algorithms, while members in the family achieve various degrees of constraint enforcement. The algorithms use relaxed constraints with nice properties (like anti-monotonicity) to filter out some unpromising patterns/candidates in their early stage.

The above interesting studies handle a few scattered classes of constraints. However, two problems remain. First, many practical constraints have not been covered. Moreover, there lacks a systematic method to push various constraints into the mining process. In other words, it is still unclear what is the general picture of constraint-based sequential pattern mining and how to handle constraints beyond the

studied classes. This forms a sharp contrast with constraint-based frequent pattern mining, where systematic studies have been performed, and constraints have been classified into a few classes, and efficient constraint-based mining methods have been developed for each class (Bayardo, Agrawal, & Gunopulos, 1999; Grahne, Lakshmanan, & Wang, 2000; Kifer, Gehrke, Bucila, & White, 2003; Ng, Lakshmanan, Han, & Pang, 1998; Pei, Han, & Lakshmanan, 2001).

To elaborate on the above observation and also motivate this study, let us consider the following example. To characterize a new disease, researchers may want to find sequential patterns about symptoms, such as "finding patterns with constraint of 2–7 days of cough followed by fever in the range of 37.5–39C for 2–5 days with average temperature of $38 \pm 0.2$C, and all these symptoms appear within 2 weeks." A pattern found could be "cough 5 days and fever 4 days with strong headache." This mining query contains a few constraints, involving sequences containing certain constants, and with average functions, etc. None of the previously developed constraint-based sequential pattern mining methods can handle all these constraints. Moreover, it is unclear how to incorporate all constraints in the mining process.

In this paper, we conduct a systematic study on constraint-based sequential pattern mining, and make the following contributions.

- First, various kinds of constraints are classified in two orthogonal ways, based on their application semantics and their roles in sequential pattern mining, respectively. The latter scheme largely follows the conventional constraint-based frequent pattern mining framework.
- Unfortunately, some commonly encountered sequence-based constraints, such as regular expression constraints, are neither monotonic, nor anti-monotonic, nor succinct. Instead of patching the classical framework, a new framework, called PG, is built based on a *prefix-monotone* property. Interestingly, all the monotonic and anti-monotonic constraints, as well as regular expression constraints, are *prefix-monotone*, and can be pushed deep into a PG-based mining algorithm. Moreover, some tough aggregate constraints, such as those involving average or general sum, can also be pushed deep into a slightly revised PG mining process.
- A performance study is conducted which demonstrates that constraint-based mining prunes a large search space effectively in sequential pattern mining, and PG is more efficient than other constraint-based sequential pattern mining algorithms studied before.

The remainder of the paper is organized as follows. In Section 2, sequential pattern mining is revisited. In Section 3, a set of frequently encountered constraints are examined. In Section 4, we examine whether extending the classical constraint-based frequent pattern mining framework can fit most of the popular constraints. To handle a popular class of constraints not fitting into the classical framework, a new framework, PG, is constructed in Section 5, by introducing a *prefix-monotone* property of constraints and developing an efficient constraint-pushing method PG. The framework is further enriched in Section 6 by handling some tough aggregate constraints and pushing them deep into a slightly revised PG mining process. Results on extensive experiments and performance study are reported and analyzed in Section 7. Extension of the PG framework are discussed in Section 8, and our study is concluded in Section 9.

## 2 Sequential pattern mining: concepts

Let $I = \{x_1, \ldots, x_n\}$ be a set of *items*, each possibly being associated with a set of *attributes*, such as value, price, profit, calling distance, period, etc. The value on attribute $A$ of item $x$ is denoted by $x.A$. An *itemset* is a non-empty subset of items, and an itemset with $k$ items is called a *k-itemset*.

A *sequence* $\alpha = \langle X_1 \cdots X_l \rangle$ is an ordered list of itemsets. An itemset $X_i$ ($1 \leq i \leq l$) in a sequence is called a *transaction*, a term originated from analyzing customers' shopping sequences in a transaction database, such as in (Agrawal, Imielinski, & Swami, 1993; Agrawal & Srikant, 1994, 1995; Srikant & Agrawal, 1996). A transaction $X_i$ may have a special attribute, *times-tamp*, denoted by $X_i.time$, which registers the time when the transaction was executed. For a sequence $\alpha = \langle X_1 \cdots X_l \rangle$, we assume $X_i.time < X_j.time$ for $1 \leq i < j \leq l$.

The number of transactions in a sequence is called the *length* of the sequence. A sequence with length $l$ is called an *l-sequence*. For an $l$-sequence $\alpha$, we have $len(\alpha) = l$. Furthermore, the $i$-th itemset is denoted by $\alpha[i]$. An item can occur at most once in an itemset, but can occur multiple times in various itemsets in a sequence.

A sequence $\alpha = \langle X_1 \ldots X_n \rangle$ is called a *subsequence* of another sequence $\beta = \langle Y_1 \ldots Y_m \rangle$ ($n \leq m$), and $\beta$ a *super-sequence* of $\alpha$, denoted by $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq i_1 < \ldots < i_n \leq m$ such that $X_1 \subseteq Y_{i_1}, \ldots, X_n \subseteq Y_{i_n}$.

A *sequence database* $SDB$ is a set of 2-tuples $(sid, \alpha)$, where *sid* is a *sequence-id* and $\alpha$ a sequence. A tuple $(sid, \alpha)$ in a sequence database $SDB$ is said to *contain* a sequence $\gamma$ if $\gamma$ is a subsequence of $\alpha$. The number of tuples in a sequence database $SDB$ containing sequence $\gamma$ is called the *support* of $\gamma$, denoted by $sup(\gamma)$.

Given a positive integer *min_sup* as the *support threshold*, a sequence $\gamma$ is a *sequential pattern* in sequence database $SDB$ if $sup(\gamma) \geq min\_sup$. The *sequential pattern mining* problem is to find the *complete* set of sequential patterns with respect to a given sequence database $SDB$ and a support threshold *min_sup*.

*Example 1* (Sequential patterns) Table 1 shows a sequence database $SDB$ with four sequences. The first sequence contains three transactions (itemsets): $\{a\}$, $\{b, c\}$ and $\{e\}$. Its length is three. For the sake of brevity, the brackets are omitted if a transaction has only one item.

As can be seen, an item can occur multiple times in various itemsets in a sequence. For example, item $b$ appears twice in sequence 20. Moreover, a sequence can even contain identical transactions, such as transaction $\{d\}$ in sequences 20, 30 and 40. However, there is not any general assumption about the repetition of items or transactions.

**Table 1** Sequence database $SDB$

| Sequence_id | Sequence |
|---|---|
| 10 | ⟨a(bc)e⟩ |
| 20 | ⟨e(ab)(bc)dd⟩ |
| 30 | ⟨c(aef)(abc)dd⟩ |
| 40 | ⟨addcb⟩ |

Sequence $\langle (ab)d \rangle$ is a subsequence of both the second sequence, $\langle e(ab)(bc)dd \rangle$, and the third one, $\langle c(aef)(abc)dd \rangle$. Thus, if the support threshold $min\_sup = 2$, $\langle (ab)d \rangle$ is a sequential pattern.

Parallel to the case of frequent itemset mining problem (Agrawal et al., 1993), there are two major difficulties in sequential pattern mining: (1) *effectiveness*: the mining may return a huge number of patterns, many of which could be uninteresting to users, and (2) *efficiency*: it often takes substantial computational time and space for mining the complete set of sequential patterns in a large sequence database.

Constraint-based mining may overcome both difficulties since constraints usually represent user's interest and focus, which confines the patterns to be found to a particular subset satisfying some strong conditions. Moreover, if constraints can be pushed deep into the mining process, it is likely to achieve efficiency since the search can be more focused. This motivates the study of constraint-based mining of sequential patterns.

## 3 Categories of constraints

A *constraint* $C$ for sequential pattern mining is a boolean function $C(\alpha)$ on the set of all sequences. The *problem of constraint-based sequential pattern mining* is to find the complete set of sequential patterns satisfying a given constraint $C$.

Constraints can be examined and characterized from different points of views. We examine them first from the application point of view in this section and then from the constraint-pushing point of view in the next section, and build up linkages between the two by a thorough study of constraint-based sequence mining.

From the application point of view, we present the following seven categories of constraints based on the semantics and the forms of the constraints. Although this is by no means complete, it covers many interesting constraints in applications.

*Constraint 1* (Item constraint) An *item constraint* specifies subset of items that should or should not be present in the patterns. It is in the form of

$$C_{item}(\alpha) \equiv (\varphi i : 1 \leq i \leq len(\alpha),\ \alpha[i]\ \theta\ V),$$

or

$$C_{item}(\alpha) \equiv (\varphi i : 1 \leq i \leq len(\alpha),\ \alpha[i] \cap V \neq \emptyset),$$

where $V$ is a subset of items, $\varphi \in \{\forall, \exists\}$ and $\theta \in \{\subseteq, \not\subseteq, \supseteq, \not\supseteq, \in, \notin\}$. For the sake of brevity, we omit the strict operators (e.g., $\subset$, $\supset$) in our discussion here. However, the same principles can be applied to them.

For example, when mining sequential patterns over a web log, a user may be interested in only patterns about visits to online bookstores. Let $B$ be the set of online bookstores. The corresponding item constraint is $C_{bookstore}\ (\alpha) \equiv (\forall i : 1 \leq i \leq len(\alpha), \alpha[i] \subseteq B)$.

*Constraint 2* (Length constraint) A *length constraint* specifies the requirement on the length of the patterns, where the length can be either the number of occurrences

of items or the number of transactions. Length constraints can also be specified as the number of distinct items, or even the maximal number of items per transactions.

For example, a user may want to find only long patterns (e.g., patterns consisting of at least 50 transactions) in bio-sequence analysis. Such a requirement can be expressed by a length constraint $C_{len}(\alpha) \equiv (len(\alpha) \geq 50)$.

*Constraint 3* (Super-pattern constraint) A *super-pattern constraint* is in the form of

$$C_{pat}(\alpha) \equiv (\exists \gamma \in P \, s.t. \, \gamma \sqsubseteq \alpha),$$

where $P$ is a given set of patterns, i.e., to find patterns that contain a particular set of patterns as sub-patterns.

For example, an analyst might want to find sequential patterns that first buy a PC and then buy a digital camera. The constraint can be expressed as

$$C_{\text{pat}}(\alpha) \equiv \langle (PC)(digital\_camera) \rangle \sqsubseteq \alpha.$$

*Constraint 4* (Aggregate constraint) An *aggregate constraint* is the constraint on an aggregate of items in a pattern, where the aggregate function can be *sum*, *avg*, *max*, *min*, standard deviation, etc.

For example, a marketing analyst may want sequential patterns where the average price of all the items in each pattern is over $100.

*Constraint 5* (Regular expression constraint) A *regular expression constraint* $C_{RE}$ 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 $C_{RE}$ if and only 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 for sequential pattern mining was first proposed in Garofalakis et al. (1999).

In some applications, one may want to have constraints on the duration of the patterns, i.e., events happening within a certain duration.

*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 sequential patterns in the sequence database must have the property such that the time-stamp difference between the first and the last transactions in a sequential pattern must be longer or shorter than a given period.

Formally, a duration constraint is in the form of

$$C_{\text{dur}} \equiv \text{Dur}(\alpha) \; \theta \; \Delta t,$$

where $\theta \in \{\leq, \geq\}$ and $\Delta t$ is a given integer. A sequence $\alpha$ satisfies the constraint if and only if $|\{\beta \in SDB | \exists 1 \leq i_1 < \cdots < i_{\text{len}(\alpha)} \leq \text{len}(\beta)$ s.t. $\alpha[1] \sqsubseteq \beta[i_1], \ldots, \alpha[\text{len}(\alpha)] \sqsubseteq \beta[i_{\text{len}(\alpha)}]$, and $(\beta[i_{\text{len}(\alpha)}].\text{time} - \beta[i_1].\text{time}) \ \theta \ \Delta t\}| \geq \text{min\_sup}$.

In some other applications, the gap between adjacent transactions in a pattern may be important.

*Constraint 7* (Gap constraint) A *gap constraint* set is defined only in sequence databases where each transaction in every sequence has a timestamp. It requires that the sequential patterns in the sequence database must have the property such that the timestamp difference between every two adjacent transactions must be longer or shorter than a given gap.

Formally, a gap constraint is in the form of

$$C_{\text{gap}} \equiv \text{Gap}(\alpha) \ \theta \ \Delta t,$$

where $\theta \in \{\leq, \geq\}$ and $\Delta t$ is a given integer. A sequence $\alpha$ satisfies the constraint if and only if $|\{\beta \in SDB | \exists 1 \leq i_1 < \cdots < i_{\text{len}(\alpha)} \leq \text{len}(\beta)$ s.t. $\alpha[1] \sqsubseteq \beta[i_1], \ldots, \alpha[\text{len}(\alpha)] \sqsubseteq \beta[i_{\text{len}(\alpha)}]$, and for all $1 < j \leq \text{len}(\alpha)$, $(\beta[i_j].\text{time} - \beta[i_{j-1}].\text{time}) \ \theta \ \Delta t\}| \geq \text{min\_sup}$.

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. For other constraints, whether the constraint is satisfied can be determined by the frequent patterns themselves without referring to the support counting process.

## 4 Characterization of constraints: a classical framework

In the recent studies of constrained frequent pattern mining (Ayres et al., 2002; Ng et al., 1998; Pei & Han, 2000; Pei et al., 2001), constraints are characterized based on the notion of monotonicity, anti-monotonicity, succinctness, and whether they can be transformed into these categories if they do not belong to them. This has become a classical framework for constraint-based frequent pattern mining. "Can we extend this framework and solve the constraint-based sequential pattern mining problem?"

A constraint $C_A$ is *anti-monotonic* if a sequence $\alpha$ satisfying $C_A$ implies that every non-empty subsequence of $\alpha$ also satisfies $C_A$. A constraint $C_M$ is *monotonic* if a sequence $\alpha$ satisfies $C_M$ implies that every super-sequence of $\alpha$ also satisfies $C_M$. The basic idea behind *succinct constraint* is that, with such a constraint, one can explicitly and precisely generate all the sets of items satisfying the constraint without recourse to a generate-everything-and-test approach. A succinct constraints is specified using a precise "formula". According to the "formula", one can generate all the patterns satisfying a succinct constraint. There is no need to iteratively check the constraint in the mining process. Limited by space, we omit the formal definitions here.

For example, length constraint $C_{\text{len}}(\alpha) \equiv \text{len}(\alpha) \leq 10$ and duration constraint $\text{Dur}(\alpha) \leq 30$ are anti-monotonic, while super-pattern constraint and the duration

constraint $\text{Dur}(\alpha) \geq 30$ are monotonic. It is easy to show that item constraints, length constraints and super-pattern constraints are all succinct.

Based on the above definition, the anti-monotonic, monotonic and succinct characteristics of some commonly used constraints for sequential pattern mining are shown in Table 2.

We have the following result.

**Theorem 1** *The monotonicity, anti-monotonicity and succinctness asserted by Table 2 are correct.*

*Proof sketch* The proof can be constructed by examining the constraints against the definitions of anti-monotonicity, monotonicity and succinctness. We only show one case here as an example, $C \equiv \text{len}(\alpha) \leq l$. For a sequential pattern $\alpha$, let $\alpha' \sqsubseteq \alpha$ be a non-empty subsequence of $\alpha$. Clearly, we have $\text{len}(\alpha') \leq \text{len}(\alpha)$. If $\alpha$ satisfies the constraint, i.e., $\text{len}(\alpha) \leq l$, then $\alpha'$ must also satisfy the constraint. Thus, the constraint is anti-monotonic, as shown in Table 2. □

From Table 2, one can see that the classical constraint-pushing framework (Ng et al., 1998) based on anti-monotonicity, monotonicity, and succinctness can be applied to many constraints. Thus the corresponding constraint-pushing strategy

**Table 2** Characterization of commonly used constraints

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

(a 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 "$*$" (every pattern satisfies this constraint) is anti-monotonic, monotonic and succinct.)

can be integrated easily into any sequential pattern mining algorithms, such as GSP, SPADE, and PSP. However, some important classes of constraints, such as RE (regular expressions), average(i.e., $avg(\alpha)\theta v$, where $\theta \in \leq, \geq$), and *g_sum* (i,e., *sum* of positive and negative values), do not fit into this framework.

This problem, with respect to commonly used regular expression constraints, has been pointed out by Garofalakis et al. (1999). They provided a solution of a set of four SPIRIT algorithms, each pushing a stronger relaxation of regular expression constraint $\mathcal{R}$ than its predecessor in the pattern mining loop. The first SPIRIT algorithm SPIRIT(N) ("*N*" for "Naive") only prunes candidate sequences containing elements that do not appear in $\mathcal{R}$. The second one, SPIRIT(L) ("*L*" for "Legal"), requires every candidate sequence to be *legal* with respect to some state of $A_{\mathcal{R}}$. The third, SPIRIT(V) ("*V*" for "Valid"), filters out candidate sequences that are not *valid* with respect to any state of $A_{\mathcal{R}}$. The fourth, SPIRIT(R) ("*R*" for "Regular"), pushes $\mathcal{R}$ all the way inside the mining process by counting support only for valid candidate sequences. SPIRIT(R) looks most promising in constraint pushing, however, when the RE constraint is not highly selective, the experiments in (Garofalakis et al., 1999) show that the number of candidates generated by SPIRIT(R) explodes and the algorithm fails to even complete execution for certain cases (run out of virtual memory). Thus finally, SPIRIT(V) was recommended as the overall winner.

The above method, though interesting, might not be applicable to some other types of constraints. Can we handle those difficult-to-push constraints in a nice and elegant way? This is the theme of the next section.

## 5 Mining sequential patterns with prefix-monotone constraints

The classical framework on frequent and sequential pattern mining is based on the anti-monotonic Apriori property of frequent patterns: "any super-pattern of an infrequent pattern cannot be frequent." A breadth-first, level-by-level search can be conducted to find the complete set of patterns.

However, as shown in Table 2, some important and popularly used constraints do not have the anti-monotonic or monotonic property. Instead of considering such constraints as tough ones and find different kinds of relaxation or patching tricks to "squeeze" them into the Apriori framework, as done by SPIRIT, we explore an intuitive way by changing the sequential pattern mining framework so that such *ugly* constraints can be naturally pushed deeply into the mining process.

Among the three typical sequential pattern mining methods, PSP seems to have the best hope to accomplish this task because sequential patterns are mined with this method by prefix-based database projection without candidate generation. A tough constraint, such as a regular expression constraint, matches naturally with prefix-based expansion and can be pushed deep into the subsequence expansion-based mining process. Moreover, the classical anti-monotonic, monotonic, and succinct constraints can be easily adapted to this evaluation framework as well.

In this section, we first present a *prefix-monotone property* for constraints and show that most of the constraints discussed so far are prefix-monotone. Then, we develop an efficient mining algorithm to push such constraints into sequential pattern mining.

### 5.1 Prefix-monotone property

Let $R$ be an order of items in a sequence database. Since the item ordering in the same transaction is irrelevant to sequential patterns, it is convenient to assume that all items in a transaction are written with respect to the order $R$. For example, let $R$ be the alphabetical order. A sequence should be written in the form of $\langle(ade)(bc)\rangle$ instead of $\langle(dae)(cb)\rangle$. The fact that item $x$ precedes item $y$ in order $R$ is denoted by $x \prec y$.

Given a sequence $\alpha = \langle X_1 \cdots X_n\rangle$, sequence $\beta = \langle X_1 \cdots X_k Y\rangle$ is called a *prefix* of $\alpha$ if (1) $k < n$, (2) $Y \subseteq X_{k+1}$, and (3) $\forall y \in Y, \forall z \in (X_{k+1} - Y)$, $y \prec z$. For example, sequence $\beta = \langle(abc)(ac)\rangle$ is a prefix of sequence $\alpha = \langle(abc)(acd)(bef)\rangle$ but sequence $\gamma = \langle(abc)(ad)\rangle$ is not a prefix of $\alpha$. Here, the alphabetical order is used.

A constraint $C_{\text{pa}}$ is called *prefix anti-monotonic* if for each sequence $\alpha$ satisfying the constraint, so does every prefix of $\alpha$. A constraint $C_{\text{pm}}$ is called *prefix monotonic* if for each sequence $\alpha$ satisfying the constraint, so does every sequence having $\alpha$ as a prefix. A constraint is called *prefix-monotone* if it is prefix anti-monotonic or prefix monotonic.

We immediately have the following result.

**Lemma 1** An anti-monotonic constraint is prefix anti-monotonic. A monotonic constraint is prefix monotonic.

*Proof* Let $C_a$ and $C_m$ be an anti-monotonic constraint and a monotonic constraint, respectively. Suppose sequence $\alpha$ satisfies $C_a$. Consider a prefix $\beta$ of $\alpha$. Clearly, $\beta$ is a subsequence of $\alpha$, i.e., $\beta \sqsubseteq \alpha$. Since $C_a$ is anti-monotonic, $\beta$ must also satisfy $C_a$. That shows $C_a$ is prefix anti-monotonic.

Similarly, suppose sequence $\alpha$ satisfies $C_m$. Consider a prefix $\gamma$ of $\alpha$. Clearly, $\gamma$ is a supersequence of $\alpha$, i.e., $\gamma \sqsupseteq \alpha$. Since $C_m$ is monotonic, $\gamma$ must also satisfy $C_m$. That is, $C_m$ is prefix monotonic. □

For example, the length constraint $\text{len}(\alpha) \leq 10$ is anti-monotonic. It must also be prefix anti-monotonic. This is because if the length of a sequence $\alpha$ is no more than ten, the length of every prefix of $\alpha$ must be no more than ten as well. Similarly, $len(\alpha) \geq 10$ is prefix monotonic since if the length of any prefix of $\alpha$ is no less than ten, $\alpha$ must be no less than ten as well.

A succinct constraint is not necessarily prefix anti-monotonic or prefix monotonic. However, since succinct constraints can be pushed deep directly into the mining process (no matter which sequential pattern mining method is applied), the pushing of such constraints will not be analyzed further here. We will further discuss the relationship between succinct constraints and prefix monotone constraints in Section 8.4.

Now, let's examine regular expression constraints. A well-known result from the formal language theory is that for every regular expression $E$, there exists a deterministic finite automata $M_E$ such that $M_E$ accepts exactly the language generated by $E$.

Given a regular expression $E$, let $M_E$ be the corresponding (deterministic finite) automata. Let $\alpha$ be a sequence. $\alpha$ is called *legal* with respect to $E$ if a state in $M_E$ can be reached following $\alpha$. From a regular expression constraint $E$, we can derive a constraint $L_E$ such that a sequence $\alpha$ satisfies $L_E$ if and only if $\alpha$ is legal with respect to $E$. Clearly, for each sequence $\alpha$ satisfying the regular expression constraint $E$,

every prefix of $\alpha$ must be legal with respect to $E$. Furthermore, for each sequence $\beta$ legal with respect to $E$, every prefix of $\beta$ must also be legal with respect to $E$. Thus, we have the following lemma.

**Lemma 2** Given a regular expression constraint $E$. Let $L_E$ be the constraint on legal prefix with respect to $E$. Constraint $L_E$ is prefix anti-monotonic.

Based on the above discussion, we have the following statement.

**Theorem 2** *All the commonly used constraints discussed in Section 3, except for g_sum and average, have prefix-monotone property.*

Theorem 2 indicates that prefix-monotone property covers more commonly used constraints than traditional anti-monotonic and monotonic properties, since prefix-monotone property is weaker than anti-monotone and monotone properties. All anti-monotonic or monotonic constraints are prefix-monotonic, but the reverse direction is not true.

Often, mining with a weaker constraint may lead to the less efficiency. Thus, one may wonder whether mining with prefix-monotone property is less efficient than mining using the classical anti-monotonicity-based Apriori methods. The rest of this paper will address this concern.

### 5.2 Pushing prefix-monotone constraints into sequential pattern mining

First, we introduce the concept of *projected database*. For sequence $\alpha \sqsubseteq \beta$, sequence $\gamma$ is said the *projection* of $\beta$ with respect to $\alpha$ if (1) $\gamma \sqsubseteq \beta$, (2) $\alpha$ is a prefix of $\gamma$, and (3) there exists no proper super-sequence $\gamma'$ of $\gamma$ such that $\gamma' \sqsubseteq \beta$ and $\gamma'$ also has $\alpha$ as a prefix. Projection is also denoted by $\gamma = \beta/\alpha$. For example, if $\alpha = \langle b\,c\rangle$, $\beta = \langle (abc)d(ace)\,f\rangle$, then $\gamma = \beta/\alpha = \langle b\,(ce)\,f\rangle$.

Projection can be computed using Algorithm 1.

**Algorithm 1** (Computing projection)

**Input:** Sequences $\alpha = A_1 \cdots A_m$ and $\beta = B_1 \cdots B_n$, where $A_i$ and $B_j$ are transactions, the global order $R$ on the set of items;
**Output:** $\gamma = \beta/\alpha$, the projection of $\beta$ with respect to $\alpha$;
**Method:**

1. $j = 1$;
2. for $i = 1$ to $n$ do
   (a) if $j > n$ then output $\langle\rangle$ (since $\alpha \not\sqsubseteq \beta$); exit;
   (b) if $A_i \not\subseteq B_j$ then $j = j + 1$; goto Step 1;
   (c) $j = j + 1$;

   end for
3. Let $x$ be the last item in $A_m$ according to order $R$;
4. Let $Z$ be the set of items in $B_{j-1}$ that follow $x$ in order $R$;
5. output $\langle A_1 \cdots A_{m-1}(A_m \cup Z)B_jB_{j+1} \cdots B_n\rangle$;

Given a sequence database $SDB$ and a sequence $\alpha$, the *$\alpha$-projected database*, denoted by $SDB|_\alpha$, is the set of projections of sequences in $SDB$ having $\alpha$ as a subsequence, i.e., $SDB|_\alpha = \{\gamma | \gamma = \beta/\alpha, \beta \in SDB \wedge \alpha \sqsubseteq \beta\}$. Since $\alpha$ appears as prefix in every sequence in $SDB|_\alpha$, for the sake of brevity, we can omit the occurrences of $\alpha$ as prefixes in $SDB|_\alpha$. For a sequence $\beta \in SDB|_\alpha$, we only record the *suffix* $\beta'$. If the last transaction of $\alpha$ and the first transaction of $\beta'$ are in the same transaction of $\beta$, then a symbol "_" is put in the first transaction of $\beta'$. Therefore, we have $\beta = \alpha \cdot \beta'$.

Now, let us examine an example of constraint pushing.

*Example 2* Let the sequence database $SDB$ be Table 1, 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 conducted in the following steps.

1. *Find length-1 patterns and remove irrelevant sequences.* Similar to sequential pattern mining without constraint $C$, one needs to scan $SDB$ once, which identifies patterns $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle d\rangle$, and $\langle e\rangle$ 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.* Without considering constraint $C$, the complete set of sequential patterns should 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 $\langle e\rangle$. However, since only patterns with prefix $\langle a\rangle$ may satisfy the constraint $C$, i.e., only $\langle a\rangle$ is legal with respect to constraint $C$, the other four subsets of patterns are 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 $\langle ad\rangle$. With the existence of constraint $C$, pattern $\langle a\rangle$ fails $C$ and thus is discarded; and $\langle(ab)\rangle$ is illegal with respect to constraint $C$, so the second subset of patterns is pruned. The remaining subsets of patterns should be explored one by one.
4. *Mine subsets recursively.* To mine patterns having $\langle ab\rangle$ as a prefix, we form the $\langle ab\rangle$-projected database $SDB|_{\langle ab\rangle} = \{\langle(\_c)dd\rangle, \langle(\_c)dd\rangle\}$. By recursively mining the projected database, we identify sequential pattern $\langle a(bc)d\rangle$ which satisfies the constraint.
   To mine patterns with prefix $\langle ac\rangle$, we form $\langle ac\rangle$-projected database $SDB|_{\langle ac\rangle} = \{\langle dd\rangle, \langle dd\rangle, \langle b\rangle$. Every sequence in the projected database contains no sub-

sequence satisfying the constrain. Thus, the search within $TDB|_{\langle ac \rangle}$ can be pruned. In other words, we will never search any projected database which does not potentially support patterns satisfying the constraint.
Similarly, we search the $\langle ad \rangle$-projected database and find $\langle add \rangle$ is a sequential pattern satisfying the constraint.
In summary, during the recursive mining, if the prefix itself is a pattern satisfying the constraint, it should be an output. The prefixes legal with respect to 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: $\{\langle a(b\,c)d \rangle, \langle add \rangle\}$.

Let's verify the correctness and completeness of the mining process in Example 2. As shown in the example, if $\langle x_1 \rangle, \ldots, \langle x_n \rangle$ are the complete set of length-1 patterns, the complete set of sequential patterns can be divided into $n$ subsets without overlap and the mining can also be reduced to mining $n$ projected databases. Such a divide-and-conquer strategy can be applied recursively. Its correctness has be shown in Pei et al. (2001). We recall it as follows.

**Theorem 3** *Correctness of depth-first search mining,* (Pei et al., 2001)

1. *Given a sequence database $SDB$. Let $\langle x_1 \rangle, \ldots, \langle x_n \rangle$ be the complete set of length-1 patterns, the complete set of sequential patterns can be divided into n subsets without overlap: the $i^{th}$ subset contains sequential patterns with prefix $\langle x_i \rangle$ and can be mined from $\langle x_i \rangle$-projected database.*
2. *In the $\langle X_1 \cdots X_l \rangle$-projected database, let $x_1, \ldots, x_n$ be the complete set of items such that $\langle X_1 \cdots X_{l-1}(X_l \cup \{x_i\}) \rangle$ $(1 \le i \le n)$ is frequent, and let $y_1, \ldots, y_m$ be the complete set of items such that $\langle X_1 \cdots X_l(y_j) \rangle$ $(1 \le j \le m)$ is frequent. The complete set of sequential patterns with prefix $\langle X_1 \cdots X_l \rangle$ can be divided into $(n + m + 1)$ subsets without overlap.*
   a. *The first subset contains pattern $\langle X_1 \cdots X_l \rangle$ itself.*
   b. *The $(i+1)^{th}$ subset $(1 \le i \le n)$ contains patterns with prefix $\langle X_1 \cdots X_{l-1}(X_l \cup \{x_i\}) \rangle$ and can be mined from $\langle X_1 \cdots X_{l-1}(X_l \cup \{x_i\}) \rangle$-projected database.*
   c. *The $(n + j + 1)^{th}$ subset $(1 \le j \le m)$ contains patterns with prefix $\langle X_1 \cdots X_l(y_j) \rangle$ and can be mined from $\langle X_1 \cdots X_l(y_j) \rangle$-projected database.*
3. $SDB|_{\langle X_1 \cdots X_{l-1}(X_l \cup \{x\}) \rangle} = (SDB|_{\langle X_1 \cdots X_{l-1} X_l \rangle})|_{\langle X_1 \cdots X_{l-1}(X_l \cup \{x\}) \rangle}$
   $SDB|_{\langle X_1 \cdots X_l(y) \rangle} = (SDB|_{\langle X_1 \cdots X_l \rangle})|_{\langle X_1 \cdots X_l(y) \rangle}$

The removal of a sequence that does not satisfy the constraint is justified by the following lemma.

**Lemma 3** Let the given constraint be $C$.

1. In a sequence database $SDB$, if a sequence $\alpha$ does not contain any subsequence satisfying $C$, then the set of sequential patterns satisfying $C$ in $SDB$ is identical to the set of sequential patterns satisfying $C$ in $SDB - \{\alpha\}$.
2. Let $\alpha \cdot \beta$ be a sequence in the $\alpha$-projected database $SDB|_\alpha$. If there exists no a subsequence $\alpha \cdot \gamma \sqsubseteq \alpha \cdot \beta$ satisfying $C$, then the set of sequential patterns

satisfying $C$ in $SDB|_\alpha$ is identical to the set of sequential patterns satisfying $C$ in $SDB|_\alpha - \{\alpha \cdot \beta\}$.

*Proof* We prove the first claim. The second claim can be proved similarly.

Since $SDB - \{\alpha\} \subset SDB$, every sequential pattern in $SDB - \{\alpha\}$ is also a sequential pattern in $SDB$. Thus, every sequential pattern in $SDB - \{\alpha\}$ satisfying $C$ is also a sequential pattern in $SDB$ satisfying $C$. Moreover, for any sequence $\beta$ that is not a subsequence of $\alpha$, the support of $\beta$ in $SDB - \{\alpha\}$ is identical to the support of $\beta$ in $SDB$.

Suppose there exists a sequential pattern $\gamma$ satisfying $C$ in $SDB$ but not in $SDB - \{\alpha\}$. $\gamma$ must be a subsequence of $\alpha$. That contradicts the assumption that $\alpha$ does not contain any subsequence satisfying $C$. Thus, every sequential pattern satisfying $C$ in $SDB$ is also a sequential pattern satisfying $C$ in $SDB - \{\alpha\}$. The claim is proved. □

With a prefix-monotone constraint, one only needs to search in the projected databases having prefixes potentially satisfying the constraint, as suggested in the following lemma.

**Lemma 4** Given a prefix-monotone constraint $C$. Let $\alpha$ be a sequential pattern.

1. When $C$ is prefix anti-monotonic, if $C(\alpha) = \mathit{false}$, then there exists no sequential patterns that have $\alpha$ as a prefix and also satisfy constraint $C$.
2. When $C$ is prefix monotonic, if $C(\alpha) = \mathit{true}$, then every sequential pattern having $\alpha$ as a prefix satisfies $C$.
3. When $C$ is a regular expression constraint, if $\alpha$ is illegal with respect to $C$, then there exists no sequential patterns that have $\alpha$ as a prefix and also satisfy constraint $C$.

*Proof* The lemma follows the prefix-monotone property immediately. In case 2, constraint testing can be waived for mining in $SDB|_\alpha$. □

Based on the above discussion, we have the constrained sequential pattern mining algorithm PG as follows.

**Algorithm 2** (PG) Mining sequential patterns with prefix-monotone constraints.

**Input:** A sequence database $SDB$, support threshold *min_sup*, and prefix-monotone constraint $C$;
**Output:** The complete set of sequential patterns satisfying $C$;
**Method:**

call $prefix\_growth(\langle\rangle, SDB)$.

**Function** $prefix\_growth(\alpha, SDB|_\alpha)$
// $\alpha$: prefix; $SDB|_\alpha$: the $\alpha$-projected database

1. Let $l$ be the length of $\alpha$. Scan $SDB|_\alpha$ once, find length-$(l+1)$ frequent prefix in $SDB|_\alpha$, and remove infrequent items and useless sequences;

2. for each length-$(l+1)$ frequent prefix $\alpha'$ potentially satisfying the constraint $C$ (Lemma 4) do
   (a) if $\alpha'$ satisfies $C$, then output $\alpha'$ as a pattern;
   (b) form $SDB|_{\alpha'}$;
   (c) call $prefix\_growth(\alpha', SDB|_{\alpha'})$

**Theorem 4** *Given a prefix-monotone constraint C, Algorithm PG finds the complete set of sequential patterns satisfying the constraint.*

*Proof* The correctness of the algorithm follows the lemmas and the analysis above immediately. □

*Is PG an efficient algorithm?* We have the following analysis.

First, Algorithm PG takes PSP as the basic sequential pattern mining algorithm and pushes prefix-monotone constraints deeply into the PSP mining process. The performance study in Pei et al. (2001) shows that PSP outperforms GSP, owing to the following factors.

- PSP adopts a prefix growth and database projection framework: for each frequent prefix subsequence, only its corresponding suffix subsequences need to be projected and examined without candidate generation.
- PSP applies a divide-and-conquer strategy so that sequential patterns are grown by exploring only local frequent patterns in each projected database.
- PSP explores further optimizations, including a pseudo-projection technique when the projected database and its associated pseudo-projection processing structure fits in main memory, etc.

Second, PG handles a broader scope of constraints than anti-monotonicity and monotonicity. A typical such example is regular expression constraints, which is difficult to be explored using an Apriori-based method, as shown in SPIRIT. By PG, such constraints can be naturally pushed deep into the mining process.

Interestingly, both PG and SPIRIT (Garofalakis et al., 1999) push regular expression constraints by relaxing the constraint to achieve some nice property facilitating the constraint-based pruning. In particular, SPIRIT(V) requires every candidate sequence to be valid with respect to some state of the automata $\mathcal{A}_{\mathcal{R}}$, which shares a similar idea with PG. However, the SPIRIT methods adopts the candidate-generation-and-test framework, which is more costly than the pattern growth methods. Moreover, the SPIRIT methods are dedicated to pushing regular expression constraints, while PG is capable in pushing many constraints more than regular expression ones. For example, anti-monotonic or monotonic constraints that are not regular expression constraints, such as super-pattern constraints and some aggregate constraints in Table 2, can be consistently pushed in PG, but cannot be handled by SPIRIT. As will be shown in the experimental results, PG outperforms SPIRIT in pushing regular expression constraints.

Third, constraint checking by Lemma 3 further shrinks projected databases effectively, due to its removal of useless sequences with respect to a given constraint during the prefix growth. This ensures that search is pursued in promising space only. Since many irrelevant sequences can be pruned in large databases, the projected databases keep shrinking quickly.

One may wonder whether Apriori-based methods, such as GSP and SPADE, can do similar prefix-based pruning using prefix-monotone constraints. Taking a non-anti-monotonic regular expression constraint as an example, for Apriori-based methods, a pattern whose prefix failing a constraint cannot be pruned since inserting more items/transactions to the pattern at *some other* positions may still lead to a valid pattern. However, by exploring prefix-monotone constraints, PG puts stronger restrictions on the possible subsequences to grow and thus prunes search space more effectively.

In summary, *although prefix-monotone property is weaker than Apriori property, since PG uses a different methodology for the mining, it still achieves better performance than Apriori-based methods.*

## 6 Handling tough aggregate constraints by PG

Besides regular expression constraints, one may wonder whether PG can effectively handle the two tough aggregate constraints in Table 2, *average* and *g_sum*? Both constraints are neither anti-monotonic nor monotonic. Even worse, they are not prefix-monotone! Let's examine such an example.

*Example 3* Let us mine sequential patterns with constraint $C \equiv \text{avg}(\alpha) \leq 25$ in a sequence database $SDB$ as shown in Table 3, with support threshold = 2. The four items in the database are of values 10, 20, 30 and 50, respectively. For convenience, the item values are used as Ids of items.

Constraint $C$ cannot be directly pushed into the PSP mining process. For example $\alpha = \langle 50 \rangle$ cannot be discarded even $\text{avg}(\alpha) \not\leq 25$, since by appending more elements to $\alpha$, we may have $\alpha' = \langle 50\ 10\ 20\ 10 \rangle$ and $\text{avg}(\alpha') \leq 25$. Also, one can easily verify that $C$ is not prefix-monotone.

In Pei et al. (2001), a technique was developed to push *convertible* constraints, like $\text{avg}(X) \geq 25$, into frequent itemset mining on *transactional databases*. The general idea is to use a proper order of frequent items, like value descending order for constraint $\text{avg}(X) \geq v$, such that the list of frequent items according to the order has a nice anti-monotonic or monotonic property.

*Can we apply the technique in* Pei et al. (2001) *to tackle the aggregate constraints for sequential pattern mining?* Unfortunately, the answer is negative. For every sequence, a temporal order has been pre-composed and we do not have the freedom to rearrange the items in sequences. The trick of simple ordering does not work well here.

Thus, new constraint pushing methods should be explored.

**Table 3** Another sequence database $SDB$

| Sequence_id | Sequence |
|---|---|
| 10 | ⟨50 10 20 20⟩ |
| 20 | ⟨30 50 20⟩ |
| 30 | ⟨50 10 20 10 10⟩ |
| 40 | ⟨30 20 10⟩ |

Let's examine how to push constraint "avg($\alpha$) $\leq v$" deep into the PG mining process.

Value-ascending order over the set of items should be used to determine the order of projected databases to be processed. An item $i$ is called a *small item* if its value $i.value \leq v$, otherwise, it is called a *big item*.

In the first scan of a (projected) database, unpromising big items in sequences should be removed according to the following two rules.

**Lemma 5** Unpromising sequence pruning rule In mining sequential patterns with constraint $avg(\alpha) \leq v$, for a sequence $\alpha$, let $n$ be the number of instances of small items and $s$ be the sum of them. If there are multiple instances of one small item, the value of that item should be counted multiple times. For any big item $x$ in $\alpha$ such that $\frac{s+x.value}{n+1} > v$, there exists no any subsequence $\beta \sqsubseteq \alpha$ that contains $x$ and $avg(\beta) \leq v$.

*Proof* Consider any $\beta \sqsubseteq \alpha$ such that $x$ is in $\beta$. Clearly, the occurrences of small items in $\beta$ is a subset of the occurrences of small items in $\alpha$. Thus, $avg(\beta) \geq \frac{s+x.value}{n+1} > v$. □

The big items identified by Lemma 5 are called *unpromising*. Removal of unpromising big items do not miss any sequential patterns satisfying the constraint. Instead, it shrinks the sequence database and facilitates the mining.

Similarly, unpromising sequence pruning rule can be recursively applied in an $\alpha$-projected database. For a projection $\gamma = \beta/\alpha$, let $n$ be the number of instances of small items appearing in $\gamma$ but not in $\alpha$ and $s$ be the sum of them. A big item $x$ in $\alpha$ is unpromising and should be removed if $(s + sum(\alpha) + x.value)/(n + \#items(\alpha) + 1)$ violates the constraint. Here, function $\#items(\alpha)$ returns the number of instances of items in sequence $\alpha$.

Moreover, an item marking method can be developed to mark and further prune some unpromising items as follows. In the $\alpha$-projected database,[1] when a pattern $\beta$ is found where the first item following $\alpha$ is a small item, we check whether that small item can be replaced by a big item $x$ frequent in the projected database and still can get average value satisfying the constraint. If so, prefix $\langle \alpha \cdot x \rangle$ is marked *promising* and does not need to be checked and marked again in this projected database. When all patterns with some small item as the first one following $\alpha$ have been found, for the prefixes with a big item $x$ following $\alpha$ having not been marked, $\langle \alpha \cdot x \rangle$ as well as the projected databases can be pruned if $\langle \alpha \cdot x \rangle$ violates the constraint. We call this the *unpromising pattern pruning rule*.

The rationale of this rule is as follows. For a big item $x$, if $\langle \alpha \cdot x \rangle$ violates the constraint but $\langle \alpha \cdot x \cdot \beta \rangle$ is a sequential pattern satisfying the constraint, then there must be some $\beta' \sqsubseteq \beta$ such that $\beta'$ starts with a small item and $\langle \alpha \cdot \beta' \rangle$ is a sequential pattern satisfying the constraint.

We elaborate on the rules and the mining procedure using the $SDB$ in Example 3 as follows.

*Example 4* Let us consider mining $SDB$ in Table 3 with constraint $C \equiv \text{avg}(\alpha) \leq 25$.

[1] The whole database $SDB$ can be regarded as $SDB|_{\langle\rangle}$.

In the first scan of $SDB$, we remove the unpromising big items in sequences by applying the *unpromising sequence pruning rule*. For example, in the second sequence, 20 is the only small item and $\frac{20+50}{2} = 35 > 25$. This sequence cannot support andy sequential patterns having item 50 and satisfying the constraint $C$. Thus, item 50 in the second sequence should be moved.

In the same database scan, we also find length-1 patterns, $\langle 10\rangle$, $\langle 20\rangle$, $\langle 30\rangle$ and $\langle 50\rangle$. The set of patterns can be partitioned into four subsets without overlap: (1) those with prefix $\langle 10\rangle$; (2) those with prefix $\langle 20\rangle$; (3) those with prefix $\langle 30\rangle$; and (4) those with prefix $\langle 50\rangle$. These subsets of patterns should be explored one by one in this order.

1. The set of patterns with prefix $\langle 10\rangle$ can be found by constructing $\langle 10\rangle$-projected database and mining it recursively. The items in $\langle 10\rangle$-projected database are small ones, so all patterns in it have average no greater than 25 and thus satisfy the constraint. There are two patterns there: $\langle 10\rangle$ and $\langle 10\ 20\rangle$.
   When pattern $\langle 10\rangle$ is found, it can be regarded as a small item 10 following a prefix $\langle\rangle$. Thus, we apply the *unpromising pattern pruning rule* to mark and prune patterns. Prefix $\langle 30\rangle$ is marked as *promising*, since $avg(\langle 30\rangle \cdot \langle 10\rangle) = 20 < 25$. Prefix $\langle 30\rangle$ will not be checked against any other pattern after it is marked.
   None of the patterns with prefix $\langle 10\rangle$ can be used to mark prefix $\langle 50\rangle$.
2. Similarly, we can find patterns with prefix $\langle 20\rangle$ by constructing and mining $\langle 20\rangle$-projected database. They are $\langle 20\rangle$ and $\langle 20\ 10\rangle$. None of the patterns with prefix $\langle 20\rangle$ can be used to mark prefix $\langle 50\rangle$.
3. 30 is a big item and prefix $\langle 30\rangle$ violates the constraint. For patterns with prefix $\langle 30\rangle$, since prefix $\langle 30\rangle$ is marked, we need to construct $\langle 30\rangle$-projected database and mine it. Pattern $\langle 30\ 20\rangle$ is found.
4. Prefix $\langle 50\rangle$ has not been marked. According to the unpromising pattern pruning rule, no pattern with prefix $\langle 50\rangle$ can satisfy the constraint. We do not need to construct or mine $\langle 50\rangle$-projected database.

Constraint $avg(\alpha); \geq; v$ is dual with respect to constraint $avg(\alpha); \leq; v$. Therefore, we can prune unpromising (small) items and patterns similarly. Moreover, with the same idea, constraint $sum(\alpha); \theta; v$ (where $\theta \in \{\leq, \geq\}$, and items can be with non-negative and negative values) can also be pushed deep into PG mining process. This is left as an exercise for interested readers.

Thus, although the two concrete rules discussed above are specific for constraint $avg(\alpha); \leq; v$, the idea is general and can be applied to some other aggregate constraints. The central point is that we can prune unpromising items and patterns as the depth-first search goes deep. In summary, with minor revision, PG can be extended to handle some tough aggregate constraints without prefix-monotone property. With such extensions, all established advantages of PG still retain and the pruning is still sharp. This is also verified by our experimental results.

## 7 Experimental results and performance study

To evaluate the effectiveness and efficiency of the algorithms, we performed an extensive experimental evaluation on both synthetic and real datasets. The results are consistent. Limited by space, in this section, we report only the results on some

synthetic datasets generated by the IBM data generator described in Agrawal and Srikant (1995). The same data generator has been used in most studies on sequential pattern mining, such as Han et al. (2000); Srikant and Agrawal (1996); Zaki (1998). We refer readers to (Agrawal & Srikant, 1995) for more details on the generation of data sets.

All the experiments are performed on a 700 MHz AMD PC machine with 256 megabytes main memory, running Microsoft Windows 2000 Server. All methods are implemented using Microsoft Visual C++ 6.0. We compare performance of four methods as follows.

- *GSP*. GSP is an efficient sequential pattern mining method based on the anti-monotonic Apriorix property proposed in association mining (Agrawal & Srikant, 1994), which states the fact that *any super-pattern of a infrequent pattern cannot be frequent*. The GSP algorithm was implemented as described in Srikant and Agrawal (1996). We also revised GSP to push anti-monotonic and monotonic constraints.
- *SPIRIT*. SPIRIT (Garofalakis et al., 1999) is a family of algorithms for sequential pattern mining with regular expression constraints. Its general idea is to use some relaxed constraint which has nice property (like Apriorix) to prune. The main distinguishing factor among the schemes is the degree to which the regular expression constraints are enforced to prune the search space. In particular, algorithm SPIRIT(V) uses a relaxed constraint "valid with respect to some state of $M_E$" for a given regular expression $E$, where $M_E$ is the deterministic finite automata corresponding to $E$. Since SPIRIT(V) has overall the best performance among the SPIRIT family, we implemented it as described in (Garofalakis et al., 1999).
- *SPADE*. It is an efficient sequential pattern mining algorithm based on vertical format (Zaki, 2001). We got the source code from the author. We only study the performance of SPADE on mining *without constraint*. Revision of SPADE to handle constraints is non-trivial.
- *Pg*. Pg is the algorithm developed in this paper. In the implementation of PG, we adopted the level-by-level projection and pseudo-projection techniques described in Pei et al. (2001).

### 7.1 Comparison between PG and GSP without constraint

We first compare the efficiency of mining sequential patterns without constraint. Figure 1 shows the scalability of PG, GSP and SPADE with support threshold on dataset $C10T5S4I1.25D200k$, which contains 100, 000 sequences with 10, 000 items. The expected average number of items within a transaction is 5 (denoted as $T5$ and the expected average number of transaction in maximal sequential pattern is 4 (denoted as $S4$).

As can be seen from the figure, PG is more efficient and scalable than GSP and SPADE, while SPADE is faster than GSP, especially when support threshold is low. When the support threshold is high, there are only a limited number of patterns and the length of patterns is short, the gaps among the three methods are small. This comparison confirms the inherent advantage of PG over GSP and SPADE. In the

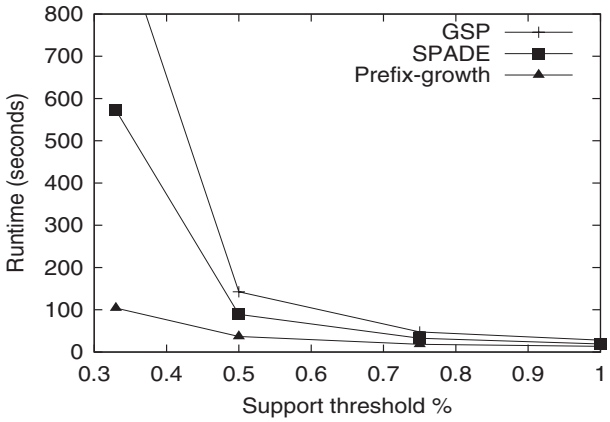


**Fig. 1** Scalability of GSP, SPADE and PG without constraint

remaining experiments, we study whether PG can carry the advantage to the extent of mining with constraints.

### 7.2 Pushing anti-monotonic constraints into sequential pattern mining

To evaluate the effect of a constraint on mining sequential patterns, we define the *selectivity* of a constraint as the ratio of the number of patterns FAILING the constraint against the total number of patterns. Therefore, a constraint with 0% selectivity filters out no pattern, while a constraint with 100% selectivity is the one filtering out all the patterns.

Anti-monotonic constraints can be pushed deep into GSP. We modified GSP such that it only generates candidates satisfying the constraint. Thus, both GSP and PG can push anti-monotonic constraints deep into the mining processes. To show the capability of GSP and PG in pushing anti-monotonic constraints into mining, we use constraint $Dur(\alpha) \leq \Delta t$ as an example here. With various values of $\Delta t$, the constraint achieves various selectivity. The support threshold is fixed to 0.7%. For GSP and PG, we compare the runtime with constraint to the one without constraint, respectively, and plot Fig. 2. The relative runtime is the ratio of the runtime of an algorithm with constraint over its runtime without constraint. In this way, the effect of constraint pushing on runtime improvement can be measured objectively.

In general, both methods are capable in pushing anti-monotone constraints. When the constraint selectivity is weak, since most patterns have to be generated and tested, not too much time can be saved. However, when the selectivity is high, i.e., many patterns do not satisfy the constraint, a major saving can be observed and PG performs better.

Comparing constraint pushing in GSP and PG, PG uses the constraint to prune both the patterns and the sequences in projected databases, while GSP has to search from the whole database all the time. When mining in large databases, the database search cost in GSP is non-trivial.

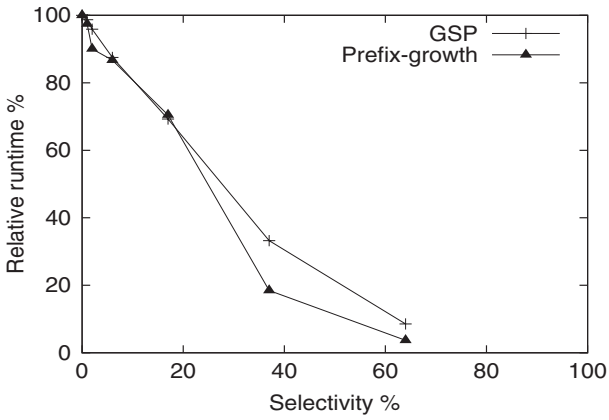


**Fig. 2** Pushing anti-monotone constraint

### 7.3 Pushing monotonic constraints into sequential pattern mining

Monotonic constraints can be used to save the cost of constraint checking, but it cannot cut the search space. In our experiments, since we use relatively simple constraints, such as $Dur(\alpha) \geq \Delta t$, the cost of constraint checking is CPU-bound. However, the cost of the whole mining process is I/O-bound. This makes the effect of pushing monotonic constraint into the mining process hard to be observed from runtime reduction. However, if we look at the number of constraint tests performed, the advantage of monotonic constraint pushing can be evaluated objectively. By pushing a monotonic constraint, PG can save a lot of effort on constraint testing. Therefore, in the experiment about pushing monotonic constraint, the number of constraint tests is used as the performance measure. We also revise GSP to handle monotonic constraints. Once a monotonic constraint is satisfied by a pattern, all candidates which are supersets of this pattern do not need to be checked anymore. Our results show that GSP and PG follow a similar trend on saving of constraint checking: the higher the constraint selectivity, the more saving. PG performs better. Since the constraint checking is often relatively efficient, the runtime saving is relatively small.

### 7.4 Pushing regular expression constraints into sequential pattern mining

The complexity of regular expression constraints can be roughly measured by the number of state changes (i.e., edges) in their corresponding deterministic finite automata. For each level of complexity, we randomly generate 1, 000 constraints and test both SPIRIT(V) and PG on them. The support threshold is set to 0.2%. The results are shown in Table 4.

With simple regular expression constraints, both SPIRIT(V) and PG are efficient. SPIRIT(V) is even better when the expression contains only two state changes. However, when the complexity of the constraints goes up, the average runtime of SPIRIT(V) increases dramatically. The increase of average runtime of PG is much

**Table 4** Experimental results on mining with regular expression constraints (runtime is measured in seconds

| Number of state changes | Average runtime of SPIRIT(V) | Average runtime of PG |
|---|---|---|
| 10 | 199.176 | 1.20 |
| 9 | 98.241 | 1.00 |
| 8 | 48.540 | 0.89 |
| 7 | 23.824 | 0.82 |
| 6 | 11.500 | 0.71 |
| 5 | 5.400 | 0.67 |
| 4 | 2.453 | 0.61 |
| 3 | 1.031 | 0.60 |
| 2 | 0.381 | 0.57 |

more moderate. Even with rather complicated constraints, PG is still very efficient. The results show that PG is more scalable and efficient than SPIRIT(V) in pushing regular expression constraints.

Based on our analysis, the difference between the two methods in performance can be explained as follows. With regular expression constraints, PG can prune both patterns and projected databases. However, SPIRIT(V) has to scan the whole sequence database repeatedly. On the other hand, even when SPIRIT(V) has pruned many candidates, it still generates some candidates and has to test them against the whole database.

### 7.5 Pushing constraints $avg(\alpha); \theta; v$ ($\theta \in \leq, \geq$) into sequential pattern mining

We also test PG on pushing constraint $avg(\alpha) \leq v$ to sequential pattern mining. The support threshold is set to 0.2%. The result is shown in Fig. 3. As can be seen, PG is efficient and scalable with respect to selectivity of the constraint.

To test the effect of prefix marking and pruning technique in PG for mining with constraint $avg(\alpha) \leq v$, we count the number of projected databases in Fig. 4. As can be seen, the prefix marking technique in PG prunes a good number of projected databases and contributes substantially to the scalability of PG. It is also interesting

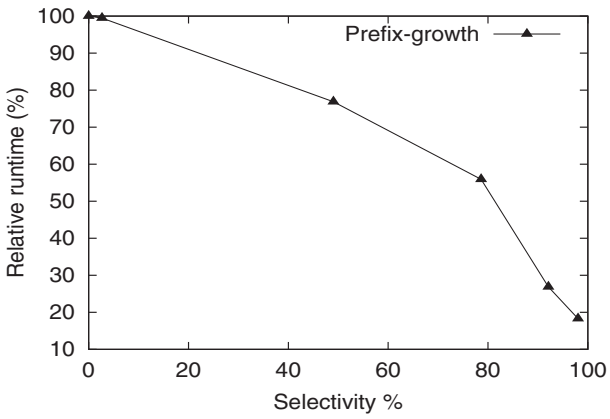


**Fig. 3** Scalability of PG with constraint $avg(\alpha) \leq v$

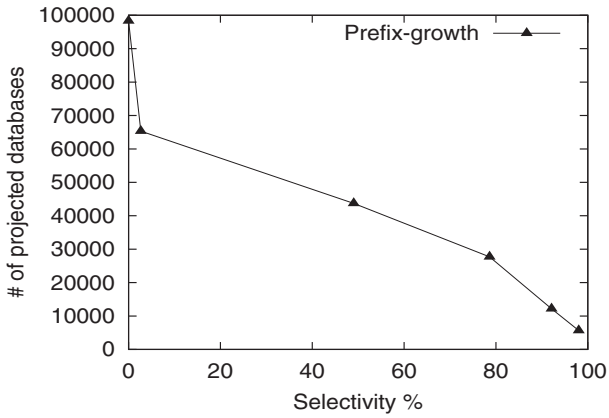


**Fig. 4** Number of projected database in PG with constraint $avg(\alpha) \leq v$

to see that the curves in Figs. 3 and 4 share similar shape. This indicates that the major cost in PG is mining projected databases. As the number of projected databases can be cut, the runtime can be brought down accordingly.

### 7.6 Scalability of PG with respect to support threshold and database size

We tested the scalability of pushing various constraints in PG with respect to support threshold. The results are shown in Fig. 5. From the figure, we can see that PG is scalable even when the support threshold is pretty low.

We also test the scalability of PG with respect to database size when mining with various constraints. The results are shown in Fig. 6. PG has linear scalability when mining with large databases.

In summary, the experimental results and performance study show that PG is efficient and scalable in mining sequential patterns with various constraints. The experimental results strongly support our theoretical analysis.

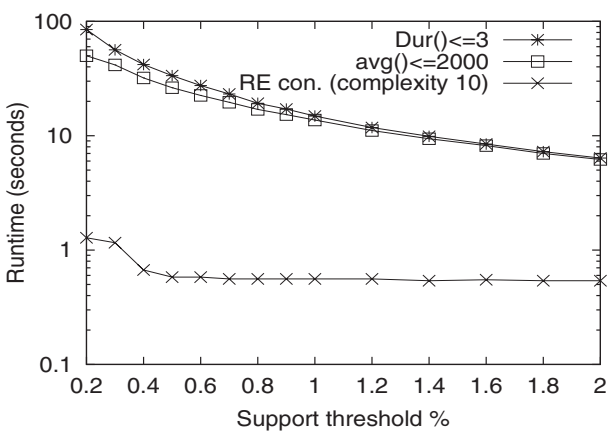


**Fig. 5** Scalability of PG with respect to support threshold

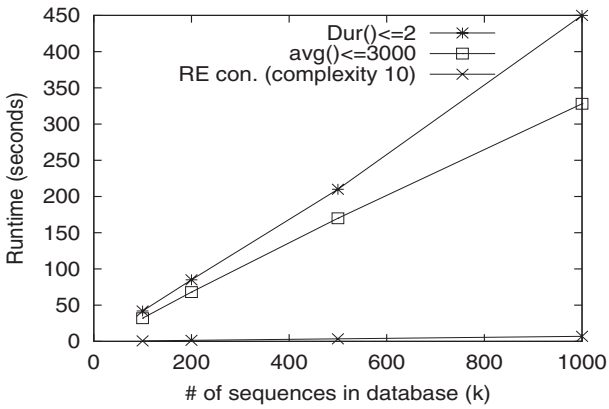


**Fig. 6** Scalability of PG with respect to database size

## 8 Discussion

### 8.1 Summary of contributions in this paper

This study distinguishes itself from previous work in the following aspects.

- Although PrefixSpan (Pei et al., 2001) is efficient in sequential pattern mining, it does not address the problem of pushing any constraint into the mining. Thus, it may suffer from returning too many patterns to the user and cannot achieve a user-specified constraint-based search.
  In this study, we push various commonly used constraints into sequential pattern mining. Even when there are many patterns in the database, PG can return those patterns really interesting to users. In this way, the effectiveness of sequential pattern mining has been improved dramatically.
  Technically, we identify properties facilitating constraint pushing and develop integrated strategies to handle various constraints.
- In this study, we present a thorough study on pushing various kinds of constraints into sequential pattern mining, not only just some particular kinds of constraints. It covers not only the constraints with nice property (like anti-monotonic, monotonic, and succinct ones), but also some tough and commonly used ones, such as regular expression constraints and those involving aggregate *avg*() and *sum*(). To our best knowledge, this is the first systematic study on sequential pattern mining with all those constraints. Thus, it enriches the utilization of constrained sequential pattern mining substantially.
  The previous work most related to this study is from Garofalakis et al. (1999), which proposes the use of regular expressions as a flexible constraint specification tool that enables user-controlled focus to be incorporated into the sequential pattern mining process. The family of SPIRIT algorithms are developed, while members in the family achieve various degrees to which the regular expression constraints are enforced to prune the search space. The general idea is that the algorithms use relaxed constraints having nice properties (like Apriorix) to filter out some unpromising patterns/candidates in their early stage.
  The approaches proposed in this study is essentially different from the SPIRIT algorithms. First, a pattern-growth pattern mining methodology is adopted.

Second, when relaxing a regular expression constraint, we do not require the relaxed constraint has an anti-monotonic property. Instead, we only looks for a weaker prefix-monotonic property. As shown in the experimental results, PG outperforms SPIRIT in pushing regular expression constraints.
Furthermore, instead of providing case by case solutions, we present a consistent framework, PG in which many kinds of constraints can be pushed deep. Users are liberated from the burden of choosing different methods depending on various constraints.

- PG improves the efficiency of constrained sequential pattern mining substantially. As verified by the theoretical analysis and performance study, PG clearly wins previous proposed methods on mining sequential patterns with anti-monotonic, monotonic and regular expression constraints. It is also efficient in mining with some tough aggregate constraints. That makes the constraint-based sequential pattern mining more powerful.

### 8.2 Mining with multiple constraints

We have studied the push of single constraints into sequential pattern mining. *Can we push multiple constraints deep into sequential pattern mining process?*

Multiple constraints in a mining query may belong to the same category (e.g., all are anti-monotonic) or to different ones. Moreover, different constraints may be on different properties of items/transactions (e.g., some could be on item price, while others could be on timestamps, etc.).

Fortunately, a constraint in the form of conjunctions and/or disjunction of prefix-monotone constraints can be pushed deep into PG mining process. We only need to keep track of which sub-constraints have been satisfied/violated. Based on that, whether the whole constraint is satisfied and whether further recursive mining of projected databases is needed can be determined. The details will not be presented here for lack of space.

For constraints involving aggregates *avg*() and *sum*() (where items can be with non-negative and negative values), PG uses a global order over all items to push them into the mining process. However, when a constraint involves more than one of such aggregates, and the orders required by these sub-constraints are conflicting, some coordination is needed. The general philosophy is to conduct a cost analysis to determine how to combine multiple order-consistent constraints and how to select a sharper constraint among order-conflicting ones. Limited by space, the details are omitted here.

### 8.3 Mining complex structures with constraints

We have shown that PG is effective and efficient for constraint-based sequential pattern mining. In many applications, such as mining XML data and bio-molecular data, constraint-based complex structure mining is required. *Is it possible to extend PG to handle those advanced tasks?*

Many structures, e.g., trees and directed acyclic graphs (DAGs), can be constructed and enumerated using some order. For example, a tree can be recorded and searched in breadth-first search or depth-first search order. By utilizing such an order, the PG idea can be extended to mine such complex structures. The set

of patterns can be divided and conquered. Accordingly, projected databases should be formed and searched. Since there could be much more combinations in complex structures than that in transactions and sequences, PG can prune search space and speed up mining dramatically, while the level-by-level candidate-generation-and-test methods may experience tremendous difficulties.

More interestingly, when mining complex structures with constraints, the order used by extended PG may facilitate constraint writing and pushing. For example, when mining frequent sub-tree structures in XML documents, it is natural to require that the upper part of the subtree (e.g., root and first level nodes) satisfies the constraint first, before pursuing the test of the lower parts. Thus, PG using the breadth-first search order is not only efficient, but also effective and semantic natural.

A systematic extension of PG can mine complex structures efficiently and many constraints can be pushed deep into the mining. As a future direction, interesting constraints specific to complex structure mining should be explored and examined further.

### 8.4 Succinct constraints and prefix-monotone constraints

Interestingly, succinct constraints can be rewritten using conjuncts and/or disjuncts of prefix anti-monotonic and prefix monotonic ones, as stated in the following lemma.

**Lemma 6** Every succinct constraint can be expressed using conjunction and/or disjunction of prefix anti-monotonic and monotonic constraints.

*Proof sketch* The proof of the lemma can be constructed by induction on the structure of $SAT(C)$ for a succinct constraint $C$. We show the two key steps here.

From the definition of succinct sequence set, for every succinct constraint $C$ such that $SAT(C)$ is a succinct sequence set, $C$ is prefix anti-monotonic.

Moreover, for a succinct constraint $C$ such that $SAT(C) = S_1 \cup S_2$, where $S_1$ and $S_2$ are two succinct sequence sets, $C$ can be expressed as $C = C_1 \vee C_2$, where $C_1$ and $C_2$ are the two prefix anti-monotonic constraints with $SAT(C_1) = S_1$ and $SAT(C_2) = S_2$.

On the other hand, let $S$ be a succinct sequence set. A succinct constraint $C$ such that $SAT(C) = \bar{S}$ is a prefix monotonic constraint, since for each sequence $\alpha$ not in $S$, every supersequence $\beta$ having $\alpha$ as a prefix will not be in $S$ either, i.e., $\beta$ satisfies the constraint. Thus, for a succinct constraint $C$ such that $SAT(C) = S_1 - S_2$, where $S_1$ and $S_2$ are two succinct sequence sets, $C$ can be expressed as a disjunction of a prefix anti-monotonic constraint and a prefix monotonic constraint. □

So far, we know that anti-monotonic, monotonic and succinct constraints all have prefix-monotone property.

## 9 Conclusions

In this paper, we have systematically studied the problem of pushing various constraints deep into sequential pattern mining. We characterize constraints for sequential pattern mining from both the application and constraint-pushing points

of views. A general property of constraints for sequential pattern mining, prefix-monotone property, is identified. It covers many commonly used constraints. An efficient algorithm, PG, is developed to push prefix-monotone constraints deep into the mining process. With some minor extensions, some tough constraints, like those involving aggregate *avg*() and *sum*(), can also be pushed deep into PG. Our extensive experimental results and performance study show that PG is efficient and scalable in mining large databases.

We have been working on a systematic implementation of constraint-based sequential pattern mining in a data mining system. Pg represents a new and promising methodology at effective and efficient mining sequential patterns with constraints. It is interesting to extend it towards mining sequential patterns with other more complicated constraints, and mining other kinds of time-related knowledge with various constraints.

**Acknowledgements** We are grateful to Dr. Mohammed J. Zaki for providing us the source code of SPADE. We also thank the reviewers for their constructive comments which help to improve the quality and presentation of the paper.

This research is supported in part by NSF Grant IIS-0308001, a President's Research Grant, an Endowed Research Fellowship Award and a startup grant in Simon Fraser University. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

## References

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93)* (pp. 207–216). New York: ACM.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)* (pp. 487–499). California: Morgan Kaufmann.

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)* (pp. 3–14). Washington, District of Columbia: IEEE Computer Society.

Ayres, J., Flannick, J., Gehrke, J., & Yiu, T. (2002). Sequential pattern mining using a bitmap representation. In *Proc. 2002 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)* (pp. 429–435). New York: ACM.

Bayardo, R.J., Agrawal, R., & Gunopulos, D. (1999). Constraint-based rule mining on large, dense data sets. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)* (pp. 188–197). Washington, District of Columbia: IEEE Computer Society.

Chiu, D.-Y., Wu, Y.-H., & Chen, A.L.P. (2004). An efficient algorithm for mining frequent sequences by a new strategy without support counting. In *Proc. of the Twentieth IEEE International Conference on Data Engineering (ICDE'04)* (pp. 275–286). Boston, Massachusetts: IEEE Computer Society.

Garofalakis, M., Rastogi, R., & Shim, K. (1999). SPIRIT: Sequential pattern mining with regular expression constraints. In *Proc. 1999 Int. Conf. Very Large Data Bases (VLDB'99)* (pp. 223–234). San Francisco, California: Morgan Kaufmann.

Grahne, G., Lakshmanan, L., & Wang, X. (2000). Efficient mining of constrained correlated sets. In *Proc. 2000 Int. Conf. Data Engineering (ICDE'00)* (pp. 512–521). Washington, District of Columbia: IEEE Computer Society.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., & Hsu, M.C. (2000). FreeSpan: Frequent pattern-projected sequential pattern mining. In *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'00)* (pp. 355–359). New York: ACM.

Kifer, D., Gehrke, J., Bucila, C., & White, W. (2003). How to quickly find a witness. In *Proc. 2003 ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'03)* (pp. 272–283). New York: ACM.

Kum, H.C.M., Pei, J., & Wang, W. (2003). Approxmap : Approximate mining of consensus sequential patterns. In *Proc. 2003 SIAM Int. Conf. Data Mining* (pp. 311–315). San Francisco, California.

Mannila, H., Toivonen, H. & Verkamo, A.I. (1997). Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, *1*:259–289.

Ng, R., Lakshmanan, L.V.S., Han, J., & Pang, A. (1998). Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)* (pp. 13–24). New York: ACM.

Pei, J. & Han, J. (2000). Can we push more constraints into frequent pattern mining? In *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'00)* (pp. 350–354). New York: ACM.

Pei, J., Han, J., & Lakshmanan, L.V.S. (2001). Mining frequent itemsets with convertible constraints. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)* (pp. 433–442). Washington, District of Columbia: IEEE Computer Society.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., et al. (2001). PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)* (pp. 215–224). Washington, District of Columbia: IEEE Computer Society.

Pei, J., Han, J., & Wang, W. (2002). Constraint-based sequential pattern mining in large databases. In *Proc. 2002 Int. Conf. on Information and Knowledge Management (CIKM'02)* (pp. 18–25). New York: ACM.

Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., & Dayal, U. (2001). Multi-dimensional sequential pattern mining. In *Proc. 2001 Int. Conf. Information and Knowledge Management (CIKM'01)* (pp. 81–88). New York: ACM.

Srikant, R. & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *Proc. 5th Int. Conf. Extending Database Technology (EDBT'96)* (pp. 3–17). Berlin Heidelberg New York: Springer.

Tzvetkov, P., Yan, X., & Han, J. (2003). Tsp: Mining top-k closed sequential patterns. In *Proc. of the Third IEEE International Conference on Data Mining (ICDM'03)* (p. 347). Washington, District of Columbia: IEEE Computer Society.

Wang, K. & Tan, J. (1996). Incremental discovery of sequential patterns. In *Proc. 1996 SIGMOD'96 Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD'96)* (pp. 95–102). New York: ACM.

Yan, X., Han, J., & Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large databases. In *Proc. 2003 SIAM Int. Conf. Data Mining* (pp. 406–417). New York: ACM.

Yang, J., Yu, P.S., Wang, W. & Han, J. (2002). Mining long sequential patterns in a noisy environment. In *Proc. 2002 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'02)* (pp. 68–75). New York: ACM.

Zaki, M.J. (1998). Efficient enumeration of frequent sequences. In *Proc. 7th Int. Conf. Information and Knowledge Management (CIKM'98)* (pp. 68–75). Washington, District of Columbia.

Zaki, M.J. (2001). Spade: An efficient algorithm for mining frequent sequences. *Mach. Learn.*, *42* (1-2),31–60.