DOI 10.1007/s10115-004-0182-5
Springer-Verlag London Ltd. © 2004
Knowledge and Information Systems (2005) 7: 499–514

Knowledge and Information Systems

Short Paper

Efficient mining of sequential patterns with time constraints by delimited pattern growth

Ming-Yen Lin[1], Suh-Yin Lee[2]

[1]Department of Information Engineering and Computer Science, Feng Chia University, Taiwan
[2]Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan

Abstract. An active research topic in data mining is the discovery of sequential patterns, which finds all frequent subsequences in a sequence database. The generalized sequential pattern (*GSP*) algorithm was proposed to solve the mining of sequential patterns with time constraints, such as time gaps and sliding time windows. Recent studies indicate that the pattern-growth methodology could speed up sequence mining. However, the capabilities to mine sequential patterns with time constraints were previously available only within the Apriori framework. Therefore, we propose the *DELISP* (delimited sequential pattern) approach to provide the capabilities within the pattern-growth methodology. *DELISP* features in reducing the size of projected databases by *bounded* and *windowed projection* techniques. *Bounded projection* keeps only time-gap valid subsequences and *windowed projection* saves nonredundant subsequences satisfying the sliding time-window constraint. Furthermore, the *delimited growth* technique directly generates constraint-satisfactory patterns and speeds up the pattern growing process. The comprehensive experiments conducted show that *DELISP* has good scalability and outperforms the well-known *GSP* algorithm in the discovery of sequential patterns with time constraints.

Keywords: Data mining; Pattern-growth; Sequence mining; Sequential patterns; Time constraint

1. Introduction

The discovery of sequential patterns is a complicated issue in data mining (Agrawal 1995; Bettini 1998; Garofalakis 1999; Mannila 1997; Pei 2002a; Rolland 2001; Srikant 1996; Tsoukatos 2001; Wang 1997; Zaki 2000, 2001). A typical example is a retail database, where each record corresponds to a customer's purchasing sequence, called *data sequence*. A data sequence is composed of all the customer's transactions

Received 5 April 2003
Revised 17 April 2004
Accepted 4 May 2004
Published online 22 October 2004

ordered by transaction time. Each transaction is represented by a set of literals indicating the set of items (called *itemset*) purchased in the transaction. The objective is to find all the frequent subsequences (called *sequential patterns*) in the sequence database.

The issue of mining sequential patterns with time constraints was first addressed in Srikant (1996). Three time constraints, including minimum gap, maximum gap and sliding time window, are specified to enhance conventional sequence discovery. For example, without time constraints, one may find a pattern $<(b, d, e)(a, f)>$. However, the pattern could be insignificant if the time interval between (b, d, e) and (a, f) is too long. Such patterns could be filtered out if the maximum gap constraint is specified.

Analogously, one might discover the pattern $<(b, d, e)(a, g)>$ from many data sequences consisting of itemset (a, g) occurring one day after the occurrence of itemset (b, d, e). Nonetheless, such a pattern is a false pattern in discovering weekly patterns, i.e. the minimum gap of 7 days. In other words, the sale of (b, d, e) might not trigger the sale of (a, g) in the next week. Therefore, time constraints including maximum gap and minimum gap should be incorporated in the mining to reinforce the accuracy and significance of mining results.

Moreover, conventional definition of an element of a sequential pattern is too rigid for some applications. Essentially, a data sequence is defined to support a pattern if each element of the pattern is contained in an individual transaction of the data sequence. However, the user may not care whether the items in an element (of the pattern) come from a single transaction or from adjoining transactions of a data sequence if the adjoining transactions occur close in time (within a specified time interval). The specified interval is named *sliding time window* (Srikant 1996). For instance, given a sliding time window of 5, a data sequence $<_{t_1}$(a,d) $_{t_2}$(b) $_{t_3}$(c)$>$ can support the pattern $<$(a,b,d)(c)$>$ if the difference between time t_1 and time t_2 is no greater than 5. Adding a sliding time window constraint to relax the definition of an element will broaden the applications of sequential patterns.

Although there are many algorithms dealing with sequential pattern mining (Agrawal 1995; Guralnik 2001; Lin 2002; Masseglia 1998; Oates 1997; Roddick 2002; Zaki 2001), few handle the mining with the addition of time constraints. *GSP* (generalized sequential pattern) algorithm (Srikant 1996) is the first algorithm that discovers sequential patterns with time constraints within the Apriori framework. To check whether a data sequence contains a certain candidate, *GSP* transforms each data sequence into items' transaction-time lists. The transformation speeds up time-constraint-related testing but introduces overheads during each database scanning.

Recent studies indicate that pattern-growth methodology could speed up sequence mining. Despite many studies on sequential pattern mining within the pattern-growth methodology (Han 2000; Lin 2002; Pei 2001, 2002a, 2002b; Pinto 2001), no algorithm fully functionally equivalent to *GSP* on time-constraint issues has been proposed so far. Especially, solving the sliding time-window constraint can be hardly found in the literature (except in the *GSP* context). In this paper, we propose a new algorithm, called the *DELISP* (delimited sequential pattern) for handling all three time constraints on sequential patterns, introduced in the context of *GSP*, within the pattern-growth framework. *DELISP* solves the problem by recursively growing valid patterns in projected subdatabases generated by subsequence projection. To accelerate mining by reducing the size of subsequences, the constraints are integrated in the projection to delimit the counting and growing of sequences. In *DELISP*, the *bounded projection* technique eliminates invalid subsequence projections caused by unqualified maximum/minimum gaps, the *windowed projection* technique reduces re-

dundant projections for adjacent elements satisfying the sliding-window constraint, and the *delimited growth* technique grows only the patterns satisfying constraints. The conducted experiments show that *DELISP* outperforms *GSP*. The scale-up experiments also indicate that *DELISP* has good linear scalability with the number of data sequences.

The rest of the paper is organized as follows. We formulate the problem in Sect. 2 and review some related work in Sect. 3. Section 4 presents the *DELISP* algorithm. The experimental evaluation is described in Sect. 5. We discuss the performance-improving factors in Sect. 6. Section 7 concludes our study.

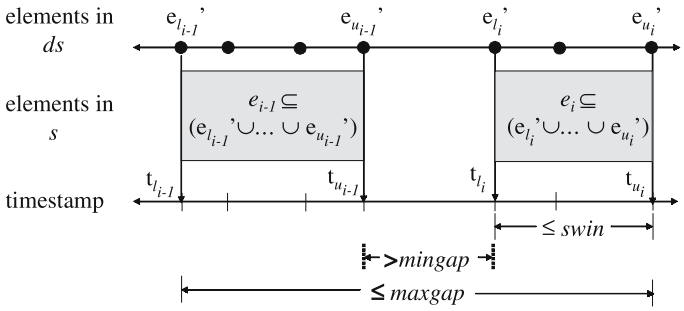
2. Problem statement

Let $\Psi = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be a set of literals, called *items*. An *itemset* $I = (\beta_1, \beta_2, \ldots, \beta_q)$ is a nonempty set of q items such that $I \subseteq \Psi$. A *sequence* s, denoted by $<e_1 e_2 \ldots e_w>$, is an ordered list of w *elements*, where each *element* e_i is an itemset. Without loss of generality, we assume the items in an element are in lexicographic order. The *size* of a sequence s, written as $|s|$, is the total number of items in all the elements in s. Sequence s is a *k-sequence* if $|s| = k$. For example, <(a)(c)(a)>, <(a,c)(a)> and <(b)(a,e)> are all 3-sequences.

The sequence database *DB* contains $|DB|$ data sequences. A *data sequence ds* having a unique identifier *sid* is represented by $sid/<_{t_1} e_1{}' \; _{t_2} e_2{}' \; \ldots \; _{t_n} e_n{}'>$, where element e_i' occurred at time t_i, $t_1 < t_2 < \ldots < t_n$. Four parameters are specified to mine the database *DB*: (1) *minsup* (**min**imum **sup**port), (2) *mingap* (**min**imum time **gap**), (3) *maxgap* (**max**imum time **gap**) and (4) *swin* (**s**liding time-**win**dow). Given *minsup*, the three constraints *mingap*, *maxgap*, *swin*, and the database *DB*, the problem is to discover the set of all time-constrained sequential patterns, i.e. sequential patterns satisfying the three time constraints.

A sequence s is a *time-constrained sequential pattern* if $s.sup \geq minsup$, where *s.sup* is the *support* of the sequence s and *minsup* is the user-specified minimum-support threshold. The *support* of s is the number of data sequences *containing* s divided by $|DB|$. A data sequence $ds = sid/<_{t_1} e_1{}' \; _{t_2} e_2{}' \; \ldots \; _{t_n} e_n{}'>$ *contains* a sequence $s = <e_1 e_2 \ldots e_w>$ if there exist integers $l_1, u_1, l_2, u_2, \ldots, l_w, u_w$ *and* $1 \leq l_1 \leq u_1 < l_2 \leq u_2 < \ldots < l_w \leq u_w \leq n$ such that the four conditions hold: (1) $e_i \subseteq (e_{l_i}{}' \cup \ldots \cup e_{u_i}{}')$, $1 \leq i \leq w$, (2) $t_{u_i} - t_{l_i} \leq swin, 1 \leq i \leq w$, (3) $t_{u_i} - t_{l_{i-1}} \leq maxgap, 2 \leq i \leq w$ and (4) $t_{l_i} - t_{u_{i-1}} > mingap, 2 \leq i \leq w$. Assume that t_j, *mingap*, *maxgap* and *swin* are all positive integers, *mingap* and *swin* can be zero, and $mingap < maxgap$. Figure 1 visualizes how a data sequence *ds* may contain the sequence s.

An example database *DB* is shown in the first column in Table 1. The data sequence $C1/<\mathbf{1}(c)\mathbf{35}(b,f)>$ has two elements (itemsets), one having a single item, c, occurring at time 1 and the other having items b and f occurring at time 35. Given $mingap = 2$, $maxgap = 30$, $swin = 2$, $C1$ contains <(c)> and <(b,f)>, but it does not contain either <(c)(b)> or <(c)(f)> because $35 - 1 > maxgap$. Similarly, $C2/<\mathbf{2}(b)\mathbf{4}(d)>$ does not contain <(b)(d)> because $4 - 2$ is not greater than *mingap*. Sequence <(a)(b)> is contained in $C3/<\mathbf{1}(\mathbf{a},d)_5(c)_6(c)\mathbf{8}(\mathbf{b})_{35}(a,f)>$ and $C5/<\mathbf{1}(\mathbf{a},b,e)_4(e)_7(f)_8(d)\mathbf{9}(\mathbf{b})>$, so that $<(a)(b)>.sup = 2/5$. With the specified *swin*, $C4/<\mathbf{2}(\mathbf{a})\mathbf{4}(\mathbf{d})_{30}(f)_{33}(a)_{61}(f)>$ may contain <(a,d)> ($4 - 2 \leq 2$) and $C5$ may contain <(b,d,f)> ($9 - 7 \leq 2$). Given $minsup = 40\%$, both <(a)(b)> and <(a,d)> are time-constrained sequential patterns while <(b,d,f)> is not. Table 1 also lists the set of all sequential patterns.



sequence $s = <e_1 e_2 \ldots e_w>$ is *contained* in data sequence $ds = sid/<{}_{t_1}e_1' \; {}_{t_2}e_2' \ldots \; {}_{t_n}e_n'>$ if all the items in e_i can be found in the element formed by combining elements between e_{l_i}' and e_{u_i}', where $1 \leq i \leq w$, and the constraints *swin*, *mingap*, *maxgap* are satisfied.

Fig. 1. Example of the sequence containment relationship

Table 1. Example sequence database (*DB*) and the time-constrained sequential patterns

Sequence	Time-constrained sequential patterns (*minsup* = 40%, *mingap* = 2, *maxgap* = 30, *swin* = 2)	Sequential patterns (*minsup=40%*)
C1/$<_1(c)_{35}(b,f)>$		
C2/$<_2(b)_4(d)>$		
C3/$<_1(a,d)_5(c)_6(c)_8(b)_{35}(a,f)>$	<(a)>,<(a)(b)>, <(a,d)>, <(a)(f)>, <(b)>, <(b,d)>, <(b,f)>, <(b)(f)>, <(c)>, <(d)>, <(f)>	<(a)>, <(a)(a)>, <(a)(b)>, <(a)(d)>, <(a)(f)>, <(b)>, <(b)(d)>, <(b)(f)>, <(c)>, <(c)(b)>, <(c)(f)>, <(d)>, <(d)(a)>, <(d)(b)>, <(d)(f)>, <(f)>
C4/$<_2(a)_4(d)_{30}(f)_{33}(a)_{61}(f)>$		
C5/$<_1(a,b,e)_4(e)_7(f)_8(d)_9(b)>$		

3. Related work

Much research has been focused in sequence mining without time constraints of *mingap*, *maxgap* and *swin* (Agrawal 1995; Ayres 2002; Han 2000; Lin 1998, 2002; Pei 2001; Shintani 1998; Zaki 2001). The *GSP* algorithm is the first algorithm that handles the time constraints in sequential patterns (Srikant 1996). Based on the Apriori framework (Agrawal 1995), the patterns are found in multiple database passes. In every database scan, each data sequence is transformed into items' time lists for fast finding of certain elements with a time tag. Because the start time and end time of an element (may comprise several transactions) must be considered, *GSP* defines contiguous subsequence for candidate generation and moves between the forward phase and backward phase for checking whether a data sequence contains a certain candidate (Srikant 1996).

A general pattern-growth framework was presented in Pei (2002b) for constraint-based sequential-pattern mining (Pei 2002a, 2002b). From the application point of view, seven categories of constraints, including *item*, *length*, *superpattern*, *aggregate*, *regular expression*, *duration* and *gap* constraints were covered. Among these

constraints, *duration* and *gap* constraints are tightly coupled with the support counting process because they confine how a data sequence contains a pattern. Orthogonally classifying constraints by their roles in mining, *monotonic*, *antimonotonic* and *succinct* constraints, were characterised and the *prefix-monotone* constraint was introduced (Pei 2002b). The *prefix-growth* framework, which pushes prefix-monotone constraints into *PrefixSpan* was also proposed in Pei (2002b). However, with respect to time constraints, *prefix-growth* only mentioned *maxgap* and *mingap* (though *duration* was addressed) with no implementation details and *swin* was not considered at all.

The *cSPADE* algorithm (Zaki 2000) extends the vertical mining algorithm *SPADE* (Zaki 2001) to deal with time constraints. Vertical mining approaches (Ayres 2002; Zaki 2000, 2001) discover sequential patterns using join operations and a vertical database layout, where data sequences are transformed into items' (sequence-id, time-id) lists. The *cSPADE* algorithm checks *mingap* and *maxgap* while doing temporal joins. Nevertheless, the huge sets of frequent 2-sequences must be preserved to generate the required classes for the *maxgap* constraint (Zaki 2000). While it is possible for *cSPADE* to handle constraints like *maxgap*/*mingap* by expanding the id lists and augmenting the join operations with temporal information (Zaki 2000), it does not appear feasible to incorporate *swin*. The *swin* constraint was not mentioned in *cSPADE*.

A different kind of time constraints, discovering patterns that involve multiple time granularities, was addressed in Bettini (1998). Simple or complex event structures, which are episodes (Mannila 1997) with time-interval restrictions similar to *mingap*/*maxgap* constraints, are discovered by the introduced timed automaton with granularities (Bettini 1998). Nevertheless, we are interested in the discovery of time-constrained sequential patterns built from itemsets.

4. DELISP: delimited sequential pattern mining

In Sect. 4.1, we introduce the terminology used in the proposed *DELISP* algorithm. Section 4.2 demonstrates the method by mining an example database. Section 4.3 describes the proposed algorithm. For convenience, we refer to a data sequence $ds = sid/\!<{}_{t_1}e_1\text{'}\ {}_{t_2}e_2\text{'}\ \ldots\ {}_{t_n}e_n\text{'}>$ as ds in the following context.

4.1. Terminology used in DELISP

Definition 1 (Frequent item). An item x is called a *frequent item* in a sequence database DB if $<(x)>.sup \geq minsup$.

Definition 2 (Stem, type-1 growth, type-2 growth, prefix). Given a sequential pattern ρ and a frequent item x in the sequence database DB, x is called the *stem item* (abbreviated as *stem*) of the sequential pattern ρ' if ρ' can be formed by (1) appending (x) as a new element to ρ or (2) extending the last element of ρ with x. The formation of ρ' is a *type-1 growth* if it is formed by appending (x), and a *type-2 growth* if it is formed by extending with x. The *prefix pattern* (abbreviated as *prefix*) of ρ' is ρ.

For example, given $<(a)>$ and the frequent item b, we may have the *type-1 growth* $<(a)(b)>$ by appending (b) to $<(a)>$ and the *type-2 growth* $<(a, b)>$ by

extending $<(a)>$ with b. The $<(a)>$ is the *prefix* and the b is the *stem* of both $<(a)(b)>$ and $<(a, b)>$. As to a *type-2 growth* $<(c)(a, \mathbf{d})>$, its *prefix* is $<(c)(a)>$ and its *stem* is d. Note that the null sequence, denoted by $<>$, is the *prefix* of any frequent 1-sequence.

Definition 3 (start-time, end-time, tag-list). The time stamp indicating the occurrence of itemset I in *ds* is marked in the projected database. If itemset I is contained in a single element ${}_{t_\delta}e_\delta$' in *ds*, the *start time* (abbreviated as **st**) and *end time* (abbreviated as **et**) pair **st**:**et** is marked as t_δ:t_δ. If I is contained in e_δ' $\cup e_{\delta+1}$' $\cup \ldots \cup e_\varepsilon$' (in *ds*), **st**:**et** is marked as t_δ:t_ε. We refer to the list of all the *st*:*et* pairs as the **tag list** of I in *ds*, which is denoted by $[\mathrm{st}_1{:}\mathrm{et}_1, \mathrm{st}_2{:}\mathrm{et}_2, \ldots, \mathrm{st}_k{:}\mathrm{et}_k]$, where $\mathrm{st}_i \leq \mathrm{et}_i$ for $1 \leq i \leq k$, $\mathrm{st}_i < \mathrm{st}_{i+1}$ and $\mathrm{et}_i < \mathrm{et}_{i+1}$ for $1 \leq i \leq k-1$.

Definition 4 (Accessible). Let the tag list of itemset I in *ds* be $[\mathrm{st}_1{:}\mathrm{et}_1, \mathrm{st}_2{:}\mathrm{et}_2, \ldots, \mathrm{st}_k{:}\mathrm{et}_k]$. An element e_a' is *accessible* from I in *ds* if its time stamp t_a satisfies: (1) $\mathrm{et}_i - swin \leq t_a \leq \mathrm{st}_i + swin$, where $i \in \{1, 2, \ldots, k\}$, or (2) $\mathrm{et}_i + mingap < t_a \leq \mathrm{st}_i + maxgap$, where $i \in \{1, 2, \ldots, k\}$, or (3) $t_b + mingap < t_a \leq t_b + maxgap$, where t_b is the time stamp of an *accessible* element e_b' from I in *ds*.

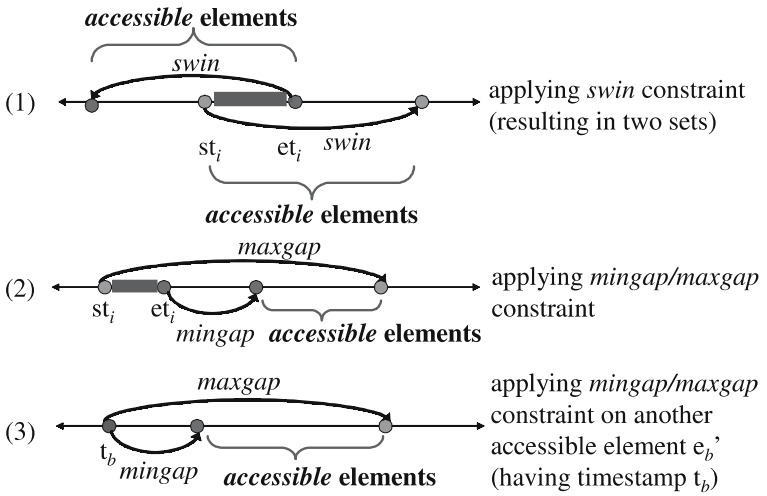


Fig. 2. Accessible elements from itemset I in *ds* with tag list $[\mathrm{st}_1{:}\mathrm{et}_1, \mathrm{st}_2{:}\mathrm{et}_2, \ldots, \mathrm{st}_k{:}\mathrm{et}_k]$

Figure 2 demonstrates the three accessible circumstances. Note that, when an accessible element is extended by condition (1) in Definition 4, the extension is checked on not violating *mingap* or *maxgap* constraints with respect to the previous itemset of I (in the pattern), denoted by I_p. The checking is to ensure that itemset I, having time stamps satisfying the *mingap*/*maxgap* constraint with I_p, does not violate the gap constraint after the type-2 extension. Such a checking requires projecting *st*:*et* of I_p, the detail of which is not shown in the following context for clearer illustration.

Lemma 1. Let *ds* contain the nonnull *prefix* $\rho = <e_1 e_2 \ldots e_p>$. Given the tag list of e_p in *ds*, a frequent item x in an element e_a' in *ds* can be a stem only if e_a' is *accessible* from e_p in *ds*.

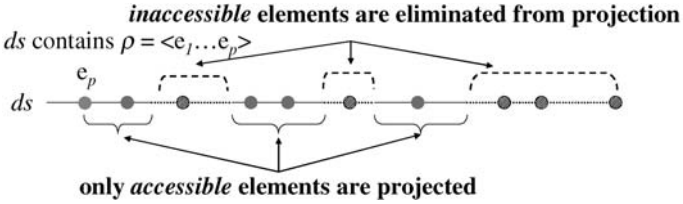


Fig. 3. The projected elements of *ds* with respect to ρ

Lemma 1 is based on the fact that a valid growth must satisfy time constraints. Hence, we may prevent the inaccessible elements from projection to speed up the growing process, as shown in Fig. 3. We further reduce projections by eliminating items in an accessible element from projection using Lemma 2, as depicted in Fig. 4.

Lemma 2. Let the last element in *prefix* ρ be e_p, the last item in e_p be x, and the tag list of e_p in *ds* be $[st_1{:}et_1, st_2{:}et_2, \ldots, st_k{:}et_k]$. Any item x' in an accessible element e_a' cannot be a stem if (1) $x' \leq x$ and (2) ${}_{t_a}e_a$' is accessible from ρ by satisfying $et_1 - swin \leq t_a \leq et_1$.

Lemma 2 is based on the fact that items are in lexicographic order within elements. Any item to be used as a stem for the type-2 growth having *prefix* ρ should have an order greater than the order of the last item in ρ. Thus, any small-ordered x' (located in ${}_{t_a}e_a$', $et_1 - swin \leq t_a \leq et_1$) need not be projected.

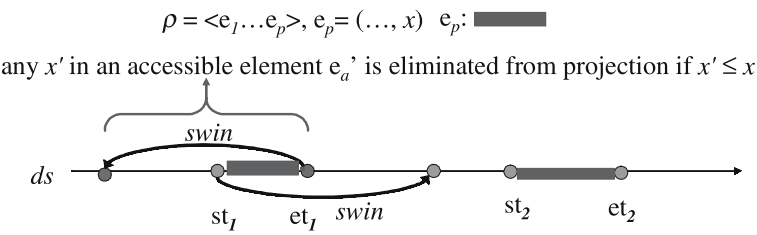


Fig. 4. Eliminating items having smaller lexicographic order from projection (Lemma 2)

4.2. Mining time-constrained sequential patterns by *DELISP*: an example

All the time-constrained sequential patterns are found by growing frequent sequences from size one to the maximum size. Frequent items in *DB* can be determined after scanning *DB* once. We then use each frequent item as a *stem* with *prefix* <> to form the set of all frequent 1-sequences. The subsequences satisfying the constraints are then projected into related subdatabases for further growing. The stems of type-1 and type-2 growth can be determined by scanning the subdatabases once. Recursively, the time-constraint integrated projection and growing techniques are applied to discover the frequent 2-sequences, 3-sequences, etc.

Example 1. Given *minsup* = 40%, *mingap* = 2, *maxgap* = 30, *swin* = 2 and the *DB* as shown in Table 1, *DELISP* mines the patterns by the following steps.

Step 1. Find frequent items. By scanning *DB* once, we have frequent items a (count = 3 for appearing in 3 data sequences $C3, C4$ and $C5$), b (count = 4), c (count = 2), d (count = 4) and f (count = 4). Nonfrequent item e is omitted from mining afterward. The five items are stems of type-1 growth, having *prefix* <>.

Step 2. Project corresponding subsequences to subdatabases. Considering the time-constrained sequential patterns having *prefix* $\rho = <(\boldsymbol{x})>$, each can be found in the subdatabase (named ρ-*DB*) generated by projecting all the data sequences having item $\boldsymbol{x}$ in *DB*. While projecting a data sequence *ds* into ρ-*DB*, we omit the nonfrequent items, those *inaccessible* elements (using Lemma 1) and those lexicographically smaller items (using Lemma 2). We tabulate the subdatabases <(a)>-*DB*, <(b)>-*DB*, <(c)>-*DB*, <(d)>-*DB* and <(f)>-*DB* in part 1 of Table 2.

Table 2. The projected subsequences in the ρ-*DB* subdatabases

***ρ*-DB**	**Projected sub-sequences**
Part 1: sub-databases of *DB*	
<(a)>-*DB*	*C3*/[1:1,35:35]/$<_1(d)_5(c)_6(c)_8(b)_{35}(a,f)>$; *C4*/[2:2,33:33]/$<_4(d)_{30}(f)_{33}(a)_{61}(f)>$; *C5*/[1:1]/$<_1(b)_7(f)_8(d)_9(b)>$
<(b)>-*DB*	*C1*/[35:35]/$<_{35}(f)>$; *C2*/[2:2]/$<_4(d)>$; *C3*/[8:8]/$<_6(c)_{35}(a,f)>$; *C5*/[1:1,9:9]/$<_7(f)_8(d)_9(b)>$
<(c)>-*DB*	*C3*/[5:5,6:6]/$<_6(c)_8(b)_{35}(a,f)>$
<(d)>-*DB*	*C3*/[1:1]/$<_5(c)_6(c)_8(b)_{35}(a,f)>$; *C4*/[4:4]/$<_{30}(f)_{33}(a)_{61}(f)>$; *C5*/[8:8]/$<_7(f)_9(b)>$
<(f)>-*DB*	*C4*/[30:30,61:61]/$<_{33}(a)_{61}(f)>$; *C5*/[7:7]/$<_8(d)_9(b)>$
Part 2: sub-databases of <(a)>-*DB*	
<(a)(b)>-*DB*	*C3*/[8:8]/$<_{35}(f)>$
<(a)(f)>-*DB*	*C5*/[7:7]/$<_8(d)_9(b)>$
<(a,d)>-*DB*	*C3*/[1:1]/$<_8(b)_{35}(f)>$; *C4*/[2:4]/$<_{30}(f)_{61}(f)>$
Part 3: sub-databases of <(b)>-*DB*	
<(b)(f)>-*DB*	*C5*/[7:7]/$<_8(d)_9(b)>$
<(b,d)>-*DB*	*None*
<(b,f)>-*DB*	*None*

Note: the notation '*st*:*et*' prior to a data sequence denotes the *start-time* and the *end-time* of the data sequence with respect to ρ projection.

Step 3. Mine each subdatabase for the subsets of time-constrained sequential patterns. In each subdatabase, we grow the patterns in each sequence according to the time constraints and determine which pattern is a valid time-constrained sequential pattern. Assume that we are growing patterns from *prefix* ρ, of which the last element is e_p and the tag list of e_p in *ds* is $[\mathrm{st}_1{:}\mathrm{et}_1, \mathrm{st}_2{:}\mathrm{et}_2, \ldots, \mathrm{st}_k{:}\mathrm{et}_k]$. The stems of potential type-1 growth come from the accessible e'_a, of which time stamp t_a satisfying $\mathrm{et}_i + mingap < t_a \leq \mathrm{st}_i + maxgap$, where $i \in \{1, 2, \ldots, k\}$. The stems of potential type-2 growth come from the accessible e'_a satisfying $\mathrm{et}_i - swin \leq t_a \leq \mathrm{st}_i + swin$, where $i \in \{1, 2, \ldots, k\}$. We may obtain the occurrence counts (i.e. supports) of stems after scanning ρ-*DB* once. Recursively, we then generate the corresponding ρ'-*DB* (having *prefix* ρ) for each stem having sufficient support count.

Step 4. Find all patterns by applying step 2 and step 3 on the subdatabases recursively. Considering the time-constrained sequential patterns having *prefix* ρ = <(a)(b)>, each can be found in the subdatabase (named <(a)(b)>-*DB*) generated by projecting all the data sequences having (b) in <(a)>-*DB*. Again, we eliminate the nonfrequent items, those *inaccessible* elements (using Lemma 1), and those lexico-

graphically smaller items (using Lemma 2). The projected subdatabases of $<(a)>$-*DB* are shown in part 2 of Table 2.

We then recursively apply the steps on $<(b)>$-*DB* for patterns having *prefix* $<(b)>$, on $<(c)>$-*DB* for patterns having *prefix* $<(c)>$, ..., and on $<(f)>$-*DB* for patterns having *prefix* $<(f)>$. By collecting the patterns found in the above process, *DELISP* efficiently discovers all the sequential patterns satisfying the time constraints.

4.3. The *DELISP* algorithm

Figure 5 presents the proposed *DELISP* algorithm. *DELISP* decomposes the mining problem by recursively growing patterns one item longer than the current patterns in the projected subdatabases. The potential items used to grow, called *delimited growth*, are subjected to *mingap* and *maxgap*. Therefore, we perform type-1 growth with items in each element ${}_{t_a}e'_a$ within range ($\mathrm{et}_i + mingap < t_a \leq \mathrm{st}_i + maxgap$), where $i \in \{1, 2, \ldots, k\}$, and type-2 growth with items in each element ${}_{t_a}e_a$' within range ($\mathrm{et}_i - swin \leq t_a \leq \mathrm{st}_i + swin$), where $i \in \{1, 2, \ldots, k\}$. The [$\mathrm{st}_1$:$\mathrm{et}_1$, st_2:et_2, ..., st_k:et_k] is the tag list of element $e_p \in$ *prefix* $<e_1 e_2 \ldots e_p>$ in *ds*. On projecting subdatabases, we avoid the bidirectional growth by imposing the item order in the type-2 growth, called *windowed-projection*. We always add a new item (in e_p) the order of which is lexicographically larger than the order of the existing items for type-2 growth.

Theorem 1. Algorithm *DELSIP* discovers the set of all time-constrained sequential patterns.

Proof. Obviously, *DELISP* discovers the set of all frequent 1-sequences in step 1. Clearly, a frequent k-sequence is formed by either a *type-1 growth* or a *type-2 growth* from a frequent $(k-1)$-sequence. Thus, the set of all time-constrained sequential patterns can be obtained by *type-1* and *type-2 growth*, from size one to the maximum size. Any item to be used as a stem must come from an accessible element; otherwise, the corresponding growth would violate either *swin* or the *mingap*/*maxgap* constraint. In Subroutine *ProjectDB*, by Lemma 1 and Lemma 2, those inaccessible items need not be projected, so they are eliminated. Subroutine *Mine* counts the supports of time-constraint-satisfied items for type-1 and type-2 growth, respectively. By recursively applying *ProjectDB* and *Mine*, *DELISP* discovers the set of all time-constrained sequential patterns. □

5. Experimental results

Extensive experiments were conducted to assess the performance of the *DELISP* algorithm. We compared the total execution times of *DELISP* and *GSP* (Srikant 1996) by varying the parameters of *mingap*, *maxgap* and *swin*. The scalability of the algorithm was also evaluated over different database sizes. The experiments were performed on an 866-MHz Pentium-III PC with 1024 MB memory running the Windows NT.

PrefixSpan (Pei 2001) does not handle the time constraints and therefore is not considered. However, note that, for gap constraints (*mingap* and *maxgap*), *PrefixSpan* could be applied with an extra pattern-counting step. In the step, patterns

Algorithm *DELISP*
Input: *DB* = a sequence database; *minsup* = minimum support; *mingap* = minimum time gap; *maxgap* = maximum time gap; *swin* = sliding time-window.
Output: the set of all time-constrained sequential patterns.
Method:
1. Scan *DB* once, find the set of all frequent items.
2. For each frequent item x,
   (a) form a time-constrained sequential pattern $\rho = <(x)>$ and output ρ.
   (b) call *ProjectDB*(ρ, *DB*) to construct sub-database ρ-*DB*.
   (c) call *Mine*(ρ-*DB*).

Subroutine *ProjectDB*(ρ, *Db*)
Parameters: ρ = pattern; *Db* = the sub-database.
Output: the sub-database ρ-*DB*.
Method:
1.For each data sequence $ds = sid/<_{t1}e_1$' $_{t2}e_2$'$\cdots$ $_{tm}e_n$'> in *Db*,
   (a) record the tag-list [st_1:et_1, st_2:et_2, …, st_k:et_k] of ρ in ds, where each st_i:et_i marks the *start-time*:*end-time* of the last element of ρ in ds.
   (b) (**Bounded-projection**) mark the list of accessible elements in ds. /* See Definition 4 (accessible) in Section 4.1 */
   (c) (**Windowed-projection**) drop item x' in an accessible element e_a' where et_1 - $swin \le t_a \le et_1$ and $x' \le x$. The item x is the last item in $e_p \in \rho = <e_1e_2\ldots e_p>$. /* Use Lemma 2 in Section 4.1 */
   (d) if the list of accessible elements is not empty, drop the non-frequent items in ds and project sid/[st_1:et_1, st_2:et_2, …, st_k:et_k]/< the list of accessible elements > to ρ-*DB*.

Subroutine *Mine*(ρ-*DB*)
Parameter: ρ-*DB* = the sub-database.
Output: time-constrained sequential patterns having prefix ρ.
Method:
1.For each data sequence $ds = sid/$[st_1:et_1, st_2:et_2, …, st_k:et_k]/$<_{t1}e_1$' $_{t2}e_2$'$\cdots$ $_{tm}e_n$'> in ρ-*DB*,
   (a) for each element e_i' with timestamp t_i in ds, insert the items in e_i'
      (i) to the stem set of potential type-1 growth if et_j+*mingap* < $t_i \le st_j$+*maxgap* where $j \in \{1, 2, \ldots, k\}$. (**Delimited-growth**/type-1)
      (ii) to the stem set of potential type-2 growth if et_j-*swin* $\le t_i \le st_j$+*swin* where $j \in \{1, 2, \ldots, k\}$. (**Delimited-growth**/type-2)
   (b) for each stem in the two sets, increase its support count by one.
2. Find the frequent items in the two sets by comparing the supports with *minsup*.
3. For each frequent item x in the two sets,
   (a) form a time-constrained sequential pattern ρ' (*prefix* ρ and stem x) and output ρ'.
   (b) call *ProjectDB*(ρ', ρ-*DB*) to construct sub-database ρ'-*DB*.
   (c) call *Mine*(ρ'-*DB*).

Fig. 5. Algorithm *DELISP*

discovered without time constraints can be verified in an extra scan of the whole database. Nevertheless, such an extension cannot be applied for *swin*. The *prefix growth* in Pei (2002b) gives no implementation details of gap constraints and no descriptions on sliding time windows, so *prefix-growth* is not compared in our experiments.

The *cSPADE* algorithm (Zaki 2000), though it accepts *mingap* and *maxgap* constraints, was not implemented in the comparison because it uses a vertical database layout. Additional storage space and computation time are required to transform the natively horizontal databases into vertical ones. In addition, the *swin* constraint is not handled in *cSPADE*. Revision of *cSPADE* to handle the *swin* constraint is not trivial. One possible implementation is to incorporate *swin* by incrementing the support for each distinct window in the vertical representation. Nevertheless, the join operation has to be extended, beyond temporal and equality join, to allow window join. For example, joining the id list of item x with that of item y, even when

their time stamps are not equal, now might generate itemset (x, y) if the time difference is no greater than *swin*. Such an extension could generate many combinations that turn out to be rejected after invoking another round of validating *mingap* and/or *maxgap*. The structure of the id list also needs to be expanded to indicate the time stamps of previous elements to enable the counting of validating *mingap*.

Like most studies on sequential pattern mining (Agrawal 1995; Ayres 2002; Han 2000; Lin 1998, 2002; Pei 2001; Zaki 2001), synthetic datasets were used and were generated using the procedure described in Srikant (1996) for these experiments. The transaction IDs were used to represent the transaction times. Table 3 shows the meaning and the values of the parameters used in the experiments. A dataset generated with $|C| = 10$, $|T| = 2.5$, $|S| = 4$, $|I| = 1.25$ is denoted by C*10*-T*2.5*-S*4*-I*1.25*.

Table 3. Parameters used in the experiments

Parameter	Description	Value
$\|DB\|$	Number of data sequences in database *DB*	100K, 200K, 400K, 800K, 1000K
$\|C\|$	Average size (number of transactions) per customer	10, 15
$\|T\|$	Average size (number of items) per transaction	2.5, 5
$\|S\|$	Average size of potentially sequential patterns	4, 8
$\|I\|$	Average size of potentially frequent itemsets	1.25, 2.5

5.1. Execution times of *GSP* and *DELISP* algorithms

First, we report the results on dataset C*10*-T*2.5*-S*4*-I*1.25* having 100,000 sequences. The execution times of *GSP* and *DELISP* in mining time-constrained sequential patterns are compared. In these experiments, *DELISP* is about 3 times faster than *GSP*. Various values of *minsup*, *mingap*, *maxgap* and *swin* are used. Note that the mining of sequential patterns without time constraints is a special case with $mingap = 0$, $maxgap = \infty$ and $swin = 0$ here. The results of varying *minsup* (2%, 1.5%, 1%, 0.75%, 0.5%) are consistent. We set the *minsup* to 0.75% and focus on the comparisons of varying time constraints in the following.

The result of varying *mingap* is shown in Fig. 6. As *mingap* increases, the number of qualified patterns existing in data sequences decreases, and thereby the total execution time decreases. The total execution time of *GSP* is 2.8 ($mingap = 0$), up to 3.3 ($mingap = 8$) times than that of *DELISP*. It shows that *DELISP* removes more inaccessible elements with larger *mingap*.

Figure 7 shows the result of varying *maxgap*. The number of time-constrained sequential patterns will decrease when the *maxgap* value increases because larger *maxgap* restricts more data sequences to contain certain patterns. In Fig. 7, the line depicting the execution time of *GSP* starts to fall steeply at $maxgap = 4$ because the sample sequences have 4 transactions ($|S| = 4$) on average. Note that *GSP* runs slightly faster without constraints (673 seconds) than with $maxgap = 12$ because most checks eventually are useless and introduce overheads. *DELISP* consistently outperforms *GSP*, from 2.9 ($maxgap = 12$) down to 1.4 ($maxgap = 1$) times, in the experiments.

Next, the *swin* was varied from 0 up to 4. The *swin* allows adjoining transactions to combine either way to form an element so that each data sequence may

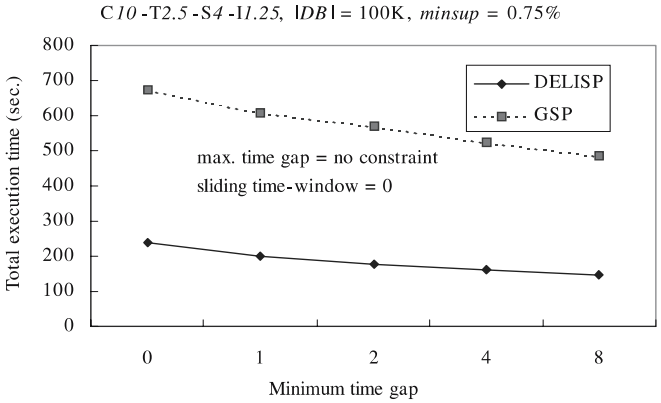


Fig. 6. Effect of the *mingap* constraint

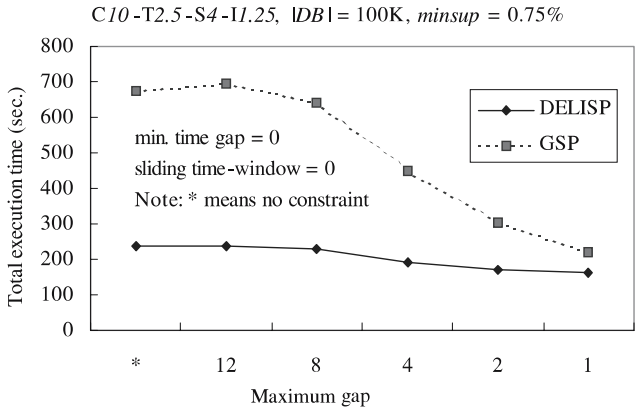


Fig. 7. Effect of the *maxgap* constraint

contain more patterns. Consequently, more execution time is required with the increased *swin*. When *swin* = 0, it took *GSP* 673 seconds and *DELISP* 238 seconds or the discovery. To mine the additional patterns that appeared with *swin* = 1, *GSP* spent 815 seconds and *DELISP* spent 272 seconds. Figure 8 displays the effect on performance when constraint *swin* is increased. Both algorithms scale up with the increased *swin*; *DELISP* performs better.

To evaluate the performance with respect to datasets of different characteristics, the series of experiments were applied on dataset C***15***-T2.5-S4-I1.25 (varying *mingap*), C10-T***5***-S4-I1.25 (varying *swin*), C10-T2.5-S***8***-I1.25 (varying *maxgap*) and C10-T2.5-S4-I***2.5*** (varying *mingap*). The results for sensitivity analysis, displayed in Fig. 9, demonstrate that *DELISP* consistently outperforms *GSP* for various data characteristics.

The effects of varying the three constraints on performance are summarized below. With respect to *mingap*, *GSP* effectively prunes the candidates utilizing the antimonotonic property of candidate generation. For instance, if $(a)(b)$ fails to be a candidate due to *mingap*, then $(a)(b)(c)$ cannot be a candidate. *DELISP* utilizes *mingap* to effectively remove the inaccessible items within the pattern-growth frame-

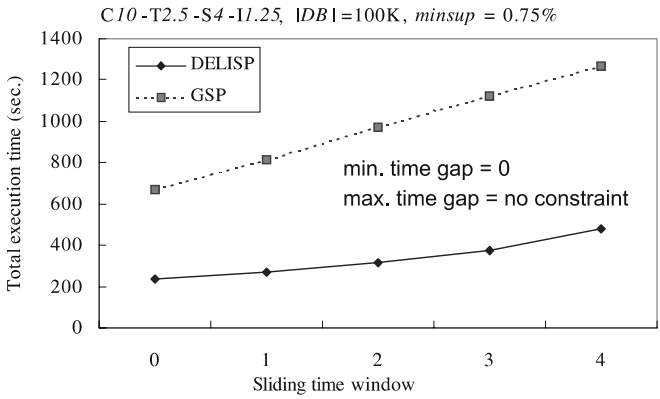


Fig. 8. Effect of the *swin* constraint

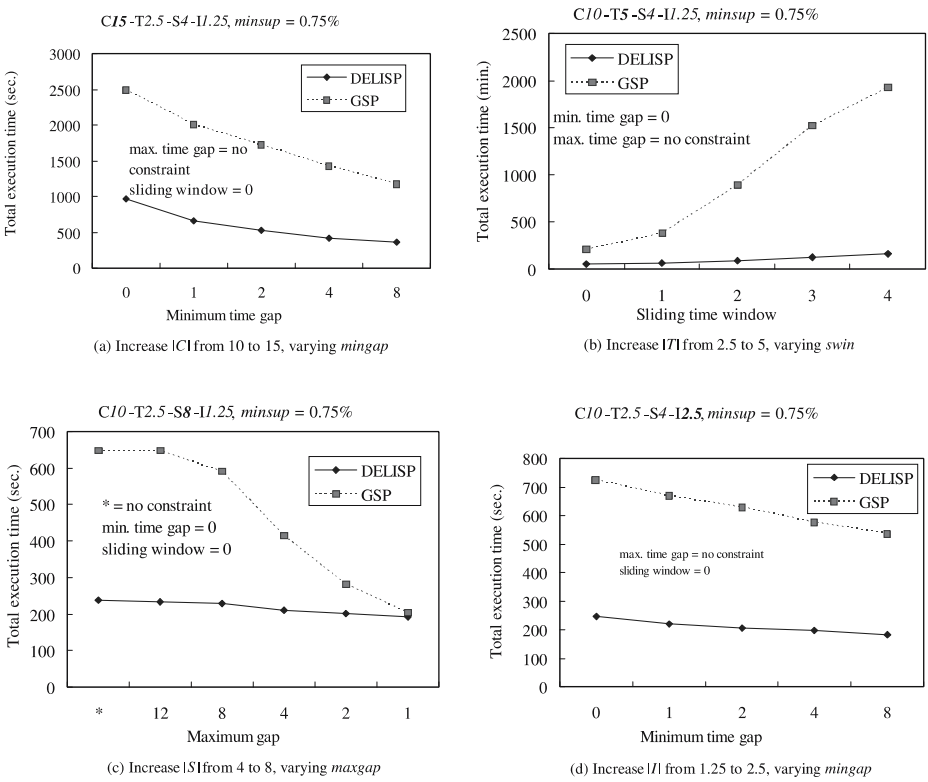


Fig. 9. Total execution time on datasets of various characteristics

work. Both *DELISP* and *GSP* can effectively handle the mining with *mingap*, while *DELISP* is at least two times faster than *GSP*.

In *GSP*, there is performance degradation when *maxgap* or *swin* is specified. The time for the containment test increases when *maxgap* is specified. Besides, the number of candidates increases when *maxgap* is used because we can no longer

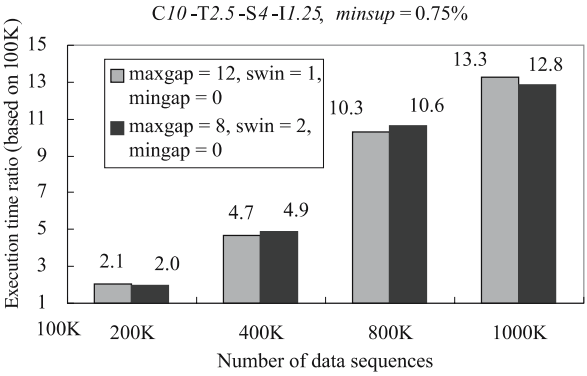


Fig. 10. Linear scalability of *DELISP*

prune noncontiguous subsequences (Srikant 1996). The time for the containment test also increases when *swin* is specified. In addition, the hash tree is less effective in reducing the number of candidates that need to be checked against a data sequence when the user specifies a larger *swin*.

However, *DELISP* effectively handles the three constraints by integrating them in sequence projecting and growing within the pattern-growth framework. Thus, the performance difference between *DELISP* and *GSP* increases when *maxgap* or *swin* increases.

5.2. Scale-up experiments on database size

To justify the scalability of *DELISP*, the number of data sequences was increased from 100 K to 1,000 K with C*10*-T*2.5*-S*4*-I*1.25*. In Fig. 10, the total execution times are normalized with respect to the execution time for $|DB| = 100$ K. When $|DB|$ increases to a very large size, like 800 K or 1,000 K, and the average number of items per transaction might be large, the projected subdatabases increase tremendously, which incurs larger overhead in disk accessing. As indicated in Fig. 10, the execution time ratio scaled up sublinearly. The execution time for $maxgap = 12$ and $swin = 1$ is 271 seconds, and that for $maxgap = 8$, $swin = 2$ is 304 seconds.

6. Discussion

We summarize the factors contributing to the efficiency of *DELISP*, by comparing with *GSP* below.

- **No candidate generation.** *DELISP* generates no candidates and saves the time for not only candidate generation but also candidate testing. Such an advantage is shared by all pattern-growth approaches, like *PrefixSpan* or *prefix-growth*.
- **Focused search.** *DELISP* searches and grows longer patterns in the smaller, promising subspace. Nevertheless, *GSP* takes every data sequence (the entire sequence) for support calculation in each pass.
- **Constraint integration.** *GSP* suffers from *maxgap*, as candidate pruning is less restrictive. For instance, given a *maxgap* constraint, a data sequence that supports candidate $(a)(e)(f)$ may not contain candidate $(a)(f)$. Nevertheless, *DELISP*

benefits from *maxgap* because some posterior elements of a sequence, once they are inaccessible, need not be considered.

- **Containment checking and sequence shrinking.** In each pass, *GSP* transforms every data sequence into items' transaction-time lists and switches between alternative phases with excess pull up of elements to check whether a data sequence contains a candidate (Srikant 1996). For instance, *GSP*, having found $(a)(b)$ in a data sequence, noticing that adding (c) would violate *maxgap*, has to pull-up (b) and maybe then (a), considering their later occurrences. Without any transformation, at each recursion, *DELISP* shrinks a data sequence by removing nonfrequent items, small items and the inaccessible elements.

DELISP benefits from the properties of pattern-growth approaches for factors like no candidate generation and focused search. However, *DELISP* eliminates the need for switching between forward and backward phases of *GSP* by extending concurrently all valid occurrences of the pattern used for projection. In addition, *DELISP* preserves the property of growing longer patterns from prefixes (i.e. avoiding the bidirectional growth) by extending pattern elements according to lexicographic order. These core techniques are specific to *DELISP* and result in the efficient discovery of time-constrained sequential patterns.

7. Conclusions

We have presented the *DELISP* algorithm to provide the full functionality of the classic *GSP* algorithm in terms of time constraints. The conducted experiments confirm that, with good scalability, *DELISP* outperforms *GSP*.

However, pattern-growth-based algorithms usually require the intermediate storage for the projected subdatabases while mining. Future improvements may include optimizations on disk projection. It is also interesting to extend the approach to deal with other time constraints, like overall time span (Pei 2002b; Zaki 2000) and various constraints (Garofalakis 1999; Mannila 1997; Pei 2002b; Zaki 2000) for effective and efficient sequential pattern mining.

Acknowledgements. The authors thank the reviewers for their valuable suggestions and comments.

References

Agrawal R, Srikant R (1995) Mining sequential patterns. In: Proceedings of the 11th international conference on data engineering. Taipei, Taiwan, pp 3–14

Ayres J, Gehrke JE, Yiu T, et al (2002) Sequential pattern mining using bitmaps. In: Proceedings of the 8th ACM SIGKDD international conference on knowledge discovery and data mining. Alberta, Canada

Bettini C, Wang XS, Jajodia S (1998) Mining temporal relationships with multiple granularities in time sequences. Data Eng Bull 21:32–38

Garofalakis MN, Rastogi R, Shim K (1999) SPIRIT: Sequential pattern mining with regular expression constraints. In: Proceedings of the 25th international conference on very large data bases. Edinburgh, Scotland, pp 223–234

Guralnik V, Garg N, Karypis G (2001) Parallel tree projection algorithm for sequence mining. In: Proceedings of the 7th international Euro-par conference on parallel processing, pp 310–320

Han J, Pei J, Mortazavi-Asl B, et al (2000) FreeSpan: Frequent pattern-projected sequential pattern mining. In: Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining, pp 355–359

Lin MY, Lee SY (1998) Incremental update on sequential patterns in large databases. In: Proceedings of 10th IEEE international conference on tools with artificial intelligence. Taipei, Taiwan, pp 24–31

Lin MY, Lee SY (2002) Fast discovery of sequential patterns by memory indexing. In: Proceedings of the 4th international conference on data warehousing and knowledge discovery (DaWaK02). Aix-en-Provence, France, pp 150–160

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

Masseglia F, Cathala F, Poncelet P (1998) The PSP approach for mining sequential patterns. In: Proceedings of the 2nd European symposium on principles of data mining and knowledge discovery, vol 1510. Nantes, France, pp 176–184

Oates T, Schmill MD, Jensen D, et al (1997) A family of algorithms for finding temporal structure in data. In: Proceedings of the 6th international workshop on AI and statistics. Fort Lauderdale, Florida, pp 371–378

Pei J, Han J (2002) Constrained frequent pattern mining: A pattern-growth view. SIGKDD Explor 4:31–39

Pei J, Han J, Pinto H, et al (2001) PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceedings of 2001 international conference on data engineering, pp 215–224

Pei J, Han J, Wang W (2002) Mining sequential patterns with constraints in large databases. In: Proceedings of the 11th international conference on information and knowledge management

Pinto H, Han J, Pei J, et al (2001) Multi-dimensional sequential pattern mining. In: Proceedings of the 10th international conference on information and knowledge management, pp 81–88

Roddick JF, Spiliopoulou M (2002) A survey of temporal knowledge discovery paradigms and methods. IEEE Trans Knowl Data Eng 14:750–767

Rolland P (2001) FlExPat: Flexible extraction of sequential patterns. In: Proceedings of the IEEE international conference on data mining 2001, pp 481–488

Shintani T, Kitsuregawa M (1998) Mining algorithms for sequential patterns in parallel: Hash based approach. In: Proceedings of the 2nd Pacific–Asia conference on knowledge discovery and data mining, pp 283–294

Srikant R, Agrawal R (1996) Mining sequential patterns: Generalizations and performance improvements. In: Proceedings of the 5th international conference on extending database technology. Avignon, France, pp 3–17 (an extended version is the IBM Research Report RJ 9994)

Tsoukatos I, Gunopulos D (2001) Efficient mining of spatiotemporal patterns. In: Proceedings of the 7th international symposium of advances in spatial and temporal databases, pp 425–442

Wang K (1997) Discovering patterns from large and dynamic sequential data. J Intell Inf Syst 9:33–56

Zaki MJ (2001) SPADE: An efficient algorithm for mining frequent sequences. Mach Learn J 42:31-60

Zaki MZ (2000) Sequence mining in categorical domains: Incorporating constraints. In: Proceedings of the 9th international conference on information and knowledge management. Washington, DC, pp 422–429

Correspondence and offprint requests to: Dr. Ming-Yen Lin, Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. Email: linmy@fcu.edu.tw