An Apriori-based Approach for First-Order Temporal Pattern Mining

Sandra de Amo[1],Daniel A. Furtado[2], Arnaud Giacometti[2], Dominique Laurent[3]

[1] Universidade Federal de Uberlândia, Computer Science Department - Uberlândia - Brazil
deamo@ufu.br, danielfurtados@yahoo.com.br
[2] LI-Université de Tours, UFR de Sciences - Blois - France
giaco@univ-tours.fr
[3] ETIS-CNRS-ENSEA-Université de Cergy Pontoise - Cergy Pontoise - France
dominique.laurent@dept-info.u-cergy.fr

Abstract. Previous studies on mining sequential patterns have focused on temporal patterns specified by some form of *propositional* temporal logic. However, there are some interesting sequential patterns whose specification needs a more expressive formalism, the *first-order* temporal logic. In this article, we focus on the problem of mining *multi-sequential patterns* which are first-order temporal patterns (not expressible in propositional temporal logic). We propose two Apriori-based algorithms to perform this mining task. The first one, the PM (Projection Miner) Algorithm adapts the key idea of the classical GSP algorithm for propositional sequential pattern mining by projecting the first-order pattern in two propositional components during the candidate generation and pruning phases. The second algorithm, the SM (Simultaneous Miner) Algorithm, executes the candidate generation and pruning phases without decomposing the pattern, that is, the mining process, in some extent, does not reduce itself to its propositional counterpart. Our extensive experiments shows that SM scales up far better than PM.

Categories and Subject Descriptors: Information Systems [**Miscellaneous**]: Databases

Keywords: Temporal Data Mining, Sequence Mining, Sequential Patterns, Temporal Data Mining, Frequent Patterns, Knowledge Discovery

1. INTRODUCTION

The problem of discovering sequential patterns in temporal data has been extensively studied in several recent papers ([Agrawal and Srikant 1995; 1996; Zaki 2001; Han et al. 2000; Pinto et al. 2001]) and its importance is fully justified by the great number of potential application domains where mining sequential patterns appears as a crucial issue, such as financial market (evolution of stock market shares quotations), retailing (evolution of clients purchases), medicine (evolution of patients symptoms), local weather forecast, telecommunication (sequences of alarms output by network switches), etc. Different kinds of temporal patterns have been proposed ([Mannila et al. 1997; Bettini et al. 1996; Das et al. 1998; Lu et al. 2000]), as well as general formalisms and algorithms for expressing and mining them have been developed ([Joshi et al. 1999; Padmanabhan and Tuzhilin 1996; Berger and Tuzhilin 1999]). Most of these patterns are specified by formalisms which are, in some extent, reducible to Propositional Temporal Logic. For instance, let us consider the (propositional) sequential pattern of form $\langle i_1, i_2, \ldots, i_n \rangle$ (where i_j, $j \in \{1, ..., n\}$, are sets of items) that have been extensively studied in the literature in the past years ([Agrawal and Srikant 1995; 1996; Zaki 2001; Han et al. 2000]). This pattern is considered *frequent* if at least $\alpha\%$ of clients (α is the minimum support specified by the user) buy the items of i_j sequentially, i.e., the items of i_1 are bought at time t_1, the items of i_2 are bought at time t_2, and so on, with $t_1 < t_2 < ... < t_n$. Denoting by p_k^j $(k = 1, ..., n_j)$ the propositional

variables representing the items in i_j, this pattern can be expressed in Propositional Temporal Logic by the formula $i_1 \wedge \Diamond(i_2 \wedge \Diamond(i_3 \wedge (\ldots \wedge \Diamond i_n)...))$, where i_j is the formula $(p_1^j \wedge p_2^j \wedge \ldots p_{n_j}^j)$ and $\Diamond$ is the temporal operator *sometimes in the future*.

Recent research in Inductive Logic Programming (ILP) ([Masson and Jacquenet 2002; Lee and Raedt 2002]) proposed temporal formalisms which are more expressive than the ones used so far for modeling sequential patterns. Indeed, usual formalisms based on propositional temporal logic are not expressive enough to specify many frequent patterns, like Unix-users logs, for instance [Jacobs and Blockeel 2001]. Undoubtedly, the need for more expressive formalisms to specify temporal patterns arise naturally, as well as methods for mining them in a large amount of data.

In this article, we propose a new kind of temporal pattern, called *multi-sequential pattern* (or *msp*), which is not expressible in Propositional Temporal Logic. Such patterns appear in several application domains like financial market, retailing, and roughly speaking aim at representing the behavior of individuals related to each other by some criteria, throughout time. The following situations are examples of multi-sequential patterns: (a) the quotations of shares x, y belonging to the same holding company frequently present the following behavior: an increase of n points of x is followed by an increase of m points of y and an ulterior decrease of k points of y; (b) clients x and y working in the same place usually present the following purchasing behavior: when client x buys a computer M, some time later his(her) colleague y buys the same computer M, then client x buys a printer P, which is followed by y purchasing the same printer P. Contrarily to the (propositional) sequential patterns studied so far ([Agrawal and Srikant 1995; 1996; Zaki 2001; Han et al. 2000]), multi-sequential patterns are not expressible in Propositional Temporal Logic and need the expressive power of First-Order Temporal Logic for their specification. For instance, the pattern expressing the behaviour of stock market shares belonging to a same holding company can be specified by the following first-order temporal formula $(\exists x_1 \exists x_2)(group(x_1, x_2) \wedge up(x_1, n) \wedge \Diamond(up(x_2, m) \wedge \Diamond down(x_1, k)))$.

Besides proposing a new kind of sequential pattern that needs a more expressive formalism than the usual Propositional Temporal Logic, we also propose two algorithms for mining them: *Projection Miner* (PM) and *Simultaneous Miner* (SM). The first one is based on a technique that decomposes the multi-sequential patterns in two propositional sequential patterns during the candidate generation and pruning phases. So, the mining process can be achieved by adapting the classical GSP algorithm for (propositional) sequential pattern mining. In the second algorithm, the mining technique is carried out without decomposing the pattern. We also present an extensive set of experimental results which allow us to conclude that SM executes three times faster than PM. In our opinion, the main contribution of this article is to show that a classical Apriori-like technique based on *candidate generation, pruning and validation* can be used to mine first-order temporal patterns, and that a *pure* first-order technique (SM) produces better results than a technique (PM) adapted from classical methods for propositional sequential pattern mining. It is important to emphasize that both algorithms are not based on temporal logic concepts. Actually, temporal logic is used just as a formalism for expressing the temporal patterns.

The article is organized as follows. In Section 2, we formalize the multi-sequential discovery problem, using a more *operational* formalism to represent them, i.e., a bi-dimensional array representation. In Section 3, we present the first algorithm PM. In Section 4, we present the algorithm SM, where the mining process does not involve decomposing the array in two *linear* sequential components. In Section 5 we analyse the experimental results carried out over synthetic datasets. Finally, in Section 6 we discuss ongoing and further research.

2. PROBLEM FORMALIZATION

In this section, we formalize the problem of discovering multi-sequential patterns. We suppose the existence of a finite set $\mathcal{I}$ of *items* and a finite set $\mathcal{C}$ of object identifiers. From now on, we will call

the elements of $\mathcal{C}$ *clients*[1]. Items are denoted by $a, b, c, ...$ and client ids by $c_1, c_2, c_3, ...$ For the sake of simplifying the presentation, we just consider sequence patterns whose elements are *items* instead of *itemsets*. Indeed, since the main goal of this article is to investigate the problem of mining *sets of sequences*, we believe that considering sequences of itemsets could add unnecessary difficulty to the problem we are interested in.

Let us consider a database schema $\mathbf{D} = \{Tr(T, IdCl, Item, IdG)\}$, and a dataset D as an instance of $\mathbf{D}$. Here, T is the time attribute whose domain (denoted dom(T)) is $\mathbb{N}$. Attributes $IdCl$, $Item$ and IdG stand for client identifiers, items and group identifiers respectively. Their domain are $\mathcal{C}$, $\mathcal{I}$ and $\mathbb{N}$ respectively. The table shown in Figure 1(a) is a dataset.

T	$IdCl$	$Item$	IdG	T	$IdCl$	$Item$	IdG
1	c_1	a	1	1	c_7	a	3
3	c_1	b	1	3	c_7	c	3
5	c_1	a	1	4	c_7	b	3
2	c_2	b	1	2	c_8	b	3
3	c_2	c	1	5	c_8	a	3
4	c_3	a	1	2	c_9	b	4
3	c_4	a	2	4	c_9	a	4
5	c_5	a	2	1	c_{10}	a	4
4	c_6	b	2	3	c_{10}	b	4
				5	c_{11}	d	4

(a)

IdG	$MSeq$
1	$\{\langle a, \perp, b, \perp, a\rangle,$ $\langle \perp, b, c, \perp, \perp\rangle,$ $\langle \perp, \perp, \perp, a, \perp\rangle\}$
2	$\{\langle \perp, \perp, a, \perp, \perp\rangle,$ $\langle \perp, \perp, \perp, \perp, a\rangle,$ $\langle \perp, \perp, \perp, b, \perp\rangle\}$
3	$\{\langle a, \perp, c, b, \perp\rangle,$ $\langle \perp, b, \perp, \perp, a\rangle\}$
4	$\{\langle \perp, b, \perp, a, \perp\rangle,$ $\langle a, \perp, b, \perp, \perp\rangle,$ $\langle \perp, \perp, \perp, \perp, d\rangle\}$

(b)

Fig. 1: Dataset

The notions of *sequence* and *multi-sequence* in our approach are defined as follows.

***Definition* 2.1** A *sequence* is a list $s = \langle i_1, \ldots, i_k\rangle$, where every element i_j is in $\mathcal{I} \cup \{\perp\}$. The symbol $\perp$ stands for "*don't care*", and k is called the *length* of s, denoted by $\mid s \mid$.

A *multi-sequence* is a finite set $\sigma = \{s^1, \ldots, s^n\}$, where every s^i is a sequence, and for all $i, j \in \{1, \ldots, n\}$ we have $\mid s^i \mid = \mid s^j \mid = k$. k is called the *length* of σ and is denoted $l(\sigma)$. The j-th component of sequence s^i is denoted s^i_j.

We notice that each sequence s^i of σ is associated to a client of the group.

A dataset can be easily transformed into a table of pairs (g, m) where g is a group identifier and m a multi-sequence. Figure 1 (b) illustrates this transformation for the dataset D of Figure 1(a). If (g, m) is in the transformed dataset then we denote m by $S(g)$.

***Definition* 2.2** A *multi-sequential pattern* (or *msp* for short) is a multi-sequence satisfying the following conditions:

(1) For every j in $\{1, \ldots, k\}$ there exists i in $\{1, \ldots, n\}$ such that s^i_j is in $\mathcal{I}$ and for all $l \neq i$, $s^l_j = \perp$.

(2) For every i in $\{1, \ldots, n\}$ there exists j in $\{1, \ldots, k\}$ such that $s^i_j \neq \perp$.

The cardinality of σ is called the *rank* of σ and is denoted by $r(\sigma)$.

[1]Depending on the application, *items* can be interpreted as articles in a supermarket, stock market *up(n)* and *down(p)*, etc, and *clients* can be interpreted as clients, stock market shares, etc.

Multi-sequences can be represented by a bi-dimensional array where rows are related to clients and columns (bottom-up ordered) are related to time. For *msps*[2], the conditions (1) and (2) above are interpreted in the array representation as follows: (1) enforces that for each row, there exists a unique position containing an item and all the other positions contain the element $\perp$. Intuitively, this condition means that at each time we focus only on one client purchases. (2) means that for each column there exists at least one position containing an item. The following example illustrates this definition.

***Example* 2.1** Let us consider the five arrays depicted below. The array (a) represents the multi-sequential pattern $\sigma = \{\langle a, \perp\rangle, \langle \perp, c\rangle\}$ whose length and rank are both 2. It is clear that the two conditions above are satisfied. The same happens for the array (e), whose length is 4 and rank is 3. Arrays (b), (c) and (d) do not represent multi-sequential patterns. Indeed, in array (b), column 2 contains no items (condition (2) is not satisfied); in array (c), row 1 contains two items (condition (1) is not satisfied); and in array (d), row 2 contains no items (condition (1) is not satisfied).

$$\begin{pmatrix} \perp & c \\ a & \perp \end{pmatrix} \quad \begin{pmatrix} c & \perp \\ b & \perp \\ a & \perp \end{pmatrix} \quad \begin{pmatrix} \perp & \perp & a \\ b & c & \perp \end{pmatrix} \quad \begin{pmatrix} b \\ \perp \\ a \end{pmatrix} \quad \begin{pmatrix} \perp & \perp & d \\ \perp & c & \perp \\ b & \perp & \perp \\ a & \perp & \perp \end{pmatrix}$$

(a) (b) (c) (d) (e)

It is clear that if σ is an *msp*, then $r(\sigma) \leq l(\sigma)$. We denote by Σ_k^n the set of *msps* of rank n and length k.

In order to compare multi-sequences, we define an order relation between elements $x, y \in \mathcal{I} \cup \{\perp\}$ as follows: $x \preceq y$ if $x = y$ or $x = \perp$.

***Definition* 2.3** Let $\sigma = \{s^1, ..., s^m\}$ and $\tau = \{t^1, \ldots, t^n\}$ be two multi-sequences. We say that σ is a *sub-multi-sequence* of τ (or σ is *included in* τ), denoted by $\sigma \subseteq \tau$, if there exist $j_1, ..., j_m$ in $\{1, ..., n\}$ and $i_1, ..., i_k$ such that:

—$j_p \neq j_q$ for $p \neq q$
—$k = l(\sigma)$ and $i_1 < ... < i_k$
—for all $p \in \{1, ..., m\}$ and $q \in \{1, ..., k\}$, $s_q^p \preceq t_{i_q}^{j_p}$.

If we consider the array representations of σ and τ, $\sigma \subseteq \tau$ means that σ can be obtained by considering columns $j_1, ..., j_m$ and rows $i_1 < ... < i_k$ in τ. So, if $\sigma \subseteq \tau$ then $l(\sigma) \leq l(\tau)$ and $r(\sigma) \leq r(\tau)$.

***Example* 2.2** Let us consider the multi-sequences σ and τ depicted below.

$$\sigma = \begin{pmatrix} \perp & c \\ a & \perp \end{pmatrix} \qquad \tau = \begin{pmatrix} f & e & c \\ \perp & b & \perp \\ a & d & d \end{pmatrix}$$

It is clear that $\sigma \subseteq \tau$ because if we consider rows 1 and 3 and columns 1 and 3 in τ, we obtain an *msp* θ (with $l(\theta) = r(\theta) = 2$) such that $\sigma_j^i \preceq \theta_j^i$ for each $i = 1, 2$ and $j = 1, 2$.

Moreover, it should be noticed that σ is an *msp*, whereas τ is not. We emphasize in this respect that the inclusion relation $\subseteq$ is defined over multi-sequences in general, and not only over *msps*.

[2]We use the notation *msps* to indicate two or more multi-sequences patterns

Frequent *msp*s are defined as follows.

***Definition* 2.4** Let D be a dataset in its transformed version and $(g, S(g)) \in D$. We say that g *supports* an *msp* σ if $\sigma \subseteq S(g)$. The *support* of an *msp* σ is defined by: $sup(\sigma) = \frac{|\{g | g \in \Pi_{IdG} D \text{ and } g \text{ supports } \sigma\}|}{|D|}$.

An *msp* σ is said to be *frequent* if $\sup(\sigma) \geq \alpha$, where α is a given minimum support threshold.

For instance, if we consider the dataset D of Figure 1, the *msp* of Example 2.1(a) is supported only by group 1. So, its support is 0.25, and for $\alpha = 0.2$, this *msp* is frequent. As shown in the following proposition, *msp* frequency is an anti-monotonic property.

Proposition 2.1 *Let σ and τ be msps and α a minimum support threshold. If $\sigma \subseteq \tau$ and $sup(\tau) \geq \alpha$ then $sup(\sigma) \geq \alpha$.*

Problem Formulation. The problem we are interested in can be described as follows: Given a dataset D and a minimum support threshold α, find all *msp*s that are frequent with respect to D and α.

3. ALGORITHM PM

In this section we describe the algorithm *PM* (Projection Miner) for mining *msp*s. The notations C_k^n and L_k^n are used to denote the set of *candidate msp*s and *frequent msp*s of length k and rank n ($n \leq k$) respectively.

First of all, we show how an *msp* can be completely characterized by two simple (propositional) sequences. Let σ be an ordered *msp* with $l(\sigma) = k$ and $r(\sigma) = n$. The *characteristic function* of σ is the function $f_\sigma : \{1, ..., k\} \rightarrow \{1, ..., n\}$ such that $f_\sigma(i)$ is the number corresponding to the (unique) column having an element of $\mathcal{I}$ in line i.

Now, we associate to σ two sequences of length k, denoted by $\Pi_t(\sigma)$ (the *item-sequence* of σ) and $\Pi_s(\sigma)$ (the *shape-sequence*), as follows: $\Pi_t(\sigma)$ is the projection of σ over the time axis and $\Pi_s(\sigma) = \langle f_\sigma(1), ..., f_\sigma(k) \rangle$.

For instance, if σ is the *msp* illustrated in case (e) of Example 2.1, then $\Pi_t(\sigma) = \langle a, b, c, d \rangle$ and $\Pi_s(\sigma) = \langle 1, 1, 2, 3 \rangle$.

It is easy to verify that an *msp* is completely characterized by its shape-sequence and item-sequence. The numbers k and n are called the *length* and *rank* of the shape-sequence $\Pi_s(\sigma)$ respectively. We denote by $\mathbf{proj}(\sigma)$ the pair $(\Pi_t(\sigma), \Pi_s(\sigma))$. The inverse function is denoted by $\boxtimes$, that is $\Pi_t(\sigma) \boxtimes \Pi_s(\sigma) = \sigma$.

In algorithm PM, the set L_{k+1}^n is obtained from L_k^n and L_k^{n-1}. In order to do so, the *msp*s in L_k^n are projected in two sequence components: a *shape-sequence* and an *item-sequence*. At (sub)-iteration $k+1$ of iteration n, the set of candidate shape-sequences and item-sequences of rank n and length $k+1$ (denoted CI_{k+1}^n and CS_{k+1}^n, respectively) are generated respectively from the sets of frequent shape-sequences LS_k^n, LS_k^{n-1} and frequent item-sequences LI_k^n, LI_k^{n-1}, obtained in previous iterations.

Figure 2 illustrates how the set L_{k+1}^n is obtained from L_k^n and L_k^{n-1}.

The algorithm PM is given in Figure 3. At steps **3.5** and **3.9.6**, the operator $\boxtimes$ is used to join the item-sequences and shape-sequences. In this process, each shape-sequence in CS_k^n is joined with all item-sequences of CI_k^n. The resulting set of *msp*s is C_k^n (also denoted by $CI_k^n \boxtimes CS_k^n$). The $Join_I$ operation joins the item-sequences in $LI_k^n \cup LI_k^{n-1}$ and returns the set CI_{k+1}^n of candidate item-sequences. Analogously, the $Join_S$ operation yields the set of shape-candidates CS_{k+1}^n by joining

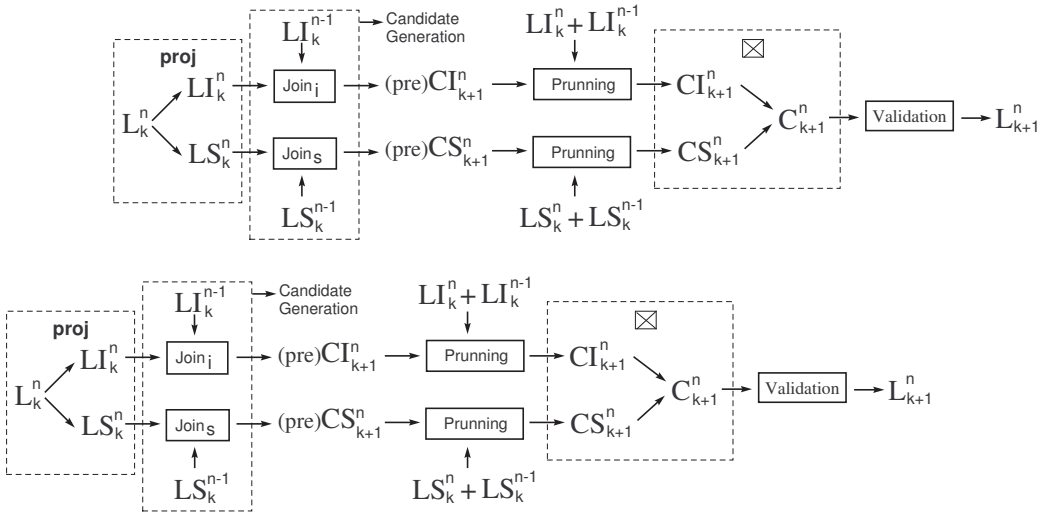


Fig. 2: L^n_{k+1} is obtained from L^n_k and L^{n-1}_k

shape-sequences in $LS^n_k \cup LS^{n-1}_k$. The details of these join operations are given in section 3.1 below. The function **proj** computes the projections Π_i and Π_s for all *msp*s in L^n_k and returns the sets LI^n_k and LS^n_k.

3.1 Candidate Generation

The $Join_I$ operation between candidate item-sequences (CI^n_k) is computed as in [Agrawal and Srikant 1996]. Concerning the generation of the candidate shape-sequences, there are three possibilities for computing the result of the $Join_S$ operation. Figure 4 illustrates each of these cases. We explain below the three cases.

Let $\sigma = \langle \sigma_1, \sigma_2, ..., \sigma_k \rangle$ and $\tau = \langle \tau_1, \tau_2, ..., \tau_k \rangle$ be two shape-sequences of rank n_σ and n_τ respectively. Let denote by σ^{first} and τ^{last} the sequences obtained by eliminating the first element of σ and the last element of τ respectively. If $\sigma^{first} = \tau^{last}$ then σ and τ are joinable. In this case (Case 1), the resulting sequence is $\langle \sigma_1, \sigma_2, ..., \sigma_k, \tau_k \rangle$. Otherwise, we test if $\tau^{last}_i + 1 = \sigma^{last}_i$ for all $i = 1, ..., k$. If this is verified (Case 2a) then σ and τ are joinable and one resulting sequence is $\langle \sigma_1, \ldots, \sigma_k, \tau_k + 1 \rangle$. We next test (Case 2b) if $\mathrm{Inc}(\tau)^{last}_i = \sigma^{last}_i$ for all $i = 1, ..., k$, where $\mathrm{Inc}(\tau)_i = \tau_i + 1$ if $\tau_i + 1 < max(n_\sigma, n_\tau)$ and $\mathrm{Inc}(\tau)_i = 1$ otherwise. If this is verified then a second resulting sequence is also obtained, defined as $\langle \sigma_1, \ldots, \sigma_k, \mathrm{Inc}(\tau)_k \rangle$.

The following example illustrates the process of the candidate generation in PM.

***Example* 3.1** Let us suppose that L^2_3 and L^1_3 contain the *msp*s σ and τ given below:

$$\sigma = \begin{pmatrix} \perp & c \\ \perp & b \\ a & \perp \end{pmatrix} \text{ and } \tau = \begin{pmatrix} d \\ c \\ b \end{pmatrix} \qquad \gamma = \begin{pmatrix} \perp & d \\ \perp & c \\ \perp & b \\ a & \perp \end{pmatrix}$$

We join σ and τ in order to obtain the candidate *msp* γ of rank 2 and length 4. We have $\Pi_i(\sigma) = \langle a, b, c \rangle \in LI^2_3$, $\Pi_s(\sigma) = \langle 1, 2, 2 \rangle \in LS^2_3$, $\Pi_i(\tau) = \langle b, c, d \rangle \in LI^1_3$ and $\Pi_s(\tau) = \langle 1, 1, 1 \rangle \in LS^1_3$. By joining the item-sequences $\langle a, b, c \rangle$ and $\langle b, c, d \rangle$ we obtain the item-sequence $\langle a, b, c, d \rangle \in LI^2_4$. By joining the

Input: α: minimum support, N: number of data multi-sequences, D: dataset
1. $k = 0$; $n = 1$;
2. Repeat
2.1 $k = k + 1$;
2.2 L_k^1 = frequent sequences of rank 1 and length k (uses GSP);
2.3 if $L_k^1 \neq \emptyset$ **then** { $LI_k^1 = L_k^1$; $LS_k^1 = \{\langle 1, 1, \ldots, 1\rangle\}$ };
Until $L_k^1 = \emptyset$;
3. Repeat
3.1 $n = n + 1$; $k = n$;
3.2 $CI_n^n = Join_I(LI_{n-1}^{n-1}, LI_{n-1}^{n-1})$; (Candidate item-sequences generated from LI_{n-1}^{n-1})
3.3 $CS_n^n = \{\langle 1, 2, ..., n\rangle\}$
3.4 delete all $\sigma \in CI_n^n$ such that $\exists \tau \subseteq \sigma$, $l(\tau) = n - 1$ and $\tau \notin LI_{n-1}^{n-1}$ (pruning);
3.5 $C_n^n = CI_n^n \boxtimes CS_n^n$ (build the candidate *msp*s)
3.6 Foreach group g in D **do**
Increment the count of all candidate *msp*s in C_n^n that are contained in $S(g)$;
3.7 L_n^n = candidates in C_n^n with count $\geq \alpha N$;
3.8 $(LI_n^n, LS_n^n) = \mathbf{proj}(L_n^n)$; (projects item-sequences and shape-sequences of L_n^n into LI_n^n and LS_n^n);
3.9 Repeat
3.9.1 $k = k + 1$;
3.9.2 $CI_k^n = Join_I(LI_{k-1}^{n-1} \cup LI_{k-1}^n, LI_{k-1}^{n-1} \cup LI_{k-1}^n)$;
(Candidate item-sequences generated from LI_{k-1}^{n-1} and LI_{k-1}^n)
3.9.3 delete all $\sigma \in CI_k^n$ such that $\exists \tau \subseteq \sigma$, $l(\tau) = k - 1$ and $\tau \notin LI_{k-1}^{n-1} \cup LI_{k-1}^n$;
(pruning item-sequences)
3.9.4 $CS_k^n = Join_S(LS_{k-1}^{n-1} \cup LS_{k-1}^n, LS_{k-1}^{n-1} \cup LS_{k-1}^n)$;
(Candidate shape-sequences generated from $LS_{k-1}^{n-1} \cup LS_{k-1}^n$)
3.9.5 delete all $\sigma \in CS_k^n$ such that $\exists \tau \subseteq \sigma$ and $\tau \notin LS_{k-1}^{n-1} \cup LS_{k-1}^n$;
(pruning shape-sequences)
3.9.6 $C_k^n = CI_k^n \boxtimes CS_k^n$
3.9.7 Foreach group g in D **do**
Increment the count of all candidate *msp*s in C_k^n that are contained in $S(g)$;
3.9.8 L_k^n = candidates in C_k^n with count $\geq \alpha N$;
3.9.9 $(LI_k^n, LS_k^n) = \mathbf{proj}(L_k^n)$; (projects item-sequences and shape-sequences of L_k^n)
Until $L_k^n = \emptyset$;
Until $L_n^n = \emptyset$;

Fig. 3: Algorithm PM

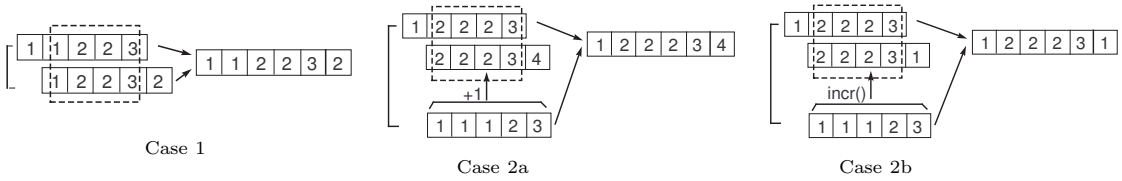


Fig. 4: Joining *shape-sequences*

shape-sequences $\langle 1, 2, 2\rangle$ and $\langle 1, 1, 1\rangle$ (Case 2a), we obtain the shape-sequence $\langle 1, 2, 2, 2\rangle \in LS_4^2$. We notice that Case 2b does not apply here. The sequences $\langle a, b, c, d\rangle$ and $\langle 1, 2, 2, 2\rangle$ uniquely characterize the *msp* $\gamma \in C_4^2$.

The following theorem, whose proof can be found in Appendix A.2, states that, at each iteration step, the frequent *msp*s are among the generated candidate *msp*s.

Theorem 3.1 *For all n, k, $1 \leq n \leq k$, we have $L_k^n \subseteq C_k^n$.*

Regarding support counting, we note that the technique used in the validation phase is the same as the one used in Algorithm SM. We refer to the end of the next section for details.

4. ALGORITHM SM

In this section, we present the algorithm *SM* (Simultaneous Miner) for mining *msp*s. We remind that C_k^n and L_k^n denote respectively the sets of *candidate msp*s and *frequent msp*s of length k and rank n, where $n \leq k$.

The general structure of the algorithm is that it generates first the frequent *msp*s of rank 1 ($L_1^1, L_2^1, L_3^1, \ldots$), using an algorithm for mining (simple) sequential patterns (for instance, the algorithm GSP ([Agrawal and Srikant 1996])). Then, for each rank n, the algorithm generates iteratively the sets C_k^n of *candidate msp*s of length $k \geq n$. For the initial case $k = n$, C_n^n is generated from L_{n-1}^{n-1}, which has been generated in the previous step corresponding to rank $n-1$. For the case $k > n$, the set C_k^n (containing the candidate *msp*s of length k and rank n) is generated from L_{k-1}^{n-1} and L_{k-1}^n which have been generated in previous steps. The supports for these candidate *msp*s are computed through a pass over the dataset. At the end of the pass, the set L_k^n (the candidate *msp*s which are actually frequent) is computed. These *msp*s become the seed for the next pass $k+1$. In Figure 5(a), we illustrate how the set L_k^n is obtained from previous steps and in Figure 5(b), we illustrate the whole mining process.

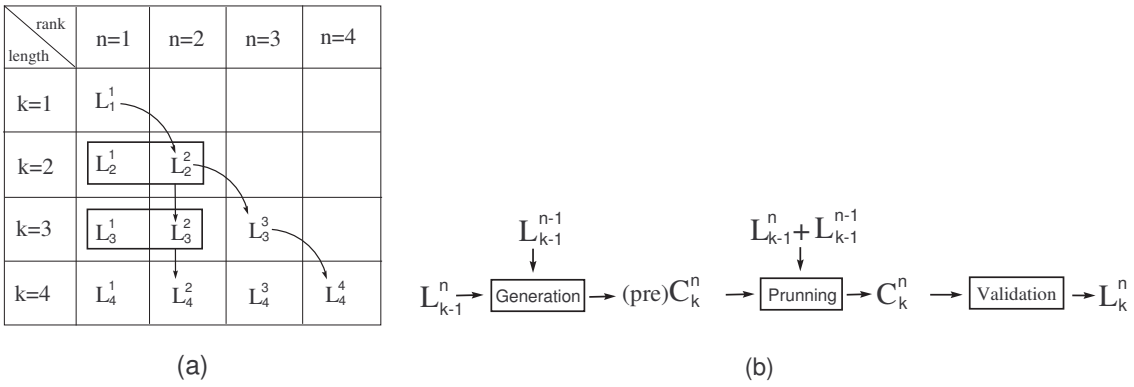


Fig. 5: L_k^n is obtained from L_{k-1}^{n-1} and L_{k-1}^n

The algorithm SM is given in Figure 6. In the pass where frequent *msp*s of length k and rank n are generated, the algorithm first generates the set C_k^n, the candidate *msp*s (steps **3.2** and **3.6.2**). After that, the *msp*s that contain *msp*s not in L_{k-1}^{n-1} and L_{k-1}^n are pruned from C_k^n. This is so because, according to Proposition 2.1, these *msp*s have no chance to be frequent (steps **3.3**, **3.6.3** and **3.6.4**). After the pruning phase, the supports of the candidates are computed through a pass over the dataset (steps **3.4** and **3.6.5**). In paragraph 4.1, we show in detail how candidate *msp*s in C_k^n are built from the sets L_{k-1}^{n-1} and L_{k-1}^n that have been calculated in previous steps.

4.1 Candidate Generation

Now, we show how to generate potentially frequent *msp*s in Σ_k^n (recall that Σ_k^n denotes the set of *msp*s of rank n and length k). In this construction, we suppose that the columns of each *msp* $\sigma \in \Sigma_k^n$ are ordered by an ordering which is naturally induced by the ordering over time. For instance, in Figure 7(e) we have a non ordered *msp* (left) and the same *msp* where columns have been ordered (right).

We use the following notation in the sequel: if $\sigma \in \Sigma_k^n$ then $\overline{\sigma}$ and $\underline{\sigma}$ denote the multi-sequences (not necessarily *msp*s) obtained by deleting, respectively, row n and row 1 from σ.

Input: (α: minimum support, N: number of data multi-sequences, D: dataset)

1. $n = 1; k = 0;$
2. Repeat
2.1 $k = k + 1;$
2.2 L_k^1 = frequent sequences of rank 1 and length k (uses GSP)
Until $L_k^1 = \emptyset;$
3. Repeat
3.1 $n = n + 1; \quad k = n;$
3.2 $C_n^n = L_{n-1}^{n-1} \bowtie L_{n-1}^{n-1}$ *(New candidate msp's of rank n and length n are generated from* L_{n-1}^{n-1}*)*
(*For details on the operator* $\bowtie$*, see Section 4.1*);
3.3 delete all candidates $\sigma \in C_n^n$ such that $\exists \tau \subseteq \sigma$, $\tau \in \Sigma_{n-1}^{n-1}$ and $\tau \notin L_{n-1}^{n-1}$ (pruning);
3.4 Foreach group g in D **do**
Increment the count of all candidate *msp's* in C_n^n that are contained in $S(g)$;
3.5 L_n^n = candidates in C_n^n with count $\geq \alpha N$;
3.6 Repeat
3.6.1 $k = k + 1;$
3.6.2 $C_k^n = (L_{k-1}^{n-1} \bowtie L_{k-1}^{n-1}) \cup (L_{k-1}^n \bowtie L_{k-1}^n) \cup (L_{k-1}^{n-1} \bowtie L_{k-1}^n) \cup (L_{k-1}^n \bowtie L_{k-1}^{n-1})$
(New candidate msp's of rank n and length k are generated from L_{k-1}^{n-1} *and* L_{k-1}^n *See Section 4.1 for details);*
3.6.3 delete all candidates $\sigma \in C_k^n$ such that $\exists \tau \subseteq \sigma$, $\tau \in \Sigma_{k-1}^{n-1}$ and $\tau \notin L_{k-1}^{n-1}$;
(pruning 1)
3.6.4 delete all candidates $\sigma \in C_k^n$ such that $\exists \tau \subseteq \sigma$, $\tau \in \Sigma_{k-1}^n$ and $\tau \notin L_{k-1}^n$;
(pruning 2)
3.6.5 Foreach group g in D **do**
Increment the count of all candidate *msp's* in C_k^n that are contained in $S(g)$;
3.6.6 L_k^n = candidates in C_k^n with count $\geq \alpha N$;
Until $L_k^n = \emptyset;$
Until $L_n^n = \emptyset$

Fig. 6: Algorithm SM

As we can see in line **3.6.2** of Algorithm SM, there are four possibilities to obtain a candidate in C_k^n:

(1) by joining two *msp*s of L_{k-1}^{n-1},
(2) by joining two *msp*s of L_{k-1}^n,
(3) by joining an *msp* of L_{k-1}^{n-1} with an *msp* of L_{k-1}^n, and
(4) by joining an *msp* of L_{k-1}^n with an *msp* of L_{k-1}^{n-1}.

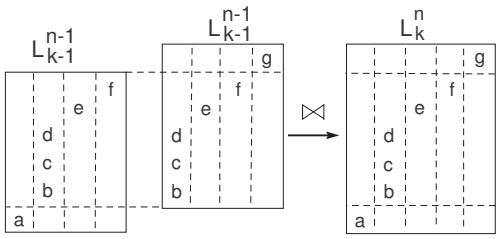
Figure 7 gives the conditions for two *msp*s being joinable (below each figure) as well as the result of the join operation, in the four cases.

The following theorem, whose proof can be found in Appendix A.1, guarantees that: (1) all frequent *msp*s of Σ_k^n are included in C_k^n and (2) C_k^n is minimal, in the sense that it contains only the potentially frequent *msp*s of Σ_k^n:

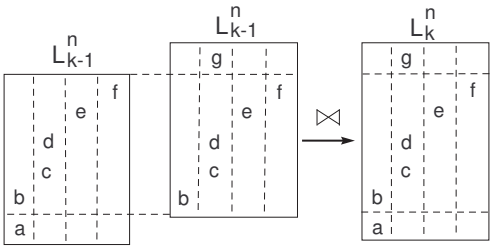
Theorem 4.1 *For all* n, k, $1 \leq n \leq k$, *we have: (1)* $L_k^n \subseteq C_k^n$. *(2) If* $\sigma \in \Sigma_k^n$ *and there exists* $\tau \subseteq \sigma$, $\tau \neq \sigma$, *such that* τ *is not frequent, then* $\sigma \notin C_k^n$.

4.2 Support Counting

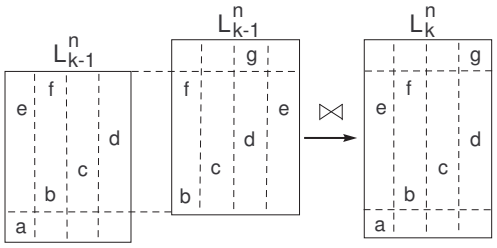
In order to reduce I/O operations over the dataset and compute the support of each *msp* in C_k^n by executing only one pass over the dataset, we find all candidates σ which are included in $S(g)$, for each data multi-sequence $S(g)$. In order to reduce the number of candidates which have to be tested for inclusion for each data multi-sequence $S(g)$, we use a technique similar to the one described in



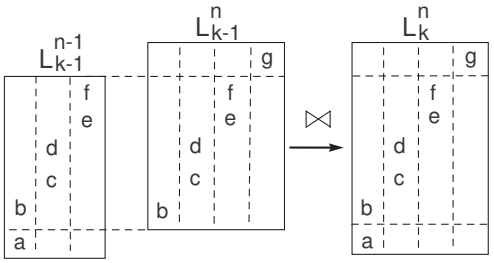
Case 1: if we eliminate the first column of $\underline{\sigma}$ and last column of $\overline{\tau}$ then the resulting *msp*s are identical.



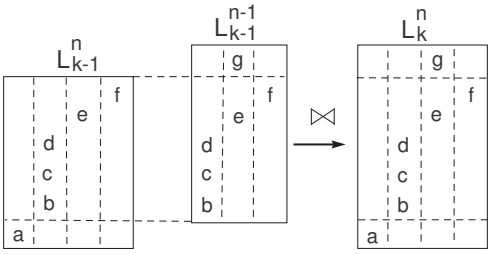
Case 2a: $\underline{\sigma}$ and $\overline{\tau}$ are identical.



Case 2b: $\underline{\sigma}$ and $\overline{\tau}$ are identical after shifting the colomns of $\overline{\tau}$ one place to the right.



Case 3: $\underline{\sigma}$ and $\overline{\tau}$ are identical and the last column of $\overline{\tau}$ oonly contain the symbol $\perp$.



Case 4: The first column of $\underline{\sigma}$ only contains the symbol $\perp$ and $\underline{\sigma}$ is identical to $\overline{\tau}$ after shifting $\underline{\sigma}$ one place to the left.

$$\begin{pmatrix} d & \perp & \perp \\ \perp & c & \perp \\ \perp & \perp & b \\ \perp & \perp & a \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} \perp & \perp & d \\ \perp & c & \perp \\ b & \perp & \perp \\ a & \perp & \perp \end{pmatrix}$$

(e) An *msp* and its ordered version

Fig. 7: Joining *msp*s

[Agrawal and Srikant 1995] for counting the support of sequences, by storing the set of candidate *msp*s C_k^n in a *set* of *hash-trees*. We omit these technical details here.

5. EXPERIMENTAL RESULTS

To evaluate the performance of the algorithms SM and PM, we ran several experiments using synthetic datasets. Our experiments have been run on a Pentium 4 of 2.4 GHz with 1GB of main memory and running Windows XP Professional.

5.1 Synthetic Data

We have developed a synthetic data generator using the idea described in [Agrawal and Srikant 1995] for the synthetic data-sequences generator. Our generator produces datasets of multi-sequences in accordance with the input parameters as shown in Table 1.

We have generated datasets by setting $N = 3000$ and $N_m = 1000$. The average rank of potentially

frequent multi-sequences ($|R|$) was set to 3 and the average length of potentially frequent multi-sequences ($|S|$) was set to 4. Table 2 summarizes the dataset parameter settings. The dataset D4-G4-C6, for instance, keeps 4000 groups with average number of clients per group equal to 4 and the average number of transactions per client equal to 6.

Table 1: Parameters used in the Synthetic Data Generator

$\|D\|$	Number of groups (size of dataset) - in '000s
$\|G\|$	Average number of customers per group
$\|C\|$	Average number of transactions per customer
$\|R\|$	Average rank of potentially frequent *msp's*
$\|S\|$	Average length of potentially frequent *msp's*
N	Number of items
N_m	Number of maximal potentially frequent *msp's*

Table 2: Synthetic Datasets - Parameter Settings

Name	$\|D\|$	$\|G\|$	$\|C\|$	Size MB
D2-G4-C3	2	4	3	1.06
D2-G4-C6	2	4	6	2.09
D2-G6-C3	2	6	3	1.57
D4-G4-C6	4	4	6	4.19

5.2 Performance Analysis

Figure 8 shows the execution times of algorithms SM and PM for the four datasets given in Table 2 as the minimum support decreases from 1% to 0.25%. As expected, the execution times for both algorithms increase. PM performs worse than SM for low support levels, mainly because more patterns of large ranks are generated in the generation phase of PM. This set is more refined in SM algorithm because the generation and pruning of the candidates *msp*s are achieved without decomposing them into the shape-sequences and item-sequences.

Both algorithms present a similar performance for high support levels. The reason is that, in this case, the generated patterns have small rank (usually less than 4) and most of the execution time is spent during the first iteration, which is the same for both algorithms.

5.3 Scale-up

We present some results regarding scale-up experiments for PM and SM. Other scale-up experiments have also been performed and similar results were obtained. Figure 9 shows how SM and PM scale up as the number of data multi-sequences ($|D|$) increases from 1000 to 5000. We show the results for the datasets Dx-G3-C3 ($x = 1, \ldots, 5$) and a minimum support set to 0,15%. The figure shows that SM scales more linearly than PM with respect to the number of data multi-sequences.

Figure 10 shows how both algorithms scale up as the number of clients per group ($|G|$) increases from 3 to 8. The datasets used were D2-Gx-C3 ($x = 3, \ldots, 8$), for which the minimum support was set to 0,25%.

We can see that the execution time for PM intensely increases as the number of clients per group increases, contrarily to SM algorithm, whose execution time increases smoothly. This behaviour is explained by the fact that PM generates more candidates than SM at each pass. On the other hand, the execution time of support calculation increases with respect to the number of clients in each group. So, it is expected that the performance of PM will be worse, since the number of candidates tested by PM at each pass is greater than the one tested by SM.

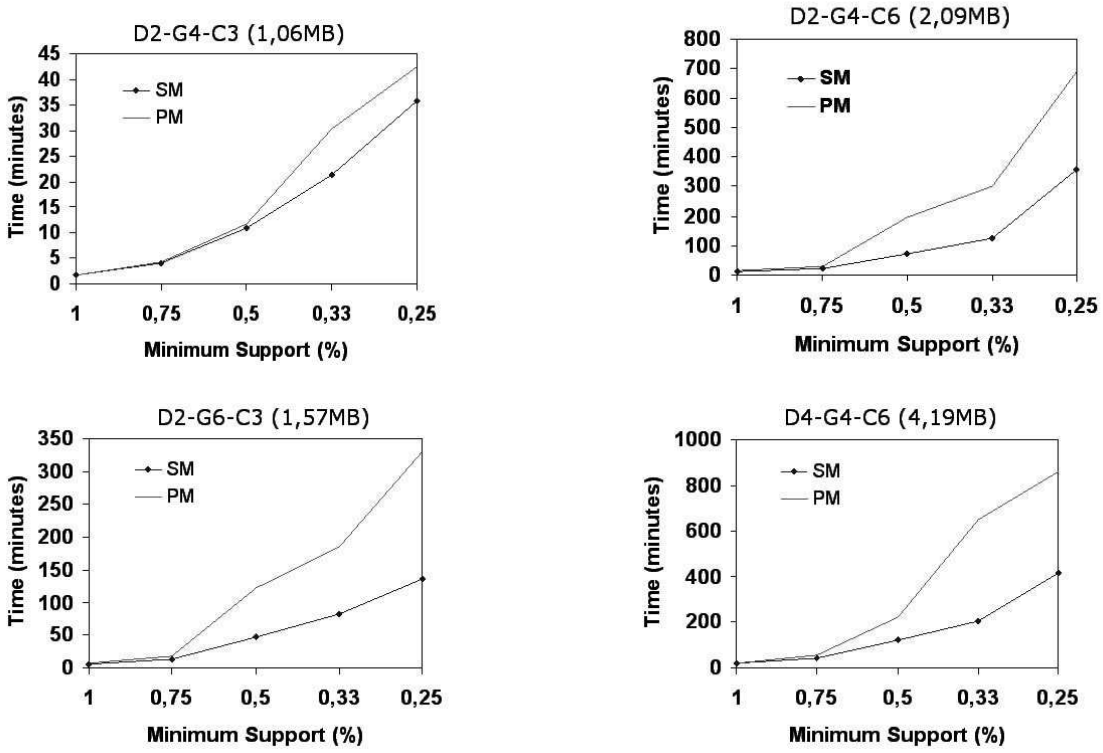


Fig. 8: Execution times: Synthetic Data

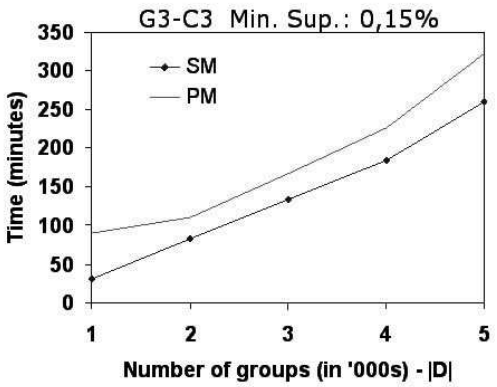


Fig. 9: Scale-up w.r.t to data multi-sequences

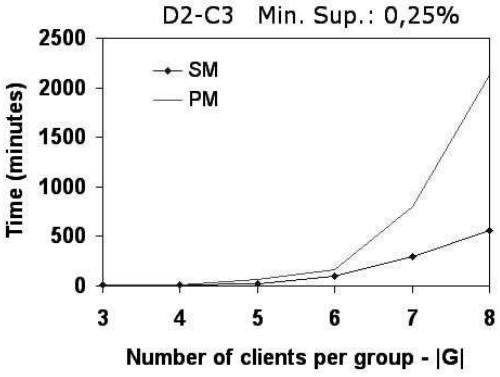


Fig. 10: Scale-up w.r.t. clients per group

6. ONGOING AND FURTHER RESEARCH

The two algorithms proposed in this article are designed to produce *all* frequent multi-sequential patterns, regardless to users' specific interests. At the present time, we are investigating constraint-based methods for restricting the candidate search space. Often, users require richer mechanisms for specifying patterns of interest, rather than the simple mechanism provided by minimum support.

Concerning the classical problem of mining sequential patterns ([Agrawal and Srikant 1995]), one of the most flexible tools enabling user-controlled focus to be incorporated into the pattern mining process has been proposed in [Garofalakis et al. 1999]: besides the dataset, the user proposes a regular

expression as input, which aims at capturing the shape of patterns he/she is interested in discovering. The automaton associated to this regular expression is incorporated into the mining process that outputs the sequential patterns exceeding a minimum support threshold and which are accepted by the automaton. We are investigating the introduction of regular expression restrictions over *msp*s and the development of algorithms to mine *msp*s satisfying such restrictions.

Another direction for future research concerns performance comparison between our Apriori-based method SM and methods for first-order sequential pattern mining based on Inductive Logic Programming. We intend also to validate our mining algorithms over real datasets.

REFERENCES

Agrawal, R. and Srikant, R. Mining Sequential Patterns. In *Proceedings of the International Conference on Data Engineering*. Taipei, Taiwan, pp. 3–14, 1995.

Agrawal, R. and Srikant, R. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the Fifth Int. Conference on Extending Database Technology*. Avignon, France, pp. 3–17, 1996.

Berger, G. and Tuzhilin, A. Discovering unexpected patterns in temporal data using temporal logic. *Information Systems Working Papers Series*, 1999.

Bettini, C., Wang, X. S., and Jajodia, S. Testing complex temporal relationships involving multiple granularities and its application to data mining (extended abstract). In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Montreal, Canada, pp. 68–78, 1996.

Das, G., ip Lin, K., Mannila, H., Renganathan, G., and Smyth, P. Rule discovery from time series. In *Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining*. AAAI Press, New York City, USA, pp. 16–22, 1998.

Garofalakis, M. N., Rastogi, R., and Shim, K. Spirit: Sequential pattern mining with regular expression constraints. In *Proceedings of the International Conference on Very Large Databases*. Edinburgh, Scotland, pp. 223–234, 1999.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M.-C. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Boston, USA, pp. 355–359, 2000.

Jacobs, N. and Blockeel, H. From shell logs to shell scripts. In *Proceedings of the International Conference on Inductive Logic Programming*. Strasbourg, France, pp. 80–90, 2001.

Joshi, M., Karypis, G., and Kumar, V. A universal formulation of sequential patterns. Tech. rep., Technical Report 99-021, Department of Computer Science and Engineering, University of Minnesota, USA, 1999.

Lee, S. D. and Raedt, L. D. Constraint Based Mining of First Order Sequences in SeqLog. In *Proceedings of the Workshop on Multi-Relational Data Mining*. ACM SIGKDD, Alberta, Canada, 2002.

Lu, H., Feng, L., and Han, J. Beyond intratransaction association analysis: mining multidimensional intertransaction association rules. *ACM Transactions on Information Systems* 18 (4): 423–454, 2000.

Mannila, H., Toivonen, H., and Verkamo, A. I. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery* 1 (3): 259–289, 1997.

Masson, C. and Jacquenet, F. Mining frequent logical sequences with spirit-log. In *Proceedings of the International Conference on Inductive Logic Programming*. Sydney, Australia, pp. 166–181, 2002.

Padmanabhan, B. and Tuzhilin, A. Pattern discovery in temporal databases: A temporal logic approach. In *Proceedings of the Second International Conference on Knowledge Discovery in Databases*. Portland, USA, pp. 351–354, 1996.

Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., and Dayal, U. Multi-dimensional sequential pattern mining. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*. Atlanta, USA, pp. 81–88, 2001.

Zaki, M. J. Spade: an efficient algorithm for mining frequent sequences. *Machine Learning Journal* vol. 42, pp. 31–60, 2001.

A. PROOFS

We first give the proof of Theorem 4.1, and then, the proof of Theorem 3.1 is provided, based on that of Theorem 4.1.

A.1 Proof of Theorem 4.1

(1) Let $\sigma \in L_k^n$. We suppose the columns of σ are ordered (See Figure 7 (e)). For $n = 1$, the result is verified, because the candidate generation procedure used in algorithm Apriori-All is correct, i.e., the set of candidates always contains all frequent sequences. For $n = k = 2$, the result is also verified because, as $\sigma \in L_2^2$, the *msps* $\{\langle\sigma_1^1\rangle\}$ and $\{\langle\sigma_2^2\rangle\}$ are in L_1^1. So, by construction of C_2^2, it is clear that $L_2^2 \subseteq C_2^2$. Let n, k be such that $2 \leq n \leq k$ and $k \geq 3$. We have the following cases to consider:

—The (unique) item of line 2 is placed in column 1 and
  (a) there exists an item in column n placed in a line i, with $i < k$: in this case, $\sigma \in L_k^n \bowtie_k^n L_k^n$,
  (b) column n contains only one item, which is placed in line k: in this case, $\sigma \in L_{k-1}^{n-1} \bowtie_k^n L_k^n$.

—The (unique) item of line 2 is placed in column 2 and
  (a) there exists an item in column n placed in a line i, with $i < k$: in this case, $\sigma \in L_k^n \bowtie_k^n L_{k-1}^{n-1}$,
  (b) column n contains only one item, which is placed in line k: in this case, $\sigma \in L_{k-1}^{n-1} \bowtie_k^n L_{k-1}^{n-1}$.

(2) Since $\tau \subseteq \sigma$ and $\tau \neq \sigma$, $\tau \in \Sigma_p^m$ with $m < n$ or $p < k$. Then:

—Let $m < n$ and $m \leq p \leq k$. In this case, τ is obtained by deleting at least one column of σ and one of the lines corresponding to the items in this column. So $p < k$. We can suppose, without loss of generality, that only one column has been deleted, i.e. $m = n - 1$. We have that τ is contained in an *msp* $\tau' \in \Sigma_{k-1}^{n-1}$, such that $\tau' \subseteq \sigma$. Since τ is not frequent, then τ' is not frequent as well, i.e., $\tau' \notin L_{k-1}^{n-1}$. Then, by construction of the *msps* in C_k^n, $\sigma \notin C_k^n$.

—Let $m = n$ and $n \leq p < k$. In this case, τ is obtained by deleting at least one line of σ. We can suppose, without loss of generality, that only one line has been deleted. Thus, $\tau \in \Sigma_{k-1}^n$. Since τ is not frequent, i.e. $\tau \notin L_{k-1}^n$, by construction of the *msps* in C_k^n, $\sigma \notin C_k^n$. □

A.2 Proof of Theorem 3.1

The proof follows from the proof of Theorem 4.1 just above. This is so because the $Join_S$ operation between shape-sequences is defined in such a way that:

$$\begin{aligned}\Pi_s((L_{k-1}^{n-1} \bowtie L_{k-1}^{n-1}) \cup (L_{k-1}^{n-1} \bowtie L_{k-1}^n) \cup (L_{k-1}^n \bowtie L_{k-1}^n) \cup (L_{k-1}^n \bowtie L_{k-1}^n)) = \\ Join_S(\Pi_s(L_{k-1}^{n-1}), \Pi_s(L_{k-1}^{n-1})) \cup Join_S(\Pi_s(L_{k-1}^{n-1}), \Pi_s(L_{k-1}^n)) \cup \\ Join_S(\Pi_s(L_{k-1}^n), \Pi_s(L_{k-1}^{n-1})) \cup Join_S(\Pi_s(L_{k-1}^n), \Pi_s(L_{k-1}^n)).\end{aligned}$$ □