Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 62 (2007) 401–420
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

First-order temporal pattern mining with regular expression constraints

Sandra de Amo *, Daniel A. Furtado [1]

Computer Science Department, Faculdade de Computacao, Universidade Federal de Uberlândia, Av. Joao Naves de Avila, 2121-Bloco, B, Sala 1B71-Campus Santa Monica, 38400 902 Uberlândia, Brazil

Received 12 August 2006; accepted 12 August 2006
Available online 2 October 2006

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, such as the *multi-sequential patterns*, whose specification needs a more expressive formalism, the first-order temporal logic. Multi-sequential patterns appear in different application contexts, for instance in spatial census data mining, which is the target application of the study developed in this paper. We extend a well-known user-controlled tool, based on regular expressions constraints, to the multi-sequential pattern context. This specification tool enables the incorporation of user focus into the mining process. We present MSP-Miner, an Apriori-based algorithm to discover all frequent multi-sequential patterns satisfying a user-specified regular expression constraint.

Keywords: Temporal data mining; Frequent sequential patterns; Regular expression constraints; Constraint-based mining

1. Introduction and motivation

The problem of discovering sequential patterns in temporal data have been extensively studied in several recent papers [1–3] 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 sequential patterns and more general temporal patterns [4–8] have been proposed, as well as general formalisms and algorithms for expressing and mining these patterns have been developed [9,10]. Most of these patterns are specified by formalisms which are, in some extent, reducible to Propositional Temporal Logic. For instance, let us consider a classical sequential pattern of the form $s = \langle\{a,b\},\{c,d\}\rangle$ (where $\{a,b\}$ and $\{c,d\}$ are set of items) which has been extensively studied in the past years [1–3]. This pattern is considered

* Corresponding author.
E-mail addresses: deamo@ufu.br (S. de Amo), daniel@facom.ufu.br (D.A. Furtado).
[1] Supported by CNPq, Brazil.

0169-023X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2006.08.009

frequent if there are a percentage α of clients (α is the minimum support specified by the user) who buy the set of items $\{a,b\}$ and $\{c,d\}$ sequentially, i.e., the items a,b are bought at time t_1 and the items c,d are bought at time $t_2 > t_1$. The sequential pattern s can be expressed as a temporal relational calculus query as following:

$$\{x | buy(x,a) \wedge buy(x,b) \wedge \Diamond(buy(x,c) \wedge buy(x,d))\}$$

where the symbol $\Diamond$ stands for the known temporal logic operator "sometimes in the futur".

Apparently, this is a first-order temporal formula. However, (1) because the only variable appearing in the formulas expressing the sequential patterns is the *free variable* x (x represents a *client who buys items*) and, as a *free variable*, constitutes the object which is counted during the support counting phase,[2] and (2) because the only predicate is *buy*, this formula can equally be expressed by a propositional temporal formula. Indeed, let us suppose propositional symbols P_i, one for each item i, whose meaning is "client x buys item i". Notice that different symbols i correspond to different propositional symbols P_i. The propositional formula below expresses our sequential pattern:

$$P_a \wedge P_b \wedge \Diamond(P_c \wedge P_d) \tag{1}$$

1.1. First-order sequential patterns

Recent research in ILP (Inductive Logic Programming) [11–13] proposed temporal formalisms which are more expressive than the propositional logic formalism used so far for modelling sequential patterns. Indeed, propositional temporal logic is not expressive enough to specify many useful and interesting patterns. Let us suppose, for instance, that we have a database storing logs of unix-user commands (see [14] for details). Examples of sequential patterns which could be discovered in such a database are:

$$\langle vi(paper.tex), latex(paper.tex), dvips(paper.dvi), lpr(paper.ps)\rangle \tag{2}$$

$$\langle vi(X), latex(X)\rangle \tag{3}$$

These patterns can be expressed by the following first-order temporal formulas, respectively:

$$vi(paper.tex) \wedge \Diamond(latex(paper.tex) \wedge \Diamond(dvips(paper.dvi) \wedge \Diamond lpr(paper.ps))) \tag{4}$$

$$\exists X(vi(X) \wedge \Diamond latex(X)) \tag{5}$$

Notice that in formulas (4) and (5), constants (paper.tex, paper.dvi, paper.ps) as well as predicate symbols (vi, latex, lpr) can be used as parameters in the mining process. In a propositional pattern (as in (1)), only constants (items) appear in its specification. Moreover, in formula (5), there is a *bounded variable* X. Differently from free variables, bounded variables are relevant in the pattern specification.

Undoubtedly, as shown in these examples, the need for more expressive formalisms to specifying temporal patterns arise naturally, as well as methods for mining them in large amount of data.

In a previous paper [15], we have introduced a new temporal pattern, called *multi-sequential pattern (msp)*. Such patterns appear in several application domains, like financial market, retailing, census data, and in some sort, aim at representing the behaviour of individuals *related to each other* by some criteria, throughout time. Contrarily to the (propositional) sequential patterns studied so far, multi-sequential patterns are not expressible in Propositional Temporal Logic and need the expressive power of First-order Temporal Logic for its specification. The following example adapted from [16] illustrates *msp* semantics.

Example 1. (Temporal census data patterns). In Brazil, a municipality is an administrative local area composed by several districts. Frequently, municipal economic administrators are interested in analyzing census data to discover information relating the behaviour of the population living in different districts, that is, how demographic indicators evolve throughout time and how the neighbour relationship[3] affects this evolution. A typical temporal pattern in this context is "*A low rate of full-time employed male living in district A, followed by a high rate of commercial activities in a neighbour district B is followed by a high rate of part-time*

[2] Free variables, representing the counting objects, are irrelevant in the pattern specification.

[3] Two districts are neighbours if they belong to the same municipality.

employed female living in district A ''. Such a pattern can be expressed by the following first-order temporal relational calculus query:

$$\{M|\exists A\exists B(N(M,A,B) \wedge \Diamond(DI(A,FTM^{-}) \wedge \Diamond(DI(B,CA^{+}) \wedge \Diamond DI(A,PTF^{+}))))\} \quad (6)$$

In the formula (6), $N(M,X,Y)$ stands for the relationship ''*A and B are districts of the municipality M*''; *DI* stands for *Demographic Indicator* and constants FTM^{-}, CA^{+} and PTF^{+} stand for *low full-time employed male*, *high commercial activities* and *high part-time employed female*, respectively. Notice that the counting object is the *free variable M*, representing some municipality (a *group* of districts). By ommiting the counting variable *M*, this pattern can also be expressed by the temporal formula:

$$\exists A\exists B(N(A,B) \wedge \Diamond(DI(A,FTM^{-}) \wedge \Diamond(DI(B,CA^{+}) \wedge \Diamond DI(A,PTF^{+}))))$$

One of the main contributions in [15] is that an Apriori-like technique based on *candidate generation, pruning and validation* can be used to mine first-order temporal patterns (*msps*) and that a *pure* first-order technique (SM-Miner) produces better results than a technique adapted from classical methods for propositional sequential pattern mining (PM-Miner).

Preliminary experiments on multi-sequential pattern mining showed an overwhelming volume of candidate patterns, most completely irrevelant to the users, resulting in a great and wasteful computational cost. This naturally led us to consider specification formalisms to allow user focus during the *msp* mining process in order to prevent the generation of uninteresting and useless patterns. In the past years, some work has been dedicated to the problem of introducing user-control features in the mining process, either during a post-processing phase [17,18], either by incorporating user-specified constraints in the mining process. The latter approach has been largely explored in the context of association rules mining [19,20]. In the sequential pattern mining field, a lot of work have been dedicated to investigate constraint-based mining [21,22]. In the inductive programming context, constraint-based mining has recently become a very appealing subject of research (see [23] for a series of surveys on the subject).

In this paper, we focus on incorporating user-specified constraints inside the mining process rather than during post-processing. Our approach follows the idea introduced in [22], where a constraint specification framework based on regular expressions (RE) is developed in the sequential pattern mining context. As it was pointed out in [22], REs provide a simple, ubiquitous and natural syntax for the specification of special classes of sequential patterns. In this paper, we extend this formalism in order to use it as a restriction specification tool in the multi-sequential pattern mining context. The following example illustrates the idea:

Example 2. Let us suppose the situation described in Example 1, referring to temporal census data mining. Let us suppose that the data analyst is interested in discovering patterns involving only two districts and fitting in the following bias: they begin by reporting a low rate of full-time employed female living in the first district and finish by reporting a high rate of part-time employed female living in the first district. This constraint (over multi-sequential patterns) can be expressed by a regular expression of the form $DI(D_1,FTF^{-})X^{*}$ $DI(D_1,PTF^{+})$. Here, X is a sequence of relational atoms of the form $DI(D,A)$, where D is D_1 or D_2, and A is a constant representing parti-cular demographic indicators. In fact, the formalism we will introduce in Section 3 is far more simple and does not involve predicate symbols (like *DI*) nor (bounded) variables (like D_1 and D_2).

Besides extending an RE formalism to the multi-sequential context, we also present the MSP-Miner algorithm to discover all frequent multi-sequential patterns satisfying a user-speci-fied regular expression constraint. MSP-Miner incorporates the RE constraints during the mining process, pushing non-antimonotone conditions (specified by regular expressions) in both the candidate generation and pruning phases. Notice that, as in [22], our pattern constraint does not satisfy the property of *antimonotonicity*. That is, the fact that a pattern satisfies the constraint does not imply that all its subpatterns also satisfy the constraint. Consequently, it is possible that some (or even *all*) candidate patterns contains subpatterns which are not in the set of interesting patterns generated in previous steps. This make the candidate generation and pruning phases a rather complex task.

We have performed an extensive set of experiments to evaluate the performance and scalability of MSP-Miner over synthetic data. Besides, we also have compared the performances of MSP-Miner and a

post-processing method for mining multi-sequential patterns (the SM-Miner algorithm introduced in [15]). Our empirical study establishes the effectiveness of exploiting RE constraints during the mining process rather than during the post-processing phase. In our opinion, the main contribution of this paper is extending a constraint-based approach for mining propositional temporal patterns (simple sequential patterns treated in [22,1,3,2]) to a similar approach to mine first-order temporal patterns.

The paper is organized as follows. In Section 2, we discuss some important work related to the two main subjects treated in this paper: first-order sequential pattern mining and constraint-based mining. In Section 3, we present the main concepts related to multi-sequential pattern mining and introduce the formalism allowing to specify the constraints. In Section 4, we briefly describe the algorithm SM-Miner. In Section 5, we present the details of the MSP-Miner algorithm which incorporates a regular expression in the candidate generation phase in order to constraint the set of candidate patterns. In Section 6, we analyse several experimental results over synthetic data and in Section 7 we conclude the paper, by discussing some future research.

2. Related work

The design of efficient algorithms for mining sequential patterns has been extensively studied in the past decade [24,1,8,3,2]. Recent work on sequential pattern mining has focused on the problem of reducing the amount of discovered patterns by introducing user control mechanisms into the mining process rather than pruning uninteresting patterns only during a post-processing phase. The use of regular expressions as a specification formalism for constraint-based sequential pattern mining has been introduced for the first time in [22]. In this article, a family of four algorithms, called SPIRIT family, has been proposed. Each algorithm in the SPIRIT family has been designed to mine all frequent patterns satisfying a given regular expression constraint. The algorithms use different relaxations of the original regular expression in order to find a reasonable balance between constraint-based pruning and support based pruning.

In [21], another formalism for reducing the candidate search space, based on con-text-free grammars, has been proposed. Indeed, there are situations where a more expressive formalism is need in order to specifying constraints. As in the SPIRIT family, several constraint-based mining algorithms have been proposed in this context, using different relaxations of the given context-free grammar.

In [25], a new framework aiming at generalizing different known constraint formalisms is proposed. This framework is able to express several classes of constraints: *item constraints* (specifying the items which are expected to appear in the sequences), *length constraints* (specifying the number of transaction in the sequences), *super pattern constraints* (find patterns which contains a given set of patterns as subpatterns), *aggregate patterns* (specifying aggregate functions on the items appearing in the sequences), *regular expressions*, *duration constraints* and *gap constraints*. The general framework is based on the concept of *prefix-monotonicity*, a property satisfied by all monotone, anti-monotone and regular expressions constraints.

All the three latter formalisms are restricted to the propositional sequential mining context. In [11], a formalism similar to the SPIRIT family, based on regular expressions, is introduced in the context of first-order sequential pattern mining. Here, a sequential pattern is a sequence of relational atoms, like in $\langle vi(X), latex(X)\rangle$. The algorithm SPIRIT-LoG has been designed to mine first-order sequential patterns satisfying a regular expression. The basic difference between this approach and ours is related to their respective contexts, since our MSP-Miner algorithm has been designed to mine multi-sequential patterns and not simple sequential patterns like in SPIRIT-LoG.

In [13], sequential patterns are sequences of relational atoms like in [11], except that they are separated by two types of symbols: $<$ and $\lhd$. For instance, the sequence $a < b \lhd c$ means b happens sometimes after a, and c happens in the next instant after b. The database is a set of *sequential* clauses (the bodies are sequential patterns). Constraints are specified by a conjunction of a monotone constraint m and an anti-monotone constraint a. These constraints are simply predicates (or properties) over sequential patterns. The levelwise algorithm MineSeqLog is proposed for mining all the sequential patterns satisfying a giving constraint. Notice that in our approach, the constraints specified by regular expressions are not monotone nor anti-monotone. The main difference between this approach and ours concerns the sequential patterns and constraints considered in each one.

Table 1
Municipalities and their districts

Municipality	Districts
Uberlândia (1)	{Martinésia (o_1), Tapuirama (o_2), Miraporanga (o_3)}
Juiz de Fora (2)	{Torreões (o_4), Rosário de Minas (o_5), Sarandina (o_6)}
Ouro Preto (3)	{Lavras Novas (o_7), Amarantina (o_8)}

3. Problem formulation

In this section, we introduce the main concepts related to the problem of discovering multi-sequential patterns satisfying a set of constraints. First, we focus on the concepts of multi-sequences datasets, multi-sequential patterns and frequency. Secondly, we introduce the formalism based on regular expressions to specify constraints on multi-sequential patterns.

3.1. Multi-sequences and multi-sequential patterns

In order to simplify the presentation, we will specify a multi-sequential pattern by means of an array and use this simple specification formalism for multi-sequential patterns throughout the paper instead of a temporal logic formalism. Nevertheless, for the sake of completeness, in the end of this section we also present multi-sequential patterns as a special class of first-order temporal logic formulas.

Let us suppose the existence of a finite set $\mathscr{I}$ of *items* and a finite set $\mathscr{O}$ of *object* identifiers. In our running example, items are interpreted as *demographic indicators*, objects are interpreted as *districts* and groups of objects are interpreted as *municipalities*.[4] Items are denoted by $a, b, c, \ldots$ object ids by $o_1, o_2, o_3, \ldots$ and groups of object ids by $g_1, g_2, g_3, \ldots$

3.1.1. Multi-sequences: the Dataset

Let us consider a database schema $\mathbf{D} = \{Tr(IdG, IdO, Item, T)\}$. A dataset D is an instance over $\mathbf{D}$. Here, T is the time attribute whose domain (denoted by dom(T)) is $\mathbb{N}$. Attributes *IdO*, *Item* and *IdG* stand for object identifiers, items and group identifiers, respectively. Table 3 illustrates a dataset containing temporal census data about three municipalities in the state of Minas Gerais, Brazil: Uberlândia (1), Juiz de Fora (2) and Ouro Preto (3). The districts of each municipality are showed in Table 1. The demographic indicators are showed in Table 2 (we use abbreviations for the names of municipalities, their districts and the demographic indicators).[5]

From now on, in order to simplify the presentation, the domain of the attributes *IdO*, *Item* and *IdG* are $\mathscr{O}$, $\mathscr{I}$ and $\mathbb{N}$, respectively.

The two algorithms we will present in the next sections require that the dataset follow a specific format, as a set of *multi-sequences*. A *sequence* is a list $s = \langle i_1, \ldots, i_k \rangle$, where each element $i_j \in \mathscr{I} \cup \{\perp\}$. The symbol $\perp$ stands for "*don't care*" (k is called the *length* of s and is denoted by $|s|$). A *multi-sequence* is a finite set $\tau = \{s^1, \ldots, s^n\}$, where each s^i is a sequence and for each $i,j \in \{1, \ldots, n\}$ we have $|s^i| = |s^j| = k$ (k is called the *length* of τ and is denoted by $l(\tau)$). The j-th component of sequence s^i is denoted by s^i_j. A dataset can be easily transformed into a table of multi-sequences. For instance, let us consider the Census Dataset of Table 3. The information stored in this dataset ranges from time 1 to time 5. So, the corresponding multi-sequences of D will be constituted of sequences of length 5 (5 instants). Let us consider, for instance, the information corresponding to the municipality of Uberlândia (group **1**). Notice that at instant 1, we have the following demographic indicators for the first district Martinésia (o_1): instant $1 \rightarrow a$ (=High % of full-time employed male), instant $3 \rightarrow d$ (=Low % of full-time employed male), instant $5 \rightarrow e$ (=High % of part-time employed female). So the sequence corresponding to the first district (o_1) of Uberlândia (group **1**) is $\langle a, \perp, d, \perp, e \rangle$. Similarly, we obtain the sequences corresponding to the other districts o_2 and o_3. The multi-sequence representing

[4] The terms *items* and *objects* have different interpretations depending on the application context.

[5] *High* and *low* percentages of each demographic indicators are established by the data analyst by means of maximum and minimum thresholds. We assume that the demographic indicators presented in Table 2 have been categorized according to these thresholds in a pre-processing phase.

Table 2
Demographic indicators

Symbol	Demographic indicator
a	High % of full-time employed male
b	Low % of full-time employed female
c	High % of commercial activities
d	Low % of full-time employed male
e	High % of part-time employed female

Table 3
Census dataset

IdG	*IdO*	*Item*	*T*	*IdG*	*IdO*	*Item*	*T*
1	o_1	*a*	1	**2**	o_4	*a*	3
1	o_1	*d*	3	**2**	o_5	*a*	5
1	o_1	*e*	5	**2**	o_6	*b*	4
1	o_2	*b*	2	**3**	o_7	*d*	1
1	o_2	*c*	3	**3**	o_7	*e*	3
1	o_3	*c*	4	**3**	o_7	*b*	4
				3	o_8	*c*	2
				3	o_8	*a*	5

the evolution of the demographic indicators of the districts of Uberlândia (group 1) is obtained by considering the set of sequences associated to o_1, o_2 and o_3. Table 3 illustrates the transformed dataset.

Multi-sequential patterns. A *multi-sequential pattern* (or *msp* for short) σ is a multi-sequence satisfying the following conditions: (1) for each $j \in \{1,\ldots,k\}$ there exists $i \in \{1,\ldots,n\}$ such that $s_j^i \in \mathscr{I}$ and for all $l \neq i$ we have $s_j^l = \perp$, (2) for each $i \in \{1,\ldots,n\}$ there exists $j \in \{1,\ldots,k\}$ such that $s_j^i \neq \perp$. The cardinality of σ is called the *rank* of σ and is denoted by $r(\sigma)$. Multi-sequences can be represented by a bi-dimensional array where rows are related to time and columns (bottom-up ordered) are related to objects. For *msps*, 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 $\perp$. In our census data example, this condition means that at each time we focus on information about only one district. (2) means that for each column there exists at least one position containing an item.

Remark. We remind that the main focus of this paper is the introduction of a certain kind of temporal pattern which cannot be specified in a propositional logic formalism. In order to avoid increasing the complexity of the presentation unnecessarily and thus, to avoid deviating from our main focus, we have decided to concentrate on events related to one object at a time. This explains the restriction (1) imposed on our multi-sequential patterns. In what concerns restriction (2): a *msp* σ with a column j containing only $\perp$ values (i.e. *don't care* values) would enclose the same amount of knowledge expressed by a simpler *msp* obtained from σ by eliminating its column j.

Example 3. The temporal pattern described in Example 1 is the multi-sequential pattern depicted below, where we have used the following abbreviations for the demographic indicators appearing in the pattern: $d =$ Low % of full-time employed male (FTM^-), $c =$ High % of commercial activities (CA^+), $e =$ High % of part-time employed female (PTF^+).

$$\sigma = \begin{pmatrix} e & \perp \\ \perp & c \\ d & \perp \end{pmatrix}$$

It is important to notice the difference between multi-sequences and multi-sequential patterns. For instance, the multi-sequence corresponding to the municipality 1 in Table 4 is not a multi-sequential pattern. As it can be seen in its array representation depicted below, row 3 contains information related to district 1 and district 2.

Table 4
Census dataset transformed into a set of multi-sequences

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

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

Definition 4. Let $a, b \in \mathscr{I}$. We say that $a \preceq b$, if $a = b$ or $a = \perp$. It is easy to verify that $\preceq$ is a partial order. Let $\sigma = \{s^1,\ldots,s^m\}$ and $\tau = \{t^1,\ldots,t^n\}$ be two multi-sequences. We say that σ is a *submulti-sequence* of τ (or σ is *included in* τ, denoted by $\sigma \subseteq \tau$) if there exist $j_1,\ldots,j_m \in \{1,\ldots,n\}$, $j_p \neq j_q$ for $p \neq q$, and $i_1 < \cdots < i_k$, where $k = l(\sigma)$, such that $s_q^p \preceq t_{i_q}^{j_p}$ for all $p \in \{1,\ldots,m\}$ and $q \in \{1,\ldots,k\}$. If we consider the array representation of σ and τ, this means that σ can be obtained by considering columns $j_1,\ldots,j_m$ and rows $i_1 < \cdots < i_k$ in τ. So, if $\sigma \subseteq \tau$ then $l(\sigma) \leqslant l(\tau)$ and $r(\sigma) \leqslant r(\tau)$. We say that σ is a *submsp* of τ if σ is a *msp* included in τ.

Example 5. Let us consider the multi-sequences σ and τ depicted in Example 3.[6] It is clear that $\sigma \subseteq \tau$ because if we consider rows 1, 2 and 3 and columns 1 and 3 in τ, we obtain a *msp* θ (with $l(\theta) = 3$ and $r(\theta) = 2$) such that $\theta_1^1 = d = \sigma_1^1$, $\theta_2^1 = \sigma_2^1 = \perp$, $\theta_3^1 = \sigma_3^1 = e$, $\theta_1^2 = \sigma_1^2 = \perp$, $\theta_2^2 = \sigma_2^2 = c$, $\theta_3^2 = \sigma_3^2 = \perp$.

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

A *msp* σ is called *frequent* if $sup(\sigma) \geqslant \alpha$, where α is a given minimum support threshold. For instance, if we consider the census dataset D of Table 4 the *msp* of Example 3 is supported by the municipalities of Uberlândia (1) and Ouro Preto (3). So, its support is 0.66. If $\alpha = 0.5$ then this *msp* is frequent.

Indeed, *msp* frequency is an antimonotone property: If σ and τ are *msps* with $\sigma \subseteq \tau$ and α a minimum support threshold such that $sup(\tau) \geqslant \alpha$, then $sup(\sigma) \geqslant \alpha$.

Multi-sequential patterns as a class of first-order temporal logic formulas. Multi-sequential patterns are characterized as a special class of First-order Temporal Logic formulas. Let us consider the first-order signature $\mathbf{D} = \{Tr\}$, where Tr is a predicate of arity 3 (in the database context, Tr corresponds to the relational schema *Tr(IdG, IdO, Item)*). Let $\mathscr{M}(g)$ be the class of existential temporal formulas, with one free variable g, of the following form:

$$\exists x_1 \cdots \exists x_n (F_1 \wedge (\Diamond(F_2 \wedge \Diamond(F_3 \cdots \Diamond F_k) \cdots)))$$

where $k \geqslant n$ and for each $i \in \{1,\ldots,k\}$ there exists $j \in \{1,\ldots,n\}$ and a constant a_i such that $F_i = Tr(g, x_j, a_i)$. We remind that the symbol $\Diamond$ stands for the temporal operator *"sometimes in the future"*. Notice that there is no explicit variable for representing the time. The notion of time is captured by the temporal operator $\Diamond$.

It is easy to see that every multi-sequential pattern can be expressed by a formula in $\mathscr{M}(g)$ and conversely, to each formula in $\mathscr{M}(g)$ corresponds a unique multi-sequential pattern. Notice that: (1) the free variable g represents the *group*, the *counting variable*, (2) the numbers n and k represent the rank and length of the

[6] In this example, σ is a *msp*, but τ is not. We notice that the inclusion relation $\subseteq$ is defined over multi-sequences in general and not only over *msps*.

multi-sequential pattern, respectively. For instance, the multi-sequential pattern given in Example 3 can be expressed by the formula $F \in \mathscr{M}(g)$:

$$F = \exists x_1 \exists x_2 (Tr(g, x_1, d) \wedge \Diamond(Tr(g, x_2, c) \wedge \Diamond(Tr(g, x_1, e))))$$

3.2. Regular expression constraints over multi-sequential patterns

In this section, we show how regular expressions can be used as a formalism for specifying user constraints and we formulate de problem of multi-sequential pattern mining. First of all, we need some extra concepts which are introduced below.

The columns of each *msp* $\sigma \in \Sigma_k^n$ are ordered by an ordering which is naturally induced by the time level ordering. For instance, in Fig. 1 the second *msp* is the ordered version of the first one.

A *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_σ: $\{1, \ldots, k\} \to \{1, \ldots, n\}$ such that $f_\sigma(i)$ is the number corresponding to the (unique) column having an element of $\mathscr{I}$ in line i. 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 $\Pi_s(\sigma) = \langle f_\sigma(1), \ldots, f_\sigma(k) \rangle$. For instance, if σ is the *msp* illustrated in Example 3, then $\Pi_t(\sigma) = \langle d, c, e \rangle$ and $\Pi_s(\sigma) = \langle 1, 2, 1 \rangle$. It is easy to verify that a *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. In our example above, $k = 3$ and $n = 2$. If C is a set of *msps*, we denote by $\Pi_t(C)$ and $\Pi_s(C)$ the sets $\{\Pi_t(\sigma) | \sigma \in C\}$ and $\{\Pi_s(\sigma) | \sigma \in C\}$, respectively. We denote by $(\Pi_t(C), \Pi_s(C))$ the set $\{(\Pi_t(\sigma), \Pi_s(\sigma)) | \sigma \in C\}$.

In what follows, we will often make no difference between the *msp* σ (resp. the set of *msps* C) and the associated pair of sequences $(\Pi_t(\sigma), \Pi_s(\sigma))$ (resp. the set $(\Pi_t(C), \Pi_s(C))$).

It is important to note that the property of antimonotonicity of *msps* propagates to item-sequences and shape-sequences.

Proposition 6. *Let D be a dataset, $0 \leqslant \alpha \leqslant 1$ and σ a msp. If σ is frequent with respect to D and α, then $\Pi_t\sigma$ (resp. $\Pi_s\sigma$) is frequent with respect to D_t and α (resp. D_s and α). Here, D_t (resp. D_s) denotes the set of item-sequences (resp. shape-sequences) obtained by projecting the multi-sequences of D on the time axis (resp. on the shape axis).*

In order to specify special classes of multi-sequential patterns, we can simply consider a pair of sequential pattern constraints, one for item-sequences and one for shape-sequences. The following example illustrates this idea.

Example 7. Let us consider the constraint $DI(D_1, b)X^*DI(D_1, e)$ of Example 2. We remind that in this example the data analyst is interested in patterns involving only two districts. The constraint can be specified by the following pair of regular expressions: $(b(a + b + c + d + e)^*e,\ 1(1 + 2)^*1)$.

Notice that, due to the particular ordering of the sequences in a *msp*, its shape-sequence has a special form: if $j \in \mathbb{N}$ appears for the first time in a position i then no $k > j$ can appear before i. For instance, 12131 is a shape-sequence but 13**2**11 is not.

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

Fig. 1. A msp and its ordered version.

Definition 8. A regular expression e over the alphabet $\mathbb{N}$ is called a *shape regular expression* if $e = 1e_1 2e_2 \cdots ne_n$, for some $n \in \mathbb{N}$, and for each $i = 1, \ldots, n$, e_i is a regular expression over $\{1, \ldots, i\}$. The number n is called *the rank of e* and is denoted by $r(e)$. The m-subexpression of e, for $m \leqslant n$, is the regular expression $e' = 1e_1 2e_2 \cdots me_m$. For instance, $122^*3(1+2)^*4$ is a shape regular expression (rank 4) but 13^*2 is not. 122^* is the 2-subexpression of $122^*3(1+2)^*4$.

An *RE-constraint* is a pair $[R_t, R_s]$ where R_t is a regular expression over the alphabet $\mathscr{I}$ (the set of items) and R_s is a shape regular expression of rank n. A *msp* σ with $r(\sigma) \leqslant r(R_s)$ satisfies an RE-constraint $[R_t, R_s]$ if $\Pi_t(\sigma)$ satisfies R_t and $\Pi_s(\sigma)$ satisfies the m-subexpression of R_s, where $m = r(\sigma)$. For instance, $[a^*b, 122^*3(1+2)^*4]$ is an RE-constraint and $\sigma = (\langle a, b, b\rangle, \langle 1, 2, 2\rangle)$ satisfies $[a^*b,\ 122^*3(1+2)^*4]$.

We say that a *msp* σ is *interesting* with respect to a dataset D, a minimum threshold α and an RE-constraint $[R_t, R_s]$ if σ is frequent with respect to D and α and σ satisfies $[R_s, R_t]$.

Problem formulation. Given a dataset D, an RE-constraint $[R_s, R_t]$ and a minimum support threshold α, find all *msps* which are interesting with respect to D, α and $[R_s, R_t]$.

4. SM-Miner algorithm

In this section we briefly describe the *SM-Miner* algorithm, which discover all frequent multi-sequential patterns and then select the patterns satisfying a user-specified RE-constraint in a post-processing phase. Since the SM-Miner algorithm is not the main topic of interest in this paper, we only give an idea of its structure. The details can be found in [15]. In what follows, C_k^n and L_k^n denotes, respectively, the set of *candidate msps* and *frequent msps* of length k and rank n. Notice that, according to the definition of multi-sequential pattern, we have necessarily that $n \leqslant k$.

The mining phase of SM-Miner follows the *Apriori* framework based on generation and validation (support counting) phases at each iteration. The general structure of the algorithm is that it generates first the frequent *msps* of rank 1 ($L_1^1, L_2^1, L_3^1, \ldots$) using for this task an algorithm for mining (simple) sequential patterns (for instance, the algorithm GSP [1]). After that, for each $n > 1$ (a given rank) it generates iteratively the sets C_k^n of *candidate msps* of length $k \geqslant n$. For the initial case $k = n$, C_n^n is generated from L_{n-1}^{n-1}, which has been already generated in the previous step corresponding to rank $n-1$. For the case $k > n$, the set C_k^n (containing the potentially frequent *msps* of length k and rank n) is generated from L_{k-1}^{n-1} and L_{k-1}^n which have been already generated in previous steps. Next, the algorithm eliminates those *msps* containing a *sub-msp* which is not frequent and then it calculates the support for the remaining candidates. At the end of the pass, it generates the set L_k^n, the candidate *msps* which are actually frequent. These *msps* become the seed for the next pass $k+1$. In Fig. 2(a), we illustrate how the set L_k^n is obtained from previous steps. For example, L_4^2 is obtained from L_3^2 and L_3^1 and the set L_3^3 is obtained from L_2^2. Fig. 2 sketches the whole mining process for L_k^n.

Fig. 3 illustrates how SM-Miner obtains a candidate *msp* of rank 5 and length 7 by joining two *msps* of L_6^4. There exists three other possibilities for obtaining candidates of rank 5 and length 7: (1) by joining a *msp* of L_6^4

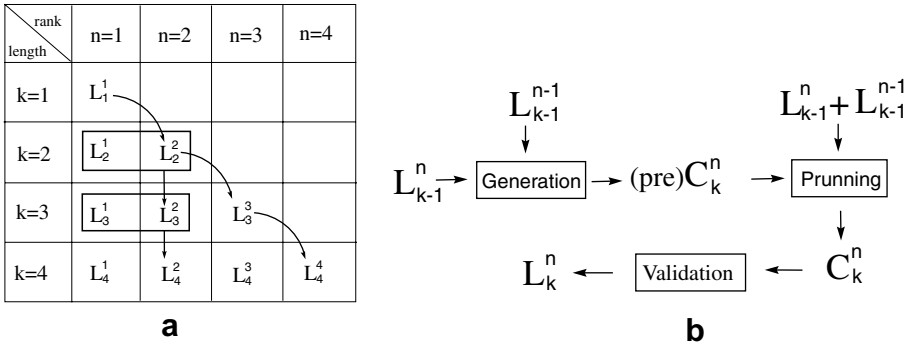

Fig. 2. L_k^n is obtained from L_{k-1}^{n-1} and L_{k-1}^n.

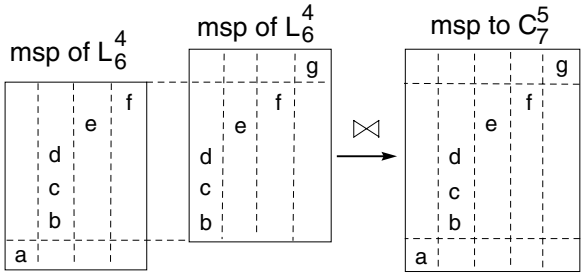

Fig. 3. Joining two msps.

and a *msp* of L_6^5, (2) by joining a *msp* of L_6^5 and a *msp* of L_6^4, (3) by joining two *msps* of L_6^5. A complete presentation of the candidate generation procedure can be found in [15].

A post-processing procedure takes place after the mining phase, when the discovered *msps* are tested and only those satisfying the RE-constraint $[R_t, R_s]$ are kept.

5. MSP-Miner algorithm

In this section we present the algorithm MSP-Miner for mining *msps* satisfying an RE-constraint $R = [R_t, R_s]$. We remind that C_k^n and L_k^n denote, respectively, the set of *candidate msps* and the set of *interesting msps* of rank n and length k, where $n \leqslant k$.

MSP-Miner is an Apriori-like double-iterative algorithm: at each stage (n,k), potentially interesting *msps* are generated, then those having no chance to be interesting are pruned and finally, the remaining ones are tested for support counting. It is important to point out that the candidate generation phase incorporates the RE-constraint R: candidates *msps* of rank n and length k are generated from previous L_{k-1}^n and L_{k-1}^{n-1} and they must satisfy R. For each n,k, we denote by $\mathscr{L}_{n,k}$ the set $\bigcup_{p=1}^{n-1}\bigcup_{q=n}^{k-1} L_q^p \cup \bigcup_{p=1}^{n-1} L_k^p$.

Fig. 4 below illustrates how candidates C_k^n are generated from previous L_{k-1}^n and L_{k-1}^{n-1}. The highlighted region corresponds to the set $\mathscr{L}_{n,k}$ for $n = 3$ and $k = 4$.

In the pruning phase, we must eliminate those *msps* containing a *submsp* which is not frequent (due to antimotonicity of *msp* frequency). Notice that, as the condition R is not *antimonotone*, we should not eliminate *msps* containing a *submsp* not satisfying R (doing so, we could eliminate interesting msps). Thus, we must concentrate only on the *msps* containing not frequent *submsps*. In order to detect not frequent *submsps* we have to find those not belonging to $\mathscr{L}_{n,k}$. A *msp* $\sigma \notin \mathscr{L}_{n,k}$ if and only if σ is not frequent or does not satisfy the constraint R. So, in order to ensure that σ is not frequent, it suffices to guaranteeing that σ satisfies R and $\sigma \notin \mathscr{L}_{n,k}$. We notice that the restrictiveness of the condition R has a negative effect on the amount of *msps* which are pruned. That is, as the strength of R increases, the amount of pruned *msps* decreases. In order to find a balance between the restrictiveness of R and the amount of pruned *msps*, we employ an idea similar

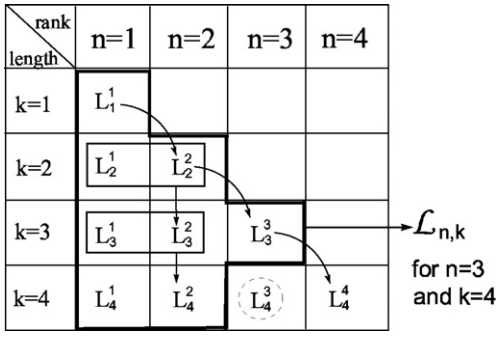

Fig. 4. L_k^n is obtained from L_{k-1}^n and L_{k-1}^{n-1}.

to the one introduced in [22], that is, we consider a relaxing condition R' which is less restrictive than R but at the same time has enough strength to eliminating a lot of candidates in the candidate generation phase. Thus, the mining process yields a set of *msps* which are interesting with respect to R'. At a post-processing phase the *msps* not satisfying R are eliminated.

In [22], four relaxing conditions have been considered: (1) the *total* relaxation where no restriction at all is incorporated in the candidate generation phase and the restriction R is considered only at the post-processing phase; (2) the *suffix-valid* relaxation, where only *valid* sequences with respect to a state of the automaton[7] (the automaton corresponding to the given regular expression R) are considered; (3) the *legal* relaxation, where only *legal* sequences with respect to a state of the automaton are considered[8]; (4) no relaxation at all: only valid sequences (those satisfying the automaton) are considered, that is, the condition R is totally pushed into the candidate generation phase and no post-processing is needed. The experimental analysis carried out in [22] allowed to conclude that the suffix-valid relaxation yields the best performance. Following this idea, we considered a similar relaxation for our RE-constraints. In what follows, we say that a sequence s over an alphabet Σ is *prefix-valid* with respect to an automaton $\mathscr{A}$ over the alphabet Σ if there exists a path in $\mathscr{A}$ starting at the initial state and producing s (not necessarily ending at a final state).

Definition 9. Let $R = [R_t, R_s]$ be an RE-constraint and A_{R_t}, A_{R_s} be the automata corresponding to R_t and R_s, respectively. We say that a *msp* σ is *prefix-valid* (*p-valid* for short) if $\Pi_t(\sigma)$ and $\Pi_s(\sigma)$ are prefix-valid with respect to A_{R_t}, A_{R_s}, respectively.

The general structure of the algorithm MSP-Miner can be summarized as following:

(1) *Candidate Generation*. The set C_k^n of p-valid *msps* is generated from the sets L_{k-1}^n, L_{k-1}^{n-1} of previous p-valid and frequent *msps*.
(2) *Pruning*. Those *msps* in C_k^n containing a p-valid *submsp* $\sigma \notin \mathscr{L}_{n,k}$ are pruned.
(3) *Support Counting*. The support of each remaining candidate is evaluated by scanning the dataset, and those *msps* with support inferior to a given minimum threshold are eliminated.
(4) *Terminating Condition*. For rank 1, the algorithm halts at stage $(1,k)$ where k is the first length for which $L_k^1 = \emptyset$. For rank $n \geqslant 2$, the algorithm halts at stage (n,k) if one of the following conditions is verified: (1) $L_p^{n-1} = \emptyset$, for all $p \in \{n-1,\ldots,k\}$ or (2) $L_{k-1}^{n-1} = L_{k-1}^n = \emptyset$ and $L_p^{n-1} = \emptyset$ for all $p \geqslant k-1$.
(5) *Post-Processing*. The remaining *msps* are tested for validity with respect to the original RE-condition R.

The algorithm MSP-Miner is described in Fig. 5. Next, we will discuss the details of the generation phase at stage (n,k).

5.1. Candidate generation

In order to generate p-valid *msps* of rank n and length k, we proceed as following:

(1) We first focus on the shape-sequences and generate all sequences of length k over the alphabet $\{1,\ldots,n\}$ which are p-valid with respect to the automaton A_{R_s} (step 7.1). Let $\mathscr{S}$ be the set of shape-sequences generated in this way.
(2) For each $s \in \mathscr{S}$, we will generate a set of item-sequences $\mathscr{I}(s)$, in such a way that the corresponding *msps* $(\mathscr{I}(s), \{s\})$ are p-valid and potentially frequent. The steps for accomplishing this task are the following: **(2a)** For each $s \in \mathscr{S}$, we consider its prefix s' of length $k-1$ by dropping its last element (step 7.2.1). Notice that $r(s') = n$ or $r(s') = n-1$ (step 7.2.2); **(2b)** For the item-sequences generation, we make use of the previous sets of p-valid and frequent *msps* L_{k-1}^n or L_{k-1}^{n-1}, depending on the rank of s' (n or $n-1$) (step 7.2.3.1). The routine *ExpandItemSeqs* (Fig. 6) is responsible for this task; **(2c)** We select in the suitable set (L_{k-1}^n or L_{k-1}^{n-1}) those *msps* corresponding to the given prefix s' (step 1. of *Expand-*

[7] A sequence s is valid with respect to a state q if there exists a path in the automaton corresponding to s, starting at q and ending at a final state.
[8] A sequence s is *legal* with respect to a state q it there is a path in the automaton corresponding to s and starting at q.

Procedure MSP-Miner(α, D, $[R_t, R_s]$) // α: minimum support, D:dataset
begin
1. N = number of data multi-sequences in D
2. A_{R_t}, A_{R_s} = automata corresponding to R_t and R_s respectively
3. CI_1^1 = item-sequences of length 1 p-valid with respect to the automaton A_{R_t}
4. $CS_1^1 = \{< 1 >\}$ // the unique shape-sequence of rank 1 and length 1
5. $C_1^1 = (CI_1^1, CS_1^1)$; L_1^1 = frequent *msps* of C_1^1
6. $n = 1$; $k = 2$; existMSPsRankCur = FALSE; kMaxIteration = 0
7. repeat
// candidate generation phase
7.1 CS_k^n = p-valid shape-sequences of rank n e length k
with respect to the automaton A_{R_s}
7.2 for each shape-sequence s of CS_k^n **do**
7.2.1 s' = $(k-1)$-prefix of s
7.2.2 x = rank(s') // the rank of s' can be n or $n-1$
7.2.3 If $s' \in \Pi_s(L_{k-1}^x)$
7.2.3.1 $\mathcal{I}(s)$ = ExpandItemSeqs(A_{R_t}, s, s',x‹ $k-1$, L_{k-1}^x)
7.2.3.2 $C_k^n = C_k^n \cup (\mathcal{I}(s), \{s\})$
// pruning phase
7.3 delete all candidates $\sigma \in C_k^n$ such that $\exists \tau \subseteq \sigma$, τ is p-valid
with respect to $[R_t,R_s]$ and $\tau \notin \mathcal{L}_{n‹k}$
7.4 for each $(g‹\tau)$ in D **do** // support counting phase
Increment the count of all candidate *msps* in C_k^n which are included in τ
7.5 L_k^n = candidates in C_k^n with count $\geq \alpha N$
7.6 If $L_k^n \neq \emptyset$
KmaxIteration = k; existMSPsRankCur = TRUE
$\mathcal{L}_{n‹k} = \mathcal{L}_{n‹k} \cup L_k^n$
7.7 $k = k+1$
7.8 if $L_{k-1}^{n-1} = \emptyset$ and $L_{k-1}^n = \emptyset$ and k > kMaxPrevIteration
7.8.1 $n = n+1$; $k = n$ // incrementing the rank
7.8.2 kMaxPrevIteration = kMaxIteration; kMaxIteration = 0
7.8.3 existMSPsPrevRank = existMSPsRankCur;
existMSPsRankCur = FALSE
Until not existMSPsPrevRank and n = k
// post-processing phase
8. delete all *msps* σ in $\mathcal{L}_{n‹k}$ such that σ does not satisfy $[R_t,R_s]$
9. return $\mathcal{L}_{n‹k}$
end

Fig. 5. MSP-Miner algorithm.

ItemSeqs); **(2d)** Then, we project these *msps* in order to obtain their item-sequences. These item-sequences are frequent (see proposition 6) and p-valid with respect to A_{R_t}. Next, the item-sequences are *expanded* according to the automaton A_{R_t} (This "expansion" will be clearer in the example below; for more details, see the procedure *ExpandItemSeqs* described in Fig. 6). At this stage, we obtain a set $\mathcal{I}(s)$ of potentially frequent and p-valid sequences with respect to the automaton A_{R_t} of length k (Step 7.2.3.1). So, the set of generated *msps* for shape-sequence s is $(\mathcal{I}(s), \{s\})$ (Step 7.2.3.2). Notice that the *msps* in $(\mathcal{I}(s), \{s\})$ are potentially frequent (proposition 6) and p-valid.

(3) The whole set C_k^n of generated *msps* of rank n and length k is $\bigcup_{s\in\mathcal{S}}(\mathcal{I}(s), \{s\})$ (Step 7.2.3.2).

5.2. An example

Let A_{R_t} and A_{R_s} depicted in Figs. 7 and 8 be the automata corresponding to the regular expressions R_t and R_s, respectively. Let q be a state of the automaton A_{R_t}. We denote by $L_k^n(q)$ the set $\{\sigma \in L_k^n \mid \Pi_t(\sigma)$ is p-valid w.r.t. $q\}$.

Procedure ExpandItemSeqs(A_{R_t}, shape-sequence s, shape-sequence s', n, k, L_k^n)
begin
1. $L_k'^n$ = *msps* in L_k^n corresponding to shape-sequence s'
// $\Pi_t(L_k'^n)$ are p-valid item-sequences with respect to some state of A_{R_t}
2. $\mathcal{I}(s) = \emptyset$;
3. **for each** state q of A_{R_t} **do**
3.1 V_q = set of all transitions ending at q
3.2 **for each** transition t of V_q **do**
3.2.1 a = label of t
3.2.2 **for each** p-valid item-seq. $u = <u_1, ..., u_k>$ from $\Pi_t(L_k'^n)$ with respect to source state of t **do**
3.2.2.1 newSeq = $< u_1, u_2, ..., u_k, a >$
3.2.2.2 $\mathcal{I}(s) = \mathcal{I}(s) \cup \{\text{newSeq}\}$
4. **return** $\mathcal{I}(s)$
end

Fig. 6. ExpandItemSeqs algorithm.

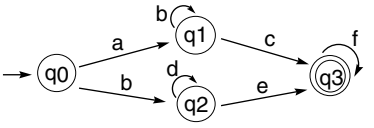

Fig. 7. Automaton A_{R_t}.

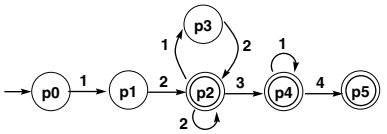

Fig. 8. Automaton A_{R_s}.

In this example, we illustrate how C_5^3 is generated from L_4^2 and L_4^3 which have already been calculated at stage (2,4) and (3,4), respectively. These sets are depicted in Table 5. The set C_5^3 is obtained as follows:

Table 5
The sets L_4^2 and L_4^3 which have already been calculated at stages (2,4) and (3,4)

$L_4^2(q_0)$	$L_4^2(q_1)$	$L_4^2(q_2)$	$L_4^2(q_3)$
	$\begin{pmatrix} \perp & b \\ \perp & b \\ \perp & b \\ a & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & \mathbf{d} \\ \mathbf{d} & \perp \\ \perp & \mathbf{d} \\ \mathbf{b} & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & \mathbf{c} \\ \mathbf{b} & \perp \\ \perp & \mathbf{b} \\ \mathbf{a} & \perp \end{pmatrix}, \begin{pmatrix} \perp & \mathbf{e} \\ \mathbf{d} & \perp \\ \perp & \mathbf{d} \\ \mathbf{b} & \perp \end{pmatrix}, \begin{pmatrix} \perp & e \\ \perp & d \\ d & \perp \\ b & \perp \end{pmatrix}$

$L_4^3(q_0)$	$L_4^3(q_1)$	$L_4^3(q_2)$	$L_4^3(q_3)$
	$\begin{pmatrix} \perp & \perp & b \\ \perp & b & \perp \\ \perp & b & \perp \\ a & \perp & \perp \end{pmatrix}$		$\begin{pmatrix} f & \perp & \perp \\ \perp & \perp & c \\ \perp & b & \perp \\ a & \perp & \perp \end{pmatrix}$

(1) We first generate all the shape-sequences of rank 3 and length 5 which are p-valid with respect to the automaton A_{R_s}. Such sequences are $\langle 1,2,1,2,3\rangle$, $\langle 1,2,3,1,1\rangle$, $\langle 1,2,2,2,3\rangle$ and $\langle 1,2,2,3,1\rangle$.

(2) For each one of these shape-sequences s_i, we will generate a set of item-sequences $\mathcal{I}(s_i)$ of length 5 in such a way that the corresponding *msps* are p-valid, as following:

(a) For the first shape-sequence $\langle 1,2,1,2,3\rangle$: Its 4-prefix (prefix of length 4) is $\langle 1,2,1,2\rangle$. This prefix has rank **2** and length ***4*** and therefore, only the item-sequences corresponding to L_4^2 will be expanded in order to obtain item-sequences of length 5. This task is performed by the procedure expandSeqsItems (Fig. 6), as following:

Step 1: Let us select the msps in L_4^2 corresponding to the shape-sequence $\langle 1,2,1,2\rangle$. They are showed in bold face in Table 5.

Step 2: These *msps* are projected in order to obtain their item-sequences. The following sets of item-sequences are obtained $\{\langle b,d,d,d\rangle\}$ (w.r.t. q_2) and $\{\langle a,b,b,c\rangle, \langle b,d,d,e\rangle\}$ (w.r.t to q_3).

Step 3: These item-sequences are expanded according to the automaton A_{R_t}. The idea is to produce p-valid item-sequences with respect to each state of A_{R_t}. Notice that only item-sequences ending at q_2 ($\langle b,d,d,d\rangle$) and q_3 ($\langle a,b,b,c\rangle$, $\langle b,d,d,e\rangle$) will be considered for expansion.

(i) *State* q_0: there is no transition ending at q_0.

(ii) *State* q_1: the transitions ending at q_1 are: $q_0 \xrightarrow{a} q_1$ and $q_1 \xrightarrow{b} q_1$. There is no p-valid item-sequences with respect to state q_0 or q_1. Therefore, no item-sequences are generated for state q_1.

(iii) *State* q_2: the transitions ending at q_2 are: $q_0 \xrightarrow{b} q_2$ and $q_2 \xrightarrow{d} q_2$.

- $q_0 \xrightarrow{b} q_2$: there is no p-valid item-sequences ending at q_0. Thus, no item-sequences will be expanded.
- $q_2 \xrightarrow{d} q_2$: the item-sequence $\langle b,d,d,d\rangle$ is p-valid with respect to state q_2. Therefore, the item-sequence $\langle b,d,d,d,\mathbf{d}\rangle$ is generated.

(iv) *State* q_3: the transitions ending at q_3 are: $q_1 \xrightarrow{c} q_3$, $q_2 \xrightarrow{e} q_3$ and $q_3 \xrightarrow{f} q_3$.

- $q_1 \xrightarrow{c} q_3$: there is no p-valid item-sequences with respect to q_1.
- $q_2 \xrightarrow{e} q_3$: the item-sequence $\langle b,d,d,d\rangle$ is p-valid with respect to q_2 and so the item-sequence $\langle b,d,d,d,\mathbf{e}\rangle$ is generated.
- $q_3 \xrightarrow{f} q_3$: the item-sequences $\langle a,b,b,c\rangle$ and $\langle b,d,d,e\rangle$ are p-valid with respect to q_3. Then, they are expanded and the item-sequences $\langle a,b,b,c,\mathbf{f}\rangle$ and $\langle b,d,d,e,\mathbf{f}\rangle$ are generated.

Thus, the algorithm *MSP-Miner* generates the item-sequences $\langle b,d,d,d,d\rangle$, $\langle b,d,d,d,e\rangle$, $\langle a,b,b,c,f\rangle$ and $\langle b,d,d,e,f\rangle$ fitting in the shape-sequence $\langle 1,2,1,2,3\rangle$.

Step 4: Each item-sequence generated in previous step is joined with the shape-sequence $<1,2,1,2,3>$. The *msps* obtained are:

$$\begin{pmatrix} \perp & \perp & d \\ \perp & d & \perp \\ d & \perp & \perp \\ \perp & d & \perp \\ b & \perp & \perp \end{pmatrix}, \begin{pmatrix} \perp & \perp & e \\ \perp & d & \perp \\ d & \perp & \perp \\ \perp & d & \perp \\ b & \perp & \perp \end{pmatrix}, \begin{pmatrix} \perp & \perp & f \\ \perp & c & \perp \\ b & \perp & \perp \\ \perp & b & \perp \\ a & \perp & \perp \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \perp & \perp & f \\ \perp & e & \perp \\ d & \perp & \perp \\ \perp & d & \perp \\ b & \perp & \perp \end{pmatrix}.$$

(b) The steps 1, 2, 3 and 4 are repeated for the shape-sequences $\langle 1,2,3,1,1\rangle$, $\langle 1,2,2,2,3\rangle$ and $\langle 1,2,2,3,1\rangle$.

(3) C_5^3 is the set of *msps* obtained in steps (a) and (b) above.

6. Experimental results

The MSP-Miner algorithm has been evaluated through an extensive set of experiments using synthetic datasets. We also compared the performances of MSP-Miner and SM-Miner. Our results confirm that incorporat-

ing RE-constraints into the mining process can lead to significant performance benefits rather than exploiting them only in the post-processing phase. Our experiments have been executed on a Pentium 4 of 3.0 GHz with 1GB of main memory and running Windows XP Professional.

6.1. Synthetic data

We have developed a synthetic data generator using the idea described in [24] for the synthetic data-sequences generator. Our generator produces datasets of multi-sequences in accordance with the input parameters. These parameters are shown in Table 6.

We have generated datasets by setting the number of items $N = 5000$. The average rank of potentially frequent multi-sequences ($|R|$) has been set to 3 and average length of potentially frequent *msps* ($|S|$) has been set to 4.

MSP-Miner is also evaluated with respect to a variety of parameters concerning the RE-constraints. We have developed an RE-constraint generator based on the regular expression generator proposed in [22] in the context of sequential pattern mining. Our method *REgen* generates RE-constraints $R = [R_t, R_s]$ according to some input parameters. Both regular expressions R_t and R_s are expressions of the form $(B_1|B_2|\cdots|B_n)^*$, where each *block* B_i is a concatenation of *terms* ($B_i = T_1T_2\cdots T_m$). A term T_i is a disjunction of items ($T_i = s_1 \mid s_2 \mid \cdots \mid s_{r_i}$).

The parameters required for generating a variety of regular expressions for our tests are: $I =$ the maximal number of items per term, $T =$ the number of terms per block, $B =$ the number of blocks, IMax = the max-

Table 6
Parameters used in the synthetic data generator

$\|D\|$	Number of groups (size of the dataset) – in '000s
$\|G\|$	Average number of objects per group
$\|C\|$	Average number of events per object
$\|R\|$	Average rank of potentially frequent *msps*
$\|S\|$	Average length of potentially frequent *msps*
N	Number of items
N_m	Number of maximal potentially frequent *msps*

Table 7
Minimum support and RE-constraint parameters

Parameter	Default value	Range
Number of blocks (B)	4	2–8
Number of terms per block (T)	3	2–8
Number of items for R_t (IMax)	5000	–
Rank of $R_s(r)$	10	–
Maximum number of items per term (I)	8	2–30
Minimum support	1.0%	0.2–1.2%

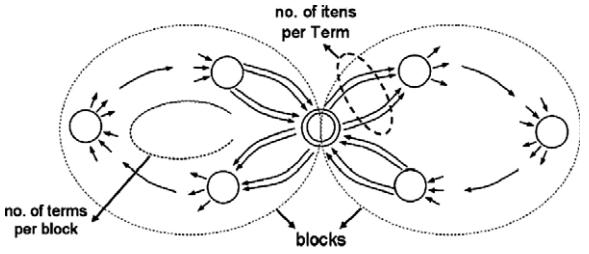

Fig. 9. Structure of automaton for RE-constraint.

imal number of items used in the regular expression R_t and r = rank of R_s. The regular expression R_t contains B blocks, T terms per block, at most I items per term and the items appearing in R_t are in $\{1,\ldots,\mathrm{IMax}\}$. The regular expression R_s is of the form $1e_1 2e_2 \cdots re_r$, where each e_i is a regular expression over $\{1,\ldots,i\}$ containing B blocks, T terms per block and at most I items per term. All the parameters used for generating the RE-constraints as well as minimum support thresholds are showed in Table 7 along with their default values and the range of values for which the experiments have been conducted.

Fig. 9 illustrates the generic structure of the automata produced by the generator. This structure allows several kinds of selectivities, yielding a large family of automata, by varying the number of items per term, the number of terms per block and the number of blocks.

6.2. Performance analysis

For the performance analysis, the experiments have been carried out over two datasets, one containing 4000 groups and the other 8000 groups. Table 8 summarizes all the dataset parameter settings for these experiments.

Fig. 10 shows the execution times for the MSP-Miner and SM-Miner algorithms for the first dataset given in Table 8. We can verify that MSP-Miner always outperforms SM-Miner. Besides, in most cases the difference between their perfomances is quite important. These results confirm the effectiveness of exploiting RE-constraints during the mining process rather than during the post-processing phase.

Table 8
Synthetic datasets – parameter settings

Name	$\|D\|$	$\|G\|$	$\|C\|$	$\|R\|$	$\|S\|$	N_m	Size (MB)
D4-G4-C6-R3-S4	4	4	6	3	4	200	4,3
D8-G4-C6-R3-S4	8	4	6	3	4	400	8,6

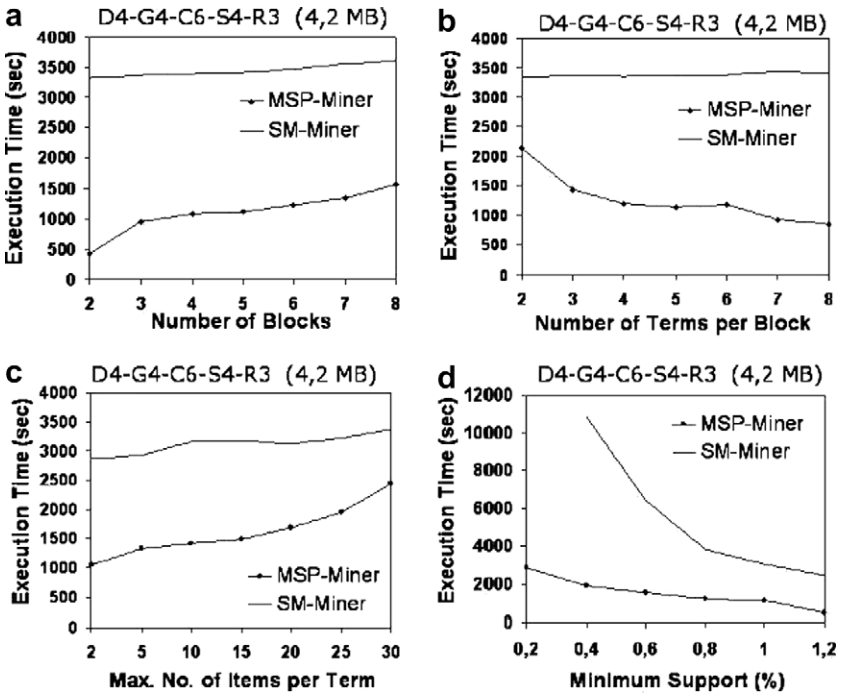

Fig. 10. Performance results for dataset D4-G4-C6-R3-S4.

The graphs in Fig. 10 plot the running times for both algorithms as the RE-constraint parameters change. Fig. 10(a) depicts the perfomance of both algorithms as the number of blocks in the regular expressions is increased from 2 to 8. As expected, as the number of blocks increases, the selectivity of the RE-constraint decreases and consequently the number of p-valid *msps* increases. Thus, the execution time of MSP-Miner also increases.

A similar analysis can be considered for the graph in Fig. 10(c). Note that in Figs. 10(a), (b) and (c) the execution times for SM-Miner raise very smoothly. This happens because SM-Miner always finds the whole set of frequent *msps* and only afterwards it selects those satisfying the constraint. The time expended in the mining process (before post-processing) does not depend on the restriction considered. Besides, as the post-processing phase is executed in main memory, its execution time is negligible when compared to the time expended in the mining process. Fig. 10(d) illustrates the performance of both algorithms as the minimum support increases from 0.2% to 1.2%. As expected, the execution times for both algorithms decreases, because fewer candidates are potentially frequent for higher values of minimum support. The graph shows that for low support levels, MSP-Miner performs much better than SM-Miner. This happens because the amount of candidate patterns reaching the support counting phase in SM-Miner is larger than the amount of candidates in MSP-Miner, where only selected candidates survive after the generation phase.

The execution time for both algorithms, as the number of terms per block in regular expressions is increased from 2 to 8, is depicted in Fig. 10(b). Note that the execution time for the MSP-Miner algorithm decreases as the number of terms per block increases. Indeed, when regular expressions R_t and R_s have few terms per block, short *msps* can be p-valid as well as long *msps* (due to the *star* in $(B_1|\cdots|B_n)^*$). However, as the number of terms per block increases, only long *msps* may be p-valid.

Fig. 11 illustrates the performance results for the second dataset of Table 8. As expected, the execution time for boths algorithms increases but MSP-Miner still outperforms SM-Miner. The execution time for MSP-Miner algorithm increases smoothly as the number of blocks in the regular expressions is increased from 2 to 6. However, when the number of blocks is greater than 6, the performance of MSP-Miner algorithm decreases considerably.

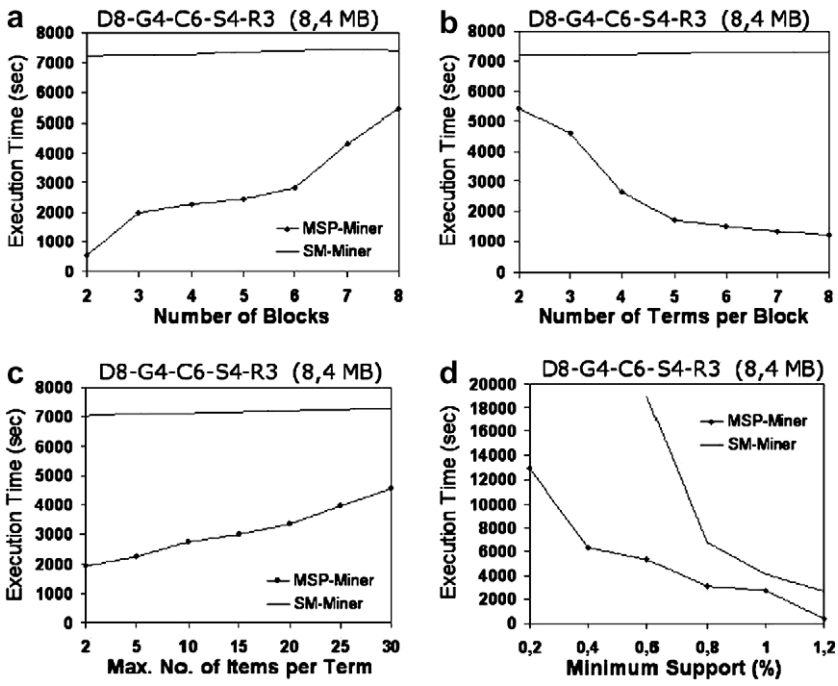

Fig. 11. Performance results for dataset D8-G4-C6-R3-S4.

6.3. Scale-up

The scalability of MSP-Miner with respect to different parameters has also been evaluated. In these experiments, we have kept the RE-constraint invariant. It has been generated using the default parameters presented in Table 7.

We first describe how MSP-Miner performs as the number of input multi-sequences in the dataset increases. Fig. 12(a) shows how MSP-Miner and SM-Miner scale up as the number of data multi-sequences ($|D|$) increases from 1000 to 6000. We show the results for the datasets Dx-G4-C3-S4-R3 ($x = 1, \ldots, 6$). The minimum support is set to 1%. The graphic shows that the execution time for MSP-Miner increases smoothly as the number of data multi-sequences (size of database) enlarges, contrarily to SM-Miner algorithm, whose execution time increases considerably.

Fig. 12(b) shows how both algorithms scale up as the number of objects per group ($|G|$) increases from 3 to 7. The datasets D4-Gx-C3-S4-R3 ($x = 3, \ldots, 7$) have been used and the minimum support has been set to 1%. When the number of objects per group is greater than 5, the performance for both algorithms decline substantially. However, increasing the size of the groups, the difference between their execution times is still large, that is, MSP-Miner is clearly faster than SM-Miner.

Besides, the execution time of support calculation increases with respect to the number of objects in each group. So, it is expected that the performance of SM-Miner will be worse, since the number of candidates tested by SM-Miner at each pass is greater than the one tested by MSP-Miner, where only selected candidates survive after the generation phase.

A similar analysis can be considered for the graph in Fig. 13, which illustrates the performance of both algorithms as the number of events per object increases from 2 to 8.

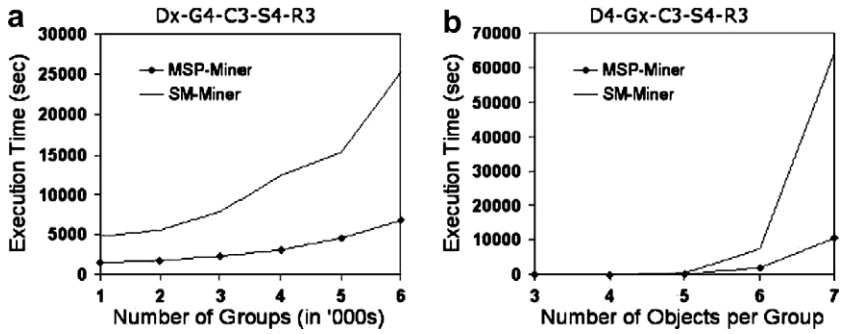

Fig. 12. Scale-up w.r.t to data multi-sequences (a) and objects per group (b).

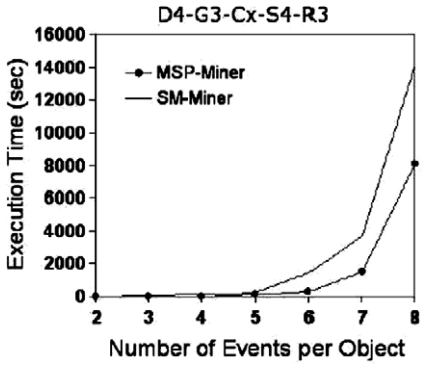

Fig. 13. Scale-up w.r.t to number of events per object.

7. Conclusion and future work

In this paper, we have presented a constraint-based approach to mine a new kind of first-order temporal patterns called *multi sequential patterns*. These patterns are characterized by the fact that support counting is based on *groups of objects* rather than single objects like in propositional sequential patterns and other known first-order temporal patterns studied in the ILP area. The main interest in using multi-sequential pattern mining is in datasets storing information about objects which are related to each other according to some criteria. A typical scenario is in mining spatial sequential patterns in temporal census data. In this scenario, objects are usually related to each other by a *neighbour* relationship, and data analysts are interested in discovering patterns telling how demographic indicators of each object (in our example, the objects are districts of a municipality) evolve throughout time and how the neighbour relationship can affect this evolution. We have introduced a formalism based on regular expressions for constraint specification and developed the algorithm MSP-Miner to discover all frequent multi-sequential patterns satisfying such constraints. MSP-Miner incorporates the constraints *during* the mining process, pushing non-antimonotone conditions (specified by regular expressions) inside both the candidate generation and pruning phases. We conducted an extensive set of experiments to evaluate and compare the performance and scalability of MSP-Miner and SM-Miner over synthetic data. SM-Miner [15] is an algorithm for mining multi-sequential patterns which incorporates the RE constraints only in post-processing phase. Our empirical study allowed to conclude the feasibility and effectiveness of exploiting RE constraints during the mining process, in a first-order temporal mining context.

Future research. In the near future, we intend to focus our research efforts in two directions: first, we need to evaluate the algorithms in a real dataset. For this purpose, we will take the census databases available in the Brazilian Institute of Geography and Statistics (IBGE). However, this task is not simple, since the datasets have to be suitably transformed before the mining process, in order to fit in the MSP-Miner mining schema. Another line of research we intend to tackle is to consider more general multi-sequential patterns. In this paper, we have considered multi-sequential patterns of the form:

$$\exists D_1 \exists D_2 \cdots \exists D_n (Q \wedge \Diamond (R_1 \wedge \Diamond (R_2 \wedge \cdots \wedge \Diamond R_m)) \cdots)$$

where Q is a formula representing the relationship between the object variables $D_1, \ldots, D_n$, and R_i is an atomic formula of the form $R(D_i, a)$, where a is an item. We think that it will be interesting to consider patterns where each R_i is a *conjunction* of atomic formulas $R(D_1, a_1) \wedge R(D_1, a_2) \wedge \cdots \wedge R(D_1, a_m)$ (only one object variable D_1 appears in this conjunction). In our census database example, this would mean to consider several demographic indicators for each district at a time. For instance, we could be interested in discovering that *"A low rate of full-time employed male in district A, followed by a high rate of commercial activities AND a high rate of full-time employed male in a neighbour district B is followed by a high rate of part-time employed female in district A AND a high rate of part-time employed male in district A"*.

References

[1] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: Proceedings of the 5th International Conference on Extending Database Technology, Avignon, France, EDBT, vol. 1057, Springer, 1996, pp. 3–17.
[2] M.J. Zaki, SPADE: an efficient algorithm for mining frequent sequences, Machine Learning 42 (1–2) (2001) 31–60.
[3] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, Freespan: frequent pattern-projected sequential pattern mining, in: Proceedings of the Sixth ACM SIGKDD, ACM Press, 2000, pp. 355–359.
[4] H. Mannila, H. Toivonen, A.I. Verkamo, Discovery of frequent episodes in event sequences, Data Mining and Knowledge Discovery 1 (3) (1997) 259–289.
[5] G. Das, K. Lin, H. Mannila, G. Renganathan, P. Smyth, Rule discovery from time series, in: Knowledge Discovery and Data Mining, 1998, pp. 16–22.
[6] F. Hoppner, Discovery of temporal patterns – learning rules about the qualitative behavior of time series, in: Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases, Freiburg, Germany, 2001, pp. 192–203.
[7] H. Lu, L. Feng, J. Han, Beyond intra-transaction association analysis: mining multi-dimensional inter-transaction association rules, ACM Transactions on Information Systems 18 (4) (2000) 423–454.
[8] Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, Umeshwar Dayal, Multi-dimensional sequential pattern mining, in: CIKM, 2001, pp. 81–88.

[9] M. Joshi, G. Karypis, V. Kumar, Universal Formulation of Sequential Patterns, University of Minnesota, Computer Science and Engineering, 1999.
[10] G. Berger, A. Tuzhilin, Discovering unexpected patterns in temporal data using temporal logic, in: Temporal Databases: Research and Practice, 1998, pp. 281–309.
[11] C. Masson, F. Jacquenet. Mining frequent logical sequences with SPIRIT-LoG, in: Proceedings of the 12th International Conference on Inductive Logic Programming, LNAI, vol. 2583, SV, 2003, pp. 166–181.
[12] S.D. Lee, L. De Raedt, Constraint based mining of first-order sequences in SeqLog, in: Workshop on Multi-Relational Data Mining, SIGKDD, University of Alberta, Edmonton, Canada, July 2002, pp. 80–96.
[13] L. De Raedt, S.D. Lee, Constraint based mining of first order sequences in seqlog, Database Support for Data Mining Applications (2004) 154–173.
[14] N. Jacobs, H. Blockeel, From shell logs to shell scripts, in: C. Rouveirol, M. Sebag (Eds.), in: ILP '00: Proceedings of the 11th International Conference on Inductive Logic Programming, vol. 2157, September 2001, pp. 80–90.
[15] S. de Amo, D.A. Furtado, A. Giacometti, D. Laurent, An apriori-based approach for first-order temporal pattern mining, in: Proceedings of the 19th Brazilian Symposium on Databases, Brasilia, Brazil, October 2004, pp. 48–61. Available from: <http://www.deamo.prof.ufu.br/arquivos/SBBD-deamo-Furtado.pdf>.
[16] D. Malerba, F. Esposito, F. Lisi, A. Appice, Mining spatial association rules in census data, Research in Official Statistics 5 (1) (2002) 19–44.
[17] B. Liu, W. Hsu, Y. Ma, Pruning and summarizing the discovered associations, in: KDD'99 Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press, 1999, pp. 125–134.
[18] D. Shah, L. Lakshmanan, K. Ramamritham, S. Sudarshan, Interestingness and pruning of mined patterns, in: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD), 1999.
[19] B. Padmanabhan, A. Tuzhilin, Small is beautiful: discovering the minimal set of unexpected patterns, in: KDD '00: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 54–63.
[20] R.T. Ng, L.V.S. Lakshmanan, J. Han, A. Pang, Exploratory mining and pruning optimizations of constrained associations rules, in: Proceedings of the ACM SIGMOD International Conference Management of Data, New York, NY, USA, 1998, pp. 13–24.
[21] C. Antunes, A.L. Oliveira, Constraint relaxations for discovering unknown sequential patterns, in: KDID, 2004, pp. 11–32.
[22] M.N. Garofalakis, R. Rastogi, K. Shim, SPIRIT: Sequential pattern mining with regular expression constraints, in: The VLDB Journal, Edinburgh, Scotland, 1999, pp. 223–234.
[23] J.-F.Boulicaut, L. De Raedt, H. Mannila (Eds.), Constraint-based mining and inductive databases, in: European Workshop on Inductive Databases, Lecture Notes in Artificial Intelligence, vol. 3848, 2006.
[24] R. Agrawal, R. Srikant, Mining sequential patterns, in: Philip S. Yu, Arbee S.P. Chen (Eds.), Eleventh International Conference on Data Engineering, IEEE Computer Society Press, Taipei, Taiwan, 1995, pp. 3–14.
[25] J. Pei, J. Han, W. Wang, Mining sequential patterns with constraints in large databases, in: J. Pei, J. Han, W. Wang (Eds.), Mining Sequential Patterns with Constraints in Large Databases, Proceedings of the 2002 ACM CIKM Conference, 2002.

Sandra de Amo received the Ph.D. degree in computer science from the University of Paris 13, France, in 1995. She is a professor in the Faculty of Computer Science at the Federal University of Uberlândia, Brazil. From 1998 to 2001 she was an invited researcher in the Computer Science Laboratory (LABRI), at the University of Bordeaux I. In the academic year 2001–2002, she was on sabbatical in the Computer Science Laboratory (LI) at the University of Tours, France. Her research interests are data mining, temporal databases, query languages, applications of logic in databases, inconsistent information management in databases. Dr. Sandra de Amo is a member of the ACM since 1999.

Daniel A. Furtado received the M.Sc. degree in computer science in 2005 from the Federal University of Uberlândia, Brazil. Currently he is a non-tenured adjunct professor in the Faculty of Computer Science at the Federal University of Uberlândia, Brazil. Since 2003, he has been working in the area of knowledge discovery in databases.