Temporal Support in Sequential Pattern Mining

Leticia I. Gómez[1] and Bart Kuijpers[2] and Alejandro A. Vaisman[3]

[1] Instituto Tecnólogico de Buenos Aires
lgomez@itba.edu.ar
[2] Buenos Aires University, Hasselt University & Transnational University of Limburg
alejandro.vaisman@uhasselt.be
[3] Hasselt University & Transnational University of Limburg
bart.kuijpers@uhasselt.be

Abstract. In sequential pattern discovery, the support of a sequence is computed as the number of data-sequences satisfying a pattern with respect to the total number of data-sequences in the database. When the items are frequently updated, the traditional way of counting support in sequential pattern mining may lead to incorrect (or, at least incomplete), conclusions. For example, if we are looking for the support of the sequence A.B, where A and B are two items such that A was created *after* B, all sequences in the database that were completed *before* A was created, can never produce a match. Therefore, accounting for them would underestimate the support of the sequence A.B. In this paper we propose to revise the classic notion of support in sequential pattern mining, introducing the concept of *temporal support of a sequential expression*(SE), intuitively defined as the number of sequences satisfying a target pattern, out of the total number of sequences that *could have possibly* matched such pattern. We then generalize this notion to regular expressions (RE) which encapsulate the definition of a collection of SEs. We present and discuss a theoretical framework for these novel notion of support, and present an algorithm to compute it.

1 Introduction and Problem Statement

Traditional sequential patterns algorithms are founded on the assumption that items in databases are *static*, and that they existed throughout the whole lifespan of the world modeled by the database. There are many real-world situations (where items are created or deleted dynamically) where these assumptions are not valid. Consider for example SPM in the context of the World Wide Web, where Web pages are frequently added or deleted. Data Mining techniques have been applied for discovering interaction patterns of WWW users, typically analyzing the URLs visited during a session, recorded in a Web server log. Figure 1 depicts a portion of a (simplified) Web log. In classic SPM, the support of a sequence S is defined as the fraction of sessions that support S. Thus, all sessions are considered as having the *same probability* to support a given sequence. For example, the support of the sequence CBC, counted in the classical way, would be 66%, since CBC is present in two of the three sessions. Analogously, the support of the sequence CB would also be 66%. However, if not all these Web pages existed all the time, would it be reasonable to ignore the evolution of the items (URLs) across time? We discuss these issues in this paper. The right hand side

session ID	interaction	
	time	URL
s1	t=1	A
	t=3	C
	t=5	B
	t=6	C
	time	URL
s2	t=3	A
	t=4	B
	t=8	A
	time	URL
s3	t=5	A
	t=10	C
	t=20	B
	t=23	C

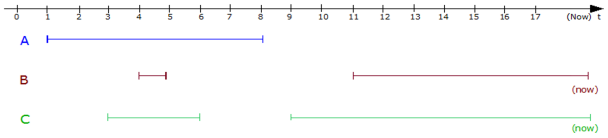

Fig. 1. Web user interaction (left) and Evolution of three URLs A, B, and C (right)

of Figure 1 shows how URLs A, B, and C in the left hand side have evolved, and the time intervals when each URL has been available. For example, URL A was available during the interval [1, 8], and URL B in intervals [4, 5] and [11, *now*]. (We use the term *now* to refer to the current time instant). During session s2, URL C did not exist at t=8, when the user clicked URL A. Thus, session s2 did not have the possibility of producing a sequence that finishes with the URL C. Sessions s1 and s3, instead, support the sequence CBC. Then, ignoring the evolution of these URLs, the support of sequence CBC would be 66%, but, if we do not count session s2, we would obtain a support of 100%. Analogously, s1 and s3 support the sequence CB but s2 does not. However, C was available during session s2, when the user clicked URLs A (t=3) and B (t=4). Thus, the user could have produced the sequence CB, although she decided to follow a different path. Session s2 must then be counted for computing the support of the sequence CB, which would be 66%.

In addition to the above, suppose now that a user is interested not only in the support of a sequence, but in the support of an RE as a whole. Let us analyze a simple example. The expression $(A|B).C$ is satisfied by sequences like A.C or B.C. Even though the semantics of this RE suggests that both of them are equally interesting to the user, if neither of them verifies a minimum support (although together they do), they would not be retrieved. The problem gets more involved if we are interested in categorical sequential patterns, i.e., patterns like *Science.Sports*, where *Science* and *Sports* are, for instance, categories of Web pages in an ontology.

In light of the above, we propose to revise, in different ways, the classic notion of support for sequential pattern mining, introducing the concept of *temporal support of regular expressions*, intuitively defined as the number of sequences satisfying a target pattern, out of the total number of sequences that *could have possibly* matched such pattern, where the pattern is defined as a RE over complex items. After reviewing related work (Section 2), we introduce the data model (Section 3). Then we present and discuss a theoretical framework for this novel notion of support, and an RE-based language (Sections 4 and 5). Finally, we adapt classic Generalized Sequential Pattern (GSP) algorithms to our setting, introducing appropriate pruning strategies (Section 6).

2 Related Work

Classic algorithms for Sequential Pattern discovery [1, 8] return all frequent sequences present in a database. However, more often than not, only a few ones are interesting from a user's point of view. Thus, post-processing tasks are required in order to discard uninteresting sequences. To avoid this drawback, languages based on regular expressions (RE) were proposed to restrict frequent sequences to the ones that satisfy user-specified constraints. Garofalakis *et al.* [2] address this problem by pruning the candidate patterns obtained during the mining process by adding user-specified constraints in the form of regular expressions over items. The algorithm returns only the frequent patterns that satisfy these regular expressions. Recently, the data mining community started to discuss new notions of support in SPM, that account for changes of the items database across time. Although this problem has already been addressed for Association Rule mining, where the concept of *temporal support* has already been introduced [4, 5, 10], this has been overlooked in SPM. To the best of our knowledge, the works we comment below are the only ones partially addressing the issue. Masseglia *et al.* [6], and Parthasarathy et al.[7], study the so-called *incremental sequential pattern mining* problem, focusing on avoiding re-scanning the entire database when new items appear. Recently, Huang *et al.* [3] addressed the problem of detecting frequent patterns *valid during a defined period of interest,* called POI. For example, if new items appear, and no new transactions were generated, old frequent sequences would still be frequent.

3 Data Model

Depending on the application domain, the items to be mined can be characterized by different attributes. Throughout the paper we refer to an example where each Web page is characterized by the following attributes: (a) *catName*, which represents the name of the category of the item; (b) *keyword*, which summarizes the page contents; (c) *filter*, specifying a list of URLs that cannot appear together with the URL of the item. Finally, *ID* is a distinguished, mandatory attribute, in this case containing the URL that univocally identifies a Web page.

Definition 1. [Category Schema] *We have a set of attribute names* $\boldsymbol{A}$*, and a set of identifier names* $\boldsymbol{I}$*. Each attribute* $a \in \boldsymbol{A}$ *is associated with a set of values in* $dom(a)$*, and each identifier* $ID \in \boldsymbol{I}$ *is associated with a set of values in* $dom(ID)$*.*

A *category schema* S is a tuple (ID, A), where $ID \in \mathbf{I}$ is a distinguished attribute denoted *identifier*, and A is a set of attributes in $\mathbf{A}$. Without loss of generality, and for simplicity, in what follows we consider the set A ordered. Thus, S has the form $[ID, attr_1, ..., attr_n]$. □

In many real-world applications, assuming that the values of attributes for a category occurrence do not change, could not be realistic. Thus, we introduce the time dimension into our data model, timestamping category occurrences. We assume that the category schema is constant across time. Formally, if T is a set,

and $<$ a discrete linear order without endpoints on T, the structure $T_P = (T, <)$ is the *Point-based Temporal Domain*. The elements in the carrier of T model the individual time instants, and the linear order $<$ models the succession of time. We consider the set T to be $\mathbf{N}$ (standing for the natural numbers). We can map individual time instants t $\in \mathbf{N}$ to calendar instants, assuming a reference point and a granularity. In what follows we use calendar time, and granularity "minute". In temporal databases terminology [9], in this paper we assume *valid time* support for categories, and *transaction time* support for items (see below).

Definition 2. [Category Occurrence and Category Instance] *Given a category schema S, a* category occurrence *for S is the tuple $[\langle ID, id\rangle, P, t]$, where ID is the ID attribute of Definition 1 above, $id \in dom(ID)$, P is the structure $[\langle attr_1, v_1\rangle ..., \langle attr_n, v_n\rangle]$, t is a point in the temporal domain T_P, and: (a) $attr_i = A(i)$ in S (remember that A is considered ordered); (b) $v_i \in dom(attr_i)$, $\forall i, i = 1..n$; (c) All the occurrences of the same category have the same set of attributes, at any given time; (d) At any instant t, the pair $\langle ID, t\rangle$ is unique for a category occurrence, meaning that no two occurrences of the same category can have the same value for ID at the same time; (e) t is the time instant when the information in the category occurrence is valid.*

A set of occurrences of the same category is denoted a category instance. *We extend the fourth condition in Definition 2 to hold for the whole set: no two occurrences of categories in the set can have the same value for ID at the same instant t (in other words, the pair (ID, t) is unique for the whole instance).* □

In what follows, for clarity, we assume that $attr_1$ stands for ID. Thus, a category occurrence is the set of pairs [$\langle$ $attr_1$,$v_1\rangle$,..., ... , $\langle$ $attr_n$,$v_n\rangle$, t]. □

Definition 3. [Interval Encoding] *Let G be a time granularity, and g a time unit for G (e.g., one minute). Given a set of $k \geq 0$ category occurrences, $[\langle attr_1, v_1\rangle, \ldots, \langle attr_n, v_n\rangle, t_1]$, $[\langle attr_1, v_1\rangle, \ldots, \langle attr_n, v_n\rangle, t_2]$,..., $[\langle attr_1, v_1\rangle \ldots$, $\langle attr_n, v_n\rangle, t_k]$, if $\forall\ i, i = 1..k-1$, it holds that $t_{i+1} = t_i + g$, we encode all these occurrences in a single tuple $[\langle attr_1, v_1\rangle, \ldots, \langle attr_n, v_n\rangle, [t_1, t_k]]$.* □

Definition 4. [Encoded Category Occurrence and Items] *Given a category instance $\mathcal{C}$ with time granularity G, and a partition $\mathcal{P}$ of $\mathcal{C}$ such that the number of sets $p_i \in \mathcal{P}$ is minimal. Each set p_i is obtained encoding the occurrences in $\mathcal{C}$ as in Definition 3, i.e., each p_i contains a set of tuples that can be encoded into a single tuple. Thus, associated to p_i there is a tuple $t_{p_i} = (\langle ID, id\rangle, \langle attr_1, v_1\rangle, \ldots \langle attr_n, v_n\rangle, t_v, t_e)$, where (a) ID, $attr_1$, ..., $attr_n$ are the attributes of the occurrences in p_i; (b) id, v_1, ... v_n are the values for the attributes in (a); (c) t_s is the smallest t of the occurrences in p_i; (d) t_e is the largest t of the occurrences in p_i. We denote t_{p_i} an encoded category occurrence (ECO) of the set of occurrences in p_i. Given an ECO e_i we denote Interval(e_i) its associated interval $[t_s, t_e]$.*

*Adding a time instant to an ECO, produces an **Item**. Let $e_o = (\langle ID, v\rangle, \langle attr_1,$ $v_1\rangle, \ldots \langle attr_n, v_n\rangle, t_s, t_e)$ be an ECO for some category instance. An* item *I associated to e_o is the set: $(\langle t, v_t\rangle, \langle ID, v\rangle, \langle attr_1, v_1\rangle, \ldots \langle attr_n, v_n\rangle, t_s, t_e\rangle)$, such that $v_t \in [t_s, t_e]$ holds. We denote t the* transaction time *of the item.* □

eco_{A1}	[(ID,'A'),	(catName,'WebPage'),	(filter,''),	(keyword,'Books'),	['11/29/2007 15:45', '11/29/2007 17:10']]
eco_{A2}	[(ID,'A'),	(catName,'WebPage'),	(filter,'P'),	(keyword,'Computers'),	['11/29/2007 18:00', '11/29/2007 19:30']]
eco_{C1}	[(ID,'C'),	(catName,'WebPage'),	(filter,'A'),	(keyword,'Books'),	['11/29/2007 16:00', '11/29/2007 16:45']]
eco_{C2}	[(ID,'C'),	(catName,'WebPage'),	(filter,''),	(keyword,'Games'),	['11/29/2007 18:00', '11/29/2007 18:50']]
eco_{C3}	[(ID,'C'),	(catName,'WebPage'),	(filter,'M'),	(keyword,'Games'),	['11/29/2007 19:30', 'Now']]
eco_{M1}	[(ID,'M'),	(catName,'WebPage'),	(filter,''),	(keyword,'Games'),	['11/29/2007 18:50', '11/29/2007 20:00']]
eco_{P1}	[(ID,'P'),	(catName,'WebPage'),	(filter,''),	(keyword,'Computers'),	['11/29/2007 16:45', '11/29/2007 18:50']]
eco_{P2}	[(ID,'P'),	(catName,'WebPage'),	(filter,'A'),	(keyword,'Computers'),	['11/29/2007 18:51', 'Now']]

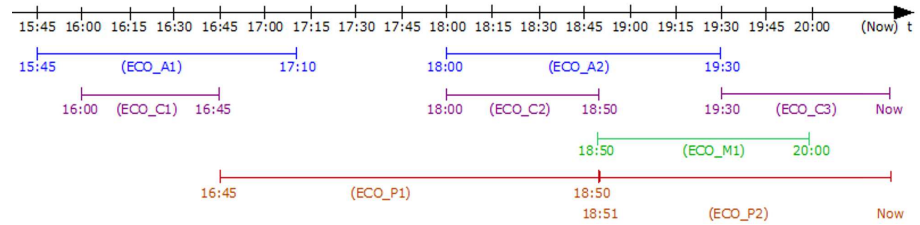

Fig. 2. Encoded Category Occurrences for the running example (granularity "minute").

Figure 2 shows an interval-encoded representation for our running example. Encoding a set of tuples requires these tuples to be consecutive over the granularity selected. Thus, if the granularity is "minute", the tuples [(ID, 'A'), ($keyword$, 'computer'), ($filter$, ''), '12/12/2000 12:31')], and [(ID, 'A'), ($keyword$, 'computer'), ($filter$, ''), '12/12/2000 12:33')], cannot be included together in the same group, since there is a two-minute gap between them. They must be encoded into two intervals.

4 A Theory for Support Count

Definition 5. [Sequential Expression] *A* Sequential Expression *(SE) of length n is an ordered list of n sub-expressions $SE_1.SE_2....SE_n$, where each SE_i is a* constraint*, $\forall i, i = 1..n$.*

A constraint *is a* formula *enclosed in squared brackets. Given a constraint $C = [\mathcal{F}]$, we denote $\mathcal{F}(\mathcal{C})$ the* formula of *C. A formula is built as a conjunction of atoms of the form term1 = term2, where term1 is an attribute (temporal or non-temporal) or a function, and term2 is a constant. More precisely, the* terms *of the language are: (a) Constants: a literal enclosed by simple quotes; (b) Non temporal Attributes: an attribute in the category schema (e.g. filter, url). (c) Temporal Attributes: t, the temporal attribute of and item; (d) Functions of n arguments: Let f_n be a function symbol, the expression $f_n(attribute, 'ct1', 'ct2', ..., 'ct_{n-1}'), n \geq 1$, is a function where the* first *parameter is an attribute (temporal or non-temporal), and all the other ones are constants.* □

Given two intervals $I_i = [ts_i, te_i]$ and $I_j = [ts_j, te_j]$ we say that I_i *follows* I_j if $ts_i \geq te_j$. For example, in Figure 2 we can see that Interval(eco_{C3}) *follows* Interval(eco_{A2}) and Interval(eco_{C2}). We can also see that Interval(eco_{C3}) *does not follow* Interval(eco_{M1}).

Definition 6. [ECO-Satisfability of a Constraint] *Given a constraint C and an ECO E, we say that E* satisfies *C if one of the following conditions hold: (a) If*

OID	Items
$Session_1$	[(t,'11/29/2007 16:30'), (ID,'C')] $\rightsquigarrow eco_{C1}$ [(t,'11/29/2007 17:00'), (ID,'P')] $\rightsquigarrow eco_{P1}$ [(t,'11/29/2007 19:45'), (ID,'C'),] $\rightsquigarrow eco_{C3}$
$Session_2$	[(t,'11/29/2007 18:20'), (ID,'C')] $\rightsquigarrow eco_{C2}$ [(t,'11/29/2007 18:50'), (ID,'P')] $\rightsquigarrow eco_{P1}$ [(t,'11/29/2007 18:51'), (ID,'M')] $\rightsquigarrow eco_{M1}$
$Session_3$	[(t,'11/29/2007 19:31'), (ID,'C')] $\rightsquigarrow eco_{C3}$ [(t,'11/29/2007 19:32'), (ID,'M')] $\rightsquigarrow eco_{M1}$ [(t,'11/29/2007 20:00'), (ID,'C')] $\rightsquigarrow eco_{C3}$

Fig. 3. An instance of the *Normalized ToI*

$\mathcal{F}(C)$ is of the form attr = 'ct' where attr is an attribute, 'ct' is a constant, and the instantiation of attr with its value in E, verifies the equality. (b) If $\mathcal{F}(C)$ is of the form $f_n(attr, 'ct1', 'ct2', ..., 'ct_{n-1}') = 'ct'$, attr and 'ct' are defined as in (a), and the instantiation of attr in f_n with its value in E, makes the equality true. *(c) If $\mathcal{F}(C)$ is of the form t = 'ct' where t is a temporal attribute, 'ct' is a temporal constant, and 'ct' $\in Interval(E)$. (d) If $\mathcal{F}(C)$ is of the form $f_n(t, 'ct1', 'ct2', ..., 'ct_{n-1}') = 'ct'$, where t and 'ct' are defined as in (c), and $\exists t_u \in Interval(E)$ such that the equality is* true. *(e) If $\mathcal{F}(C)$ is a formula $F1 \wedge F2$, and $F1$ and $F2$ are satisfied by E.* □

Definition 7. [ECO-Satisfability of SE] *Let $EO = (EO_1, EO_2, ...EO_n)$ be a list of ECOs such that $\forall\ i, j,\ i < j \Rightarrow Interval(E_i)$ does not follow $Interval(E_j)$. We denote EO a* t-ordered *list of ECOs. A sequential expression $SE=SE_1.SE_2....SE_n$ is satisfied by EO if EO_i satisfies SE_i, $\forall\ i, i = 1..n$. We denote $\mathcal{S}_{L_k}(SE)$ the set composed of the n lists of ECOs that satisfy an SE of length k.* □

Definition 8. [ToI and Normalized ToI] *Let $\mathcal{I}$ be a finite set of items. A* Table Of Items *(ToI) for $\mathcal{I}$ is a table with schema $T = (OID, Items)$, where* Items *is the name of an attribute whose instances are items, and an instance of T is a finite set of tuples of the form $\langle O_j, i_k \rangle$ where $i_k \in \mathcal{I}$ is an item associated to the object O_j. Moreover, given $\langle O_j, i_k \rangle$ and $\langle O_j, i_m \rangle$, two tuples corresponding to the same object, and t_k and t_m the transaction times of the items, then $t_k \neq t_m$ holds. A* normalized ToI *is a database containing a table with schema (OID, t, ID) (the* Normalized ToI*), and one table per category, each one with schema $(ID, attr_1, ..., attr_n, t_s, t_e)$.* □

Figure 3 shows an instance of a normalized ToI where items are related to the category instances of Figure 2. There are three sessions (sequences), $Session_1$, $Session_2$ and $Session_3$, each one with an associated list of items. The three sessions clicked on URL C, but only $Session_1$ would satisfy the constraint [ID = 'C' $\wedge$ $catName$ = 'Books'] (see Figure 2).

Definition 9. [Temporal Matching of a S.E] *Consider a* normalized ToI *(from now on, nToI), with schema (OID, t, ID). An object identified by OID_m* temporally matches *a SE of length k, if there exist k tuples in nToI, $\langle OID_m, t_1, ID_1 \rangle$, $\langle OID_m, t_2, ID_2 \rangle$, ..., $\langle OID_m, t_k, ID_k \rangle$, where for at least one $L_p \in \mathcal{S}_{L_k}(SE)$, $L_p = (eco_1, eco_2, \ldots, eco_k)$, $t_i \in Interval(eco_i), \forall\ i = 1..k$.* □

Example 1. Let us analyze SE = $[ID$ = 'P'$].[$*filter* = 'M'], using Figures 2 and 3. The first *constraint* is satisfied by eco_{P1} and eco_{P2}; the second one, by eco_{C3}. Thus, the t-ordered lists of ECOs that satisfy SE are $L_1 = \{eco_{P1}, eco_{C3}\}$ and $L_2 = \{eco_{P2}, eco_{C3}\}$. The object $Session_1$ *temporally matches* SE, since there exist two different tuples in $Session_1$ whose transaction times belong to $Interval(eco_{P1})$ and $Interval(eco_{C3})$, respectively. With a similar analysis, $Session_2$ *does not match* the SE (eco_{C3} did not exist when the user in this session clicked the last two URLs). Finally, $Session_3$ temporally matches SE. □

Definition 10. [Temporal Satisfability of a Constraint] *Given a constraint C and a normalized ToI, with schema* (OID, t, ID)*, we say that a tuple in nToI* $\mu = \langle OID_m, t_m, ID_m \rangle$ temporally satisfies *C if at least one of the following conditions hold: (a) if* $\mathcal{F}(C)$ *is of the form* t = *'ct' where* t *is a temporal attribute, 'ct' is a temporal constant, and 'ct'* $= t_m$*; (b) if* $\mathcal{F}(C)$ *is of the form* $f_n(t,$ *'ct1', 'ct2', ..., 'ct*$_{n-1}$*')* = *'ct', and* $f_n(t_m,$ *'ct1', 'ct2', ..., 'ct*$_{n-1}$*')* = *'ct' ; (c) if* $\mathcal{F}(C)$ *does not contain a temporal attribute; (d) if* $\mathcal{F}(C)$ *is a formula* $F1 \wedge F2$ *and F1 and F2 are satisfied by* μ*.* □

Definition 11. [Total Matching of a Sequential Expression] *Given a sequential expression SE=*$SE_1.SE_2...SE_k$ *of length k, and a* normalized ToI *with schema* (OID, t, ID)*, we say that an object identified by* OID_m totally matches *SE, if there exists k different tuples* $\mu_1, \ldots, \mu_k$ *in nToI, of the form* $\mu_1 = \langle OID_m, t_1, ID_1 \rangle$, $\mu_2 = \langle OID_m, t_2, ID_2 \rangle$,...,$\mu_k = \langle OID_m, t_k, ID_k \rangle$*, and there is at least one list* $L_p = (eco_1, eco_2, ...eco_k), L_p \in \mathcal{S}_{L_k}(SE)$*, where the following conditions hold: (a)* $t_i \in Interval(eco_i)$, $\forall\ i, i = 1..k$*; (b)* ID_i *is the identifier of the encoded category occurrence* (eco_i), $\forall\ i, i = 1..k$*; (c)* SE_i *is temporally satisfied by* μ_i, $\forall\ i, = 1..k$*. We denote each* L_p *a* list of interest *for SE.* □

From Definition 9, if a list of ECOs does not *satisfy* a sequential expression SE, no object in the nToI can use this list to temporally match SE. Thus, given that the lists in $\mathcal{S}_{L_k}(SE)$ are computed over the category occurrences, which usually fit in main memory, unnecessary database scans can be avoided. In addition, from Definitions 11 and 10, if an object OID_j in $\mathcal{T}$ does not temporally match SE, then it cannot *totally match* SE, and, if an object OID_j in $\mathcal{T}$ that *totally matches* SE, then OID_j *temporally matches* SE.

Example 2. Object $Session_1$ in Example 1, totally matches SE, using the second and third tuples, together with list L_1. On the other hand, $Session_2$ does not totally match SE, since it does not temporally match the expression. Finally, $Session_3$ temporally matches SE, but it does not totally match it, because L_2 does not satisfy the second condition in Definition 11. □

Definition 12 (Temporal Support of SE). *The* temporal support *of a sequential expression SE, denoted* $\mathcal{T}_s(SE)$*, is the quotient between the number of different objects that* totally match SE *and the number of different objects that* temporally match SE*, if the latter is different than zero. Otherwise* $\mathcal{T}_s(SE) = 0$*.* □

5 Temporal Support of Regular Expressions

Definition 13. [R.E. over constraints] *A* regular expression *over constraints is an expression generated by the grammar*

$$E \longleftarrow C \mid E|E \mid E? \mid E^* \mid E^+ \mid E.E \mid E \mid \epsilon$$

where C is a constraint, ϵ is the symbol representing the empty expression, ' $|$ ' means disjunction, ' . ' means concatenation, ' ? ' "zero or one occurrence", ' $+$ ' "one or more occurrences", and ' $*$ ' "zero or more occurrences". □

Since REs produce SEs, extending the definitions of Section 4 using the grammar of Definition 13 is straightforward.

Definition 14 (Temporal Support of a RE). *Given a regular expression $\mathcal{R}$ generated by the grammar of Definition 13, the DFA $\mathcal{A}_\mathcal{R}$ that accepts $\mathcal{R}$, and a normalized ToI with schema (OID, t, ID), we say that OID_m* temporally matches *$\mathcal{R}$, if there exists some $n \in \mathbb{N}$ such that there exists at least one string of length n accepted by $\mathcal{A}_\mathcal{R}$, and OID_m* temporally matches *this string. Moreover, OID_m* totally matches *RE, if there exists some $n \in \mathbb{N}$ such that there is at least one string of length n accepted by $\mathcal{A}_\mathcal{R}$ and OID_m* totally matches *this string.*

The temporal support *of a regular expression $\mathcal{R}$, denoted $\mathcal{T}_r(\mathcal{R})$, is the quotient between the number of different objects that totally match $\mathcal{R}$ and the number of different objects that temporally match $\mathcal{R}$, if the latter is different than zero. Otherwise $\mathcal{T}_r(\mathcal{R}) = 0$.* □

Example 3. Consider the following SEs: SE_1=[keyword='Games'], SE_2 = [keyword='Games'].[filter=''], and SE_3 = [keyword='Games'].[filter=''].[filter=''] (i.e., for each SE_i, $i > 3$, a condition [filter=''] is added). They are produced by the RE $\mathcal{R}$ = [*keyword* = 'Games'].([$filter$ = ''])*. To compute the Temporal Support of $\mathcal{R}$, we need to check the satisfability of all the SEs that the RE can produce. Since no session has four tuples, in our running example we only need to compute the support of SE_1 through SE_3, proceeding like in Example 2. For example, for SE_1, $\mathcal{S}_{L_1}(SE) = \{L_1 = \{eco_{C2}\}, L_2 = \{eco_{C3}\}, L_3 = \{eco_{M1}\}\}$. From the third tuple in $Session_1$, and $L_2 = \{eco_{C3}\}$, we conclude that $Session_1$ totally (and hence, temporally) matches SE_1. Analogously, $Session_2$ and $Session_3$ totally match SE_1. Finally, the temporal support of SE_1 is $3/3 = 1$. Repeating this procedure for all the possible SEs that the RE can produce, we can compute the temporal support of the RE. □

6 Computing the Temporal Support of RE

We propose an algorithm, based on GSP [8] and SPIRIT [2], that generates sequential expressions of incremental length using $\mathcal{A}_\mathcal{R}$. The general idea of the algorithm is the following. At each step k, *candidate lists* of ECOs of length k are produced, using the lists of length k-1 produced in the previous step. Accessing

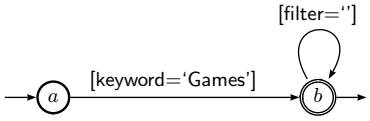

Fig. 4. DFA for $\mathcal{R}$

the ToI, we could check which of these lists of ECOs are temporally and/or totally matched by some object in that table. To prune as early as possible the ECOs that will not produce a match, we use the DFA $\mathcal{A}_\mathcal{R}$. *Building and early pruning these lists of ECOs are the keys of the algorithm.* Each step k is composed of three phases: *Phase 1* consists in building a set CS_k of candidate sequences of ECOs of length k; in *Phase 2*, these CS_k's are pruned using the automaton; in *Phase 3,* the sequences in CS_k that are not temporally matched by any object in nToI are pruned. A sequence eco_{ik} of ECOs (of length k) is pruned if it could not be temporally matched by any object. If at a step p, $CS_p = \emptyset$, or there is no nToI instance with at least p tuples for the same object (i.e., eco_p cannot be matched by any object in the nToI), the algorithm exits the loop. Two further steps are needed. The first one uses $\mathcal{A}_\mathcal{R}$ again, this time to prune all lists obtained in intermediate steps that it does not accept. The final step uses nToI to compute the temporal support of $\mathcal{R}$.

The DFA $\mathcal{A}_\mathcal{R}$, which recognizes the RE $\mathcal{R}$, generates sequential expressions of different lengths, such that each *constraint* in an SE labels an edge of $\mathcal{A}_\mathcal{R}$. We use $\mathcal{A}_\mathcal{R}$ to prune the lists of ECOs produced at each step. Since this automaton typically fits in main memory, the number of accesses to the nToI is reduced. The algorithm accepts, at iteration k, a sequence that matches any sub-path in the automaton (thus, a kind of "relaxed" constraint is used). For example, for $SE = [keyword =$ 'Games'$].([filter =$ ''$])^*$, $\mathcal{A}_\mathcal{R}$ is depicted in Figure 4. At $k = 1$ we accept any sequence that satisfies the constraints $[keyword =$ 'Games'$]$ or $[filter =$ ''$]$ (respectively, paths a-b and b-b in the automaton of Figure 4. In the final phase $\mathcal{A}_\mathcal{R}$ is used for *strict* verification (i.e., to match a path starting and ending in a initial and final state, respectively).

Example 4. Consider the regular expression $\mathcal{R}$=[keyword='Games'].([filter='])*. The algorithm first builds $\mathcal{A}_\mathcal{R}$, the automaton that accepts $\mathcal{R}$, shown in Figure 4. The first phase populates CS_1 with the lists of candidate ECOs that could satisfy an SE of length one (i.e., all the eco_s). In the second phase, we use $\mathcal{A}_\mathcal{R}$ to generate words of length 1, in this case [keyword='Games'] and [filter=']. Then, we prune CS_1 keeping only the lists of ECOs that satisfy any of these words. Note that L_2, L_3 and L_8 are dropped, because neither of them has a 'Games' keyword, nor an empty filter. In the third phase, we access nToI to check which lists of ECOs are not temporally matched by any object. In this example there is no pruning. For example, L_1 is temporally matched by the first tuple of $Session_1$. Figure 5 shows the results at each phase of iteration $k = 1$. The other iterations proceed analogously, with the exception that an apriori verification is performed,

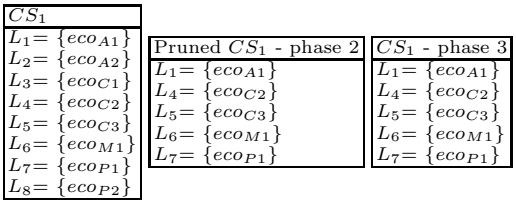

Fig. 5. Initial CS_1 (left), CS_1 pruned with $\mathcal{A}_{\mathcal{R}}$ (center), and pruned with nToI (right).

that is whether or not the lists of ECOs in CS_j are $t - ordered$. For example, for $k = 2$, we check if given two joining lists $L_1 = \{eco_i\}$ and $L_2 = \{eco_j\}$, $L_{12} = \{eco_i, eco_j\}$ is $t - ordered$. All the non-t-ordered lists are pruned, since they cannot produce a match. □

7 Future Work

We expect to extend our work in two ways. On the one hand, the theoretical framework introduced here allows to think in a more general definition of support, with different semantics (not only temporal), that may enhance current data mining tools. On the other hand, we will develop an optimized implementation of the algorithm that can support massive amounts of data.

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the Int'l Conference on Data Engineering (ICDE)*, 1995.
2. M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *Proceedings of the 25th VLDB Conference*, 1999.
3. J. Huang, C. Tseng, J. Ou, and M. Chen. A general model for sequential pattern mining with a progressive database. *IEEE Transactions on Knowledge and Data Engineering*, 20(9):1153–1167, 2008.
4. C. Lee, C. Lin, and M. Chen. On mining general temporal association rules in a publication database. In *ICDM*, pages 337–344, 2001.
5. Y. Li, P. Ning, X.S. Wang, and S. Jajodia. Discovering calendar-based temporal association rules. *Data Knowl. Eng.*, 44(2):193–218, 2003.
6. F. Masseglia, P. Poncelet, and M. Teisseire. Incremental mining of sequential patterns in large databases. *Data Knowl. Eng.*, 46(1):97–121, 2003.
7. S. Parthasarathy, M. J. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and interactive sequence mining. In *CIKM '99*, pages 251–258, 1999.
8. R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, 1996.
9. A. Tansel, J. Clifford, and S. Gadia (eds.). *Temporal Databases: Theory, Design and Implementation*. Benjamin/Cummings, 1993.
10. Abdullah Uz Tansel and Susan P. Imberman. Discovery of association rules in temporal databases. In *ITNG*, pages 371–376, 2007.