HAL
open science

Softening the Blow of Frequent Sequence Analysis: Soft Constraints and Temporal Accuracy

Céline Fiot, Anne Laurent, Maguelonne Teisseire

To cite this version:

Céline Fiot, Anne Laurent, Maguelonne Teisseire. Softening the Blow of Frequent Sequence Analysis: Soft Constraints and Temporal Accuracy. International Journal of Web Engineering and Technology, 2009, Web-based Knowledge Representation and Management, pp.24-47. 10.1504/IJWET.2009.025012 . lirmm-00196964

HAL Id: lirmm-00196964

https://hal-lirmm.ccsd.cnrs.fr/lirmm-00196964

Submitted on 18 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Softening the blow of frequent sequence analysis: soft constraints and temporal accuracy

Céline Fiot*, Anne Laurent
and Maguelonne Teisseire

LIRMM
161 rue Ada, 34080 Montpellier, France
E-mail: fiot@lirmm.fr
E-mail: laurent@lirmm.fr
E-mail: teisseire@lirmm.fr
*Corresponding author

Abstract: Mining temporal knowledge has many applications. Such knowledge can be all the more interesting as some time constraints between events can be integrated during the mining task. Both in data mining and machine learning, some methods have been proposed to extract and manage such knowledge using temporal constraints. In particular, some work has been done to mine Generalised Sequential Patterns (GSPs). However, such constraints are often too crisp or need a very precise assessment to avoid erroneous information. Within this context, we propose an approach based on sequence graphs derived from soft temporal constraints. These relaxed constraints enable us to find more GSPs. We also propose a temporal accuracy measure to provide the user with a tool for analysing the numerous extracted patterns.

Keywords: data mining; sequential patterns; time constraints; fuzzy set theory; temporal accuracy; web mining.

Reference to this paper should be made as follows: Fiot, C., Laurent, A. and Teisseire, M. (2009) 'Softening the blow of frequent sequence analysis: soft constraints and temporal accuracy', *Int. J. Web Engineering and Technology*, Vol. 5, No. 1, pp.24–47.

Biographical notes: Céline Fiot completed a Master's programme in Computer Engineering and Decision Support Systems at the Ecole des Mines de Nantes, France. Since 2004, she has been working on mining approximative frequent sequences from imprecise, uncertain or incomplete data as a PhD student in the Data Mining Group at the Laboratory of Computer Science, Robotics and Microelectronics (LIRMM) in Montpellier, France. She received her PhD in September 2007.

Anne Laurent completed her PhD at the Computer Science Laboratory of the Department of Learning and Knowledge Extraction, University of Paris VI, France, under the supervision of Bernadette Bouchon-Meunier. She has been an Assistant Professor at the Montpellier II University since September 2003, as a member of the Data Mining Group at LIRMM. She is interested in fuzzy data mining, multidimensional databases and sequential patterns, investigating and proposing new methods to tackle the problem of remaining scalable when dealing with fuzziness and complex data.

Copyright © 2009 Inderscience Enterprises Ltd.

Maguelonne Teisseire received her PhD in Computing Science from the Méditerrané University, France, in 1994. Her research interests are focused on behavioural modelling and design. She is currently an Assistant Professor of Computer Science and Engineering at Montpellier II University and Polytech' Montpellier, France. She has been the Head of the Data Mining Group at LIRMM since 2000. Her research is focused on advanced data mining approaches involving time-ordered data.

1 Introduction

The quantity of data from the World Wide Web is growing dramatically: requested Uniform Resource Locators (URLs), number of requests or connexion duration, *etc.*, are gathered automatically by web servers and stored in access log files. Analysing these data can provide useful information for performance enhancement or customer targetting. In this context, many research studies have been proposed to mine usage patterns and user profiles (Spiliopoulou and Faulstich, 1998; Yan *et al.*, 1996; Zaiane *et al.*, 1998). In particular, Masseglia *et al.* (1999) provided knowledge from the databases of visited page sequences. Discovering such patterns can show, for example, that "60% of users visiting the Disneyland website and Eiffel Tower pages visit later traveling websites".

The information thus discovered can often be improved by looking for temporal knowledge (*e.g.*, detection of frauds, failures, behaviour analysis). Some data mining techniques thus aim at extracting recurrent episodes from a long sequence (Mannila *et al.*, 1997) or from sequence bases (Agrawal and Srikant, 1995). Searching for such information becomes all the more interesting that different constraints between events can be taken into account, such as the minimal or maximal duration separating two events (Srikant and Agrawal, 1996; Zaki, 2000; Meger and Rigotti, 2004) or constraints on repetitions (Capelle *et al.*, 2002; Leleu *et al.*, 2003).

Within this framework, Generalised Sequential Pattern (GSP) mining was introduced in Srikant and Agrawal (1996). This data mining technique extracts frequent sequences that meet user-specified time constraints from a sequence database (*e.g.*, successive purchases of customers in a supermarket). The specification of such constraints between events enables the user to select some typical sequences, *e.g.*, repeated access to a website at really close intervals.

However, although these methods are effective and robust, the user has to know the exact constraint values to be specified. Then, there is a risk that erroneous or useless knowledge may be gathered. Moreover, in some cases, these values are somewhat uncertain. Time constraints, as they are defined, thus allow the user to find new sequential patterns, but they are still too stiff. Consequently, it may become necessary to make several attempts with various combinations of these parameters before getting satisfactory results. Meger and Rigotti (2004) proposed to automatically determine the optimal window of observation for repetitive episode mining in a sequence, but this is hardly adaptable to sequential pattern mining. In this domain, to our knowledge, no papers have proposed the automatic determination of the most appropriate time constraints.

Besides, for some applications, it could also be interesting to soften the constraints specified by the experts of the domain to refine their knowledge: the expert knowledge is used as a starting point and mining results complete it. For instance, nonhuman access to webpages can be characterised by repeated requests at short intervals. The task is then to formalise this expert knowledge with time constraints and especially to translate the notion of *short interval* into a crisp duration. Softening time constraints could facilitate this formalisation, such that an approximate value could be specified for a short period (between one and three seconds, for instance). To make the constraint specification easier, we propose a method that softens user-specified time constraints. We also propose an efficient algorithm, *i.e.*, Graph for Extended Time Constraints (GETC), to handle these constraints.

Otherwise, the discovered sequential patterns, according to the specified time constraints, can quickly become so numerous that their analysis becomes less effective. In this regard, a measure that could facilitate the analysis of GSPs would be a valuable tool. We thus propose to provide the end user with a time-satisfaction degree that will indicate how well the user-specified initial constraints are fulfilled.

To tackle several of these problems, here we describe our proposal including the following points. First, we define extended time constraints for GSPs. Secondly, we state a definition of time accuracy for frequent sequence analysis. Lastly, we focus on the algorithm we designed to handle soft time constraints and compute time accuracy.

In the next section, we define the fundamental concepts associated with sequential patterns and GSPs. In Section 3, after briefly introducing the fuzzy set theory, we present the first part of our proposal, *i.e.*, defining the soft time constraints and the temporal accuracy of a sequence. Then, Section 4 details the second part of our work, *i.e.*, the algorithm that implements the handling of soft time constraints. Section 5 develops our proposal with an example. We then propose some experiments on both synthetical data and web access logs in Section 6, thus showing the benefits of these soft time constraints and measures. Finally, we conclude in Section 7 on the prospects opened by our work.

2 Sequential patterns and time constraints

This section defines the concepts used in the GSP mining task. It broadly summarises the formal description of the problem introduced in Agrawal and Srikant (1995) and Srikant and Agrawal (1996) and broadly summarises the various proposals of algorithms to handle such time constraints.

2.1 Sequential patterns

Sequential patterns were initially defined in Agrawal and Srikant (1995) as maximal frequent sequences, as follows.

Let O be a set of objects. Each object o is described by a list of records r consisting of three information elements: an object ID, a record timestamp and a set of items in the record. Let $I = \{i_1, i_2, ..., i_q\}$ be a set of items. An *itemset* is a non-empty non-ordered set of items, denoted by $(i_1 i_2 \ldots i_k)$. A *sequence* s is a non-empty ordered list of itemsets, denoted by $< s_1 s_2 ... s_p >$. An *n-sequence* is a sequence of n items (or of size n).

Example 1 *Let us consider an example of market basket analysis. The object is a customer and the records are the transactions made by this customer. The timestamps are the dates of transactions. If a customer purchases products* 1, 2, 3, 4 *and* 5 *according to the sequence* $s = <(1)\ (2\ 3)\ (4)\ (5)>$*, then all the items of the sequence were bought separately, except products* 2 *and* 3*, which were purchased at the same time. In this example, s is a 5-sequence.*

One sequence $<s'_1 s'_2 \ldots s'_m>$ is a *subsequence* of another one $<s_1\ s_2 \ldots s_p>$ if there are integers $l_1 < l_2 < \ldots < l_m$ such that $s'_1 \subseteq s_{l_1}, s'_2 \subseteq s_{l_2}, \ldots, s'_m$. We should also mention that s' is *included* in s.

Example 2 *The sequence* $s' = <(2)\ (5)>$ *is a subsequence of the s in Example 1, because* $(2) \subseteq (2\ 3)$ *and* $(5) \subseteq (5)$*. However,* $<(2)\ (3)>$ *is not a subsequence of s.*

All records from the same object are grouped together and sorted in increasing order according to their timestamp. They are called a data sequence. In order to efficiently aid decision making, the aim is to discard nontypical behaviours according to the user's viewpoint. Performing such a task requires allocating any data subsequence in O with a frequency value $freq(s)$. The *frequency* of a sequence is defined as the percentage of objects supporting s with respect to the number of objects in the database. An object *supports* a sequence s if s is included within the data sequence of this object.

In order to decide whether a sequence is frequent or not, a minimum frequency value *minFreq* is specified by the user and the sequence is said to be frequent if the condition $freq(s) > minFreq$ holds. Given a database of object records, the goal of sequential pattern mining is to find all the maximal sequences whose frequency is greater than a specified threshold (*minFreq*) (Agrawal and Srikant, 1995). Each of these sequences represents a sequential pattern, also called a maximal frequent sequence.

This sequence definition is rather strict and turns out to be inappropriate for many applications because time constraints are not handled. When verifying whether a candidate sequence is included within another one, record partitioning enforces a strong constraint since only pairs of itemsets are compared. However, if the interval between two records of an object is short enough, they could be considered simultaneous. On the contrary, two events that are too distant could have no link together. That is why GSPs were proposed in Srikant and Agrawal (1996), introducing time constraints in order to improve the subsequence definition.

2.2 Generalised sequential patterns

Time constraints restrict the time gap between sets of records that contain consecutive elements of the sequence. There are three different constraints. First, *minGap* is the minimal time gap that *must* separate two consecutive itemsets in a sequence. Then, *maxGap* is the maximal time gap within which two consecutive itemsets of a sequence *must* occur. Finally, *windowSize* is a sliding window during which several records *may* be grouped into one itemset. Handling time constraints, Srikant and Agrawal (1996) redefined when a data sequence supports a sequence.

Definition 1 *Given user-specified windowSize, minGap and maxGap values, a data sequence* $d = < d_1 \ldots d_m >$ *supports a sequence* $s = < s_1 \ldots s_n >$ *if there exist integers* $l_1 \leq u_1 < l_2 \leq u_2 < \ldots < l_n \leq u_n$ *such that:*

i $s_i \subset \cup_{k=l_i}^{u_i} d_k,\ 1 \leq i \leq n$

ii $timestamp(d_{u_i}) - timestamp(d_{l_i}) \leq windowSize,\ 1 \leq i \leq n$

iii $timestamp(d_{l_i}) - timestamp(d_{u_{i-1}}) > minGap,\ 2 \leq i \leq n$

iv $timestamp(d_{u_i}) - timestamp(d_{l_{i-1}}) \leq maxGap,\ 2 \leq i \leq n.$

We will refer to timestamp(d_{l_i}) as *start-time*(s_i) and timestamp(d_{u_i}) as *end-time*(s_i). In other words, *start-time*(s_i) and *end-time*(s_i) correspond to the first and last timestamps of the set of records that contains s_i. These time constraints, as well as the minimum frequency condition, are parameterised by the user.

Time constraints allow a more flexible handling of records, insofar as the end user is then provided with the following advantages for mining sequences:

- to group together itemsets when their timestamps are sufficiently close via the *windowSize* constraint
- to regard itemsets as too close to appear in the same frequent sequence with the *minGap* constraint (*i.e.*, to be considered related)
- to regard itemsets as too distant to appear in the same frequent sequence with the *maxGap* constraint (*i.e.*, to be considered related).

Example 3 *Consider logs of a website requiring identification. Id1 and Id2 are two identified users and Table 1 describes their various browsing sessions. The session sequence of each user is a data sequence.*

Table 1 Browsing sessions of two web users

User Id	*Hour 1*	*Hour 2*	*Hour 3*	*Hour 4*	*Hour 5*	*Hour 6*	*Hour 7*
Id1	1	2	3 4		5 6 7	8	9
Id2	1	2 3 4	5 6	7 8			

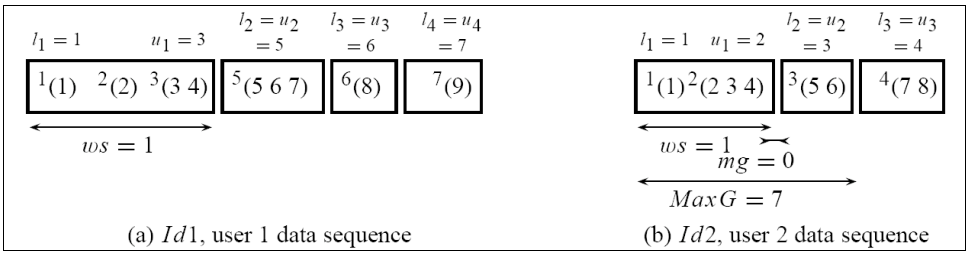
Let $s = <(1\ 2\ 3\ 4)>$ *be a sequence and the following time constraint parameters:*

- *minGap* = 0*, consecutive itemsets must have a distance of at least 1 h*
- *maxGap* = 7*, consecutive itemsets must have a maximum distance of 7 h*
- *windowSize* = 1*, accesses may be grouped together over at most two consecutive hours.*

This means that when analysing profiles, two consecutive hours may be considered one and that two consecutive sessions must happen within a seven-hour period to be considered related. The numbers 1 to 9 represent the website URLs.

Then, Figure 1 shows how these time constraints are applied in order to determine whether data sequences Id1 and Id2 support the candidate sequence s = <(1 2 3 4)> or not.

Figure 1 The description of time constraints, *windowSize* (*ws*), *minGap* (*mg*) and *maxGap* (*MaxG*), on data sequences Id1 and Id2



(a) $Id1$, user 1 data sequence (b) $Id2$, user 2 data sequence

Note: $^i(a\ b)$ denotes the itemset $(a\ b)$ browsed at hour i.

To make sequence s appear in data sequence Id1, the sessions of the first, second and third hour must be grouped together. However, this itemset does not meet constraint (ii), since end-time(s_1) – start-time(s_2) = 3 – 1 = 2 > windowSize. There are no other possibilities to find s in this data sequence. Thus, data sequence Id1 does not support sequence s.

To make sequence s appear in data sequence Id2, the sessions of the first and second hour must be grouped together. This itemset meets the windowSize constraint, since it was built over two consecutive hours. The minimum gap between this first itemset and the next is then hour 3 – hour 2 = 1 > 0 = minGap, which meets the minGap constraint (iii). So does the maxGap constraint (iv). The data sequence Id2 supports sequence s.

Note that if the specified values are *minGap* = 0, *maxGap* = ∞ and *windowSize* = 0, we get back the notion of sequential patterns, as introduced in Section 2.1, where there are no time constraints and where the items in an itemset come from a single record.

2.3 Related work

Various algorithms were proposed to handle these constraints. Some push them directly into the mining process, like the GSP algorithm (Srikant and Agrawal, 1996) and the DELISP algorithm (Lin *et al.*, 2002). In contrast, some others propose a preprocess to apply the constraints to the sequences, which are then analysed by some sequential pattern tool. The Graph for Time Constraints (GTC) algorithm, proposed in Masseglia *et al.* (2004), is based on this principle.

The GSP algorithm proposed in Srikant and Agrawal (1996) aims to mine GSPs. It extends previous proposals for sequence mining by handling time constraints and taxonomies (*is-a* hierarchies). It uses a generate-and-prune approach that uses the frequent sequences of size k to generate candidate sequences of size $k + 1$. Then, the frequency of these $(k + 1)$ sequences is calculated. Time constraints are handled when parsing a data sequence. For each candidate sequence, the GSP algorithm checks whether it is contained in the data sequence. Because of the sliding windows and minimum and maximum time gaps, it is necessary to switch between forward and backward phases during examination. Forward phases are performed to deal progressively with items and, while selecting items, *windowSize* is used for resizing records partitioning. Backward phases are required as soon as the *maxGap* constraint is no longer fulfilled. In such a case, it is necessary to discard all items for which the *maxGap* constraint is violated and to resume parsing the sequence, starting with the earliest item meeting the *maxGap* condition.

In Masseglia *et al.* (1998), another approach called Prefix Tree for Sequential Patterns (PSP) was proposed. It again fully utilises the fundamental principles of GSP while using a different structure for organising the candidate sequences, which thus improves retrieval efficiency.

More recently, the DELISP algorithm (Lin *et al.*, 2002) was proposed for mining sequential patterns with time constraints. It is based on the mining scheme of PrefixSpan. Actually, the original database is divided into multiple subsets for each prefix of a potential sequential pattern. While writing the subsets, DELISP reduces the size of the projected databases by *bounded* and *windowed* projection techniques. The experiments proposed by the authors show a clear improvement of DELISP over GSP. However, this technique is restricted to prefix-growth algorithms.

Therefore, the GTC algorithm (Masseglia *et al.*, 2004) was developed. It improved the PSP approach by its more efficient handling of time constraints. The GTC algorithm, based on a data sequence, precalculates a relevant set of sequences to be tested. By precalculating this set, the time spent analysing a data sequence when verifying candidate sequences is reduced. The efficiency of this approach was demonstrated.

We need an efficient implementation to handle our extended time constraint. Moreover, this implementation should not be restricted to one sequential pattern mining approach. For these reasons, we decided to develop the implementation of our soft time constraints introduced in Section 3 based on the GTC principle. This implementation is detailed in Section 4, where we present our GETC algorithm.

3 Soft time constraints

The main drawback of the previously described time constraints is that they are user-specified. They require the data and constraint values to be *a priori* well-known. The results thus depend on the good knowledge of the end user. Misvalued time constraints could indeed lead to erroneous or incomplete knowledge. However, to our knowledge, no studies have been proposed to automatically determine the optimal time constraints for sequential pattern mining. We propose extending the time constraints above for GSPs using some fuzzy set theory principles.

Moreover, extracted patterns become increasingly numerous, particularly during sequential pattern mining. It becomes necessary to provide the end user with tools to analyse the obtained sequential patterns. In the case of GSPs, some useful information could be derived from the duration of data sequences corresponding to time constraints. This is the purpose of soft time constraints, which we define in this section.

These soft time constraints enable us to define a measure of temporal accuracy expressing how well a sequence fulfils the initial user-specified values of time constraints.

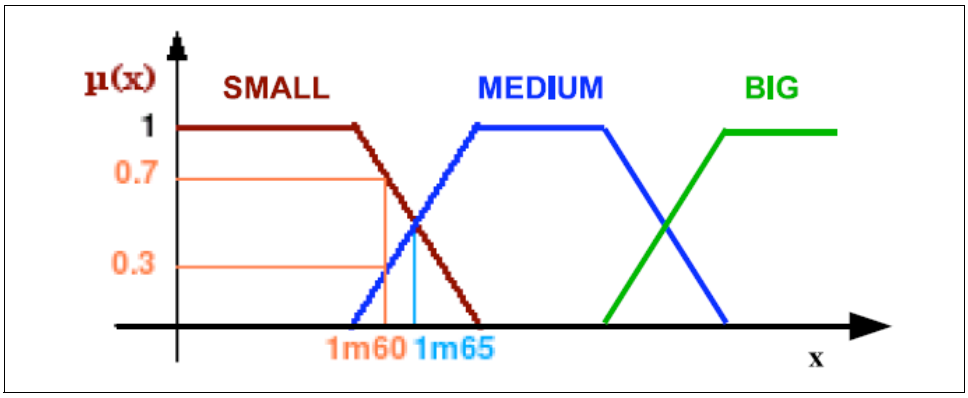
We thus provide the user with flexibility in time constraint specification and a tool to help in analysing the extracted patterns.

3.1 Fuzzy set theory

The fuzzy set theory was introduced by Zadeh (1965). This theory generalises crisp set theory and assumes intermediary situations between all and nothing. Whereas in the classical theory, an element a belongs or not to a set A, in fuzzy set theory, a may partially belong to A (then called a fuzzy set) and, thus, partially belong to its complement. Besides enabling this partial membership, fuzzy set theory allows a gradual transition of an object from one state to the next.

Example 4 *Let X be the universe of all possible sizes for a human being. One fuzzy set A (e.g.,* small, medium *or* big*) is defined by a membership function* μ_A *expressing for every x of X the degree with which x belongs to A. This degree is in interval* [0,1]. *Examples of these three fuzzy sets is graphically represented in Figure 2. Thus, a person of height* $x = 1\text{m}60$ *can be simultaneously small and medium-sized with, for example, a degree of* 0.7 *for the fuzzy set* small *(*$\mu_M(x) = 0.7$*) and a degree of 0.3 for the fuzzy set* medium *(*$\mu_M(x) = 0.3$*).*

Figure 2 Big and small fuzzy sets describing a person's height (see online version for colours)



Fuzzy logic operators are a generalisation of crisp logic operators. In particular, we consider negation, intersection and union. The operator $\top$ or t-norm operator (triangular norm) is the fuzzy equivalent of the binary intersection $\mu_{A\cap B}(x) = \top(\mu_A(x), \mu_B(x))$. The operator $\perp$ or t-conorm operator (triangular conorm) is similar to the binary union:

$\mu_{A\cup B}(x) = \perp(\mu_A(x), \mu_B(x))$. We denote $\overline{\top}$ (resp. $\underline{\perp}$) as operator $\top$ (resp. $\perp$) generalised to the n-ary case. Different operators can be used as a t-norm (*min*, *product*, *etc.*). They are associated with their dual operator for the t-conorm (*e.g.*, *max* is the t-conorm for the *min* t-norm). As the *min* operator is idempotent, we use it for the t-norm and, consequently, the *max* operator will be our t-conorm.

3.2 Principles and notations

Our proposal of soft time constraints for sequential patterns is built by analogy with fuzzy sets. Thus, a sequence will no longer meet a constraint in a binary way because the user may relax these time constraints. Each constraint can then be regarded as a fuzzy set, with its membership function giving a temporal satisfaction degree. This degree is thus calculated for each possible value of that time constraint and tells the end user the extent to which the initially specified constraint value has been fulfilled.

In order to fulfil the user's needs and make our approach flexible, a minimum temporal satisfaction degree ρ_x can be specified for constraint X of initial value x_{init}.

As the satisfaction degree of constraints is based on the membership function of each constraint, specified coefficients are in the interval [0,1]. It is also possible to set a constraint with certainty:

- If $\rho_x = 0$, the user wants constraint X to take each possible value; the temporal satisfaction degree will depend on the constraint value generating the sequence.
- If $\rho_x = 1$, the specified minimum temporal satisfaction degree is 1, *i.e.*, the user does not want the value of constraint X to vary. That constraint will be set at the initial value and will not change; all generated sequences will have a temporal satisfaction degree equal to 1.
- In all other cases, $\rho_x \in]0,1[$ and the value x of constraint X will vary from its user-specified value x_{init} and a limit value x_ρ, for which the temporal satisfaction degree is $\rho(x) = \rho_x$.

Note that if the specified values for the minimum satisfaction degree of each time constraint is 1, we get back crisp time constraints and, thus, the notion of GSPs as introduced in Section 2.2.

The useful limit values of time constraints (extreme values) first have to be determined. These values correspond to the variation on the whole search space (*i.e.*, for $\rho_x = 0$) and are computed from crisp time constraints (ii), (iii) and (iv) in Section 2.2. They are given by the limit value allowed by these definitions.

The *windowSize* and *maxGap* constraints define the maximal gap between two itemsets. For any given object, the maximal value they can be set at is thus the duration between the first and last record. For the whole database, this common extreme value will thus be the maximal gap, over all objects, between the minimal and maximal record timestamp for the same object:

$$M = \max_{o\in O}(timestamp(r_{o_{max}}) - timestamp(r_{o_{min}})).$$

The *minGap* constraint defines the minimal gap between two consecutive itemsets. We have defined the limit value of this constraint by taking the crisp inequality it implies into account. As explained in Fiot *et al.* (2006a), the limit value for *minGap* is given by:

$$m = \max(\min_{o \in O}(\min_{r \in R_o}(timestamp(r+1) - timestamp(r))) - 1, 0).$$

In the remainder of this section, we use the following notations to distinguish the three time constraints. Parameters ws_{init}, mg_{init} and MG_{init} are the initial user-specified values for the constraints *windowSize*, *minGap* and *maxGap* and ρ_{ws}, ρ_{mg} and ρ_{MG} are the minimum temporal satisfaction degrees associated with them. These coefficients enable the user to limit the time constraint variation according to his/her own requirements. The identifier *ws* (resp. *mg* or *MG*) denotes the variable associated with the *windowSize* (resp. *minGap* or *maxGap*) constraint and $\rho(ws)$ (resp. $\rho(mg)$ or $\rho(MG)$) denotes the satisfaction degree obtained by the value *ws* (resp. *mg* or *MG* values) of this variable.

3.3 Extending time constraints to soft time constraints

3.3.1 Soft windowSize

The value *ws* of the *windowSize* constraint may vary from its user-specified value ws_{init} to its limit value *M*. This soft constraint is described with a fuzzy set for which the membership function (Equation 1) gives the accuracy for a specific value *ws*:

$$\rho(ws) = \begin{cases} 1 & \text{if} \quad ws \le ws_{init} \\ \dfrac{1}{ws_{init} - M} ws - \dfrac{M}{ws_{init} - M} & \text{if} \quad ws_{init} < ws \le M. \\ 0 & \text{else} \end{cases} \tag{1}$$

The user can then choose to allow the temporal satisfaction degree of the *windowSize* constraint to be somewhere between 1 and the lowest acceptable value ρ_{ws}. This lowest degree will be attained for a value $ws_\rho > ws_{init}$ of *ws*.

More specifically, this largest acceptable window size is given by:

$$ws_\rho = \lfloor (ws_{init} - M)\, \rho_{ws} + M \rfloor. \tag{2}$$

Example 5 *Consider the two data sequences shown in Example 3 and the sequence* $s = <(1\ 2\ 3\ 4)>$. *We have* $M = \max((7-1), (4-1)) = 6$. *Suppose that the user-specified time constraint parameters are:*

- $ws_{init} = 1$ *with* $\rho_{ws} = 0.7$, *windowSize varies; applying Equation (2), we get* $ws_\rho = \lfloor (1-6) * 0.7 + 6 \rfloor = 2$. *The value ws of windowSize will successively be 1 and 2*
- $mg_{init} = 0$ *with* $\rho_{mg} = 1$, *then mg does not vary*
- $MG_{init} = 7$ *with* $\rho_{MG} = 1$, *then MG does not vary.*

For ws = 1, *grouping the sessions of the first to third hour of data sequence Id1 in order to accept the candidate sequence s would violate the windowSize constraint. However, with ws* = 2, *we can indeed group them and this ws value, by Equation (1), yields a temporal satisfaction degree* $\rho(ws) = \rho(2) = 0.8$. *The other constraints are fulfilled as well, then data sequence Id1 supports candidate sequence s. For data sequence Id2, the sessions of the first and second hour are grouped*

together. Note that the windowSize constraint is satisfied with $ws = ws_{init} = 1$ *and the corresponding satisfaction degree is* $\rho(ws) = 1$. *The other constraints are also respected and data sequence Id2 supports sequence s.*

3.3.2 Soft maxGap

Like the *windowSize* constraint, the *maxGap* constraint value *MG* may vary from its user-specified value MG_{init} and its limit value *M*.

This soft constraint is described with a fuzzy set whose membership function (Equation 3) gives the accuracy for a specific value *MG*:

$$\rho(MG) = \begin{cases} 1 & \text{if } MG \le MG_{init} \\ \dfrac{1}{MG_{init} - M} MG - \dfrac{M}{MG_{init} - M} & \text{if } MG_{init} < MG \le M. \\ 0 & \text{else} \end{cases} \tag{3}$$

The user can then choose to allow the temporal satisfaction degree of the *maxGap* constraint to be somewhere between 1 and a lowest acceptable value ρ_{MG}. This lowest degree will be attained for a value $MG_\rho > MG_{init}$ of *MG*.

More specifically, this largest acceptable gap is given by:

$$MG_\rho = \lfloor (MG_{init} - M)\, \rho_{MG} + M \rfloor. \tag{4}$$

Example 6 *Consider the two data sequences shown in Example 3 and the candidate sequence* $s = \langle (1\ 2\ 3\ 4)\ (7\ 8) \rangle$. *Suppose that the user-specified time constraint parameters are:*

- $ws_{init} = 2$ *with* $\rho_{ws} = 1$, *then ws does not vary.*
- $mg_{init} = 0$ *with* $\rho_{mg} = 1$, *then mg does not vary.*
- $MG_{init} = 3$ *with* $\rho_{MG} = 0.3$, *then maxGap is the only varying constraint. Applying Equation (4) to* $M = 6$, *we get* $MG_\rho = 5$. *The value MG will successively be 3, 4 and 5.*

For data sequence Id1, sessions of the first to third hour, as well as the fifth and sixth hour, are grouped together in order to accept candidate sequence s. However, with $MG = 3$, *the maxGap constraint is violated, as with* $MG = 4$. *With* $MG = 5$, *this constraint is satisfied and the temporal satisfaction degree is then* $\rho(\mathrm{MG}) = \rho(5) = 0.3$ (3). *The other constraints are also fulfilled, then the data sequence Id1 supports candidate sequence s. For data sequence Id2, the records of the first and second hour are grouped together, meeting the maxGap constraint with* $MG = MG_{\mathrm{init}} = 3$ *and the corresponding satisfaction degree is* $\rho(MG = 3) = 1$. *The other constraints are also fulfilled and data sequence Id2 supports sequence s.*

3.3.3 Soft minGap

The value *mg* of the *minGap* constraint may vary from its limit value *m* to its user-specified value mg_{init}. This soft constraint is described with a fuzzy set whose membership function (Equation 5) gives the accuracy for a specific value *mg*:

$$\rho(mg) = \begin{cases} 1 & \text{if} \quad mg \geq mg_{init} \\ \dfrac{1}{mg_{init} - m} mg - \dfrac{m}{mg_{init} - m} & \text{if} \quad mg_{init} > mg \geq m. \\ 0 & \text{else} \end{cases} \tag{5}$$

The user can then choose to allow the temporal satisfaction degree of the *minGap* constraint to be somewhere between the lowest acceptable value ρ_{mg} and 1. This lowest degree will be attained for a value $mg_{\rho} < mg_{init}$ of *mg*.

More specifically, this smallest acceptable gap is given by:

$$mg_{\rho} = \lceil (mg_{init} - m)\, \rho_{mg} + m \rceil. \tag{6}$$

Example 7 *Consider the two data sequences shown in Example 3 and the candidate sequence s* = <(1 2 3 4) (5 6)>. *Suppose that the user-specified time constraint parameters are:*

- $ws_{init} = 2$ *with* $\rho_{ws} = 1$, *then ws does not vary.*
- $mg_{init} = 2$ *with* $\rho_{mg} = 0$, *then minGap is the only varying constraint. Applying Equation (6) to m* = 0, *we get* $mg_{\rho} = 0$. *The value mg will successively be* 2, 1 *and* 0.
- $MG_{init} = 7$ *with* $\rho_{MG} = 1$, *then MG does not vary.*

For data sequence Id1, the sessions of hours 1, 2 and 3. The minGap constraint is not fulfilled with mg = 2, *but it is with mg* = 1. *In this case, the satisfaction degree for minGap is* $\rho(mg) = \rho(1) = 0.5$ *(Equation 5). The other constraints are fulfilled, so data sequence Id1 supports sequence s. For the second data sequence, the sessions of hours 1 and 2. The minGap constraint is not fulfilled while mg is greater than* 0. *mg* = 0 *yields, by Equation (5), a temporal satisfaction degree* $\rho(mg) = \rho(0) = 0$. *The other constraints are also satisfied and data sequence Id2 supports s.*

Note that these soft constraints are defined in Sections 3.3.1, 3.3.2 and 3.3.3 by using fuzzy sets where the temporal satisfaction degree is described by a linear membership function between the initial constraint value and its extreme value *M* or *m*. However, these functions could also be defined in a different way, *e.g.*, by a step function or by a function representing the proportion of objects in the dataset meeting each constraint value.

3.4 Temporal accuracy of a sequence

We now define the level of time constraint satisfaction for a sequence considering constraints (ii), (iii) and (iv) together. At the end of the mining task, we get a list of frequent sequences. For each object, each frequent sequence has been generated using specific time constraint values *ws*, *mg* and *MG*. These values are used to compute the satisfaction degree of each constraint. These satisfaction degrees are then combined into a global measure associated with the sequence.

For an object *o*, the *temporal accuracy of a sequence s* is defined as the satisfaction degree yielded by the three time constraints considered simultaneously. It is calculated using a t-norm operator ($\top$). For each object, several occurrences of *s* may appear. The occurrence satisfying the most the initial values (*i.e.*, with the highest temporal satisfaction degree) is searched through the set ς_o of subsequences of *o* using a t-conorm operator ($\perp$).

We define the temporal accuracy of a sequence $s = <s_1 \ldots s_n>$ for object *o* by the following equation:

$$\begin{aligned}\varrho(s, o) = \underline{\perp}_{s \in \varsigma_o}(&\overline{\top}_{i \in 1,n}(\rho_{ws}(\textit{end-time}(s_i) - \textit{start-time}(s_i))),\\ &\overline{\top}_{i \in 2,n}(\rho_{mg}(\textit{end-time}(s_i) - \textit{start-time}(s_{i-1})),\\ &\rho_{MG}(\textit{end-time}(s_i) - \textit{start-time}(s_{i-1})))).\end{aligned} \tag{7}$$

For the whole dataset, the temporal accuracy of a sequence *s* is given by the average aggregation of each object accuracy, *i.e.*:

$$\Upsilon(s) = \frac{1}{|O|}\sum_{o \in O} \varrho(s, o). \tag{8}$$

Example 8 *Consider the two data sequences from Example 3 and the frequent sequence* $s = <(1\ 2\ 3\ 4)\ (5\ 6)>$ *with the following parameters for soft constraints:* $ws_{init} = 1$, $mg_{init} = 2$, $MG_{init} = 4$ *and* $\rho_{ws} = 0.6$, $\rho_{mg} = 0.4$ *and* $\rho_{MG} = 0.5$. *We still have* $M = 6$ *and* $m = 0$. *We use the min and max operators for the generalised t-norm* ($\overline{\top}$) *and the generalised t-conorm* ($\underline{\perp}$), *respectively. For data sequence Id1, s appears by grouping together the first to third hour on the one hand and the fifth and sixth hour on the other. Then,* $\textit{start-time}(s_1) = 1$, $\textit{end-time}(s_1) = 3$, $\textit{start-time}(s_2) = 5$ *and* $\textit{end-time}(s_2) = 6$. *It is the single occurrence of s in this data sequence. Thus, for Id1, the temporal accuracy of s is (details omitted):*

$$\begin{aligned}\varrho(s, Id1) &= \min(\rho_{ws}(2), \rho_{ws}(1), \min(\rho_{mg}(1), \rho_{MG}(5)))\\ &= \min(0.8, 1, \min(0.5, 0.5))\\ &= 0.5.\end{aligned}$$

Then, the same computation is done for data sequence Id2. Similarly, we get $\varrho(s, Id2) = 0.5$. *The temporal accuracy of sequence s for the whole database is thus given by:*

$$\Upsilon(s) = \frac{\varrho(s, Id1) + \varrho(s, Id2)}{2} = 0.5.$$

4 Graph for extended time constraints

As we wanted our implementation of soft time constraints to be efficient and fully compatible with any sequential pattern mining approach, we developed our GETC algorithm using the same kind of data structure as the GTC algorithm proposed in Masseglia *et al.* (2004) and detailed in Section 2.3. The main idea is to transform the data sequence of an object into a sequence graph in which each path is a subsequence that fulfils the time constraints. The sequence graphs of the data sequences are then used to determine the frequent sequences by a sequential pattern mining algorithm.

Since the handling of time constraints is done prior to and separate from the counting frequency step of a data sequence, we propose to use this method to implement the soft time constraints. The graph structure will thus be used both for sequential pattern mining and for computing the temporal accuracy of sequential patterns in a second step.

4.1 General strategy of the algorithm

Our approach includes all the fundamental principles of GTC. It contains a number of iterations. Each iteration finds all the frequent sequences of the same size. GETC is used as a preprocess for handling soft time constraints. Once a data sequence has been transformed into a sequence graph that fulfils the soft time constraints, frequent sequences are searched within the subsequence set of the sequence graph. As a result, using the sequence graph, checking the time constraints becomes useless during candidate parsing, *i.e.*, only inclusion must be verified. Once the sequential patterns are extracted, the sequence graphs are weighted, then explored one last time to calculate the temporal accuracy of each GSP.

4.2 Sequence graph building

From an input data sequence *d*, the GETC algorithm (Algorithm 2) builds a sequence graph *Gd*(*V*, *E*) in which vertices are itemsets and paths represent the subsequences fulfilling the time constraints. First, each itemset of the input sequence is associated with a vertex. Then, the subfunction *addWindowSize* combines records in an attempt to meet the soft *windowSize* constraint. It adds to the graph any satisfactory combination as a new vertex. Vertices are allocated to 'levels' according to their *end-time* in order to reduce the time spent in checking gap constraints. The next step consists of building the edges that fulfil both the *minGap* and *maxGap* soft constraints. For each vertex, the first 'level' of vertices meeting the soft *minGap* constraint is thus retrieved. For each vertex of this set, the *minGap* constraint is fulfilled and the *maxGap* constraint is checked. If it is fulfilled, a new edge is built between both vertices. Some optimisation is done by the *addEdge* and *propagate* subfunctions to reduce the number of sequence inclusions. Finally, the remaining included subpaths are deleted from the graph by the subfunctions *pruneMarked* and *convertEdges*. All of these subfunctions are detailed in Fiot *et al.* (2006a).

We have proven in Fiot *et al.* (2006a) that at the end of this process, GETC has built exactly all the longest sequences, fulfilling the soft time constraints *windowSize*, *minGap* and *maxGap* generated from the input data sequence. The GETC algorithm can

thus be used as a preprocessing phase to handle soft time constraints before sequential patterns are mined. After this step, the candidate sequence support is computed on these sequence graphs.

4.3 Temporal accuracy computation

Once the sequence graphs have been built, we know which sequences are allowed by the time constraints and which are forbidden. However, some sequences fulfil crisp constraints while others are built only by applying soft constraints. Thus, their 'quality' is not the same. Therefore, we propose to calculate the temporal accuracy level of each longest path of the sequence graph (each maximal sequence) and to allocate it to each subsequence composing it.

Algorithm 1 GETC

```
GETC - Input: d, a data sequence
       Ouput: G_d(V, E), d graph sequence,
              S vertice set of G_d, A edge set

S ← buildVertices(d);
addWindowSize(S) ;
While (x ≠ S.first()) do
  l ← x.level().prec() ; mg ← mg_init ;
  While (x.start-time() - l.end-time() ≤ mg) do
    contmg ← FALSE ;
    If (x.start-time() > l.end-time()) Then
      While (mg ≥ mg_ρ) do
        If (constming(x,l)) Then
          contmg ← TRUE ;
          mg ← mg_ρ-1 ;
        Else
          mg - - ;
        End If
      End While
    End If
    If (contmg == FALSE) Then
      propagate(x,l) ; l ← l.prec() ;
    End If
  End While
  For chq w ∈ l do
    included ← TRUE ;
    MG ← MG_init ;
    While (MG ≤ MGρ) do
      If (constMaxG(x,w)) Then
        addEdge(w,x) ;
        MG ← MG_ρ + 1;
      Else
        MG++;
      End If
    End While
  End For
  x ← S.next(x) ;
End While
pruneMarked(G_d(S, A)) ;
convertEdges(G_d(S, A)) ;
return G_d(S, A);
```

In order to determine the time constraint values satisfied by the paths in the graph, each edge (x,y) is weighted by $\top(\mu_{mg}(y.\text{begin}()-x.\text{end}()),\mu_{MG}(y.\text{end}()-x.\text{begin}()))$ depending on the *mg* and *MG* values used to build this edge; each vertex is similarly weighted by μ_{ws}. These weights are computed by the *valueGraph* function detailed in Fiot *et al.* (2006a). The temporal accuracy of a sequence is then given by Equation (5) in Section 3.4. This computation requires an additional iteration after sequential pattern mining to return each of them with its temporal accuracy.

5 A short example

Consider the dataset in Table 2 (from the data, M = 17 and m = 0) and the following parameters for soft time constraints:

- for *windowSize*, $ws_{init} = 2$ and $\rho_{ws} = 0.86$, then $ws_{\rho} = 4$
- for *maxGap*, $MG_{init} = 4$ and $\rho_{MG} = 0.84$, then $MG_{\rho} = 6$
- for *minGap*, $mg_{init} = 2$ and $\rho_{mg} = 0.5$, then $mg_{\rho} = 1$.

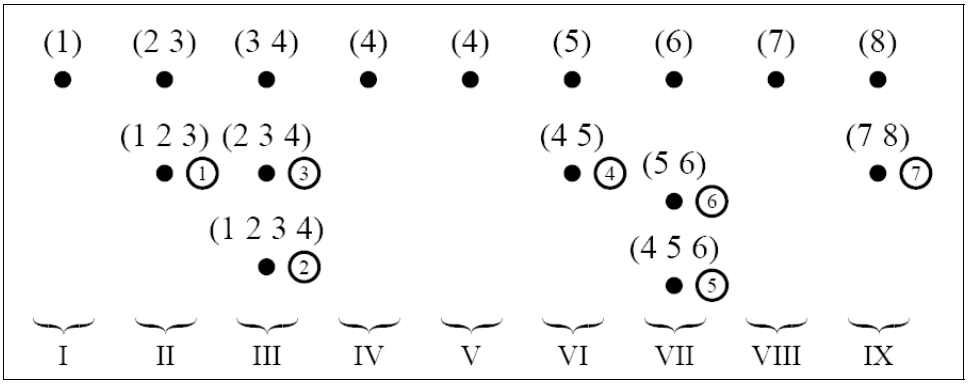
Table 2 The dataset

Timestamp	*1*	*3*	*4*	*5*	*6*	*8*	*9*	*10*	*12*	*17*	*18*
Id1	1	–	2 3	3 4	4	4	–	5	6	7	8
Id2	2 3	4	–	–	5	6	–	–	–	–	–
Id3	1 2	–	3	3 4	4	–	5 6	–	–	–	–

5.1 Sequence graph building

The first step consists of building the sequence graph for data sequence Id1. First, the vertex set is initialised: each record is associated with one vertex. This is the first line of the graph in Figure 3. Then, the *windowSize* constraint is applied on each possible combination of vertices using *addWindowSize*. Only the combinations fulfilling the soft *windowSize* constraint (*i.e.*, *end-time*($O1_i$)-*start-time*($O1_i$) $\leq ws_{\rho} = 4$) are kept.

Figure 3 The sequence graph for *O*l at the end of vertex set creation by *addWindowSize*



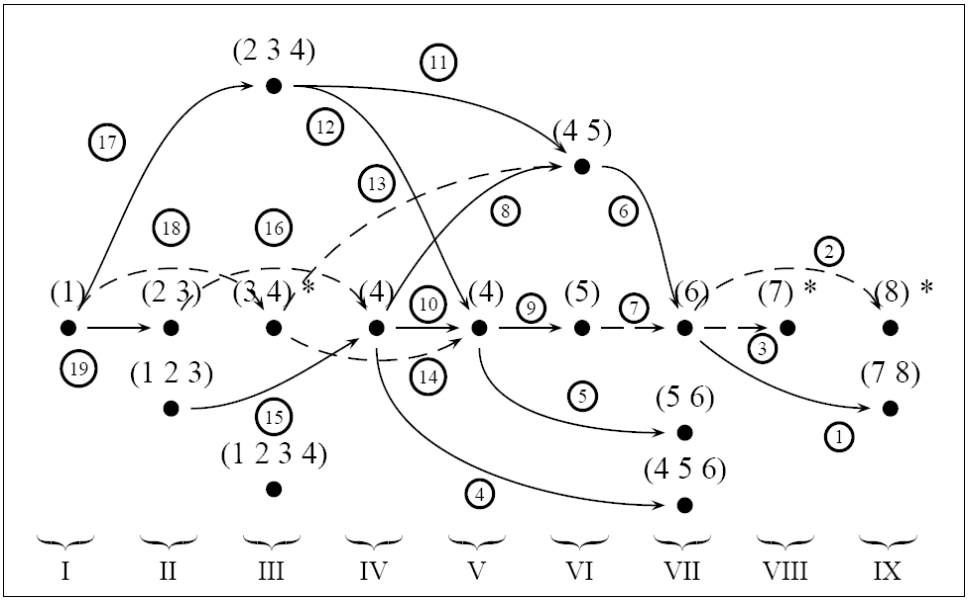
Notes: ⊙ denotes the building order.
VI denotes the sixth end-time 'level'.

Then, the edges fulfilling both the *minGap* and *maxGap* soft constraints are added to the graph using the main function and the *propagate* and *addEdge* subfunctions. The building of edges starts with the last vertex (7 8). The first level that can access (7 8) that fulfils *minGap* is Level VII, then we build an edge for each vertex in this level if *maxGap* is fulfilled. The first edge is then from (6) to (7 8).

When a level cannot attain a vertex *v* because of *minGap*, we need to check if the vertices of this level can access the vertices that are successors of *v*. This is done by the function *propagate*.

After this step, every subsequence of the initial data sequence meeting the three soft constraints is in the graph in Figure 4.[1]

Figure 4 The sequence graph for $O1$ after edge creation, showing the GETC edge creation order



However, some inclusions[2] may remain. The last step consists of deleting these inclusions using the *pruneMarked* subfunction. Once valued, the final sequence graph obtained from data sequence $O1$ is described by Figure 5.

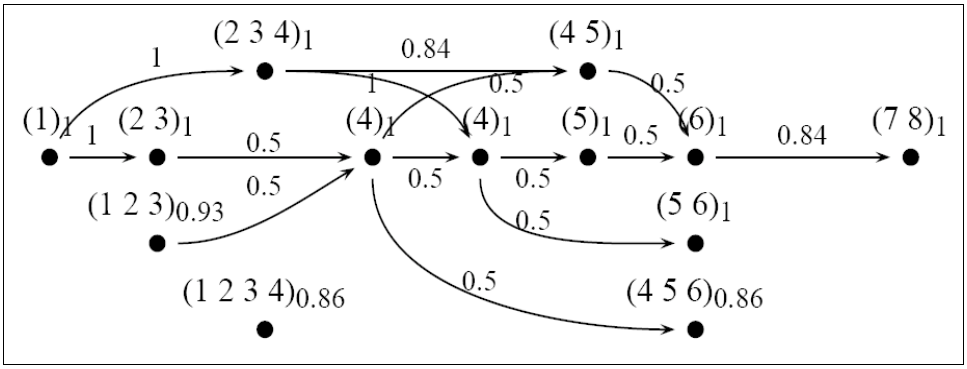
5.2 Temporal accuracy of extracted patterns

The sequence graph of each data sequence is built from the database in Table 2 and the soft time constraints specified in the example statement. Then, sequential patterns are mined for. The GSPs obtained with $minFreq = 70\%$ are: <(3)(4 5)>, <(2 3 4)>, <(2 3)(4)(5 6)>, <(2)(4 5)>, <(3 4)(5)> and <(3 4)(6)>, each having a 100% frequency. To analyse their relevance according to the user's needs, their temporal accuracy is computed. To do so, the sequence graphs are weighted, as described in Section 4.3. The vertices built with $ws = 0,1$ or 2 have a weight of 1, while those built with $ws = 3$ have a weight of 0.93 and those built with $ws = 4$, a weight of 0.86. For *minGap*, the edges built with $mg = 1$ have a weight of 0.5 and a weight of 1 if $mg = 2$. For *maxGap*, the weight is 1 if $MG \leq 4$, 0.92 if MG = 5 and 0.84 if MG = 6. The resulting weighted sequence graph for $O1$ is shown in Figure 5.

Finally, these weights are used to compute the temporal accuracy of extracted patterns. For instance, the temporal accuracy for sequential pattern <(3 4) (5)> is given by the average temporal accuracy of supporting sequences in the graph of each Id: $\varrho_{Id1} = 0.84$, $\varrho_{Id2} = 1$ and $\varrho_{Id3} = 1$, then $\Upsilon = 0.95$.

Once patterns have been obtained with their temporal precision, we can more accurately analyse the constraints used to generate them. The closer the precision is to 1, the more the initial user-specified values correspond to the timestamps in the database. On the contrary, a low precision value indicates that the constraints are not well suited to this dataset.

Figure 5 The weighted sequence graphs for data sequence Id1



6 Experiments

In this section, we compare the performances of the GETC algorithm for soft constraints with those of the GTC algorithm for crisp constraints. In Section 6.1, we compare the behaviours of these algorithms, while also using an implementation of PSP handling time constraints. In a second phase, we compare the patterns extracted using either soft or crisp constraints, first with a synthetical dataset then with web access logs (Section 6.2). All of these experiments were carried out on a PC with a Linux 2.6.7 OS, CPU 2,8 GHz and 2 GB of DDR memory. All the algorithms were implemented in C++ and use the PSP principle and structure to search for sequential patterns.

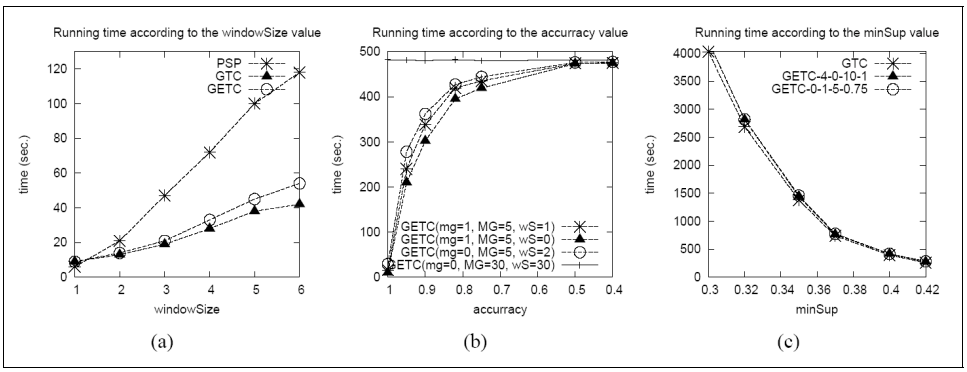
6.1 Synthetical datasets: GETC overall behaviour

The results presented here were obtained through processing several synthetical randomly generated datasets, with each containing approximately 1000 data sequences of 20 records on average. Each of these records contains an average of 15 items chosen among 1000 possible ones.

The first phase involved comparing runtimes without time constraints (*windowSize* = 0, *minGap* = 0 and *maxGap* = ∞) for GTC and GETC, with a minimum accuracy equal to 1 for each soft constraint. We thus compared the runtime of our algorithm with those of PSP and GTC and showed that the GETC behaviour is similar to that of GTC, *i.e.*, while extracting the same set of patterns with GTC or GETC, runtime performances are quite the same.

We then repeated these measures by processing crisp time constraints (with an accuracy of 1 for GETC) to compare the behaviours of GETC and GTC. Figure 6(a) shows the runtime pattern as a function of the *windowSize* value. GETC has a linear behaviour close to that of the GTC. The difference is due to the temporal accuracy calculation step, by which the time increases slightly with *windowSize* because the number of vertices in the sequence graphs increases accordingly. Finally, Figure 6(b) shows (for GETC alone) the runtime pattern according to the accuracy, for a minimum frequency of 0.37. Note that the runtime reaches a maximum value which corresponds to the extreme values of the soft time constraints *M* and *m*.

Figure 6 The runtimes



Notes: (a) as a function of *windowSize* with *minGap* = 2, *maxGap* = ∞ and *minFreq* = 0.35 (for GETC, $\rho_{ws} = \rho_{MG} = \rho_{mg} = 1$); (b) as a function of accuracy depending on several time constraints (*minFreq* = 0.37); (c) as a function of *minFreq* regarding soft time constraints with accuracy equal to 1 or not.

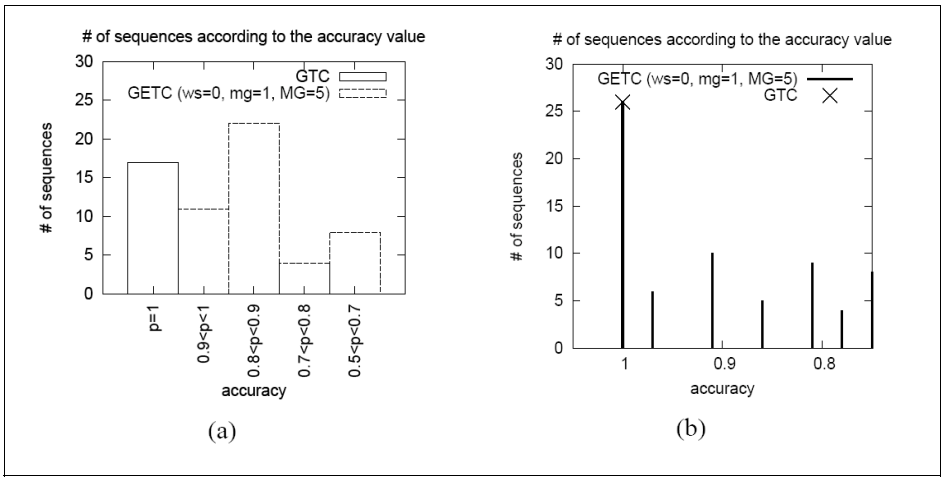
The second part of our experiments on synthetical datasets dealt with an analysis of the sequential patterns extracted by GETC compared to those extracted by GTC,[3] according to the accuracy required for the various soft constraints. Figure 6(c) presents the runtimes for GTC and GETC according to the minimum frequency, depending on the sample values chosen for each parameter. These values were calculated so that the time constraints used for GTC (crisp constraints) and GETC with an accuracy of 1 (simulated crisp constraints) correspond to the GETC limit values (soft constraints) with a precision that differs from 1. These parameters are respectively:

- GTC with $ws = 4$, $mg = 0$ and MG = 10
- GETC with $\rho = 1$, with $ws_{init} = 4$, $mg_{init} = 0$ and $MG_{init} = 10$
- GETC with $\rho = 0.75$, with $ws_{init} = 0$, $mg_{init} = 1$ and $MG_{init} = 5$, yielding $ws_p = 4$, $mg_\rho = 0$ and $MG_\rho = 10$.

Note that under these conditions, GETC with soft time constraints is as fast as GTC with the same constraint limit values although, in addition to retrieving the same sequential patterns, it uncovers their temporal accuracy.

Besides, if we ignore the optimal value of one or several time constraints, it could be interesting to use GETC with a minimum accuracy level different from 1 to extend the search space. An analysis of the retrieved patterns and their accuracy can inform us about a more adequate time constraint value. We compared the patterns extracted by GTC with the patterns extracted by GETC with the same initial time constraints. The number of detected patterns is then greater, as shown in Figure 7(a). By classifying them in decreasing order of accuracy, we got all the patterns extracted by GTC (which have an accuracy of 1), then a list of patterns of lower temporal accuracies corresponding to the soft constraints. This histogram also shows that for this synthetical dataset, the constraints allowing us to extract the largest number of patterns correspond to an accuracy of between 0.8 and 0.9.

Figure 7 Pattern distribution depending on their temporal accuracy on synthetical data ($ws_{init} = 0$, $mg_{init} = 1$, $MG_{init} = 5$ and $\rho_{ws} = \rho_{mg} = \rho_{MG} = 0.5$) (a); Pattern distribution depending on their temporal accuracy on web access logs ($ws_{init} = 0$, $mg_{init} = 0$, $MG_{init} = 0$ and $\rho_{ws} = \rho_{mg} = 1$, $\rho_{MG} = 0.75$, $minFreq = 0.2$ (b)



6.2 *Soft constraints to mine atypical web accesses*

The aim of these experiments was to show that the GSPs extracted under soft time constraints bring more precise information than those obtained with crisp time constraints.

In these experiments, the access logs from a private photo gallery website have been mined to analyse atypical behaviours. This website runs on an Apache server and uses a mySQL database accessed through PHP. It is divided into two parts: one is accessible anonymously and the other requires identification via login and password. We analysed the error logs and isolated the access logs corresponding to these errors and also those corresponding to non-usual access (*i.e.*, accesses not browsing pages or not identified by web browsers). These isolated access logs are globally called *atypical access logs*.

These atypical access logs are preprocessed, with one log per record: the object ID is the requesting IP, the items are the connection request, request type, returned error and connecting software and the timestamp is the access date measured in seconds since 1 January 1970 at 00:00:00.

Example 9 *Table 3 represents the accesses attempted on 1 January 1970 at 00:00:05 AM from the IP encoded by 253 to URL '/PictureGallery/home.htm' with a request 'GET' by the software 'Mozilla/4.0'. The error returned was Error 404 (page does not exist). This access is followed 6 sec later by the same request to URL '/PictureGallery/index.htm'. The software was not identified and no error was returned.*

Table 3 Examples of access logs

IP id	*Time*	*Request*	*URL*	*Error*	*Software*
253	5	GET	'/PictureGallery/home.htm'	404	'Mozilla/4.0'
253	11	GET	'/PictureGallery/index.htm'	–	–

Our atypical access log represents about 22 000 connection attempts over approximately one year from 510 different IPs to 1181 different URLs. First, we compared the performances of GETC with those of GTC. We found the same global runtime behaviour for both algorithms, even though GETC still is a little slower than GTC because of the temporal accuracy computation. The advantage of GETC shown by these experiments is the additional information given by the temporal accuracy. In fact, as in the second step of our experiments on synthetical datasets, we applied crisp constraints to GTC that correspond to the initial value set for GETC and we compared the extracted sequential patterns.

First, we had to choose the values to be specified as initial ones. Since our mining goal was to describe atypical behaviours, we decided to identify nonhuman profiles, *i.e.*, program-generated profiles. This kind of profile can be characterised, for example, by repeated requests over a short period. In this sense, a sequence should be composed of itemsets that have been recorded over one second ($ws_{init} = 0$ and $\rho_{ws} = 1$); no grouping over records is allowed. Similarly, the minimum gap between two itemsets will take its minimum possible value without varying ($mg_{init} = 0$ and $\rho_{mg} = 1$). The constraint corresponding to the behaviour we want to highlight is *maxGap*. Automated requests can be viewed as requests that are too close to be done by human beings, so the maximum gap separating two itemsets should be the shortest possible one: *maxGap*=1 sec. However, we wanted to be sure that we would not ignore other profiles, so we softened this constraint by specifying a temporal accuracy of less than 1 for *maxGap*, here $\rho_{MG} = 0.75$. We thus used the flexibility of the *maxGap* soft constraint to more precisely describe the atypical profiles.

Then, we compared the different results obtained by varying the temporal accuracy of *MG* from 1 (the same results for GETC and GTC) to 0.75 (more patterns extracted by GETC). As for synthetical data, we obtained more patterns thanks to our use of soft constraints, with part of them being the same as those discovered with crisp constraints. Figure 7(b) shows the number of patterns for each temporal accuracy extracted by GETC and those extracted by GTC: every pattern extracted by GTC is also found by GETC with a temporal accuracy of 1 and vice versa. Moreover, GETC uncovers some additional patterns whose temporal accuracy is less than 1.

Regarding the sequential patterns that are found by both algorithms, the advantage of using GETC instead of GTC is the additional information given by the temporal accuracy of each sequence. We thus got more relevant information, since each descriptive pattern for atypical behaviour is provided with its temporal accuracy. Here is the description of some atypical behaviours we found with both algorithms:

- The bot pxyscand sent five CONNECT requests to URL 1185. Each time, it received Error 405 (unauthorised method) (<("CONNECT", URL 1185, 405 Error, "pxyscand/2.1")...(3 times)...("CONNECT", URL 1185, 405 Error, "pxyscand/2.1")>).

- One sequential pattern observed with both GTC and GETC is the insistent access to the URL '/mambo...', which does not exist. It thus seems to be a hacking attempt (<("/cvs/mambo/index2.php?_REQUEST...", 404 Error)("/cvs/mambo/index2.php?_REQUEST...", 404 Error)>).

Regarding the additional patterns uncovered by GETC but not found by GTC, two examples are:

- The software Pompös tried to retrieve some non-existent pages: <("cvs/index2.php?request...",404 Error,Pompos)...once...("cvs/index2.php?request...",404 Error,Pompos)>. This sequential pattern has a temporal accuracy of 0.95 and its frequency is 35%.
- One typical nonhuman access is bot scanning, in particular, from Google's Image Bot. This bot does not appear when we use crisp constraints. When analysing the access logs, we see that the average gap between two requests is 2 sec <("/PictureGallery/thumbnail.php3?...",Googlebot-Image/1.0)...once...("/PictureGallery/thumbnail.php3?...',Googlebot-Image/1.0)...>.

7 Conclusion and prospects

The GSPs presented in Srikant and Agrawal (1996) redefine the inclusion of sequences in a broader way by introducing time constraints. These constraints, which allow the user to gather records or separate them into different sequences, can highlight less immediate knowledge and closer to his/her needs. However, this definition is still too rigid, particularly if the user has only a vague idea of the time constraints which bind his data. In this article, we thus proposed to soften these time constraints for GSPs, by using some fuzzy set theory principles. We thus give more flexibility to the specification of time constraint parameters. The implementation of our approach is based on the construction of sequence graphs to handle time constraints during the sequential pattern mining process. We showed the efficiency of our GETC algorithm to solve the problem of mining for generalised sequences under crisp or soft time constraints. We also highlighted the flexibility offered by our soft time constraints, as well as the advantages of the temporal accuracy measure to analyse sequential patterns by running experiments on both synthetical and real-life datasets. Finally, we intend to extend the fuzzy sequential patterns presented in Fiot *et al.* (2006b) to GSPs with time constraints (crisp or soft) in order to mine quantitative timestamped data under time constraints.

Acknowledgements

The authors would like to acknowledge the anonymous webmaster who made our experiments possible by providing us with web access logs and, thus, helped us demonstrate the relevance of our work.

References

Agrawal, R. and Srikant, R. (1995) 'Mining sequential patterns', *11th Int. Conf. on Data Engineering*, Taipei, Taiwan, 6–10 March, pp.3–14.

Capelle, M., Masson, C. and Boulicaut, J-F. (2002) 'Mining frequent sequential patterns under a similarity constraint', *3rd Int. Conf. on Intelligent Data Engineering and Automated Learning* (*IDEAL '02*), Manchester, UK, 12–14 August, pp.1–6.

Fiot, C., Laurent, A. and Teisseire, M. (2006a) 'Extended time constraints for generalized sequential pattern mining', Technical Report 06051, LIRMM.

Fiot, C., Laurent, A., Teisseire, M. and Laurent, B. (2006b) 'Why fuzzy sequential patterns can help data summarization: an application to the INPI trademark database', *15th IEEE International Conference on Fuzzy Systems* (*FuzzIEEE '06*), Vancouver, Canada, 16–21 July.

Leleu, M., Rigotti, C., Boulicaut, J-F. and Euvrard, G. (2003) 'Constraint-based mining of sequential patterns over datasets with consecutive repetitions', *7th Eur. Conf. on Principles and Practice of Knowledge Discovery in Databases* (*PKDD '03*), Cavtat – Dubrovnik, Kroatia, 22–26 September, pp.303–314.

Lin, M-Y., Lee, S-Y. and Wang, S-S. (2002) 'DELISP: efficient discovery of generalized sequential patterns by delimited pattern-growth technology', *6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining* (*PAKDD '02*), Taipei, Taiwan, 6–8 May, pp.198–209.

Mannila, H., Toivonen, H. and Verkamo, A.I. (1997) 'Discovery of frequent episodes in event sequences', *Data Mining and Knowledge Discovery*, Vol. 1, No. 3, pp.259–289.

Masseglia, F., Cathala, F. and Poncelet, P. (1998) 'The PSP approach for mining sequential patterns', in *Principles of Data Mining and Knowledge Discovery*, pp.176–184.

Masseglia, F., Poncelet, P. and Cicchetti, R. (1999) 'An efficient algorithm for web usage mining', *Networking and Information Systems Journal*, Vol. 2, Nos. 5–6, pp.571–603.

Masseglia, F., Poncelet, P. and Teisseire, M. (2004) 'Pre-processing time constraints for efficiently mining generalized sequential patterns', *11th Int. Symp. on Temporal Representation and Reasoning* (*TIME '04*), Tahitou, France, 1–3 July, pp.87–95.

Meger, N. and Rigotti, C. (2004) 'Constraint-based mining of episode rules and optimal window sizes', *8th Eur. Conf. on Principles and Practice of Knowledge Discovery in Databases* (*PKDD '04*), Pisa, Italia, 20–24 September, pp.313–324.

Spiliopoulou, M. and Faulstich, L.C. (1998) 'WUM: a tool for web utilization analysis', *Proceedings of the World Wide Web and Databases International Workshop* (*WebDB '98*), LNCS 1590, Valencia, Spain, 27–28 March, pp.184–2003.

Srikant, R. and Agrawal, R. (1996) 'Mining sequential patterns: generalizations and performance improvements', *5th Int. Conf. on Extending Database Technology* (*EDBT '96*), Avignon, France, 25–29 March, pp.3–17.

Yan, T-W., Jacobsen, M., Garcia-Molina, H. and Dayal, U. (1996) 'From user access patterns to dynamic hypertext linking', *Proceedings of the Fifth International World Wide Web Conference on Computer Networks and ISDN Systems*, Paris, France, 6–10 May, pp.1007–1014.

Zadeh, L. (1965) 'Fuzzy sets', *Information and Control*, Vol. 3, No. 8, pp.338–353.

Zaiane, O-R., Xin, M. and Han, J. (1998) 'Discovering web access patterns and trends by applying OLAP and data mining technology on web logs', *Proceedings of the Advances in Digital Libraries Conference* (*ADL '98*), IEEE Computer Society, p.19.

Zaki, M.J. (2000) 'Sequence mining in categorical domains: incorporating constraints', *9th Int. Conf. on Information and Knowledge Management* (*CIKM '00*), Washington, DC, USA, 6–11 November, pp.422–429.

Notes

1 To improve graph legibility, in Figure 4, some vertices have been moved up from where they were in Figure 3.

2 The potentially included subsequences are shown with dashlines in Figure 4. The potentially included vertices are marked (*) during the edge creation step.

3 The patterns discovered using GTC are the same as would be obtained by DELISP. The only difference comes from the way both algorithms run.