Extended Time Constraints for Generalized Sequential Patterns

Céline Fiot

Extended Time Constraints for Generalized Sequential Pattern Mining

Céline Fiot

Abstract

Mining temporal knowledge has many applications. Such knowledge can be all the more interesting as some time constraints between events can be pushed into during the mining task. As well in data mining as in 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 generalized sequential patterns. 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 graphs of sequences derived from extended temporal constraints. These relaxed constraints enable us to find more generalized sequential patterns. We also propose a measure of the temporal accuracy of the extracted sequences compared to the initial constraints; this measure will provide the user with a tool to analyse the numerous extracted patterns.

1 Introduction

The quantity of data from the World Wide Web is growing dramatically: requested 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 works have been proposed to mine usage patterns and user profiles (Spiliopoulou and Faulstich, 1999; Yan et al., 1996; Zaiane et al., 1998). In particular, (Masseglia et al., 1999) provides knowledge from databases of visited page sequences.

The information thus discovered can often be improved by looking for temporal knowledge. In certain cases (detection of frauds, failures, behavior analysis), it can not only be useful but even necessary. Some learning techniques allow to manage and to discuss such knowledge, such as (Allen, 1990), which defines operations on rules associated to temporal intervals. The goal of those data mining techniques is to extract recurring episodes from a long sequence (Mannila et al., 1997; Raissi et al., 2005) or from bases of sequences (Agrawal et al., 1993; Agrawal and Srikant, 1995; Masseglia et al., 1998). Searching for such information becomes all the more interesting since it allows to take into account different constraints between events, such as the minimal or maximal duration separating two events (Srikant and

Agrawal, 1996; Zaki, 2000; Masseglia et al., 2004; Meger and Rigotti, 2004), or constraints on regular expressions or rehearsals (Garofalakis et al., 2002; Capelle et al., 2002; Leleu et al., 2003; Albert-Lorincz and Boulicaut, 2003).

Within this framework, generalized sequential pattern mining was introduced in (Srikant and Agrawal, 1996). This data mining technique extracts frequent sequences satisfying user-specified constraints from a database of sequences (e.g. successive purchases of customers in a supermarket). 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 applying the constraints to the sequences, which are then analyzed by some sequential pattern tool. The GTC algorithm (Graph for Time Constraint), proposed in (Masseglia et al., 2004), is based on this principle.

However, although these methods are effective and robust, more especially the approach based on sequence graphs, they require the user to know exactly the constraint values to be specified. The risk may then be to gather erroneous or useless knowledge. Moreover, in some cases, these values are somewhat uncertain. Thus, time constraints, as they are defined, allow 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) has proposed to determine automatically 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 work proposes an automatical 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.

Finally, the discovered sequential patterns, according to the specified time constraints, can quickly become so numerous that their analysis becomes less effective. In that regard, a measure that could help the analysis of generalized sequential patterns would be a valuable tool.

To tackle several of the aforementioned problems, in this article we propose a method that can soften user-specified time constraints and provides the end-user with a satisfaction degree of this initial time constraints.

After having presented the fundamental concepts associated to sequential patterns and to generalized sequential patterns in next section, we present in section 3 a brief introduction to the fuzzy set theory and we define soft time constraints and the temporal accuracy of a sequence. Then section 4 details our algorithm to implement the handling of soft time constraint and section 6 develops our proposal on an example. We then propose some experiments on both synthetical data and web access logs in section 7. Finally, we conclude in section 8 on the perspectives opened by our work.

2

2 Sequential Patterns and Time Constraints

This section defines the concepts used in the generalized sequential pattern mining task. It broadly summarizes the formal description of the problem introduced in (Agrawal and Srikant, 1995; Srikant and Agrawal, 1996).

2.1 Sequential patterns

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

Let $\mathcal{O}$ 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 $\mathcal{I} = \{i_1, i_2, ..., i_q\}$ 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 $< s_1 s_2 ... s_p >$. A *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, records are the transactions made by this customer. Timestamps are the date of transactions. If the customer Smith purchases products* 1, 2, 3, 4, *and* 5 *according to the sequence* $s =<$ (1) (2 3) (4) (5) $>$, *then all 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 $< s_1' \, s_2' \, ...s_m' >$ is a *subsequence* of another one $< s_1 \, s_2...s_p >$ if there aere integers $l_1 < l_2 < ... < l_m$ such that $s_1' \subseteq s_{l_1}$, $s_2' \subseteq s_{l_2}$, ..., $s_m' \subseteq s_{l_m}$. We shall also write that s' is *included* in s.

Example 2 *The sequence* $s' = <(2)(5)>$ *is a subsequence of s above, 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 of their timestamp. They are called a data sequence. In order to efficiently aid decision making, the aim is to discard non-typical behaviors according to the user's viewpoint. Performing such a task requires providing any data subsequence in $\mathcal{O}$ 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 database. An object *supports* a sequence s iff 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 problem of sequential pattern mining is to find all 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 definition of sequences is rather strict and turns out not to be appropriate 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 as simultaneous. On the contrary, two events too distant could have no link together. That is why generalized sequential patterns have been proposed in (Srikant and Agrawal, 1996), introducing time constraints in order to improve the definition of subsequences.

2.2 Generalized 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) redefines when a data sequence supports a sequence as follows.

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

Time constraints allow a more flexible handling of records, insofar as the end user is then equipped 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 as related);

- to regard itemsets as too distant to appear in the same frequent sequence with the $maxGap$ constraint (i.e. to be considered as related).

Example 3 *In this example, we take the same context as for the previous examples: records refer to the purchases of customers in a supermarket. Consider the data sequences C1 and C2 of customers 1 and 2 given in Table 1.*

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
C1	1	2	3 4		5 6 7	8	9
C2	1	2 3 4	5 6	7 8			

Table 1: Purchases made by two customers during one week

Let $s = <(1\ 2\ 3\ 4)>$ be a sequence and the following time constraint parameters:
- *$minGap=0$, consecutive itemsets must be at least one-day distant,*
- *$maxGap=7$, consecutive itemsets must be at most seven-day distant,*
- *$windowSize=1$, purchases may be grouped together over at most two consecutive days.*

Then Figure 1 shows how these time constraints are applied in order to determine whether data sequences C1 and C2 support the candidate sequence $s = < (1\ 2\ 3\ 4) >$ or not.

Figure 1: Description of time constraints, $windowSize$ (ws), $minGap$ (mg) and $maxGap$ ($MaxG$), on data sequences C1 and C2

(a) $C1$, customer 1 data sequence   (b) $C2$, customer 2 data sequence
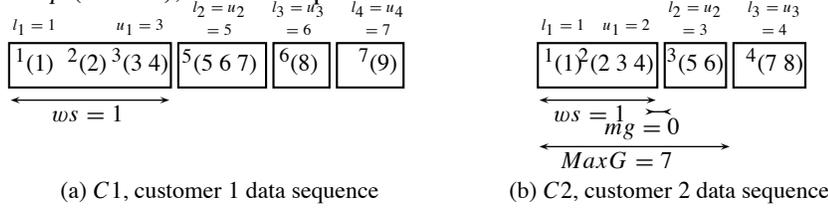
$^i(a\ b)$ denotes the itemset $(a\ b)$ bought at date i

To make sequence s appear in data sequence C1, the purchases of days 1, 2 and 3 must be grouped together. However, this itemset does not meet constraint (ii), since end-time(s_1) - start-time(s_2) = day 3 - day 1 = 2 > $windowSize$. There exists no other possibility to find s in this data sequence. Thus the data sequence C1 does not support sequence s.
To make sequence s appear in data sequence C2, the purchases of days 1 and 2 must be grouped together. This itemset meets the $windowSize$ constraint, since it has been built over two consecutive days. The minimum gap between this first itemset and the next is then day 3 - day 2 = 1 > 0 = $minGap$, which meets the $minGap$ constraint (iii). So does the $maxGap$ constraint (iv). The data sequence C2 supports sequence s.

Note that if the specified values are $minGap=0$, $maxGap=$inf 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 items in an itemset come from a single record.

3   Soft Time Constraints

The main drawback of such constraints is that they are user-specified. They require the data and constraint values to be well-known a priori. Results are thus subject to a good knowledge of the end-user. Misvalued time constraints could indeed lead to erroneous or incomplete knowledge. However no work, to our knowledge, has been proposed to determine automatically optimal time constraints for sequential pattern mining. We here propose to extend the above time constraints for generalized sequential patterns using some principles of fuzzy set theory.

Moreover, extracted patterns become more and more numerous, particularly during sequential pattern mining. So it becomes necessary to provide the end-user with tools to analyse the sequential patterns he get.

In the case of generalized sequential patterns, some useful information could be given by the duration of data sequences corresponding to time constraints. This is the purpose of soft time constraints, that we define in this section.

Furthermore, these soft time constraints enable us to define a measure of temporal accuracy expressing how well a sequence satisfies the initially user-specified time constraint values.

We thus provide the user with flexibility in time constraint specification and with a tool helping him to analyze the extracted patterns.

3.1   Fuzzy Set Theory

The fuzzy set theory was introduced by (Zadeh, 1965). This theory generalizes crisp set theory, admitting intermediary situations between all and nothing. Whereas in the classical theory an element a belongs or not to a set A, in the 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, the fuzzy set theory allows the transition of an object from one state to the next to be gradual.

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]. An exemple of these three fuzzy sets is graphically represented on Figure 2. Thus a person with height x=1m60 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
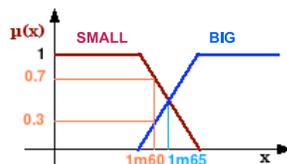
Operators in fuzzy logic are a generalization of crisp logic operators. In particular, we consider negation, intersection and union. The operator $\top$ or t-norm operator (triangular norm) is the fuzzy equivalent of the binary intersection: $\mu_{A \cap B}(x) = \top(\mu_A(x), \mu_B(x))$. The operator $\bot$ or t-conorm operator (triangular conorm) is similar to the binary union: $\mu_{A \cup B}(x) = \bot(\mu_A(x), \mu_B(x))$. We denote $\overline{\top}$ (resp. $\underline{\bot}$) the operator $\top$ (resp. $\bot$) generalized to the n-ary case. Different operators can be used as a t-norm (min, $product$...). They are associated to their dual operator for the t-conorm (e.g. max is the t-conorm for the min t-norm). The min operator being 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 not satisfy or not a constraint in a binary way anymore, because the user may relax these time constraints. Each constraint can then be regarded as a fuzzy set, its membership function giving a temporal satisfaction degree for a given sequence. This degree is thus calculated for each possible value of that time constraint, and tells the end-user to which extent the initially specified constraint value has been fulfilled.

In order to answer user needs and to make our approach flexible, a minimum temporal satisfaction degree ρ_x can be specified for constraint $\mathcal{X}$ of initial value x_{init}.

The satisfaction degree of constraints being 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 $\rho_x = 1$, the specified minimum temporal satisfaction degree is 1, i.e. the user does not want the value of constraint $\mathcal{X}$ to vary. That constraint will be set to the initial value and will not change; all generated sequences will have a temporal satisfaction degree equal to 1.

- If $\rho_x = 0$, the user want constraint $\mathcal{X}$ to take each possible value; the temporal satisfaction degree will depend on the constraint value generating the sequence.

- In all other cases, $\rho_x \in]0,1[$ and the value x of constraint $\mathcal{X}$ 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 minimum satisfaction degree of each time constraint is 1, we get back the crisp time constraints, and thus, the notion of generalized sequential patterns as introduced in section 2.2.

First of all, the useful limit value of time constraints (extreme values) have to be determined. These values may correspond to the variation on the whole search space (i.e. for ρ_x=0).

These values are computed from the crisp time constraints (ii), (iii) and (iv) in section 2.2. They are given by the limit value allowed by those 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 record and the last one. For the whole database, this commen extreme value will thus be the maximal gap, over all objects, between the minimal and maximal record timestamp for the same object: $M = \max_{o \in \mathcal{O}}(timestamp(r_{o_{max}}) - timestamp(r_{o_{min}}))$.

The $minGap$ constraint defines the minimal gap between two consecutive itemsets. We have defined the limit value of this constraint taking into account the crisp inequality it implies. The minimum gap between two records of an object o is given by $\forall o \in \mathcal{O}, \min_{r \in \mathcal{R}_o}(timestamp(r + 1) - timestamp(r))$. For the whole database, this gap corresponds to the minimal value for the set of objects. It means: $\min_{o \in \mathcal{O}}(\min_{r \in \mathcal{R}_o}(timestamp(r + 1) - timestamp(r)))$. Going to the limit, the $minGap$ constraint is expressed by the inequality:

$$
\begin{aligned}
\min_{o \in \mathcal{O}}(\min_{r \in \mathcal{R}_o}(timestamp(r + 1) - timestamp(r))) &> mingap \\
\min_{o \in \mathcal{O}}(\min_{r \in \mathcal{R}_o}(timestamp(r + 1) - timestamp(r))) &\geq mingap + 1 \\
\min_{o \in \mathcal{O}}(\min_{r \in \mathcal{R}_o}(timestamp(r + 1) - timestamp(r))) - 1 &\geq mingap
\end{aligned}
$$

we obtain that the limit value for $minGap$ is $\min_{o \in \mathcal{O}}(\min_{r \in \mathcal{R}_o}(timestamp(r + 1) - timestamp(r))) - 1$.

But, in the case of one object having one single record, it would imply this value to be -1. It would then imply that there could be no gap between two consecutive itemsets. In other words, the sequence $< (a\ b\ c) >$ could support the sequence $< (a)\ (b)\ (c) >$, which is not compatible with the definitions given section in 2. That is why we lay down that, in this particular case, the limit value of $minGap$ would be 0. The limit value for $minGap$ is then given by $m = \max(\min_{o \in \mathcal{O}}(\min_{r \in \mathcal{R}_o}(timestamp(r + 1) - timestamp(r))) - 1, 0)$.

In the remainder of this section, we use the notations given in Table 2, in order 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 to them. These coefficients will enable the user to limit the variation of time constraints, according to his 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 that variable.

M	maximal possible value for $windowSize$ and $maxGap$
	$M = \max_{o \in \mathcal{O}}(timestamp(r)_{o_{max}} - timestamp(r)_{o_{min}})$
ws	variable for the $windowSize$ constraint; may vary from ws_{init} to M
ws_{init}	initial value of parameter $windowSize$, user-specified
ws_{ρ}	limit acceptable value for ws, computed from ρ_{ws}
ρ_{ws}	lowest acceptable satisfaction degree for $windowSize$
$\rho(ws)$	temporal satisfaction degree obtained by the value ws of $windowSize$
MG	variable for the $maxGap$ constraint; may vary from MG_{init} to M
MG_{init}	initial value of parameter $maxGap$, user-specified
MG_{ρ}	limit acceptable value for MG, computed from ρ_{MG}
ρ_{MG}	lowest acceptable satisfaction degree for $maxGap$
$\rho(MG)$	temporal satisfaction degree obtained by the value MG of $maxGap$
m	minimal possible value for $minGap$
	$m = \max(\min_{o \in \mathcal{O}}(\min_{r \in \mathcal{R}_o}(timestamp(r+1) - timestamp(r))) - 1, 0)$
mg	variable for the $minGap$ constraint; may vary from m to mg_{init}
mg_{init}	initial value of parameter $minGap$, user-specified
mg_{ρ}	limit acceptable value for mg, computed from ρ_{mg}
ρ_{mg}	lowest acceptable satisfaction degree for $minGap$
$\rho(mg)$	temporal satisfaction degree obtained by the value mg of $minGap$

Table 2: Notations

3.3 Extending Time Constraints to Soft Time Constraints

We now detail how each soft time constraint is built and we illustrate it with a short example.

3.3.a Soft $windowSize$

The value ws of $windowSize$ constraint may vary from its user-specified value ws_{init} and its limit value M, as shown on Figure 3.

Figure 3: Temporal satisfaction degree of $windowSize$ according to the soft constraint
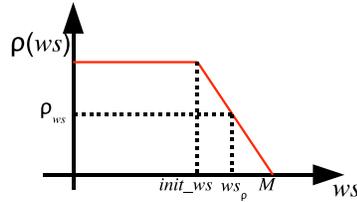
This soft constraint is described with a fuzzy set of which the membership

function (1) gives the accuracy for a specific value ws:

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

The user can then choose to allow the temporal satisfaction degree of the $windowSize$ constraint to be somewhere between 1 and a lowest acceptable value ρ_{ws}. That 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 $s = <(1\ 2\ 3\ 4)>$. Suppose that the user-specified time constraint parameters are $ws_{init} = 1$, $mg_{init} = 0$ and $MG_{init} = 7$, with $\rho_{ws} = 0.7$, $\rho_{mg} = 1$ and $\rho_{MG} = 1$.*

Thus, ρ_{ws} is the only lowest acceptable satisfaction degree different from 1, and $windowSize$ is the only constraint possibly having several values. Applying equation (2) with $M = \max((7-1), (4-1)) = 6$ (Table 2, we get $ws_\rho = \lfloor (1-6) * 0.7 + 6 \rfloor = 2$. The value ws of $windowSize$ will successively be 1 then 2.

For $ws = 1$, grouping the pruchases of days 1 to 3 of data sequence $C1$ in order to accept the candidate sequence s would violate the $windowSize$ constraint. However with $ws = 2$, we can indeed group them and that ws value, by equation (1), yields a temporal satisfaction degree $\rho(ws) = \rho(2) = 0.8$. The other constraints are satisfied as well, then the data sequence $C1$ supports the candidate sequence s.

For data sequence C2, the purchases of days 1 and 2 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 constraint are also respected, the data sequence C2 supports the candidate sequence s.

3.3.b Soft $maxGap$

The value MG of $maxGap$ constraint may vary from its user-specified value MG_{init} and its limit value M.

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

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

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}. That lowest degree will be attained for a value $MG_\rho > MG_{init}$ of MG.

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

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

10

Example 6 *Consider the two data sequences shown in example 3 and the candidate sequence $s=<(1\ 2\ 3\ 4)\ (7\ 8)>$. Suppose that the user-specified time constraint parameters are $ws_{init}=2$, $mg_{init}=0$ and $MG_{init}=3$, with $\rho_{ws}=1$, $\rho_{mg}=1$ and $\rho_{MG}=0.3$.*

Thus ρ_{MG} is the only lowest acceptable satisfaction degree different from 1 and $maxGap$ is thus the only constraint varying. Applying equation (4) with $M=6$, we get $MG_{\rho}=5$. The value MG will successively be 3, 4 and 5.

For data sequence C1, purchases of days 1 to 3, as well as days 5 and 6, are grouped together in order to accept the 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 $\rho(MG)=\rho(5)=0.3$ (3). The other constraints are also satisfied, then the data sequence C1 supports the candidate sequence s.

For the data sequence C2, the records of days 1 and 2 are grouped together, satisfying the $maxGap$ constraint with $MG=MG_{init}=3$ and the corresponding satisfaction degree is $\rho(MG=3)=1$. The other constraint are also satisfied, the data sequence C2 supports the sequence s.

3.3.c Soft $minGap$

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

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

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

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

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

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 $ws_{init}=2$, $mg_{init}=2$ and $MG_{init}=7$, with $\rho_{ws}=1$, $\rho_{mg}=0$ and $\rho_{MG}=1$.*

Thus, ρ_{mg} is the only lowest acceptable satisfaction degree different from 1 and $minGap$ is thus the only constraint varying. Applying equation (6) with $m=0$, we get $mg_{\rho}=0$. mg will successively be 2, 1 and 0.

For the data sequence C1, the purchases of days 1, 2 and 3. The $minGap$ constraint is not satisfied 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 satisfied, so the data sequence C1 supports the sequence s.

For the second data sequence, the purchases of days 1 and 2. The $minGap$ constraint is not satisfied while mg is greater than 0. $mg=0$ yields, by equation (5), to a temporal

11

satisfaction degree $\rho(mg) = \rho(0) = 0$. *The other constraints are also satisfied, the data sequence C2 supports s.*

Note that these soft constraints are defined in 3.3.*a*, 3.3.*b* and 3.3.*c* by using fuzzy sets where the temporal satisfaction degree is described by a linear membership function between the constraint initial 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 data set satisfying each value of the constraints.

3.4 Temporal Accuracy of a Sequence

We now define the level of time constraint satisfaction for a sequence considering the three constraints (ii), (iii) and (iv), together. At the end of the mining task, we get a list of frequent sequences. For each object, each of them 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 now combined into a global measure associated to the sequence.

For an object o, the ***temporal accuracy of a sequence*** s is defined for an object o 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. So 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 ($\bot$).

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

$$\varrho(s,o) = \bot_{s \in \varsigma_o} \Big(\overline{\top}_{i \in 1,n} \big(\rho_{ws}(\textit{end-time}(s_i) - \textit{start-time}(s_i)) \big),$$
$$\overline{\top}_{i \in 2,n} \big(\rho_{mg}(\textit{end-time}(s_i) - \textit{start-time}(s_{i-1})), \tag{4}$$
$$\rho_{MG}(\textit{end-time}(s_i) - \textit{start-time}(s_{i-1})) \big) \Big)$$

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

$$\Upsilon(s) = \frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} \varrho(s,o) \tag{5}$$

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* ρ_{ws}=0.6, ρ_{mg}=0.4 *and* ρ_{MG}=0.5. *We still have M=6 and m=0. In this example, we use the min and max operators respectively for the generalized t-norm ($\overline{\top}$) and the generalized t-conorm ($\underline{\bot}$).*
For data sequence C1, s appears by grouping together days 1 to 3 on the one hand and days 5 and 6 on the other hand. Then, start-time(s_1)=1, end-time(s_1)= 3, start-time(s_2)=5

and end-time(s_2)= 6. It is the single occurrence of s in this data sequence. Thus for C1, the temporal accuracy of s is (details omitted):

$$\varrho(s, C1) = min(\rho_{ws}(2), \rho_{ws}(1), min(\rho_{mg}(1), \rho_{MG}(5)))$$
$$= min(0.8, 1, min(0.5, 0.5))$$
$$= 0.5$$

Then the same computation is done for data sequence C2. Similarly, we get $\varrho(s, C2) = 0.5$. The temporal accuracy of sequence s for the whole database is thus given by:

$$\Upsilon(s) = \frac{\varrho(s, C1) + \varrho(s, C2)}{2} = 0.5$$

4   Graph for Extended Time Constraints

Our implementation of these soft time constraints is based on the GTC algorithm (Graph for Time Constraints) proposed in (Masseglia et al., 2004). The main idea is to transform the data sequence of an object into a sequence graph in which each path is a subsequence satisfying the time constraints. The sequence graphs of the data sequences are then used to determine the frequent sequences by a sequential pattern mining algorithm.

The efficiency of this approach was demonstrated. We chose to develop our solution from this idea. We thus propose an algorithm allowing to build a sequence graph for the soft time constraints which will also allow us, in a second phase, to calculate the temporal accuracy of the extracted generalized sequential patterns.

4.1   Related Work

Several previous approaches have tackled the generalized frequent sequences problem.

The GSP algorithm proposed in (Srikant and Agrawal, 1996) is aimed at mining Generalized Sequential Patterns. 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, GSP 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 during examination between forward and backward phases. Forward phases are performed for dealing 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 satisfied. In such a case, it is necessary to discard all the items for which the $maxGap$ constraint is violated and to resume parsing the sequence starting with the earliest item satisfying the $maxGap$ condition.

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

More recently, the DELISP algorithm (Lin et al., 2002) has been proposed for mining sequential patterns with time constraints. It is based on the mining scheme of Prefix-SPAN. Actually, the original database is divided into multiple subsets for each prefix of a potential sequential pattern. While writing of 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, such a technique is restricted to prefix-growth algorithms. Therefore the GTC algorithm (Masseglia et al., 2004) has been developped. It improved the PSP approach by a more efficient handling of time constraints.

The GTC (Graph for Time Constraints) algorithm, taking 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. Because 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. Thus, the graph structure used by GTC, described in the next section will then be used both for sequential pattern mining and for computing temporal accuracy of frequent sequences.

4.2 General Strategy of the Algorithm

Our approach described in Algorithm 1 takes up 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 satisfying 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 the candidate parsing: only inclusion must be verified. This preprocess optimizes the GSP algorithm by handling separately time constraints, thus avoiding backward and forward exploration while searching for frequent sequences. Once the sequential patterns are extracted, the sequence graphs are weighted, then explored one last time to calculate the temporal accuracy of each generalized sequential patterns extracted.

4.3 Sequence Graph Building

From an input data sequence d, the GETC algorithm (Algorithm 2) builds a sequence graph $\mathcal{G}_d(V, E)$ in which vertices are itemsets and paths represent subsequences

14

Main - **Input:** $minFreq, DB$
Ouput: F, frequent sequences on DB,
k longest length of frequent sequences

$F_0 \leftarrow \varnothing \; ; k \leftarrow 1 \; ;$
$F_1 \leftarrow \{\{< i >\}/i \in \mathcal{I} \& freq(i) > minFreq\}\};$
addWindowSize(S) ;

While $(Candidate(k) \neq \varnothing)$ **do**
   For each $d \in DB$ **do**
      $(G) \leftarrow \text{GETC}(d)$;
      countFrequency$(Candidate(k), minFreq, (G))$;
   End For
   $F_k \leftarrow \{s \in Candidate(k)/freq(s) > minFreq\};$
   $Candidate(k + 1) \leftarrow \text{generate}(F_k)$;
   $k++;$
End While
ComputeAccuracy(F_k);
return $F \leftarrow \bigcup_{j=0}^{k} F_k$

Algorithm 1: Main algorithm

satisfying the time constraints.

First each itemset of the input sequence is associated to a vertex. Then the sub-function *addWindowSize* combines records, trying to satisfy the soft $windowSize$ constraint and adds to the graph any satisfying combination as a new vertex. Vertices are affected to "levels" according to their *start-time* in order to reduce the time spent for building edges. The next step consits in building the edges satisfying both *minGap* and *maxGap* soft constraints. Thus for each vertex, the first "level" of vertices satisfying the soft *minGap* constraint is retrieved. For each vertex of this set, the *minGap* constraint is satisfied and the *maxGap* constraint is checked. If it is satisfied, a new edge is built between both vertices. Some optimization is done by the *addEdge* and *propagate* subfunctions in order to reduce the number of sequence inclusions. Finally, the remaining included subpaths are deleted from the graph by the subfunctions *pruneMarked* and *convertEdges*.

The $addWindowSize$ function scans each vertex *x* and determines for each of them the other vertices *y* to which it can be combined (i.e. if *y*.date() - *x*.date() $\leq$ ws). Each vertex then corresponds to an itemset *i* associated to a starting timestamp *i*.begin() and an ending timestamp *i*.end(). The vertices are grouped together in *levels* by ending timestamp of the itemsets. It allows to analyse if the constraints are satisfied by level and not for each vertex anymore. We thus reduce the runtime.

GETC - **Input:** *d*, a data sequence
      Ouput: $G_d(V, E)$, *d* graph sequence,
           S vertex 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
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);

Algorithm 2: GETC

The *addEdge* algorithm builds the edges between vertices satisfying the *minGap* and *maxGap* constraints. A definitive edge is created if the vertices are not already linked in a sequence or by the inclusion of their successors or predecessors. In that case, the built edge is temporary and becomes definitive if the sequence it builds is maximal.

It is also during the running of this algorithm that included vertices are marked, they will be later deleted if they are unnecessary.

propagate is used when a jump over a level *l* occurs because of the non-satisfaction of the *minGap* constraint. It is used to build the sequences between the vertices in level *l* and the successors of *x* which can not reach this level. Like in *addEdge*, temporary or definitive edges are built depending on the likely inclusion of the built sequence.

For each temporary edge from *x* to *y*, if *y* is included in a successor *z* of *x* and if the successors of *y* are also successors of *z*, then there is an included subsequence, the edge is unnecessary, it is so deleted.

In other cases, the edge is necessary to obtain every maximum sequences. It is then converted into a definitive edge.

addWindowSize - **Input:** *S*, vertex set to scan (ordered ascendently with respect to starting timestamp, ending timestamp) ;

```
copyS ← S ; α ← S.first() ;
While (α ≠ S.last()) do
    αnext ← S.next(α) ; β ← S.next(α) ; ws ← a ;
    While (ws ≤ aρ) do
        While (β.end() - α.begin() ≤ ws) do
            i ← group(α, β) ;
            [Add the vertex i after the vertex β
            to copyS
            To avoid rehearsals due to ws, we
            add the condition if i ≠ α]

    If (i.itemset ≠ α.itemset) Then
        added ← FALSE ;
        For each z ∈ β.level() do
            If (z.itemset == i.itemset) Then
                [then z.begin() < i.begin(), by building]
                copyS.modify(z,i) ;
                [z.begin() is modified to start-time(i)]
                added ← TRUE ;
            End If
        End For
                        If (added == FALSE) Then
                            copyS.insert(i, β);
                            β.level().add(i) ;
                        End If
                        α ← i ;
                        [If α ≠ S.last(), end loop]
                    End If
                    β ← S.next(β) ;
                    [If β ≠ S.last(), end loop]
                End While
                ws ++ ;
            End While
            α ← αnext ;
        End While
        S ← copyS ;
```

Algorithm 3: addWindowSize

During building, the marked vertices are those of included subsequences, the function *pruneMarked* deletes them.

We proove in section 5 that, at the end of this process, GETC has built exactly all the longest sequences, satisfying the soft time constraints *windowSize*, *minGap* and *maxGap*, generated from the input data sequence. The GETC algorithm thus can be used as a preprocessing phase to handle the soft time constraints, before the sequential patterns are mined. After this step, the candidate sequence support is computed on these sequence graphs.

addEdge - **Input:** S, two vertices r and s, A edge set of the graph;

If (r.succ() $== \varnothing$) **Then**
  If (s.prev() $== \varnothing$) **Then**
    A $\leftarrow$ A $\cup\{(r,s)\}$;
    unmark(r) ; unmark(s) ;
  Else
    For each $p \in s$.prev() **do**
      If (($r \subset p$) & (r.succ() $\subset p$.succ())) **Then**
        $included \leftarrow$ TRUE ;
      End If

    End For
    If ($included ==$ TRUE) **Then**
      arcTmp(r,s) ;
      mark(r) ;
      [if no definitive edge exists from *or* to *r*]
    Else
      A $\leftarrow$ A $\cup\{(r,s)\}$;
      unmark(r) ; unmark(s) ;
    End If
  End If
End If

Else
  For chq $t \in r$.succ() **do**
    If (($s \subset t$) && (s.succ() $\subset t$.succ())) **Then**
      $included \leftarrow$ TRUE ;
    End If
  End For
  If ((s.prev() $\neq \varnothing$) & ($included ==$ FALSE)) **Then**
    For each $p \in s$.prev() **do**
      If (($r \subset p$) & (r.succ() $\subset p$.succ())) **Then**
        $included \leftarrow$ TRUE ;
      End If
    End For
  End If
  If ($included ==$ TRUE) **Then**
    arcTmp(r,s) ;
    mark(r) ;
    [if no definitive edge exists from *or* to *r*]
  Else
    A $\leftarrow$ A $\cup\{(r,s)\}$;
    unmark(r) ; unmark(s) ;
  End If
End If

Algorithm 4: addEdge

---

propagate - **Input:** x, un sommet du graphe de séquences, l, un niveau du graphe de séquences

If (x.begin() $> l$.end()) **Then**
  For chq $u \in l$ **do**
    For chq $v \in x$.succ() **do**
      If (v.begin() - u.end() $> mg$)) **Then**
        $MG \leftarrow G$;
        While ($MG \leq G_\rho$) **do**
          If (v.end() - u.begin() $\leq MG$) **Then**
            If (u.succ() $\cap v$.prev()) **Then**
              For chq $t \in v$.prev() **do**
                If ($u \subsetneq t$) **Then**
                  $included \leftarrow$ TRUE ;
                End If
              End For
              If ($included ==$ FALSE) **Then**
                For chq $w \in u$.succ() **do**
                  If (w.succ() $\subset u$.succ()
                  & $w \subset u$) **Then**
                    $included \leftarrow$ TRUE ;
                  End If
                End For
              End If
            End If

            If ($included ==$ TRUE) **Then**
              arcTmp(u,v) ;
              mark(u) ; *[si aucun arc défi-*
                    nitif de *ou* vers *r*]
            Else
              A $\leftarrow$ A $\cup\{(u,v)\}$;
              unmark(u) ; unmark(v) ;
            End If
           End If
          $MG \leftarrow G_\rho + 1$;
        Else
          $MG + +$;
        End If
        End While
      End If
    End For
  End For
End If

Algorithm 5: propagate

18

convertEdge - **Input:** $G(S, A)$, a sequence graph

For each arcTmp(x,y) **do**
   For each $z \in x$.succ$()\backslash y$ **do**
     If $((y \subset z)$ && $(y$.succ$() \subset z$.succ$()))$ **Then**
       included $\leftarrow$ TRUE
     End If
   End For
   If (*included* == TRUE) **Then**
     prune(arcTmp(x,y)) ;
   Else
     A $\leftarrow$ A $\cup \{(x, y)\}$;
     prune(arcTmp(x,y)) ;
   End If
End For

Algorithm 6: convertEdges

---

pruneMarked - **Input:** $G(S, A)$, a sequence graph

For each $s \in S$ **do**
   If (s.marked == TRUE) **Then**
     prune(s) ;
   End If
End For

Algorithm 7: pruneMarked

4.4 Temporal Accuracy Computation

Once the sequence graphs has been built, we know which sequences are allowed by the time constraints and which are forbidden. However, some sequences satisfy the crisp constraints while others were built only by applying the 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.

In order to determine the time constraint values satisfyed 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 that edge; each vertex is similarly weighted by μ_{ws}. These weights are computed by the $valueGraph$ function. 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, in order to return each of them with its temporal accuracy.

With the function $valueGraph$, the graph is scanned. For each vertex, the weight is computed with respect to $windowSize$ and each of the edges with respect to $minGap$ and $maxGap$. Then for each frequent sequence, the algorithm $calcGenPrec$ computes the temporal accuracy thanks to the formula 5.

---

valueGraph - **Input:** $G_d(S, A)$, sequence graph (not weighted)
of a data sequence d
$\mu_{ws}, \mu_{mg}, \mu_{MG}$, membership functions.
Output: $G_d(S, A)$, weighted sequence graph

For each $s \in S$ **do**
    s.valuate($\mu_{ws}(s.\text{end}()-s.\text{begin}())$) ;
    For each $t \in s$.succ() **do**
        arc(s,t).valuate($\top(\mu_{mg}(t.\text{begin}()-s.\text{end}()), \mu_{MG}(t.\text{end}()-s.\text{begin}()))$) ;
    End For
End For

Algorithm 8: valueGraph

---

calcGenPrec - **Input:** $\mathcal{C}$, set of objects and their sequence graph
T set of frequent sequences,
$\mu_{ws}, \mu_{mg}, \mu_{MG}$, membership functions.
Output: T set of frequent sequences, with their temporal accuracy.

For each $c \in \mathcal{C}$ **do**   **For each** $t \in T$ **do**
    valueGraph($g_c,\mu_{ws},\mu_{mg},\mu_{MG}$); t.degPrec $\leftarrow t$.degPrec/$\mathcal{C}$;
    calcPrec(g_c,T) ;   **End For**
End For

Algorithm 9: calcGenPrec

5 All the maximal sequences?

Definition 2 *An itemset i is included in another itemset j iff both following conditions hold:*

- *$i.begin() \geq j.begin()$,*

- *$i.end() \leq j.end()$,*

We proove in this section that GETC builds exactly all the frequent sequences of maximal length supported by the input data sequence.

Theorem 1 *The inclusion of a sequence is not possible after GETC running.*

Proof 1 *Suppose that it exists two sequences s_1 and s_2 in the sequence graph such that $s_1 \subset s_2$.*
It means that the sequence graph contains a vertex y such that one of its predecessors x is accessible from t, also predecessor of y, i.e. it exists a path in the graph $(t, ..., y)$ of length greater or equal to 2 and an edge (t,y).
By building conditions, this case can only appear during the $propagate$ function running. The existence of such a path implies an intersection betwenn the vertices preceeding y and those following t. However if this situation appears, $propagate$ does not create a new edge from t to y. Thus GETC only builds the maximal path and deletes the included ons.

Theorem 2 *The GETC algorithm exactly builds all the solutions of maximal length satisfying both constraints $minGap$ and $maxGap$.*

Proof 2 *The GETC algorithm scans each vertex of the graph. That implies that if a path satisfying $minGap$ and $maxGap$ contains a vertex x, then this path is included into the graph after running the GETC algorithm (even if its only vertex is x).*
Every vertices are included in a path. The inclusion of path is impossible and if two paths $(x, ..., y)$ and $(y', ..., z)$ can be combined (i.e. if $y'.start\text{-}time() - y.end\text{-}time() > g_\rho$ and $z.end\text{-}time() - x.start\text{-}time() > G_\rho$), they will be, because the algorithm will build the edge (y,y') while exploring the vertex y.
So the GETC algorithm builds exactly all the solutions of maximal length satisfying the $minGap$ and $maxGap$ constraints.

Theorem 3 *The $addWindowSize$ algorithm builds exactly every vertices that may be used for building every maximal length sequences satisfying the $minGap\tilde{o}$ and $maxGap$ constraints.*

Proof 3 *Suppose that every vertices resulting of the itemsets combination satisfying $windowSize$ are built. Some of them are unecessary for the maximal length sequence search.*
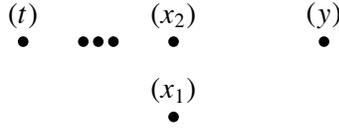
Figure 4: $addwindowSize$

First of all, consider the case in which the graph contains a level with two vertices x_1 and x_2 such that the itemsets of both vertices are the same and $x_1.end() = x_2.end()$ and $x_1.begin() < x_2.begin()$. Consider also a vertex t such that $t.begin() \leq t.end() < x_1.begin()$ and a vertex y such that $x_1.end() < y.begin() \leq y.end()$ (figure 4).
We show that every sequences including x_1 can also be obtained by including the vertex x_2 and that, consequently, just the vertex x_2 is necessary for the search of maximal sequences.

Suppose that it exists an edge between t and x_1, it means that the $minGap$ and $maxGap$ constraints are satisfied. So:

$$\begin{cases} x_1.end() - t.begin() & \leq & maxgap \quad a \\ x_1.begin() - t.end() & > & mingap \quad b \end{cases} \qquad (6)$$

And $x_1.end() = x_2.end()$, so equation 6-a becomes $x_2.end() - t.begin() \leq maxgap$. The $maxGap$ constraint is then also satisfied between t and x_2.
On a également $x_1.begin() < x_2.begin()$, c'est-à-dire :

$$\begin{aligned} x_2.end() - t.begin() & > & x_1.end() - t.begin() \\ \text{and so:} & & \\ x_2.begin() - t.end() & > & mingap \end{aligned} \qquad (7)$$

The $minGap$ constraint is then also satisfied between t and x_2. So a path going through t then x_2 will be built.

22

Now, suppose that such an edge cannot be built between t and x_1. It means that one of the minGap and maxGap constraints is not satisfied.
If the maxGap constraint is not satisfied, as $x_1.end() = x_2.end()$, the constraint will not be satisfied between t and x_2. If, on the contrary, it is the constraint minGap that is not satisfied, it may be satisfied with x_2. Indeed:

$$
\begin{array}{rcl}
x_1.begin() & < & x_2.begin() \\
x_1.begin() - t.end() & < & x_2.begin() - t.end()
\end{array}
\tag{8}
$$

It means that even if $x_1.begin() - t.end() \leq mingap$, we cannot conclude that $x_2.begin() - t.end() \leq mingap$.

We have prooven that every sequence including x_1 is a subsequence of a sequence including x_2 and also that if a path do not go through x_1, it may go through x_2. The vertex x_1 is then redundant regarding the vertex x_2 for the building of a path arriving to the itemset represented by the vertices x_1 and x_2.

Now, suppose that there is an edge between x_1 and y, it means that both constraints minGap and maxGap are satisfied, so:

$$
\left\{
\begin{array}{rcll}
y.end() - x_1.begin() & \leq & maxgap & a \\
y.begin() - x_1.end() & > & mingap & b
\end{array}
\right.
\tag{9}
$$

And, $x_1.end() = x_2.end()$, equation 9-b becomes $y.begin() - x_2.end() > mingap$. The minGap constraint is also satisfied between x_2 and y.
We also have $x_1.begin() < x_2.begin()$, i.e. :

$$
\begin{array}{rcl}
-x_1.begin() & > & -x_2.begin() \\
y.end() - x_1.begin() & > & y.end() - x_2.begin() \\
\textit{and so:} & & \\
y.begin() - x_2.end() & \leq & maxgap
\end{array}
\tag{10}
$$

The maxGap constraint is then also satisfied between x_2 and y. So a path going through x_2 then y will be built.

Suppose now that no edges can be buiilt between x_1 and y. It means that one of the constraints minGap and maxGap is not satisfied.
If the minGap constraint is not satisfied, as $x_1.end() = x_2.end()$, it will not be satisfied between x_2 and y. If, on the contrary, it is the maxGap constraint that is not satisfied, it may be satisfied with x_2. Indeed:

$$
\begin{array}{rcl}
x_1.begin() & < & x_2.begin() \\
-x_1.begin() & > & -x_2.begin() \\
y.end() - x_1.begin() & > & y.end() - x_2.begin()
\end{array}
\tag{11}
$$

It means that even if $t.end() - x_1.begin() > maxgap$, we cannot conclude that $t.end() - x_2.begin() > maxgap$.

We have shown that all sequences including x_2 is a subsequence of a sequence including x_2 and also that if a path does not go through x_1, then it may go through x_2. The vertex x_1 is redundant with respect to the vertex x_2 for the building of a path coming from the itemset represented by the vertices x_1 and x_2.

<u>*Conclusion:*</u> *for two vertices representing the same itemset and having the same ending time, the vertex having the latest starting time is the only one being necessary for the research of the maximal sequences.*
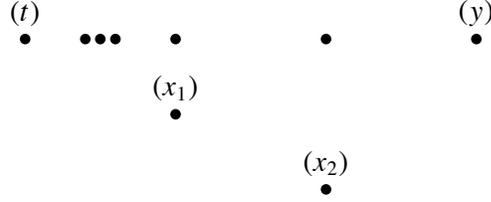
Figure 5: $add\,window\,Size$

Consider now the case in which the graph contains into two successive levels two vertices x_1 and x_2 such that the itemsets of both vertices are the same and that $x_1.begin() = x_2.begin()$ and $x_1.end() < x_2.end()$. Consider also a vertex t such that $t.begin() \leq t.end() < x_1.begin()$ and a vertex y such that $x_2.end() < y.begin() \leq y.end()$ (figure 5).
We shox that all sequences including x_2 can also be obtained by including the vertex x_1 and that, consequently, only the vertex x_1 is necessary for the research of maximal length sequence.

Suppose that there is an edge from t to x_2, it means that the constraints $minGap$ and $maxGap$ constraints are satisfied, so:

$$\begin{cases} x_2.end() - t.begin() & \leq & maxgap & a \\ x_2.begin() - t.end() & > & mingap & b \end{cases} \tag{12}$$

However, $x_1.begin() = x_2.begin()$, equation 12-b becomes $x_1.begin() - t.end() > mingap$. The $minGap$ constraint is then also satisfied between t and x_1.
We also have $x_1.end() < x_2.end()$, i.e. :

$$\begin{aligned} x_1.end() - t.begin() & < & x_2.end() - t.begin() \\ & \text{and so:} \\ x_1.begin() - t.end() & \leq & maxgap \end{aligned} \tag{13}$$

The $maxGap$ constraint is then also satisfied from t to x_1. So, a path will be built going through t then through x_1.

Suppose now that no edge can be built between t and x_2. It means that one of the constraints $minGap$ and $maxGap$ is not satisfied.

24

If it is the $minGap$ constraint that is not satisfied, as $x_1.begin() = x_2.begin()$, it will also not be satisfied between t and x_1. If, on the contrary, it is the $maxGap$ constraint that is not satisfied, it may be satisfied with x_1. Indeed :

$$
\begin{array}{lcl}
x_1.end() & < & x_2.end() \\
x_1.end() - t.begin() & < & x_2.end() - t.begin()
\end{array}
\tag{14}
$$

It means that even if $x_2.end() - t.begin() > maxgap$, we cannot conclude that $x_1.end() - t.begin() > maxgap$.

We have shown that a sequence including x_2 is a subsequence of a sequence including x_1 and also that if a path does not go through x_2, it will not go through x_1. The vertex x_2 is then redundant with respect to the vertex x_1 for the building of a path arriving to the itemset represented by the vertices x_1 and x_2.

Suppose now that there is an edge from x_2 to y, it means that the $minGap$ and $maxGap$ constraints are satisfied. So:

$$
\left\{
\begin{array}{lcll}
y.end() - x_2.begin() & \leq & maxgap & a \\
y.begin() - x_2.end() & > & mingap & b
\end{array}
\right.
\tag{15}
$$

However, $x_1.begin() = x_2.begin()$, equation 15-a becomes $y.end() - x_1.begin() \leq maxgap$. The $maxGap$ constraint is then also satisfied between x_1 and y.
We also have $x_1.end() < x_2.begin()$, i.e.:

$$
\begin{array}{lcl}
-x_1.begin() & > & -x_2.begin() \\
y.end() - x_1.begin() & > & y.end() - x_2.begin() \\
\text{and so:} & & \\
y.end() - x_2.begin() & \leq & maxgap
\end{array}
\tag{16}
$$

The $maxGap$ constraint is then also satisfifed between x_1 and y. A path will then goes through x_1 then through y.

Suppose now that no edges can be built from x_2 and y. It lmeans that one of the two constraints is not satisfied.
If it is the $maxGap$ constraint that is not satisfied, as $x_1.begin() = x_2.begin()$, it will also not be satisfied between x_1 and y. If, on the contrary, it is the $minGap$ constraint that is not satisfied, it may be satisfied with x_1. Indeed :

$$
\begin{array}{lcl}
x_1.end() & < & x_2.end() \\
-x_1.end() & > & -x_2.end() \\
y.begin() - x_1.end() & > & y.begin() - x_2.end()
\end{array}
\tag{17}
$$

It means that even if $y.begin() - x_2.end() \leq mingap$, it cannot be concluded that $y.begin() - x_1.end() \leq mingap$.

We have shown that all sequences including x_2 is a subsequence of a sequence including the vertex x_1 and also that if a path does not go through x_2, it may go

through x_1. The vertex x_2 is then redundant with respect to the vertex x_1 for the building of path going from the itemset representd by the vertices x_1 and x_2.

    Conclusion: for two vertices representing the same itemset and starting at the same timestamp, only the one with the smallest ending time is necessary for the search of maximal length sequences.

    The $addWindowSize$ algorithm builds exactly all the vertices that could contribute in the building of all the maximal length solutions satisfying the $minGap$ and $maxGap$ constraints.

Theorem 4 *The GETC algorithm builds exactly every sequences of maximal length satisfying the $windowSize$, $minGap$ and $maxGap$ constraints.*

Proof 4 *From the theorem 2, GETC builds exactly every solutions of maximal length satisfying the $minGap$ and $maxGap$ constraints. We still have to verify that the processing of the $windowSize$ constraint does not allow inclusion.*
Suppose that the sequence graph computed by GETC contains two sequences s_1 and s_2 such that $s_1 \subset s_2$. It means that the sequence graph contains a subgraph such that one of its vertices y, included into another vertex z and such that y.next() $\subseteq$ z.next().
But the algorithm $addEdge$ and $propagate$ have marked such a vertex y during the building of the edges and the algorithm $pruneMarked$ deletes every marked vertices. By the building process, such inclusion is then impossible.

6   An Example

Consider the dataset in Table 3 (from the data, $M = 17$ and $m = 0$) and the following parameters for soft time constraints:

- for $windowSize$, $ws_{init}=2$ and $\rho_{ws} = 0.87$, which then yields $ws_\rho=4$;
- for $maxGap$, $MG_{init}=4$ and $\rho_{MG} = 0.84$, which then yields $MG_\rho=6$;
- for $minGap$, $mg_{init}=2$ and $\rho_{mg} = 0.5$, which then yields $mg_\rho=1$.

6.1   Sequence Graph Building

The first step consists in building the sequence graph for data sequence O1. First, the vertex set is initialized: each record is associated to one vertex. This is the first line of the graph on Figure 6. Then the $windowSize$ constraint is applied on

Timestamp	1	3	4	5	6	8	9	10	12	17	18
O1	1	-	2 3	3 4	4	4	-	5	6	7	8
O2	2 3	4	-	-	5	6	-	-	-	-	-
O3	1 2	-	3	3 4	4	-	5 6	-	-	-	-

Table 3: A dataset

each possible combination of vertices using *addWindowSize*. Only the combinations satisfying the soft $windowSize$ constraint (i.e. *end-time*($O1_i$)-*start-time*($O1_i$) $\leq ws_\rho$=4) are kept.

Figure 6: Sequence graph for $O1$ at the end of vertex set creation by *addWindowSize*
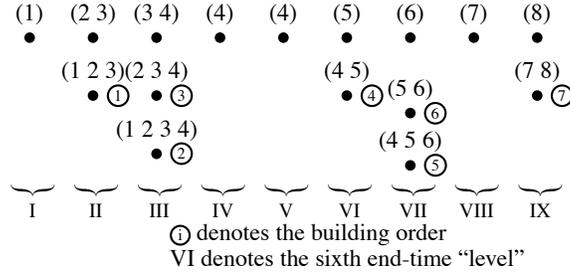
Then edges satisfying both $minGap$ and $maxGap$ soft constraints, are added to the graph using the main function and the *propagate* and *addEdge* subfunctions. After this step, every subsequence of the initial data sequence satisfying the three soft constraints is in the graph on Figure 7[1].

However some inclusions can remain. Last step consists in deleting these inclusions, using subfunction *pruneMarked*. The final sequence graph obtained from data sequence $O1$ is described by Figure 8.

The longest sequences supported by data sequence $O1$ are then:
- <(1 2 3)(4)(4 5 6)>   - <(1 2 3)(4)(4)(5)(6)(7 8)>   - <(1)(2 3)(4)(4)(5)(6)(7 8)>
- <(1 2 3)(4)(4 5)(6)>   - <(1)(2 3)(4)(4 5 6)>   - <(1)(2 3 4)(4)(5 6)>
- <(1 2 3)(4)(4)(5 6)>   - <(1)(2 3)(4)(4 5)(6)>   - <(1)(2 3 4)(4)(5)(6)(7 8)>
- <(1 2 3 4)>   - <(1)(2 3)(4)(4)(5 6)>   - <(1)(2 3 4)(4 5)(6)>

---

[1] Note that in Figure 7, some vertices have been moved up from where they were in Figure 6, to improve the graph legibility.

Figure 7: Sequence graph for $O1$ after edge creation, the circled numbers show the edge creation order by GETC
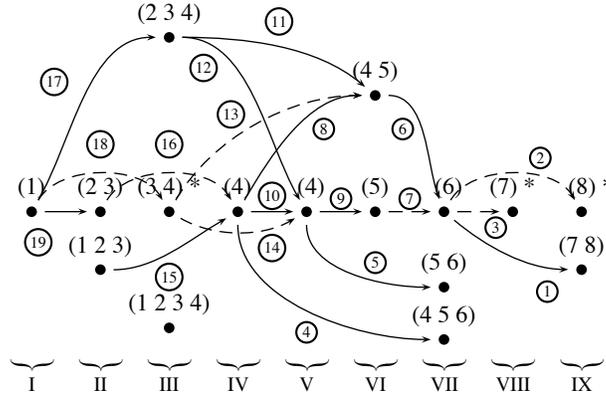
Figure 8: Final sequence graph for data sequence $O1$
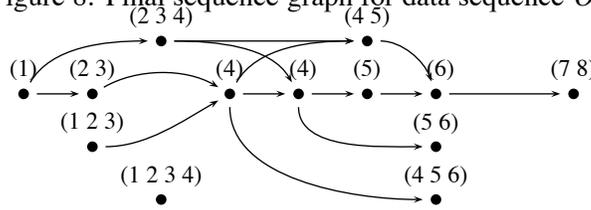
6.2 Temporal Accuracy of Extracted Patterns

From database Table 3 and soft time constraints specified in section 6.1, sequence graph of each data sequence is built. Then sequential patterns are mined for. Generalized sequential patterns obtained with *minFreq* = 70%, are: $< (2\ 3\ 4) >$, $< (2\ 3)(4)(5\ 6) >$, $< (2)(4\ 5) >$, $< (3\ 4)(5) >$, $< (3\ 4)(6) >$ and $< (3)(4\ 5) >$, each having a 100% frequency.

In order to analyse their relevancy according to user needs, their temporal accuracy is computed. To do so, the sequence graphs are weighted, as described in section 4.4. Vertices built with *ws*=0,1,2 has a weight of 1, with *ws*=3, a weight of 0.93 and with *ws*=4, a weight of 0.87. Edges built with *mg*=1 are weighted to 0.5 for *minGap* and to 1 if *mg*=2. For *maxGap*, the wieght is 1 if $MG \leq 4$, 0.92 if MG=5 and 0.84 if MG=6.

These weights are used to compute the temporal accuracy of extracted patterns. Results are presented on Table 4.

Once patterns were obtained with their temporal precision, we can analyze more exactly the constraints used to generate them. The more the precision is close to 1,

28

Figure 9: Weighted sequence graphs for data sequences O1, O2 and O3

$(2\ 3\ 4)_1$ $(4\ 5)_1$
1 0.84 0.5 0.5
$(1)_1$ $(2\ 3)_1$ $(4)_1$ $(4)_1$ $(5)_{1}$ $(6)_1$ $(7\ 8)_1$
1 1 0.5 0.5 0.5 0.84
$(1\ 2\ 3)_{0.93}$ 0.5 0.5 0.5
0.5 $(5\ 6)_1$
$(1\ 2\ 3\ 4)_{0.87}$ 0.5 $(4\ 5\ 6)_{0.87}$ data sequence O1

$(4\ 5)_{0.93}$ $(5\ 6)_1$
0.5 1 0.5
$(2\ 3)_{1}$ $(4)_1$ $(5)_1$ $(6)_1$
0.5 1 1 0.5
$(2\ 3\ 4)_1$ 1

data sequence O2

$(4\ 5\ 6)_{0.93}$
0.5
$(1\ 2)_1$ $(3)_1$ $(3\ 4)_1$ $(4)_1$ $(5\ 6)_1$
1 1 1 1
1 $(1\ 2\ 3)_{0.93}$ 0.5
0.5
$(1\ 2\ 3\ 4)_{0.87}$
data sequence O3
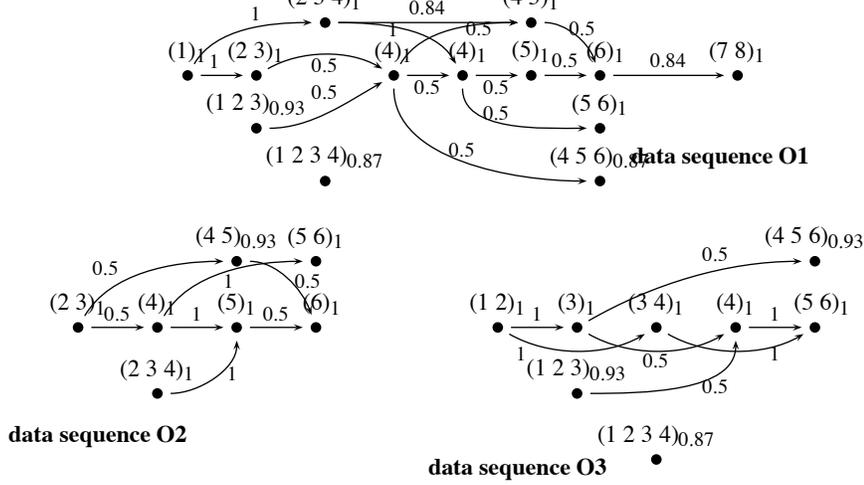
Sequential patterns	ϱ_{Cl1}	ϱ_{Cl2}	ϱ_{Cl3}	Υ
$< (2\ 3\ 4) >$	1	1	0.87	0.96
$< (2\ 3)(4)(5\ 6) >$	0.5	0.5	0.5	0.5
$< (2)(4\ 5) >$	0.84	0.5	1	0.78
$< (3\ 4)(5) >$	0.84	1	1	0.95
$< (3\ 4)(6) >$	0.5	0.5	1	0.67
$< (3)(4\ 5) >$	0.84	0.5	0.5	0.61

Table 4: Temporal accuracy computation of discovered sequential patterns

the more the initial values specified by the user correspond to the timestamps in the database. On the contrary, a weak precision indicates that the constraints are not well suited to this dataset.

7   Experiments

In this section, we present a comparison of performances of the algorithm GETC (for soft constraints) and the GTC algorithm (for crisp constraints). In a first part we compare the behaviors of these algorithm, also using an implementation of PSP, integrating or not the management of the constraints of time. In a second phase we compare what patterns are extracted using soft and crisp constraints, first with synthetical dataset then on web access logs. These experiments were carried out on a PC - Linux 2.6.7 OS, CPU 2,8GHz with 2GB of DDR memory. All the algorithms were implemented in C++ and use the PSP principle and structure to mine for seqeuntial patterns.

7.1 Synthetic Datasets and GETC overall behavior

Results presented here were obtained from the processing of several synthetic dataset containing approximately 1000 sequences of 20 records on average. Each of these records contains an average of 15 items chosen among 1000.

The first phase consisted in comparing runtime without time constraint: $windowSize$=0, $minGap$=0 and $maxGap$=∞ for GTC as well as for GETC, with a minimum accuracy equal to 1 for each soft constraint. So, we compared the runtime of our algorithm with those of PSP and GTC and we shown that GETC behavior is similar to that of GTC and that runtime are almost identical, to extract the same patterns.

We then repeated these measures by processing time constraints of time, with an accuracy of 1 to compare the behavior of GETC and GTC for the handling crisp time constraint. Figure 10(a) show the runtime evolution according to the value of $windowSize$. GETC has a linear behavior close to that of the GTC. The difference results from the temporal accuracy calculation step, by which the time increases slightly with $windowSize$, the number of vertices in the sequence graphs increasing with this parameter.

Finally, Figure 10(b) shows the runtime evolution according to the accuracy for a minimum frequency of 0.37. Note that the runtime reaches a limit value which corresponds to the limit values of the soft time constraints (*M* and *m*).

Figure 10: Runtime: (a) according to $windowSize$ with $minGap$=2, $maxGap$=∞ and $minFreq$=0.35 (for GETC, ρ_{ws}=ρ_{MG}=ρ_{mg}=1); (b) according to accuracy depending on several time constraints ($minFreq$=0.37); (c) according to $minFreq$ regarding soft time constraints with accuracy equal to 1 or not.

(a)   (b)   (c)
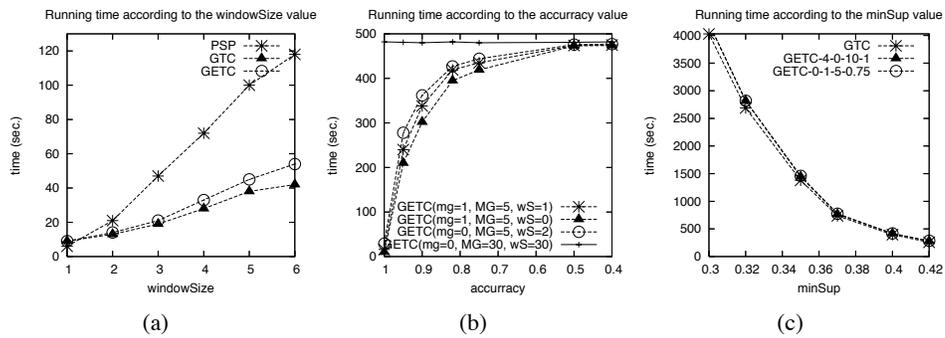
The second part of our experiments concerned the analysis of the sequential patterns extracted by GETC, with regard to the patterns extracted by GTC, according to the accuracy of the various soft constraints. Figure 10(c) presents runtime for GTC and GETC according to the minimum frequency according to values chosen

for each parameter. These values were calculated so that the time constraints used for GTC and for GETC with an accuracy of 1 correspond to the limit values of GETC with a precision different from 1. These parameters are:

- GETC with ws_{init}=0, mg_{init}=1 and MG_{init}=5, with ρ=0.75, then $ws_\rho = 4$, $mg_\rho = 0$ and $MG_\rho = 10$,

- GETC with ws_{init}=4, mg_{init}=0 and MG_{init}=10 with $\rho = 1$,

- GTC with ws_{init}=4, mg_{init}=0 and MG_{init}=10,

We note that GETC with soft time constraints is not more expensive than GTC with the limit values of the constraints, while obtaining the same sequential patterns, detailed by their temporal accuracy.

Besides, it can be interesting, in case we ignore the optimal value of one or several time constraints, to use GETC with an minimum accuracy level different from 1, to extend the search space. The analysis of the discovered patterns and their accuracy can inform about a more adequate time constraint value. So, we compared patterns extracted by GTC with the patterns extracted by GETC, with the same initial time constraints. The number found patterns is then greater, as shown by Figure 11(a). By ordering them in decreasing order of accuracy, we found all the patterns extracted by GTC (which have an accuracy equal to 1) then a list of patterns of lower temporal accuracy corresponding to the soft constraints. This histogram also shows that for this synthetic dataset, the constraints allowing to extract the most patterns correspond to an accuracy in 0.8, 0.9.

Figure 11: (a): Patterns repartition depending on their temporal accuracy on synthetical data (ws_{init}=0, mg_{init}=1, MG_{init}=5 and ρ_{ws}=ρ_{mg}=ρ_{MG}=0.5); (b): Patterns repartition depending on their temporal accuracy on synthetical data (ws_{init}=0, mg_{init}=0, MG_{init}=0 and ρ_{ws}=ρ_{mg}=1, ρ_{MG}=0.75, $minFreq$=0.2.

(a) (b)
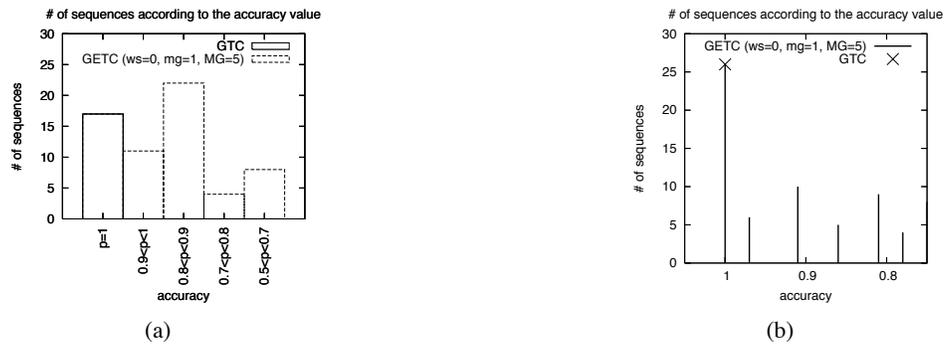
7.2 Soft Constraints to Mine Atypical Web Access

The aim of these experiments is to show that generalized sequential patterns extracted under soft time constraints bring more precise information compared to those obtained with crisp time constraints.

In this case, access logs from a private photo gallery website have been mined to analyse atypical behaviors. 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, the other requires an identification via login and password. We have analyzed the errors logs and have isolated access logs corresponding to these errors and also to non usual access (i.e.: access not browsing pages and not identified by web browsers).

The "atypical" access logs are preprocessed: each IP adress becomes the object_id, each connection request is coded as an item, as well as the request type, the returned error and connecting software, timestamp is the date since January, 1st 1970 in seconds.

Example 9

IP id	time	request	URL	error	software
253	5	GET	"/PictureGallery/home.htm"	404	"Mozilla/4.0"
253	11	GET	"/PictureGallery/index.htm"	-	-

Table 5: Example of access logs

Table 9 represents the access of the IP encoded by 253 to URL "/PictureGallery/home.htm" with request "GET" by the software "Mozilla/4.0". Error returned was error 404 (page does not exit). This access is followed 6 seconds later by the same request to URL "/PictureGallery/index.htm". The software was not identified and no error was returned.

This atypical access log represents around 22,000 connection attempts during approximately one year, by 510 different IPs, to 1181 different URLs. First we compared performances of GETC to those of GTC. We have found the same global behavior for both algorithm even if GETC still is a little slower tha GTC, because of temporal accuracy computation. The interestingness of GETC shown by these experiments stands in the additionnal information given by temporal accuracy. In fact, we have reproduced the second step of experiments on synthetical datasets: we applied to GTC crisp constraints that correspond to the initial values for GETC. We have then compared extracted sequential patterns.

First of all we had to chose the values to specify as initial ones. As we were mining for description of atypical behaviors we have decided to identify non human profiles. This kind of profiles can be for example characterized by repeated requests over a short period. That means that 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). The minimum gap between two itemsets will take its minimum possible

32

value, without varying (mg_{init}=0 and ρ_{mg}=1). The constraint corresponding to the behavior we want to highlight is $maxGap$. Automatical requests can be viewed as requests to close to be done by human beings, so the maximum gap separating two itemsets should be the shortest, 1 second. However we would like to be sure not to ignore other profiles, so we decide to soften this constraint: we specify a temporal accuracy less than 1 for $maxGap$(ρ_{MG}=0.75). We thus use the flexibility of soft constraint $maxGap$ in order to more precisely describe the atypical profiles.

Then we have compared different results obtained by varying 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, when we obtained more patterns corresponding to soft constraints, one part of the being also discovered with crisp constraints. Histogram 11(b) shows the number of patterns for each temporal accuracy extracted by GETC and those mined by GTC: every patterns extracted by GTC is found by GETC with temporal accuracy equals to 1.

We also have a more relevant information as we have more descriptive patterns for atypical behavior and each of them is detailed with its temporal accuracy. We give some description of atypical behaviors we found with both algorithms:

- The bot pxyscand sent a CONNECT request to the URL 1185 of the web site. It received the 405 Error (Unauthorized 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 access to the URL "/mambo..." which does not exist. It seems to be a hacking attempt ($<$("/cvs/mambo/index2.php?_REQUEST...", 404 Error)("/cvs/mambo/index2.php?_REQUEST...", 404 Error)$>$).

Additionnal information obtained by GETC and not found by GTC is for example that:

- This sequential pattern has a temporal accuracy of 0.95 and its frequency is 35%. It means that the software Pompos tried to retrieve unexisting pages:
  $<$("cvs/index2.php?request...",404 Error,Pompos)...once...
  ("cvs/index2.php?request...",404 Error,Pompos)$>$.

- One non-human typical access is the bot scanning, in particular the Image Bot from Google. This bot does not appear when we use the crisp constraints. Analysing the access logs we see that the average gap between two requests is 2 seconds
  $<$("/PictureGallery/thumbnail.php3?...",Googlebot-Image/1.0)...once...
  ("/PictureGallery/thumbnail.php3?...',Googlebot-Image/1.0)...$>$.

8   Conclusion and Perspectives

Generalized sequential patterns presented by (Srikant and Agrawal, 1996) redefined the inclusion of sequences in a broader way by introducing the use of time constraints. These constraints, which allow to gather records or to separate them in different sequences, enable the user to discover less obvious knowledge and closer to his needs. However, this definition remains still too rigid, in particular if the user only has a vague idea of the time constraints which bind its data. In this article

we thus propose to soften these time constraints for generalized sequential patterns, using some principles of the fuzzy set theory. We so give more flexibility to the specification of the time constraint parameters. The implementation of our approach is based on the construction of sequence graphs to handle the time constraints during the sequential pattern mining process. The feasibility and the robustness of this method were shown for GTC in (Masseglia et al., 2004). The principle of GETC being the same, we were able to show its efficiency to solve the problem of mining for generalized sequences with crips or soft time constraints and the resemblance of its behavior with that of the GTC. We also highlighted the flexibility offered by the soft constraints, as well as the interest of the temporal accuracy measure to analyse sequential patterns by running experiments on both synthetical and real-life datasets; it still remains us to validate the robustness of it. Finally, we intend to extend the fuzzy sequential patterns presented in (Fiot et al., 2006b) to generalized sequential patterns, with time constraints (crisp or soft).