Extended Time Constraints for Sequence Mining

Céline Fiot, Anne Laurent, Maguelonne Teisseire

Extended Time Constraints for Sequence Mining

Céline Fiot Anne Laurent Maguelonne Teisseire
LIRMM - Univ. Montpellier II, CNRS
161 rue Ada, 34392 Montpellier, France
{fiot, laurent, teisseire}@lirmm.fr

Abstract

Many applications require techniques for temporal knowledge discovery. Some of those approaches can handle time constraints between events. 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. Therefore, in this paper we propose to soften temporal constraints used for generalized sequential pattern mining. To handle these constraints while data mining, we design an algorithm based on sequence graphs. Moreover, as these relaxed constraints may extract more generalized patterns, we propose temporal accuracy measure for helping the analysis of the numerous discovered patterns.

1 Introduction

In many fields searching for temporal knowledge is necessary, for instance in order to identify temporal profiles or detecting frauds and failures. Some learning techniques can handle such knowledge. [4] defines operations and rules associated with time intervals. In data mining, some approaches can extract recurring episodes within long sequences [12, 17] or sequence databases [1, 2, 13]. Searching for such information is all the more interesting that it can handle constraints between events: shortest or longest time-gap between two events [18, 20, 14, 15], or regular expression and repetition constraints [9, 6, 10, 3].

Within this framework, generalized sequential pattern mining was introduced in [18]. This data mining technique extracts frequent sequences that meet user-specified constraints from a sequence database (e.g. successive purchases of customers in a supermarket).

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. In this domain, to our knowledge, no papers have proposed an 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.
To make the constraints specification easier, we propose a method that softens user-specified time constraints. We also propose an efficient algorithm, GETC (Graph for Extended Time Constraints), in order 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 generalized sequential patterns 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.

In this paper, we describe our proposal including the following points. First, we define extended time constraints for generalized sequential patterns. Secondly, we state a definition of time accuracy for frequent sequence analysis. Lastly we focus on the algorithm we designed in order to handle soft time constraints and to compute time accuracy.

In the next section, we define the fundamental concepts associated with sequential patterns and generalized sequential patterns and we introduce existing algorithms that manage time constraints. In section 3, we briefly introduce the fuzzy set theory. We define the soft time constraints and the temporal accuracy of a sequence. Then section 4 details our algorithm to implement the handling of soft time constraint. Section 5 illustrates our proposal on a running example. We then propose some experiments on synthetic data and we give brief results about experiments run on web access logs in section 6, thus showing the benefits of our soft time con-

straints and measure. 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 generalized sequential pattern mining task. It broadly summarizes the formal description of the problem introduced in [2, 18].

2.1 Sequential patterns

Sequential patterns were initially defined in [2] 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, which is an integer, 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 $(i_1 i_2 \dots i_k)$. 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 a customer 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 are 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 should also mention 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 allocating 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 the 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)* $\geq$ *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*) [2]. 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 as simultaneous. On the contrary, two events that are too distant could have no link together. That is why generalized sequential patterns were proposed in [18], introducing time constraints in order to improve the subsequence definition.

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, [18] redefined 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* $\forall i \in [1, n]$*:*

i. $s_i \subseteq \cup_{k=l_i}^{u_i} d_k$*;*
ii. timestamp(d_{u_i}*) - timestamp(*d_{l_i}*)* $\leq windowSize$*;*
and $\forall i \in [2, n]$,
iii.timestamp(d_{l_i}*) - timestamp(*$d_{u_{i-1}}$*)* $> minGap$*;*
iv. timestamp(d_{u_i}*) - timestamp(*$d_{l_{i-1}}$*)* $\leq maxGap$*;*

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.

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 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).

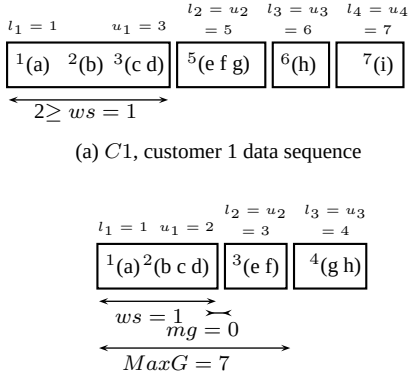
These time constraints, as well as the minimum frequency condition, are parameterized by the user.

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.*

Table 1. Purchases made by two customers during one week

Days	1	2	3	4	5	6	7
C1	a	b	c d		e f g	h	i
C2	a	b c d	e f	g h			

Let s =<$(a\ b\ c\ d)$> 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 Fig. 1 shows how these time constraints are applied in order to determine whether data sequences C1 and C2 support the candidate sequence s =< $(a\ b\ c\ d)$ > or not. 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 are no other possibilities to find s in this data sequence. Thus data sequence C1 does not support 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 was 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). Data sequence C2 supports sequence s.



(a) $C1$, customer 1 data sequence

(b) $C2$, customer 2 data sequence

$^i(a\ b)$ denotes the itemset $(a\ b)$ bought at date i

Figure 1. Time constraints, $windowSize$ (ws), $minGap$ (mg) and $maxGap$ ($MaxG$), on data sequences C1 and C2

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 items in an itemset come from a single record.

2.3 Related work

In the next sections (3 and 4) we introduce our extension of time constraints and how handling them with our algorithm GETC. Before that, we describe in this section some approaches for generalized sequential pattern mining under time constraints.

Various algorithms were proposed to handle these constraints. Some push them directly into the mining process, like the GSP algorithm [18] and the DELISP algorithm [11]. 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, proposed in [14], is based on this principle.

The GSP algorithm proposed in [18] aims 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 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 [13], another approach called PSP (Prefix-Tree for Sequential Patterns) was proposed. It again fully utilizes the fundamental principles of GSP, while using a different structure for organizing the candidate sequences, which thus improves retrieval efficiency.

More recently, the DELISP algorithm [11] was 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 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, although there exists a lot of different approaches and structures for sequential pattern mining [5, 21, 16].

Therefore the GTC algorithm [14] was developed. 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. Since this approach is efficient and adapted to any sequential pattern mining algorithm, we instigated our own algorithm GETC, detailed in section 4, using the same structure and principles as GTC.

3 Soft time constraints

The main drawback of time constraints, as defined in section 2.2, 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 optimal time constraints for sequential pattern mining. We here propose to extend the above time constraints for generalized sequential patterns using some Fuzzy Set Theory principles.

Moreover, extracted patterns become increasingly numerous, particularly during sequential pattern mining. So it becomes necessary to provide the end-user with tools to analyse the sequential patterns obtained. In the case of generalized sequential patterns, 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 fulfills the initial user-specified values of the time constraints. We thus provide the user with flexibility in time constraint specification and with a tool to help analyze the extracted patterns.

3.1 Fuzzy Set Theory

The Fuzzy Set Theory was introduced by [19]. This theory generalizes crisp set theory, assuming 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]. An exemple of these three fuzzy sets is graphically represented on Fig. 2. Thus a person of 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).*

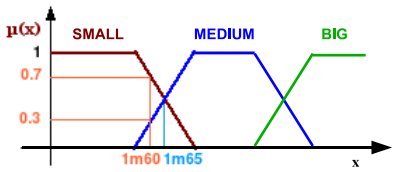


Figure 2. BIG **and** SMALL **fuzzy sets describing a person's height**

Fuzzy logic operators 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 $\perp$ 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 the operator $\top$ (resp. $\perp$) generalized to the n-ary case. Different operators can be used as a t-norm (min, $product$...). 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 as 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 fulfill users' 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}.
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 want constraint $\mathcal{X}$ to take each possible value; the temporal satisfaction degree depends 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 $\mathcal{X}$ to vary. That constraint is set at the initial value and will not change; all generated sequences have a temporal satisfaction degree of 1.
- In all other cases, $\rho_x \in]0,1[$ and the value x of constraint $\mathcal{X}$ may 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 generalized sequential patterns as introduced in section 2.2.

The useful limit value of time constraints (extreme values) first have to be determined. These values 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 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:

$$M = \max_{o \in \mathcal{O}}(timestamp(r_{o_{max}}) - timestamp(r_{o_{min}})) \quad (1)$$

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. Then, the limit value m for $minGap$ is:

$$\max(\min_{o \in \mathcal{O}}(\min_{r \in \mathcal{R}_o}(timestamp(r+1) - timestamp(r))) - 1, 0) \quad (2)$$

All details about this definition can be found in [7].

In the remainder of this section, we use the notations given in TABLE 2, 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 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.

Table 2. Notations

M	maximal possible value for $windowSize$ and $maxGap$ given by equation 1
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 $minGap$ given by equation 2
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 $minGap$

3.3 Extending time constraints to soft time constraints

We now detail how the soft time constraints are built, using the extension of $windowSize$ as an example. Then we illustrate each soft time constraint with a brief example.

The value ws of $windowSize$ 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 (3) gives the accuracy for a specific

value ws:

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

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:

$$ws_\rho = \lfloor (ws_{init} - M)\rho_{ws} + M \rfloor \quad (4)$$

Example 5 *Consider the two data sequences shown in example 3 and the sequence s=<(a b c d)>. Suppose that the user-specified time constraint parameters are ws_{init}=1, mg_{init}=0 and MG_{init}=7, with ρ_{ws}=0.7, ρ_{mg}=1 and ρ_{MG}=1.*
Thus, ρ_{ws} is the only lowest acceptable satisfaction degree different from 1, and $windowSize$ is thus the only constraint possibly having several values. Applying equation (4) 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 this ws value, by equation (3), yields a temporal satisfaction degree $\rho(ws) = \rho(2) = 0.8$. The other constraints are fulfilled as well, then data sequence C1 supports candidate sequence s.
For data sequence C2, the purchases of days 1 and 2 are grouped together. Note that the $windowSize$ constraint is fulfilled with ws=ws_{init}=1 and the corresponding satisfaction degree is $\rho(ws) = 1$. The other constraints are also respected, data sequence C2 supports sequence s.

The $maxGap$ constraint is described with a fuzzy set whose the membership function (5) gives the accuracy for a specific value MG:

$$\rho(MG) = \begin{cases} 1 & \text{if } MG \leq MG_{init} \\ \frac{MG-M}{MG_{init}-M} & \text{if } MG_{init} < MG \leq M \\ 0 & \text{else} \end{cases} \quad (5)$$

As a results, the largest acceptable gap is given by:

$$MG_\rho = \lfloor (MG_{init} - M)\rho_{MG} + M \rfloor \quad (6)$$

Example 6 *Consider the two data sequences shown in example 3 and the candidate sequence s=<(a b c d) (g h)>. Suppose that the user-specified time constraint parameters are ws_{init}=2, mg_{init}=0 and MG_{init}=3, with ρ_{ws}=1, ρ_{mg}=1 and ρ_{MG}=0.3.*
Thus ρ_{MG} is the only lowest acceptable satisfaction degree different from 1 and $maxGap$ is thus the only varying constraint. Applying equation (6) 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 candidate sequence s. However, with MG=3, the $maxGap$ constraint is violated, as with MG=4. With MG=5, this constraint is fulfilled and the temporal satisfaction degree is then $\rho(MG) = \rho(5) = 0.3$ (5). The other constraints are also fulfilled, then data sequence C1 supports candidate sequence s.
For data sequence C2, the records of days 1 and 2 are grouped together, meeting the $maxGap$ constraint with MG=MG_{init}=3 and the corresponding satisfaction degree is $\rho(MG = 3) = 1$. The other constraints are also fulfilled, and data sequence C2 supports sequence s.

The $minGap$ constraint is described with a fuzzy set whose the membership function (7) gives the accuracy for a specific value mg:

$$\rho(mg) = \begin{cases} 1 & \text{if} \quad mg \geq mg_{init} \\ \frac{mg-m}{mg_{init}-m} & \text{if} \quad mg_{init} > mg \geq m \\ 0 & \text{else} \end{cases} \quad (7)$$

As a results, the smallest acceptable gap is given by:

$$mg_\rho = \lceil (mg_{init} - m)\rho_{mg} + m \rceil \quad (8)$$

Example 7 *Consider the two data sequences shown in example 3 and the candidate sequence s=<(a b c d) (e f)>. Suppose that the user-specified time constraint parameters are ws_{init}=2, mg_{init}=2 and MG_{init}=7, with ρ_{ws}=1, ρ_{mg}=0 and ρ_{MG}=1.*
Thus, ρ_{mg} is the only lowest acceptable satisfaction degree different from 1 and $minGap$ is thus the only varying constraint. Applying equation (8) with $m = 0$, we get mg_ρ=0. mg will successively be 2, 1 and 0.
For data sequence C1, the purchases of days 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 (7)). The other constraints are fulfilled, so data sequence C1 supports sequence s. For the second data sequence, the purchases of days 1 and 2. The $minGap$ constraint is not fulfilled while mg is greater than 0. mg=0 yields, by equation (7), to a temporal satisfaction degree $\rho(mg) = \rho(0)$= 0. The other constraints are fulfilled, data sequence $C2$ supports s.

Note that these soft constraints defined above are based on 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 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 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. 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 ($\perp$).

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

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

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

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

Example 8 *Consider the two data sequences from example 3 and the frequent sequence $s =< (a\ b\ c\ d)\ (e\ f) >$ 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 for the generalized t-norm ($\overline{\top}$) and the generalized t-conorm ($\overline{\perp}$) respectively.*
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. 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):

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

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 [14] and sketched 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 fulfills 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 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 described in Fig. 3 includes all the fundamental principles of GTC. It contains a number of iterations. Each iteration finds all 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 fulfills 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. 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.

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

Figure 3. Main algorithm

4.2 Sequence Graph Building

From an input data sequence d, the GETC algorithm (Fig. 4) builds a sequence graph $\mathcal{G}_d(V, E)$ in which vertices are itemsets and paths represent subsequences satisfying 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 satisfying 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 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 fulfilled and the $maxGap$ constraint is checked. If it is fulfilled, 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*, [7].

We have proven 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, [7]. 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 fulfill the crisp constraints while others are 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 $\top(\mu_{mg}(y.\text{begin}()\text{-}x.\text{end}()),\mu_{MG}(y.\text{end}()\text{-}x.\text{begin}()))$ 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, Fig. 5. 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.

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

V ← buildVertices(d); addWindowSize(V) ;
While ($x \neq$ V.first()) **do**
  $l \leftarrow x$.level().prec() ; $mg \leftarrow mg_{init}$;
  While (x.start-time() - l.end-time() $\leq mg$) **do**
    $contmg \leftarrow$ FALSE ;
    If (x.start-time() $> l$.end-time()) **Then**
      While ($mg \geq mg_\rho$) **do**
        If (constming(x,l)) **Then**
          $contmg \leftarrow$ TRUE ; $mg \leftarrow mg_\rho$-1 ;
        Else
          mg - - ;
        End If
      End While
    End If
    If ($contmg$ == FALSE) **Then**
      propagate(x,l) ; $l \leftarrow l$.prec() ;
    End If
  End While
  For chq $w \in l$ **do**
    $included \leftarrow$ TRUE ; $MG \leftarrow MG_{init}$;
    While ($MG \leq MG\rho$) **do**
      If (constMaxG(x,w)) **Then**
        addEdge(w,x) ; $MG \leftarrow MG_\rho + 1$;
      Else
        MG++;
      End If
    End While
  End For
  $x \leftarrow$ S.next(x) ;
End While
pruneMarked($G_d(V, E)$) ; convertEdges($G_d(V, E)$) ;
return $G_d(V, E)$;

Figure 4. GETC

valueGraph - **Input:** $G_d(V, E)$, non valued sequence graphe
for a data sequence d
$\mu_{ws}, \mu_{mg}, \mu_{MG}$, membership functions
for time constraints.
Output: $G_d(V, E)$, valued sequence graph for d

For each $s \in V$ **do**
  s.valuate(μ_{ws}(s.end()-s.begin()) ;
  For each $t \in s$.succ() **do**
    edge(s,t).valuate($\top$(μ_{mg}(t.begin()-s.end()),
    μ_{MG}(t.end()-s.begin())) ;
  End For
End For

Figure 5. valueGraph

5 A Running 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 $\rho_{ws} = 0.86$, then ws_ρ=4; for $maxGap$, MG_{init}=4 and $\rho_{MG} = 0.84$, then MG_ρ=6; for $minGap$, mg_{init}=2 and $\rho_{mg} = 0.5$, then mg_ρ=1.

Table 3. Dataset: data sequences of three objects over 18 Days

	1	3	4	5	6	8	9	10	12	17	18
O1	a	-	b c	c d	d	d	-	e	f	g	h
O2	b c	d	-	-	e	f	-	-	-	-	-
O3	a b	-	c	c d	d	-	e f	-	-	-	-

5.1 Sequence Graph Building

The first step consists of building the sequence graph for data sequence O1. First, the vertex set is initialized: each record is associated with one vertex. This is the first line of the graph in Fig. 6. Then the $windowSize$ constraint is applied on each possible combination of vertices using *addWindowSize*. Only combinations fulfilling the soft $windowSize$ constraint (i.e. *end-time*($O1_i$)-*start-time*($O1_i$) $\leq ws_\rho$=4) are kept.

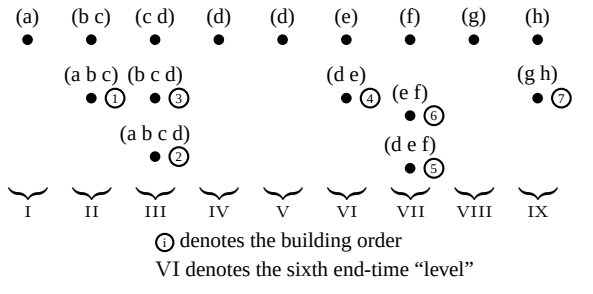


Figure 6. Sequence graph for $O1$ at the end of vertex set creation by $addWindowSize$

Then edges fulfilling both $minGap$ 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 (g h). The first level that can access (g h) that fulfills $minGap$ is VII, then we build an edge for each vertex in this level if $maxGap$ is fulfilled. The first edge is then from (f) to (g h). When a level cannot attain a vertex v because of $minGap$, we need to check if the vertices of this level can access 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 7[1].

However some inclusions[2] may remain. The last step consists of deleting these inclusions, using subfunction *pruneMarked*. The final sequence graph obtained from data sequence $O1$ is described by Fig. 8.

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

[2]The potentially included subsequences are shown with dashlines in Fig. 7. The potentially included vertices are marked (*) during the edge creation step.

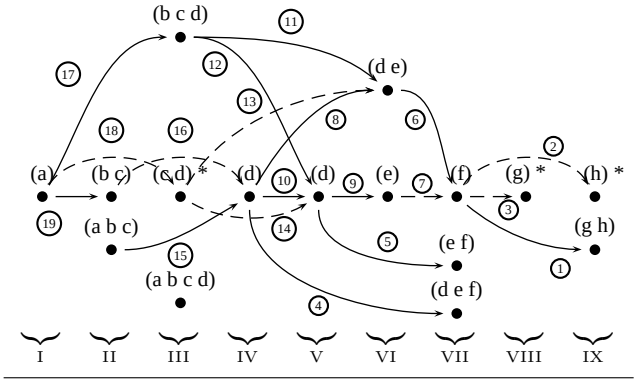


Figure 7. Sequence graph for $O1$ after edge creation, showing edge creation

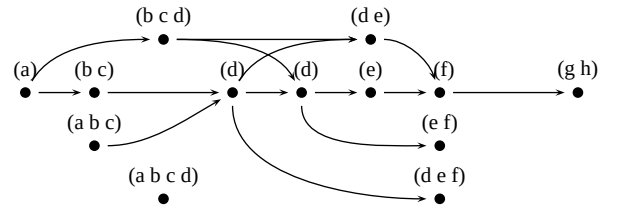


Figure 8. Final sequence graph for data sequence $O1$ of TABLE 3

The longest sequences supported by data sequence $O1$ are then:

- <(a b c)(d)(d e f)>
- <(a b c)(d)(d e)(f)>
- <(a b c)(d)(d)(e f)>
- <(a b c)(d)(d)(e)(f)(g h)>
- <(a)(b c)(d)(d)(e)(f)(g h)>
- <(a)(b c d)(d)(e)(f)(g h)>
- <(a b c d)>
- <(a)(b c)(d)(d e f)>
- <(a)(b c)(d)(d)(e f)>
- <(a)(b c d)(d)(e f)>
- <(a)(b c d)(d e)(f)>

5.2 Temporal Accuracy of Extracted Patterns

The sequence graph of each data sequence is built, from TABLE 3 and soft time constraints specified in the example statement. Then sequential patterns are mined for. The generalized sequential patterns obtained with *minFreq* = 70%, are those in TABLE 4, with each having a 100% frequency.

In order 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.

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, have a weight of 0.87. 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 on Fig. 5.2.

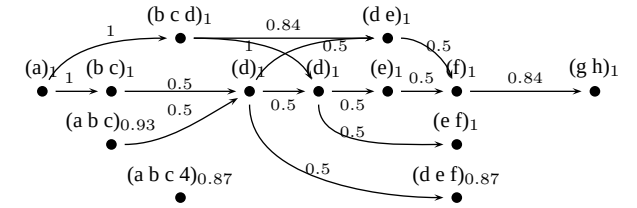


Figure 9. Weighted sequence graphs for data sequences $O1$

Finally, these weights are used to compute the temporal accuracy of extracted patterns. The corresponding results are presented in TABLE 4.

Table 4. Temporal accuracy computation of discovered sequential patterns

Sequential patterns	ϱ_{Cl1}	ϱ_{Cl2}	ϱ_{Cl3}	Υ
$< (b\ c\ d) >$	1	1	0.87	0.96
$< (b\ c)(d)(e\ f) >$	0.5	0.5	0.5	0.5
$< (b)(d\ e) >$	0.84	0.5	1	0.78
$< (c\ d)(e) >$	0.84	1	1	0.95
$< (c\ d)(f) >$	0.5	0.5	1	0.67
$< (c)(d\ e) >$	0.84	0.5	0.5	0.61

Once patterns have been obtained with their temporal precision, we can more accurately analyze 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.

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. We compare the behaviors of these algorithms, while also using an implementation of PSP, while integrating the handling of time constraints. In a second phase, we compare the patterns extracted using either soft or crisp constraints on synthetic data. All of these experiments were carried out on a PC - Linux 2.6.7 OS, CPU 2,8 GHz with 2 GB of DDR memory. All the algorithms were implemented in C++ and use the PSP principle and structure to search for sequential patterns.

The results presented here were obtained through processing of several synthetic 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 runtime without time constraints ($windowSize$=0, $minGap$=0 and $maxGap$=∞) for GTC and for 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 behavior is similar to that of GTC, i.e. runtimes are almost identical to extract the same patterns.

We then repeated these measures by processing crisp time constraints (with an accuracy of 1 for GETC) to compare the behaviors of GETC and GTC. Fig. 10 shows the runtime pattern as a function of the $windowSize$ value. GETC has a linear behavior 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.

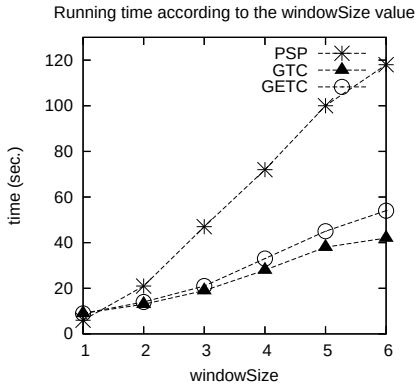


Figure 10. Runtime as a function of $windowSize$ **with** $minGap$**=2,** $maxGap$**=**∞ **and** $minFreq$**=0.35 (for GETC,** ρ_{ws}=ρ_{MG}=ρ_{mg}=1**)**

Finally, Fig. 11 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.

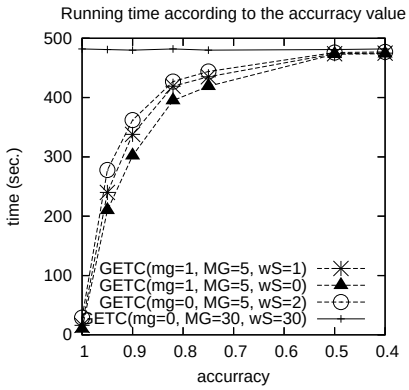


Figure 11. Runtime as a function of accuracy depending on several time constraints ($minFreq$**=0.37)**

The second part of our experiments on synthetic datasets dealt with an analysis of sequential patterns extracted by GETC, compared to those extracted by GTC, according to the accuracy required for the various soft constraints.

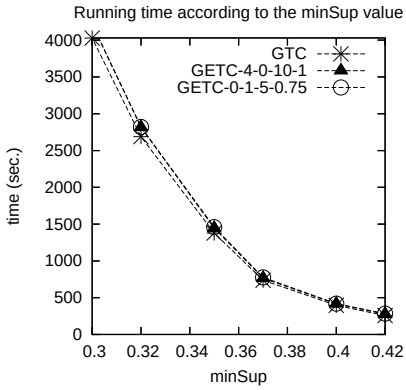


Figure 12. Runtime as a function of $minFreq$ **regarding soft time constraints with accuracy equal to 1 or not**

Fig. 12 presents 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 for 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 ρ=0.75, with ws_{init}=0, mg_{init}=1 and MG_{init}=5, yielding $ws_\rho = 4$, $mg_\rho = 0$ 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, in order to extend the search space. Analysis of the retrieved patterns and their accuracy can inform us about a more adequate time constraint value. So we compared patterns extracted by GTC with patterns extracted by GETC, with the same initial time constraints. The number of detected patterns is then greater, as shown on Fig. 13.

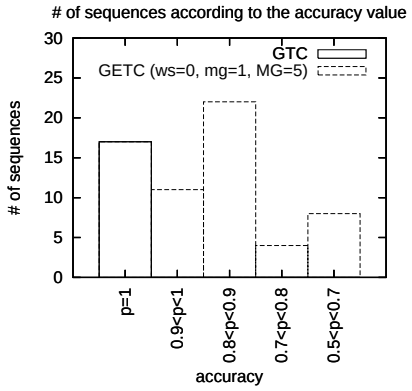


Figure 13. Pattern distribution depending on their temporal accuracy on synthetic data (ws_{init}**=0,** mg_{init}**=1,** MG_{init}**=5 and** ρ_{ws}**=**ρ_{mg}**=**ρ_{MG}**=0.5)**

By classifying them in decreasing order of accuracy, we got all 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 synthetic dataset, the constraints allowing us to extract the largest number of patterns corresponding to an accuracy of between 0.8 and 0.9.

Some experiments were done on web access logs to analyse atypical behaviors. Comparison of performances and runtimes between GETC and GTC gave quite the same results as with synthetic data. Regarding the quality of sequential patterns discovered by GETC, we obtained more relevant information, since we got more descriptive patterns for atypical behavior, each of them being provided with its

temporal accuracy. Moreover, some additional information is obtained by GETC, but not uncovered by GTC.

7 Conclusion and perspectives

The generalized sequential patterns presented in [18] redefine the inclusion of sequences in a broader way by introducing time constraints. These constraints, which allow the user to gather records or to separate them into different sequences, can highlight less immediate knowledge and closer to his/her needs. However, this definition is still too rigid, in particular if the user has only a vague idea of the time constraints which bind his data. In this article, we thus propose to soften these time constraints for generalized sequential patterns, 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 show the efficiency of our GETC algorithm to solve the problem of mining for generalized sequences under crisp or soft time constraints. We also highlight 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 synthetic and real-life datasets. Finally, we intend to extend the fuzzy sequential patterns presented in [8] to generalized sequential patterns, with time constraints (crisp or soft) in order to mine quantitative timestamped data under time constraints.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[2] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *11th International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[3] H. Albert-Lorincz and J.-F. Boulicaut. Mining Frequent Sequential Patterns under Regular Expressions: a Highly Adaptative Strategy for Pushing Constraints. In *3rd SIAM Int. Conf. on Data Mining (SIAM DM'03)*, pages 316–320, 2003.

[4] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Readings in qualitative reasoning about physical systems*, pages 361–372, 1990.

[5] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick. Sequential pattern mining using bitmaps. In *8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2002.

[6] M. Capelle, C. Masson, and J.-F. Boulicaut. Mining Frequent Sequential Patterns under a Similarity Constraint. In *3rd Int. Conf. on Intelligent Data Engineering and Automated Learning (IDEAL'02)*, pages 1–6, 2002.

[7] C. Fiot, A. Laurent, and M. Teisseire. Extended time constraints for generalized sequential patterns. Technical Report 6051, LIRMM, 2005.

[8] C. Fiot, A. Laurent, and M. Teisseire. From crispness to fuzziness: Three algorithms for soft sequential pattern mining. *IEEE Transactions on Fuzzy Systems*, to appear, 2007.

[9] M. Garofalakis, R. Rastogi, and K. Shim. Mining Sequential Patterns with Regular Expression Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):530–552, 2002.

[10] M. Leleu, C. Rigotti, J.-F. Boulicaut, and G. Euvrard. Constraint-Based Mining of Sequential Patterns over Datasets with Consecutive Repetitions. In *7th Eur. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, pages 303–314, 2003.

[11] M.-Y. Lin, S.-Y. Lee, and S.-S. Wang. DELISP: Efficient discovery of generalized sequential patterns by delimited pattern-growth technology. In *6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD '02)*, pages 198–209, 2002.

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

[13] F. Masseglia, F. Cathala, and P. Poncelet. The PSP Approach for Mining Sequential Patterns. In *Principles of Data Mining and Knowledge Discovery*, pages 176–184, 1998.

[14] F. Masseglia, P. Poncelet, and M. Teisseire. Pre-processing time constraints for efficiently mining generalized sequential patterns. In *11th International Symposium on Temporal Representation and Reasoning (TIME '04)*, pages 87–95, 2004.

[15] N. Meger and C. Rigotti. Constraint-based mining of episode rules and optimal window sizes. In *8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'04)*, pages 313–324. Springer-Verlag, 9 2004.

[16] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefixprojected pattern growth. In *Int. Conf. Data Engineering (ICDE'01)*, pages 215–224, 2001.

[17] C. Raissi, P. Poncelet, and M. Teisseire. Need for SPEED: Mining Sequential Patterns in Data Streams. In *21 ème Journée Bases de Données Avancées (BDA '05)*, 2005.

[18] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *5th International Conference on Extending Database Technology (EDBT '96)*, pages 3–17, London, UK, 1996. Springer-Verlag.

[19] L. Zadeh. Fuzzy sets. *Information and Control*, 3(8):338–353, 1965.

[20] M. J. Zaki. Sequence mining in categorical domains: incorporating constraints. In *9th International Conference on Information and Knowledge Management (CIKM '00)*, pages 422–429, New York, NY, USA, 2000. ACM Press.

[21] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Mach. Learn.*, 42(1-2):31–60, 2001.