Alcom-FT Technical Report Series
ALCOMFT-TR-03-197

Discovering Unbounded Episodes in Sequential Data *

G. Casas-Garriga

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona Salgado 1-3, Barcelona
gcasas@lsi.upc.es

Abstract. One basic goal in the analysis of time-series data is to find frequent interesting episodes, i.e, collections of events occurring frequently together in the input sequence. Most widely-known work decide the interestingness of an episode from a fixed user-specified window width or interval, that bounds the length of the subsequent sequential association rules. We present in this paper, a more intuitive definition that allows, in turn, interesting episodes to grow during the mining without any user-specified help. A convenient algorithm to efficiently discover the proposed unbounded episodes is also implemented. Experimental results confirm that our approach results useful and advantageous.

1 Introduction

A well-defined problem in Knowledge Discovery in Databases arises from the analysis of sequences of data, where the main goal is the identification of frequently-arising patterns or subsequences of events. There are at least two related but somewhat different models of the sequential pattern mining. In one of them each piece of data is a sequence (such as the aminoacids of a protein, the banking operations of a client, or the occurences of recurrent illnesses), and one desires to find patterns common to several pieces of data (proteins with similar biological functions, clients of a similar profile, or plausible consequences of medical decisions). See [2] or [7] for an introduction to this model of a sequential database. The second model of sequential pattern matching is the slightly different approach proposed in [6], where data come in a single, extremely long stream, e.g. a sequence of alarms in a telecommunication network, in which some recurring patterns, called *episodes*, are to be found.

Both problems seem similar enough, but we concentrate here on the second one of finding episodes in a single sequence. Abstractly, such ordered data can be viewed as a string of events, where each event has an associated time of occurrence. An example of an event sequence is represented in Figure 1. Here A, B and C are the event types, such as the diferent types of user actions marked on a time line.

* This work is supported in part by EU ESPRIT IST-1999-14186 (ALCOM-FT), and MCYT TIC 2002-04019-C03-01 (MOISES)

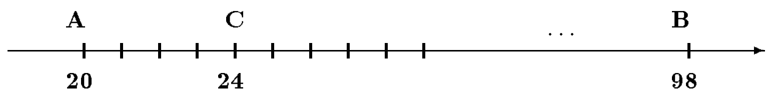

Fig. 1. A sequence of events.

We briefly describe the current approaches to interesting episodes, point out some disadvantages, and then propose, as our main contribution, an alternative approach for defining a new kind of serial episodes, i.e *unbounded episodes*. We finally explain how previous algorithms for finding frequent sets can be applied to our approach, and suggest an interpretation of parallel episodes as summaries of serial episodes, with the corresponding algorithmic consequences. Finally, we describe the results of a number of preliminary experiments with our proposals.

2 Framework Formalization

To formalize the framework of the time-series data we follow the terminology, notation, and setting of [6]. The input of the problem is a *sequence* of events. Given a set E of event types, an *event* is a pair (A, t) where $A \in E$ is an event type and t is its occurrence time.

An event sequence is a triple (s, T_s, T_e), where T_s is called the starting time of the sequence, T_e is the ending time, and s has the form: $s = \langle (A_1, t_1), \ldots, (A_n, t_n) \rangle$ where A_i is an event type, and t_i is the associated occurrence time, with $T_s \leq t_i < t_{i+1} \leq T_e$ for all $i = 1, \ldots, n-1$. The time t_i can be measured in any time unit, since this is actually irrelevant for our algorithms and proposals.

2.1 Episodes

Our desired output for each input sequence is a set of frequent episodes. An *episode* is a partially ordered collection of events occurring together in the given sequence. Episodes can be described as directed acyclic graphs. Consider, for instance episodes α, β and γ in Figure 2. Episode $\alpha = B \rightarrow C$ is a *serial episode*: event type B occurs before event type C in the sequence. Of course, there can be other events occurring between these two in the sequence. Episode $\beta = \{A, B\}$ is a *parallel episode*: events A and B occur frequently close in the sequence, but there are no constraints about the order of their appearences. Finally, episode γ is an example of *hybrid episode*: it occurs in a sequence if there are occurences of A and B and these precede an occurrence of C, possibly, again, with other intervening events.

More formally, an episode can be defined as a triple $(V, \leq, g)$ where: V is a set of nodes, $\leq$ is a partial order relation on V, and $g : V \rightarrow E$ is a mapping associating each node with an event type. We also define the size of an episode as the number of events it contains, i.e, $|V|$. The interpretation of an episode is that events in $g(V)$ must occur in the order described by $\leq$. In this paper we will only deal with serial and parallel episodes.

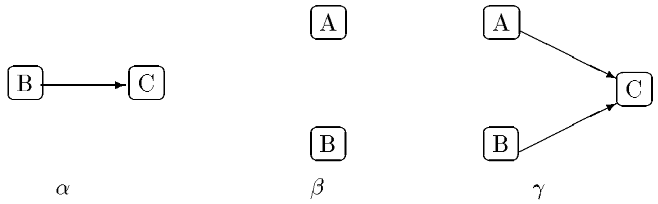

Fig. 2. Types of episodes

Definition 1. *An episode $\beta = (V', \leq', g')$ is a* **subepisode** *of $\alpha = (V, \leq, g)$, noted by $\beta \subseteq \alpha$, if there exists an injective mapping $f : V' \to V$ such that $g'(v) = g(f(v))$ for all $v \in V'$, and for all $v, w \in V'$ with $v \leq' w$ also $f(v) \leq f(w)$.*

3 Classical Approaches to Define Interesting Episodes

In the analysis of sequences we are interested in finding all *frequent* episodes from a class of episodes which can be *interesting* to the user. In this section we will mainly take the classical widely-used work of [6] as a reference, i.e, we state that to be considered interesting, the events of an episode must occur close enough in time.

3.1 Winepi

In the first approach of [6], the user defines how close the events of an interesting episode should be by giving the width of the *time window* within which the episode must occur. The number of possible windows of a certain width win in the sequence (s, T_s, T_e) is exactly: $T_e - T_s + win - 1$, and we denote by $W(s, win)$ the set of all these windows of size win. Thereby, the frequency of an episode α in s is defined to be:

$$fr(\alpha, s, win) = \frac{|\{w \in W(s, win) | \alpha \ occurs \ in \ w\}|}{|W(s, win)|}$$

So, an episode is frequent according to the number of windows where that episode has occured, or to its ratio to the total number of possible such windows in the sequence. To be frequent, the ratio $fr(\alpha, s, win)$ of an episode must be over a minimum user-specified real value. The Winepi approach applies the Apriori algorithm to find the frequency of all the candidate episodes in the sequence as though each sliding window were a transaction with ordered events.

Once the frequent interesting episodes are discovered from the sequence, the second goal of the approach is to create the **episode association rules** that hold over a certain minimum confidence. For all episodes $\beta \subseteq \alpha$, an **episodal rule $\beta \Rightarrow \alpha$ holds with confidence**:

$$conf(\beta \Rightarrow \alpha) = \frac{fr(\alpha, s, win)}{fr(\beta, s, win)}$$

3.2 Minepi

Minepi is based on *minimal occurrences* of episodes in a sequence. For each frequent episode, the algorithm finds the location of its minimal occurrences. Given an episode α and an event sequence s, we say that the interval $w = [t_s, t_e)$ is a *minimal occurrence* of α in s, if:

(1) α occurs in the window w.
(2) α does not occur in any proper subwindow on w.

Basically, the applied algorithm is Apriori: it locates, for every episode going from the smaller ones to larger ones, its minimal occurrences. In the candidate generation phase, the location of minimal occurrences of a candidate α is computed as a temporal join of the minimal occurrences of two subepisodes of α.

This approach differs from Winepi in the fact that it does not use a frequency ratio to decide when an episode is frequent. Instead, an episode will be considered frequent when its number of minimal occurrences is over an *integer* value given by the user. This is a consequence of the fact that the lengths of the minimal occurrences vary, so that a uniform ratio could be misleading. One advantage of this approach is that allows the user to find final rules with two windows widths, one for the left-hand side and one for the whole rule, such as "if A and B occur within 15 seconds, then C follows withing 30 seconds". So, in this approach an **episode association rule** is an expression $\beta[win_1] \Rightarrow \alpha[win_2]$, where β and α are episodes such that $\beta \subseteq \alpha$, and win_1 and win_2 are integers specifying interval widths. The informal interpretation of the rule is that if episode β has a minimal occurrence at interval $[t_s, t_e)$ with $t_e - t_s \leq win_1$, then episode α occurs at interval $[t_s, t'_e)$ for some t'_e such that $t'_e - ts \leq win_2$.

The **confidence of an episode association rule** $\beta[win_1] \Rightarrow \alpha[win_2]$ with $\beta \subseteq \alpha$ and two user-specified interval widths win_1 and win_2 is the following:

$$conf(\beta[win_1] \Rightarrow \alpha[win_2]) = \frac{|\{[t_s,t_e) \in mo(\beta)\ s.t\ t_e - t_s \leq win_1\ and\ [t_s, t_s + win_2) \in mo(\alpha)\}|}{|\{[t_s,t_e) \in mo(\beta)\ and\ t_s - t_e \leq win_1\}|}$$

where $mo(\alpha)$ are the set of minimal occurrences of the episode α in the original input sequence. So, even if there is no fixed window size (as occurred in Winepi approach) and apparently minimal occurrences are not restricted in length, now the user needs to specify the time bounds win_1 and win_2 for the generation of the subsequent episode rules and their confidences. These values force minimal occurrences to be bounded in a fixed interval size of at most win_2 time units during the mining process.

3.3 Some Disadvantages of these Previous Approaches

We summarize below some of the observed disadvantages in Winepi and Minepi.

- In Winepi the window width is fixed by the user and it remains fixed throughout the mining. Consequently, the size of the discovered episodes is limited. Winepi just reduces the problem of mining the long event sequence to a sequential database (such as in [2]), where now each transaction is a fixed window.
- In Minepi the user specifies two time bounds for the creation of the subsequent episode association rules. These intervals make the final minimal occurrences to be bounded in size, since just those occurrences contained within the bounds are counted.
- Both Minepi and Winepi require the end user to fix one parameter with not much guidance on how to do it. Intervals or windows too wide can lead to misleading episodes where the events are widely separated among them; so, the subsequent rules turn out to be uninformative. On the other hand, interval or windows set too tight give rise to overlapping episodes: if there exists an interesting episode, α, whose size is larger than the fixed window width, then that episode will never fit in any window and, consequently, α will be discovered just partially.
- Minepi does not use a frequency ratio to decide whether an episode is frequent. This makes difficult the application of sampling in the algorithms of finding frequent episodes.
- In case the user decides to find the episode association rules for a different time bound (a different window size in Winepi or a different interval length for Minepi), then the algorithm that finds the source of frequent episodes has to be run again, incurring in a inconvenient overhead.
- Both approaches do not seem truly compatible for those problems where the adjancency of the events in the discovered episodes is a must (such as protein function identification). Neither Winepi or Minepi allow to set this kind of restriction between the events of an interesting episode.

4 Unbounded Episodes

In order to avoid all these drawbacks and be able to enlarge the window width automatically throughout the mining process, we propose the following approach. We will consider a serial or parallel episode interesting if it fulfills the following two properties:

(1) Its *correlative events* have a gap of at most tus time units (see figure 3).
(2) It is frequent.

$$\begin{array}{c} \quad tus \quad tus \\ A \rightarrow B \rightarrow C \\ \\ \alpha \end{array} \qquad\qquad \begin{array}{c} \quad tus \quad tus \\ \{A, \quad B, \quad C\} \\ \\ \beta \end{array}$$

Fig. 3. Example of serial and parallel unbounded episodes

So, in our proposal, the measure of interestingness is based on tus, the **time-unit separation** between correlative events in the episode. This number of time units must be specified by the user. The above two episodes α and β are examples of the interpretation of our approach. In the serial episode α, the distance between A and B is tus, and the distance between B and C is also tus time units. Besides, despite not specifying the distance between events A and C, it can be clearly seen they are at most $2 \times tus$ time units away. In the parallel episode β, distance between correlative events A, B and C, regardless of the order of their appearences in the sequence, must be of at most tus time units. More generally, an episode of size e may span up to $(e-1) \times tus$ time units.

Now, every episode that is candidate to be frequent, will be searched in windows whose width will be delimited by the episode size: an episode with e events will be searched in all windows in the sequence of width $(e-1) \times tus$ time units. Thus, the window width is not bounded, nor is the size of the episode, and both will grow automatically, if necessary, during the mining. This explains the name chosen: we are mining *unbounded episodes*.

At this point, it is worth mentioning the work of [8] (contributing with the algorithm cSPADE) and [7] (the algorithm GSP). These two papers integrate inside the mining process the possibility to define a max-gap constraint between the elements of the frequent sequences found in a sequential database. However, this max-gap constraint in [8] or [7] does not lead to an unbounded class of patterns as we present here. The reason is that they work on the sequential database problem, and so, the frequent mined patterns turn out to be naturally bounded by the lenght of the transactions in the database.

With our approach the window width is allowed to grow automatically without any predetermined limits. The frequency of an episode can be defined in the following way: let us denote by $W_k(s, win)$ the total set of windows in a sequence (s, T_i, T_f) of a fixed width $win = k \times tus$ time units (the number of such windows in the sequence is $T_e - T_f + win - 1$). Then:

Definition 2. *The* frequency *of an episode α of size $k{+}1$ in a sequence (s, T_i, T_f) is:*

$$fr(\alpha, s, tus) = \frac{|\{w \in W_k(s, win) | \alpha \ occurs\ in\ w\}|}{|W_k(s, win)|}$$

where $win = (|\alpha| - 1) \times tus = k \times tus$.

Note that the dependence on win, for fixed α, is here simply a more natural way to reflect the dependence on the user-supplied parameter tus, but both correspond to the same fact since win and tus are linearly correlated.

To sum up, every episode α will be frequent if its frequency is over a minimum user-specified frequency, that is, according to the number of windows in which it occurs; however, the width of that window depends on the number of events in α. So, the effect in the algorithm is that, as an episode size becomes bigger and the number of its events increases, the proper window in which that episode is searched also increases its width; and simultaneously the ratio that has to be compared with the user-specified desired frequency is appropriately adjusted.

4.1 Episode Association Rule with Unbounded Episodes

The approach of mining unbounded episodes will be flexible enough to allow the generation of association rules according to two interval widths (one for the left hand side, and one for the whole rule as occurred with Minepi).

An **unbounded episode rule** will be an expresion $\beta[n_l] \Rightarrow \alpha[n_r]$, where β and α are unbounded episodes such that $\beta \subseteq \alpha$, and n_l and n_r are integers such that $n_l = |\beta|$ and $n_r = |\alpha| - |\beta|$. The informal interpretation of these two new variables n_l and n_r is the number of events occurring in the left hand side (n_l) and new events implied in the right hand side (n_r) of the rule respectively.

So, we can rewrite any unbounded episode rule $\beta[n_l] \Rightarrow \alpha[n_r]$ in terms of a rule with two window widths $\beta[w_1] \Rightarrow \alpha[w_2]$ by considering $w_1 = (n_l - 1) \times tus$ and $w_2 = n_r \times tus$. This transformation will lead to an easy and informative interpretation of the rule: "if events in β occur within w_1 time units, then, the rest of the events in α will follow within w_2 time units".

One of the advantages of this proposed approach is that focusing our episode search on the time-unit separation between events, will allow to generate the best unbounded episode rule $\beta[n_l] \Rightarrow \alpha[n_r]$ (and so, the best rule $\beta[w_1] \Rightarrow \alpha[w_2]$) without fixing any other extra parameter: neither n_l or n_r will be user-specified for any rule, since these values will be chosen from the best antecedent and consequent maximizing the value of confidence for that rule (or in other words, n_l and n_r will be uniquely determined by the size of the episode being the antecedent and the size of the episode being the consequent in the best rule according to confidence ratio).

Since in our approach we have a ratio of frequency support, we can define the **confidence of a rule** $\beta \Rightarrow \alpha$ for $\beta \subseteq \alpha$ as:

$$conf(\beta \Rightarrow \alpha) = \frac{fr(\alpha, s, tus)}{fr(\beta, s, tus)}$$

where the value of $fr(\alpha, s, tus)$ for a fixed α, depends on the occurrences of α in all windows of lenght $(|\alpha| - 1) \times tus$ in the sequence. Note that since β is a subepisode of α, the rule right-hand side α contains information about the relative location of each event in it, so the "new" events in the rule right-hand can actually be required to be positioned between events in the left-hand side. The rules defined here are also rules that point forward in time (rules that point backwards can be defined in a similar way).

As we see, the values n_l and n_r of a rule do not affect the confidence, and they can be determined after having chosen the best rule by following the procedure:

for each maximal episode α,
$$\beta[|\beta|] \Rightarrow \alpha[|\alpha| - |\beta|] = arg.max\{conf(\beta \Rightarrow \alpha)\ s.t\ \beta \subseteq \alpha\}$$

So, the final windows widths ($w_1 = |\beta| \times tus$ and $w_2 = (|\alpha| - |\beta|) \times tus$) are determined by the best rule in terms of confidence, and this can vary from one rule to the other, adapting always to the best combination. Note that instead of confidence, any other well-defined metric for episodes could be used to select the best rule in this procedure, and so, different unbounded rules would be taken.

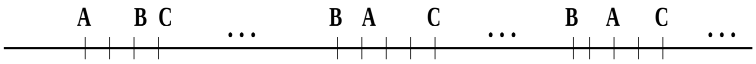

Fig. 4. Example of an event sequence

Example in figure 4 will serve to illustrate the advantatges of our unbounded episode approach. The sequence of this figure shows that we could consider frequent the episodes: $\beta = \{A, B\}$ and $\gamma = \{A, B\} \rightarrow C$ (as they are represented as a graph in figure 2). The best association rule we can find in this example is the following: $\{A, B\} \Rightarrow C$, that should have a confidence of 1 for this presented piece of sequence.

For Winepi, at least a fixed window of 5 time units of width should be specified to find both β and γ. But this parameter depends on the user and it is not intuitive enough to chose the right value. In this example, if the user decides a window width of 3 time units, then the episode γ would never be fully discovered and the rule will never be generated.

With Minepi, the problem comes when specifying the two windows widths for the episode association rule. In case the user decides $win_1 = 3$ and $win_2 = 4$, the generated rule would be $\beta[3] \Rightarrow \gamma[4]$, that has a confidence of just 1/3 in this example. It is not the best association rule, and it is due to the value of $win_2 = 4$, that it is set too tight. Besides, if the user wants to specify a wider win_2, the algorithm finding frequent minimal occurrences has to be run again.

For the unbounded approach however, one would find both β and γ by just specifing a big enough value for tus. This is an intuitive parameter, and the best subsequent episode rule in terms of confidence would be $\beta[2] \Rightarrow \gamma[1]$, with a confidence of 1. This rule can be transformed in terms of two window widths and interpret the following "if A and B occur within tus time units, then C will follow in next tus time units".

4.2 Advantages of Our Approach

We shortly summarize some advantages of our unbounded proposal.

- Since the window increases its width along with the episode size, the final frequent episodes do not overlap unnecesarily, and their size is not limited.
- The unbounded episode rules have better quality in terms of confidence without any previous user help.
- Unbounded episodes generalizes Minepi and Winepi in that episodes found with a window width of x time units can be found with our approach using a distance of $x - 1$ time units between correlative events.
- The application of sampling techniques are allowed.
- Once the frequent unbounded episodes are mined, finding the episode rules with two windows widths is straight. What is more, the user can try different windows widths for the rules, and chose the best width according to some

statistical metric. This does not affect the previous mining and the discovered unbounded episodes, and they are always the same once we are in the generating rule phase.
– Our proposal can be adapted to the sequential-database style by imposing a wide gap between the different pieces of data.

On the whole, we can say that unbounded episodes are more general and intuitive than Minepi or Winepi approaches. In particular, these unbounded episodes can be very useful in contexts such as the classification of documents or the intrusion detection systems. As argued in [5], a drawback of subsequence patterns is that they are not suitable for classifying long strings over small alphabet, since a short subsequence pattern matches with almost all long strings. So, the larger the episodes found in a text the better for the future predictions.

5 Algorithms to Mine Unbounded Episodes

Our proposed definition of unbounded episodes is flexible enough to still allow the use of previous algorithms. Besides, to prove the flexibility of the proposal, we also adapt here our strategy from [3], Best-First strategy, which is a non-trivial evolution of Dynamic Itemset Counting (DIC, [4]) and provides better performance than both Apriori and DIC. For better understanding, we give a brief account of how our Best-First strategy works.

Similarly to DIC, our strategy keeps cycling through the data as many times as necessary, counting the support of a number of candidate itemsets. Whenever one of them reaches the threshold that declares it frequent, it immediately "notifies" this fact to all itemsets one unit larger than it. In this way, potential future candidates keep being informed of whether each of their immediate predecessors is frequent. When all of them are, the potential candidate is promoted to candidate and its support starts to be counted. DIC follows a similar pattern but only tries to generate new candidates every M processed transactions: running it with $M = 1$ would be similar to Best-First strategy, but would incur overheads that our algorithm avoids thanks to the previous online information of which subsets of the potential candidates are frequent at each moment.

To follow the same structure, our new algorithm for mining episodes, called Episodal Best-First (**EpiBF**), will distiguish two sets of episodes: 1/ **candidate episodes** whose frequency is being counted, and 2/ **potential candidates** that will be incorcorated as candidates as soon as the monotonicity property of frequency is fulfilled. Hovewer, given that now we are using our unbounded approach of interestingness, we must relax the monotonicity property frequency for pruning unwanted candidates. Other Breadth-First algorithms, like Apriori or DIC, can be also easily applied by taking into account that at each new scan of the database for candidates of size k, the window width must be incremented conveniently (i.e, $(k-1) \times tus$). Apart from that, we also have to relax here the monotonicity property of frequency used in the candidate generation phase as we will see in short. We discuss separately the case of serial episodes first.

5.1 Discovering Serial Episodes

In case of using our approach of unbounded episodes, the well-known monotonicity property of frequency (stating that any frequent episode has all its subepisodes also frequent) does not hold: that is, a frequent unbounded episode could have some subepisode not frequent. For instance, let us consider the unbounded serial episode $A \rightarrow B \rightarrow C$, and its subepisode $A \rightarrow C$. They refer to two different classes of unbounded interestingness: while in $A \rightarrow B \rightarrow C$, events A and C are separated for at most $2 \times tus$ time units, in its subepisode $A \rightarrow C$ the events are separated at most tus time units. So, it might well be that since the gap between events is different in both episodes, $A \rightarrow C$ is not frequent while $A \rightarrow B \rightarrow C$ is frequent. We cannot use this property to prune unwanted candidates.

But we will relax this notion here and we will just consider those subepisodes whose events follow an adjacency of tus time-unit separation. For instance, to consider the episode $A \rightarrow B \rightarrow C$ a good candidate that deserves to be counted in the data, one has to find frequent just the subepisodes $A \rightarrow B$ and $B \rightarrow C$ (i.e, the overlapping parts of an unbounded episode). Then, it is true that any frequent unbounded episode has all its overlapping parts frequent.

Now, the algorithm EpiBF for serial episodes goes in the following way. It starts by initializing the set of candidate episodes with all episodes of size 2, and the set of potential candidates with all episodes of size 3. Then, it goes on counting the frequency of all the candidate episodes until this set becomes empty. When one of these candidate episodes of size k achieves the state of frequent, it increments counters corresponding to all the potential candidates of size $k + 1$ that we can obtain by adding one more event before it or after it. This growth leads to unbounded episodes. On the other hand, when a potential candidate of size $k + 1$ finds that both subepisodes of size k, obtained by chopping off either end, have been declared frequent, then it will be incorporated in the set of candidate episodes.

It is important to highlight that, in this algorithm, the set of candidate episodes can be made up of episodes of diferent sizes, and each episode α of size k must be searched and counted in all windows of width $(k-1) \times tus$ time units. This fact forces EpiBF to handle windows of different sizes at the same time by simply taking, at every step, the largest window for the longest episode in the set of candidate episodes. The rest of episodes in the set of candidate episodes will be searched in the proper subwindows.

5.2 Discovering Serial and Parallel Episodes Simultaneously

In case of mining parallel episodes the problem can be reduced efficiently to mining serial episodes in the following way. Every parallel episode of size k lumps together up to $k!$ serial episodes. For instance, the parallel episode $\{A, B\}$ gathers the following two serial episodes: $A \rightarrow B$ and $B \rightarrow A$. In this case, a serial episode will be called *participant* of a parallel episode. Clearly, any serial episode is participant of one, and only one, parallel episode.

Let us discuss what could be the meaning of parallel episode mining. Clearly, if a frequent parallel episode has some (but not all) participants already frequent, the desired output is the list of such frequent serial episodes: the parallel one, given alone, provides less information. In such cases we should not move from the serial episodes to the parallel one, unless actually *all* of them are frequent: in this last case, the parallel episode is an effective way of representing this fact. Thus, according to our proposal, in order to be considered interesting, a parallel episode α must fullfil one of the two following conditions: either

1. by adding up the frequency of the serial episodes that are participants of α, we reach the user-specified minimal frequency, *but* no serial episode participant of α is frequent alone; or
2. every serial episode participant of α is frequent.

From the point of view of the algorithm, any used strategy will mine serial episodes, but these serial episodes can refer to parallel ones too. Thereby, the set of candidate episodes will be made of serial ones, while the set of potential candidates will be composed of parallel episodes. This means that the algorithm will be counting the support of serial candidates, as in the previous case; however, when declaring one of these serial episodes, α, frequent or non-frequent, the notification must go to that parallel episode which α is participant of.

6 Experiments

In this section we present the results of running (a probabilistic version of the) EpiBF algorithm on a variety of different data collections.

First, we experimented, as in [6], with protein sequences. We used data in the PROSITE database of the ExPASy WWW molecular biology server of the Geneva University Hospital and University of Geneva [10]. The purpose of this experiment is to identify specific patterns in sequences so as to determine to which family of protein they belong. The sequences in the family we selected ("DNA mismatch repair proteins I", PROSITE entry PS00058, the same one used in [6] for comparison), are known to contain the string `GFRGEAL`. This string represents a serial episode of seven consecutive symbols separated by 1 unit of time among them. Parameter tus was set to 1, and the support threshold was set to 15, for the 15 individual sequences in the original data. Note that no previous knowledge of the pattern to be found is involved in this parameter setting.

As expected, we found in the database the pattern `GFRGEAL` along with 3,755 more serial episodes (whether maximal or not), most of them much shorter. When comparing our approach against previous ones, we see that both Winepi and Minepi need to know in advance the length of the expected pattern in the protein sequence, in order to fix the window width. However, it is usual that we do not know which pattern is to be found in a sequence; so, one must try the experiment with different window widths.

In order to see the flexibility of the unbounded episodes, we also run experiments with text data collections. In particular, we used a part of a text extracted from "Animal Farm" by Orwell [9]. Once again, setting tus close to 1 and considering each letter a new event, we are able to find frequent prepositions, articles, suffixes of words, and concatenations of words (such as "to", "in", "at", "ofthe", "was", "her", "ing" ...). This experiment could have been done considering an event to be every new word in the text; this will lead to unbounded episodes as a tool to classify other new texts.

When it comes to the general performance of the method, we found that, naturally, the larger the value of the parameter tus, the more discovered episodes. Besides, discovering our serial and parallel episodes simultaneously, allows the algorithm to discover parallel patterns when hardly serial patterns are found in the database. For example, fed with the first 40,000 digits of the Champernowne sequence (012345678910111213141516 ...), with a high frequency threshold of 50% and digits far apart at most 15 positions in the episodes ($tus = 15$), only 3 serial episodes were found but we discovered 15 other parallel episodes.

7 Conclusions

We present in this paper a more intuitive approach for interesting episodes. This proposal overcomes the disadvantages of the widely-used previous approaches (Minepi and Winepi), and it turns out to be an adaptative approach for categorical time-series data. The algorithmic consequences of the unbounded episodes are also discused and implemented. Finally, we have also introduced a new way of considering parallel episodes as a set of participant serial episodes. First experiments prove to be promising, but more experimentation on the different values of tus and their consequences in the subsequent rules is on the way.

References

1. R.Agrawal, H.Mannila, R.Srikant, H.Toivonen and I.Verkamo. Fast Discovery of Association Rules. *Advances in Knowledge Discovery and Data Mining*. 1996.
2. R.Agrawal and R.Srikant. Mining Sequential Patterns. *Proc. of the Int. Conf. on Data Engineering*. 1995.
3. J.Baixeries, G.Casas-Garriga, and J.L.Balcázar. A Best First Strategy for Finding Frequent Sets. *Extraction et gestion des connaissances* (EGC'2002), 100–106. 2002.
4. S.Brin, R.Motwani, J.Ullman and S.Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *Int. Conf. Management of Data*. 1997.
5. M.Hiaro, S.Inenaga, A.Shinohara, M.Takeda and S.Arikawa. A Practical Algorithm to Find the Best Episode Patterns. *Int. Conf. on Discovery Science*, 235–440. 2001.
6. H.Mannila, H.Toivonen and I.Verkamo. Discovery of frequent episodes in event sequences. *Proc. Int. Conf. on Knowledge Discovery and Data Mining*. 1995.
7. R.Srikant and R.Agrawal. Mining Sequential Patterns: Generalizations And Performance Improvements.*Proc. 5th Int. Conf. Extending Database Technology*. 1996.
8. M.J.Zaki. Sequence Mining in Categorical Domains: Incorporating Constrains. *Proc. Int. Conf. on Information and knowledge management* ,.422–429. 2000.
9. Data Analysis Challenge, http://centria.di.fct.unl.pt/ida01/
10. Geneva University Hospital and University of Geneva, Switzerland. ExPASy Molecular Biology Server. http://www.expasy.ch/