Discovering Unbounded Episodes in Sequential Data *

G. Casas-Garriga
Departament de LSI
Universitat Politècnica de Catalunya
gcasas@lsi.upc.es

April 23, 2003

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 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. This extracted knowledge is expected to be of interest in several contexts: such as the aminoacids of a protein, the banking operations of a client, DNA sequences, alarms in a telecomunication network, occurrences of recurrent illnesses and so on.

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],[3], [6] or [4] for an introduction to this model of a sequential database. The second model of sequential pattern matching is the slightly different approach proposed in [9], 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 sequence 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.

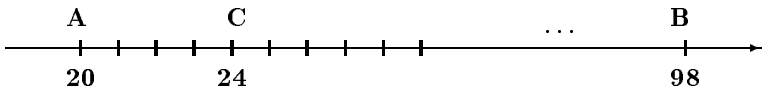

Figure 1: A sequence of events.

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

The frequent episode mining goal consists of identifying collections of events occurring frequently together in the event sequence. These frequent episodes are the interesting patterns in the sequence, and they are expected to provide useful information about it. Such information can then be used to better explain problems, to suppress redundant events or to predict severe faults. We briefly describe the approaches to interesting episodes in the classical [9], 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*, which we consider more intuitive and informative than previous ones. Our proposal for this new approach on unbounded episodes will be specially useful in several fields such as classification of texts or the intrusion detection problem as we will argue. We finally explain how a previous algorithm for finding frequent sets proposed in [12] can be applied to this problem, 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 experiments with our proposals.

2. Framework Formalization

To formalize the framework of the time-series data we follow the terminology, notation, and setting of [9]. The input of the problem is a *sequence* of events, where each event has an associated time of occurrence. 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:

$$s = \langle (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n) \rangle$$

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 α 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. The notation used for a serial episode will be: $B \rightarrow C$. Episode β 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. The notation used for a parallel episode will be: $\{A, B\}$. 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.

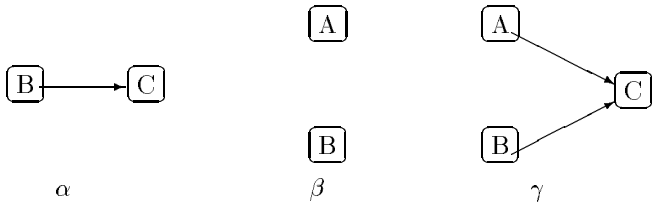

Figure 2: Types of episodes

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 $\leq$. An episode α is parallel if the partial order $\leq$ is trivial (i.e., $x \not\leq y$ for all $x, y \in V$ such that $x \neq y$) and an episode α is serial if the relation $\leq$ is a total order (i.e., $x \leq y$ or $y \leq x$ for all $x, y \in V$). In this paper we will only deal with serial and parallel episodes.

Definition 2.1 *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 $f : V' \rightarrow V$ such that $g'(v) = g(f(v))$ for all $v \in V'$, and for all $v, w \in V'$ with $v \leq' w$ also $f(v) \leq f(w)$.*

For most of our work, we deal with injective episodes, where the function g is injective; this means that events are not repeated within the episodes. However, we do treat recurring events (a particular form of non-injective serial episodes) since they show important in some applications.

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 work of [9] as a reference, i.e, we state that to be considered interesting, the events of an episode must occur close enough in time; of course this does not imply that they are fully adjacent, although for some specific applications (such as protein function identification) it is usual to impose adjacency. Our later approach will allow to implement this requirement but does not enforce it. In this section, we also describe how the fundamental algorithms like Apriori [1], are appropriately adapted to mine those episodes.

3.1. Winepi

In the first approach of [9], 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: $T_e - T_s + win - 1$. Thereby, the frequency of an episode α in s is defined to be:

$$fr(\alpha, s, win) = \frac{windows\ in\ which\ \alpha\ occurs}{total\ windows\ of\ width\ win\ from\ s}$$

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 episode must appear an specific number of times in the sequence, i.e., its number of appearences must be over a minimum user-specified value.

The approach consists in applying the Apriori algorithm in sequences as though a window were a transaction with ordered events. The goal is finding out which are the interesting frequent episodes; so, in every processed window, the frequency of those episodes that fit in the window is incremented whereas the frequency of those episodes that do not occur in the window is decremented. The algorithm starts by computing the frequency of episodes of size 2, which are candidates to be frequent. After this stage, the frequent episodes from this set of candidates, are combined to form candidates of size 3, and so on. An episode α will be frequent when $fr(\alpha, s, win) \geq minfr$.

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. The minimal occurrences with their

time intervals are identified in the following way. 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 episode 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 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". 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. 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:

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

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), 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; which restricts the the minimal occurrences of episodes to a fixed window of lenght win_2 during the 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. For instance, if the window width is fixed with a value of 3 time units, the maximum size of the discovered episodes is of 3 events, which are exactly the number of events that can fit in the window.

- In Minepi the user specifies two time bounds for the creation of the subsequent episodal 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 to be uninformative.

- Both Winepi and Minepi can 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. For instance, if the window is of width 3 time units and a larger frequent episode in the sequence is $A \rightarrow B \rightarrow C \rightarrow D$, the result of the mining will be: $A \rightarrow B \rightarrow C$ on the one hand, and $B \rightarrow C \rightarrow D$ on the other hand. These two episodes are actually overlapping parts of the same episode.

- Minepi does not use a frequency ratio to decide whether an episode is frequent. This makes difficult the application of sampling.
- In case the user decides to find the episodal rules for a different time bound (a different window size in Winepi or a different interval lenght for Minepi), then the algorithm that finds the source of frequent episodes has to be run again, incurring in a inconvenient waste of time.
- 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* are separated at most tus time units between them (as in figure 3).

$$\underset{\alpha}{A \overset{tus}{\rightarrow} B \overset{tus}{\rightarrow} C} \qquad\qquad \underset{\beta}{\{A, \overset{tus}{} B, \overset{tus}{} C\}}$$

Figure 3: Example of serial and parallel unbounded episodes

(2) It is frequent.

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 that between A and C there are at most $2 \times tus$ time units away. More generally, an episode of size e may span up to $(e-1) \times tus$ time units.

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.

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 algorithms. This explains the name chosen: we are mining unbounded episodes.

With this approach the frequency of an episode can be defined in the following way. Let us denote by $W_k(s, win) = T_i - T_f + win - 1$ the total number of windows in a sequence (s, T_i, T_f) of a fixed width $win = k \times tus$ time units. Then:

Definition 4.1 *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 $win = (|\alpha| - 1) \times tus$.

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). So, the final association rules will be like "if A and B occur within 25 seconds, then C follows within 30 seconds".

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 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 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 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 that maximizes 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).

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:

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

where the value of win, for fixed α, is linearly correlated with the user-specified tus as we have seen in the previous section. 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 do not affect the value of the confidence, and they will be determined after having chosen the best rule. We will chose the best unbounded episode rule in terms of confidence for each *maximal frequent* unbounded episode. The procedure is the following:

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

As we see, 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.

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

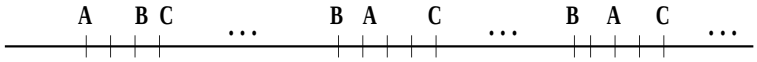

Figure 4: Example of an event sequence

$\gamma = \{A, B\} \rightarrow C$ (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 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 specifies $win_1 = 3$ and $win_2 = 4$, the association rule generated 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.

For the unbounded approach however, the algorithm would find both β and γ by just specifing a big enough value for tus (the minimum value would be 2 in this example). 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 "if A and B occur within tus time units, then C will follow in next tus time units".

4.2. Advantages of Our Approach

Here we shortly summarize some advantages of our proposal for unbounded episodes.

- 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 association rules that can be mined from a sequence can contain more information because the discovered episodes that form that rule can be longer.
- 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.
- Once the unbounded episodes are mined, finding the episode rules with two windows widths (such as in Minepi) can be easily done. What is more, the user can try different windows widths for the rules, and chose the best width for the antecedent and consequent according to the rule. 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 (which did not happen with Minepi or Winepi).
- We can define the frequency of an episode, easing the application of sampling techniques.
- Our proposal can be adapted to other types of sequential data, such as those in the approach of [2],[3], [6] or [4].

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 [19], 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 subsequent classifications, and our unbounded episodes could be very useful in this task. Also for the intrusion detection problem [20] there is a need to mine long sequences in order to get more informative patterns.

5. Algorithms to Mine Unbounded Episodes

The main aim of this section is to describe algorithms for mining interesting episodes from sequences, based on our interestingness measure. Previously used algorithms were based essentially on the same intuitions as the Apriori algorithm [1]. We adapt instead our strategy from [12], Best-First strategy, which is a nontrivial evolution of Dynamic Itemset Counting (DIC, [13]) and provides better performance than both Apriori and DIC (that follows a Breadth-First approach). For better understanding, we give a brief account of how our strategy works.

Similarly to DIC, our algorithm 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:

- The **candidate episodes**, those episodes which form the set of current candidates and whose frequency is being counted through the sequence.
- The **potential candidates**, those episodes which have some subepisodes which are candidates.

The EpiBF algorithm, following the Best-First strategy, will incorporate the future candidates to the counting as soon as they are ready. In case of mining unordered data, a future candidate itemset was ready to be counted when all its subsets were frequent. Hovewer, in case of sequential data, and given that we are using our approach to interesting episodes, we must redefine once again the concept of "being ready". We discuss separately the case of serial episodes first.

5.1. Discovering Serial Episodes

For serial episodes, the algorithm EpiBF 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.

Upon reaching the end of the sequence, the process starts over from its beginning like in the case of DIC. We know that a candidate is not frequent when the wrapped-up search comes to the point where counting started for that candidate.

It is important to highlight that, in this algorithm, the set of candidate episodes can be made up of episodes of diferent sizes, and EpiBF must also be able to process windows of different sizes for every episode. Nevertheless, this problem is not too difficult to solve: since an episode α of size k must be searched and counted in all windows of width $(k-1) \times tus$ time units, EpiBF will take, 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: $A \to B$ and $B \to A$. 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, and given with them becomes redundant. 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.

Thus, condition 1 reflects the case in which no serialized form of α is frequent, but the aggregation of all those non-frequent episodes into a parallel one, makes a contribution of useful information. Condition 2 is different: here the parallel episode serves as a gathering of all the different forms in which the serial episodes are actually frequent.

Globally, this means that, from the point of view of the algorithm, EpiBF 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.

Moving now into the algorithm, the parallel episodes, which are potential candidates, will simply wait for the notification of all their serial participants, which are the candidate episodes. This means that the algorithm will be counting the support of serial episodes, 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.

So, once one of these potential candidates, which are parallel episodes, has received the necessary notifications, it will be incorporated into the counting in the form of all its participating serial episodes.

5.3. Recurring Events

There is an important class of non-injective episodes worth consideration: serial episodes of the form $A \to A$. This is called "recurring events". Of course such episodes, if frequent, can be of interest, and there is no reason to avoid them; but in our approach of unbounded episodes they lead to the following result: they will appear in the output in the recurrent form $A \to A \to A \to \cdots \to A$ of size k such that the frequency threshold just separates the frequencies of the episode with the analogous one of size $k+1$. Usually, this would be an unintended effect of the frequency threshold. For instance, a highly branching web page such as the entry point of a photo gallery is a place likely to recurse in a web log sequence, and this fact is better expressed simply by $A \to A$ rather than by the longest subsequence of A's that fits the frequency threshold.

This also illustrates the inherent difficulty of our approach with non-injective episodes, where a similar but less understandable phenomenon may appear. Our current implementation of EpiBF only finds injective episodes and recurring events. Our feeling is that this encompasses most of the really interesting episodes we want to find along the input sequence.

6. Experiments and Conclusions

In this section we present the results of running (a probabilistic version of the) EpiBF algorithm on a variety of different data collections. The goal of these experiments was mainly to prove the usefulness and advantages of our presented framework for episodes against previous approaches (Winepi and Minepi).

First, we experimented, as in [9], 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 [18]. 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 [9] 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, 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 don't know which pattern is to be found in a sequence; so, one must try the experiment with different window widths. We can avoid this problem using our approach, which sets the parameter tus to 1, and this serves to any protein family.

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 [17]. Once again, setting tus close to 1, we are able to find frequent prepositions, articles, suffixes of words, and concatenations of words (such as "to", "in", "at", "ofthe", "was", "her", "ing" ...). We also experimented with WWW data collections extracted from the WWW server log from the LSI Department at the Universitat Politècnica de Catalunya, a 8Mb sequence. Our approach seems more intuitive for this kind of data: the parameter tus represents here the units of time spent from one page to the next. With support 0.05 and a value of 2 minutes for tus, the algorithm invested about 7h20' and found several dozens of correlated accesses. As an example, it is frequent that, at most 2 minutes after accessing to `/~rbaeza/handbook`, the user moves to `/~rbaeza/handbook/algs/3` (from where source code for many algorithms can be downloaded right away).

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 paral.lel episodes simultaneously, allows the algorithm to discover paral.lel patterns when hardly serial patterns are found in the database. For example, fed with the first 40,000 digits of the Champernowne sequence (0123456789101112131415161718192021 ...), 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. This task took only a couple of minutes.

7. Future Work

Hybrid episodes seem a natural extension. They do not present too high a difficulty for fixed window width algorithms, as indicated in [9], but in our setting their handling is not fully obvious.

This is due to our mesure of interestingness, that is, fixing tus, the maximum time between correlative events. Because of this measure, any episode α must take into account only those subepisodes of α whose events were located in a correlative way in α. Those subepisodes are called the parts of size $k-1$ of α. An episode α of size k would be *ready* to become a candidate episode, and so, begin the counting of its frequency in the sequence, when α has all its parts of size $k-1$ frequent instead of all its subepisodes. The parts of size $k-1$ of an episode α are a subset of all the subepisodes of size $k-1$ of the same episode α; that subset depends on whether α is serial, parallel, or hybrid, and reasons to output hybrid episodes must be related to their parts in a similar way as parallel episodes correspond to the set of participating serial episodes.

A second open line is the careful and detailed study of non-injective episodes and their behavior under the unbounded episode search; repeated events may lead to larger episodes whenever the frequency threshold allows it, but this behavior may lead to inappropriate information for human understanding.

Finally, there are several proposals in the association rules area to compute quantities like correlations or deviations as ratios of frequencies, and some of them might suggest better choices to select, or at least to rank, the mined serial, parallel, or hybrid episodes. Another approach to reduce the number of mined episodes is to use the notion of closure for the sequential patterns and adapt it to find the closed unbounded episodes ([14],[15],[16]).

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*, p.307–328. AAAI Press, 1996.

[2] R. Agrawal and R. Srikant. "Mining Sequential Patterns". *Proc. of the Int'l Conf. on Data Engineering*. 1995.

[3] R. Srikant and R. Agrawal. " Mining Sequential Patterns: Generalizations And Performance Improvements".*Proc. 5th Int. Conf. Extending Database Technology*. 1996.

[4] S. Parthasarathy, M.J. Zaki, M. Ogihara and S. Dwarkadas. "Incremental and Interactive Sequence Mining". *Proc. of the 8th International Conference on Information and Knowledge Management*, p.251–258, 1999.

[5] M.J. Zaki. "SPADE: An Efficient Algorithm for Mining Frequent Sequences". *Machine Learning Journal*, p.31-60, vol 42. 2001.

[6] J. Pei, J. Han and W. Wang. "Mining Sequential Patterns with Constraints in Large Databases". *ACM SIGKDD Explorations*. 2000.

[7] H. Kum, J. Pei, W. Wang and D.Duncan. "ApproxMAP: Approximate Mining of Consensus Sequential Patterns". *Proc. The 2003 SIAM International Conf. on Data Mining*. 2003.

[8] J. Pei, J. Han, et al. "PrefixSpan: Mining Sequential patterns efficiently by preffix-projected pattern growth". *Int'l Conference on Data Engineering*, pages 215-224. 2001.

[9] H. Mannila, H. Toivonen and I. Verkamo. "Discovery of frequent episodes in event sequences". *Proc. of the First International Conference on Knowledge Discovery and Data Mining*. 1995.

[10] H. Mannila and D. Rusakov. "Decompositon of Event Sequences into Independent Components". *International Conference on Data Mining*. 2001.

[11] H. Mannila and J.K. Seppänen. "Finding similar situations in sequences of events via random projections". *International Conference on Data Mining*. 2001.

[12] 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.

[13] S. Brin, R. Motwani, J. Ullman and S. Tsur. "Dynamic Itemset Counting and Implication Rules for Market Basket Data". *Int. Conf. Management of Data*, p. 255–264. 1997.

[14] X.Yan, J.Han and R.Afshar. "CloSpan: Mining Closed Sequential Patterns in Large Databases". *Int'l Conference on SIAM Data Mining*. 2003.

[15] G. Casas-Garriga. "Characterization of Concept Lattices for Ordered Contexts". 2003.

[16] S.Harms, J.Deogun, J.Saquer and T.Tadesse. "Discovering Representative Episodal Association Rules from Event Sequences Using Frequent Closed Episode Sets and Event Constraints". *Int'l Conference on Data Miminig*. 2001.

[17] Data Analysis Challenge, http://centria.di.fct.unl.pt/ida01/

[18] Geneva University Hospital and University of Geneva, Switzerland. ExPASy Molecular Biology Server. http://www.expasy.ch/

[19] M. Hiaro, S. Inenaga, A. Shinohara, M. Takeda and S. Arikawa. "A Practical Algorithm to Find the Best Episode Patterns". *Int'l Conference on Discovery Science*, pages 235-440. 2001.

[20] W. Lee and S.J. Stolfo. "A Framework for Constructing Features and Models for Intrusion Detection Systems". *Int'l Conf on Data Engineering*. 1999.