Mining Closed Strict Episodes

Nikolaj Tatti and Boris Cule
University of Antwerp
Antwerp, Belgium
{*nikolaj.tatti,boris.cule*}*@ua.ac.be*

***Abstract*—Discovering patterns in a sequence is an important aspect of data mining. One popular choice of such patterns are episodes, patterns in sequential data describing events that often occur in the vicinity of each other. Episodes also enforce in which order events are allowed to occur.**

In this work we introduce a technique for discovering closed episodes. Adopting existing approaches for discovering traditional patterns, such as closed itemsets, to episodes is not straightforward. First of all, we cannot define a unique closure based on frequency because an episode may have several closed superepisodes. Moreover, to define a closedness concept for episodes we need a subset relationship between episodes, which is not trivial to define.

We approach these problems by introducing strict episodes. We argue that this class is general enough, and at the same time we are able to define a natural subset relationship within it and use it efficiently. In order to mine closed episodes we define an auxiliary closure operator. We show that this closure satisfies the needed Galois connection so that we can use the existing framework for mining closed patterns. Discovering the true closed episodes can be done as a post-processing step. We combine these observations into an efficient mining algorithm and demonstrate empirically its performance in practice.

***Keywords*-Frequent Episode Mining, Closed Episodes, Level-wise Algorithm**

I. Introduction

Discovering frequent patterns in an event sequence is an important field in data mining. A pattern in a sequence is usually considered to be a set of events that reoccurs in the sequence within a window of a specified length. Gaps are allowed between the events and the order in which the events occur is often also considered important. Frequency, the number of sliding windows in which the episode occurs, is monotonically decreasing so we can use the well-known WINEPI [1] method, essentially a level-wise approach, to mine all frequent episodes.

The order restrictions of an episode are described by a directed acyclic graph (DAG): the set of events in a sequence covers the episode if and only if each event occurs only after all its parent events (with respect to the DAG) have occurred (see the formal definition in Section II). Usually, only two extreme cases are considered. A parallel episode poses no restrictions on the order of events, and a window covers the episode if the events occur in the window, in any order. In such a case, the DAG associated with the episode contains no edges. The other extreme case is a serial episode. Such an episode requires that the events occur in one, and only one, specific order in the sequence. Clearly, serial episodes are more restrictive than parallel episodes. If a serial episode is frequent, then its parallel version is also frequent.

General episodes have, in practice, been over-shadowed by parallel and serial episodes, despite being defined at the same time [1]. The main reason for this is the pattern explosion demonstrated in the following example.

Example 1.1: As an example of pattern explosion we will use text data, namely inaugural speeches by presidents of the United States (see Section VI for more details). By setting the window size to 15 and the frequency threshold to 60 we discovered a serial episode with 6 symbols,

$$(\text{preserv} \rightarrow \text{protect} \rightarrow \text{defend} \rightarrow \text{constitut} \rightarrow \text{unit} \rightarrow \text{state}).$$

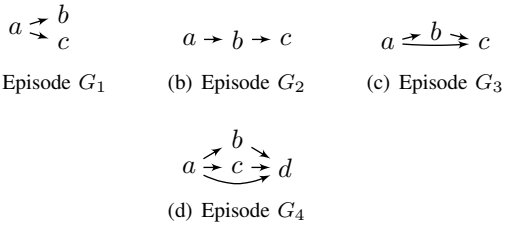
In total, we found another 4823 subepisodes of size 6 of this episode. However, all these episodes had only 3 distinct frequencies, indicating that the frequencies of most of them could be derived from the frequencies of only 3 episodes, so 4821 episodes could safely be left out of the output.

Motivated by this example, we approach the problem of pattern explosion by using a popular technique of closed patterns. A pattern is closed if there exists no more specific pattern with the same frequency. Mining closed patterns has been shown to reduce the output. Moreover, if we can establish a specific property called the Galois connection, we can discover closed patterns efficiently. However, adopting the concept of closedness to episodes is not without problems.

Subset relationship: Firstly, in order to define closed patterns we need a subset relation between patterns to describe whether a pattern G is a subpattern of pattern H. Essentially the same episode can be described by multiple DAGs and if we would base our definition of closedness simply on a subset relationship of DAGs we will run into problems as demonstrated in the following example.

Example 1.2: Consider episodes G_1, G_2, and G_3 given in Figure 1. Episode G_1 states that for a pattern to occur a must precede b and c. G_2 and G_3, meanwhile, state that a must be followed by b and then by c. Note that G_2 and G_3 represent essentially the same pattern that is more restricted than the pattern represented by G_1. However, G_1 is a subgraph of G_3 but not a subgraph of G_2. This reveals a problem if we base our definition of a subset relationship of episodes solely on the edge subset relationship. We solve this by generating

only transitively closed graphs, thus ignoring graphs of form G_2. We will not lose any generality since we are still going to discover episodes of form G_3.



(a) Episode G_1 (b) Episode G_2 (c) Episode G_3

(d) Episode G_4

Figure 1. Toy episodes used in Examples 1.2 and 1.3.

Frequency closure: Secondly, frequency does not satisfy the Galois connection. In fact, given an episode G there can be *several* more specific closed episodes that have the same frequency. So the closure operator cannot be defined as a mapping from an episode to its frequency-closed version.

Example 1.3: Consider sequence $s = abcbdacbcd$ and episode G_4 given in Figure 1(d). Assume that we use a sliding window of size 5. There are two windows that cover episode G_4, namely $s_1 \cdots s_5$ and $s_6 \cdots s_{10}$. Hence, the frequency of G_4 is 2. There are *two* serial episodes that are more specific than G_4 and have the same frequency, namely, $H_1 = (a \to b \to c \to d)$ and $H_2 = (a \to c \to b \to d)$. Moreover, there is no superepisode of H_1 and H_2 that would has the frequency 2. In other words, we cannot define a unique closure for G_4.

The contributions of our paper address these issues:

1) We introduce *strict* episodes, a new subclass of general episodes. We will argue that this class is large, contains all serial and parallel episodes, and most of the general episodes, yet using only strict episodes eases the computational burden.
2) We introduce a natural subset relation between episodes based on the subset relation of sequences covering the episode. We will prove that for strict episodes this relation corresponds to the subset relation between transitively closed graphs. For strict episodes such a graph uniquely defines the episode.
3) We introduce a milder version of the closure concept called the *instance-closure*. We will show that this closure satisfies the Galois connection and that a frequency-closed episode is always instance-closed.
4) Finally, we present an algorithm that generates strict instance-closed episodes with transitively closed graphs. Once these episodes are discovered we can further prune the output by removing the episodes that are not frequency-closed.

II. Preliminaries and Notation

We begin by presenting the preliminary concepts and notations that will be used throughout the paper. In this section we introduce the notions of sequence and episodes.

A *sequence* $s = (s_1, \ldots, s_L)$ is a string of symbols coming from an *alphabet* Σ, so that for each i, $s_i \in \Sigma$. An episode G is represented by an acyclic directed graph with labelled nodes, that is, $G = (V, E, lab)$, where $V = (v_1, \ldots, v_K)$ is the set of nodes, E is the set of directed edges, and lab is the function $lab : V \to \Sigma$, mapping each node v_i to its label. We denote the set of nodes of an episode G with $V(G)$, and its set of edges with $E(G)$.

Given a sequence s and an episode G we say that s *covers* G if there is an *injective* map f mapping each node v_i to a valid index such that the node v_i in G and the corresponding sequence element $s_{f(v_i)}$ have the same label, $s_{f(v_i)} = lab(v_i)$, and that if there is an edge (v_i, v_j) in G, then we must have $f(v_i) < f(v_j)$. In other words, the parents of v_j must occur in s before v_j.

Episode mining is based on searching for episodes that are covered by windows of certain fixed size often enough. The *frequency* of a given episode is then defined as the number of such windows that cover it.

We now provide a canonical form for episodes, which will help us in further theorems and algorithms. We define an episode that has the maximal number of edges using a fundamental notion familiar from graph theory.

Definition 2.1: The *transitive closure* of an episode $G = (V, E, lab)$ is an episode $tcl(G)$, where G and $tcl(G)$ have the same set of nodes V, the same lab function mapping nodes to labels, and the set of edges in $tcl(G)$ is equal to

$$E(tcl(G)) = E \cup \{\, (v_i, v_j) \mid \text{ a path exists in } G \text{ from } v_i \text{ to } v_j \,\}.$$

Note that, despite its name, the transitive closure has nothing to do with the concept of closed episodes.

III. Subset Relationship

Generally, a pattern is considered closed if there exists no more specific pattern having the same frequency. In order to speak of more specific patterns, we must first have a way to describe episodes in these terms. In this section we define a subset relationship among episodes, that would allow us to describe one episode as more specific than another one.

Definition 3.1: Assume two transitively closed episodes G and H with the same number of nodes. An episode G is called a *subset* of episode H, denoted $G \preceq H$ if the set of all sequences that cover H is a subset of the set of all sequences that cover G. If G is a proper subset of H, we denote $G \prec H$. If $|V(G)| < |V(H)|$, then G is a subset of H if there is a subgraph H' of H such that $G \preceq H'$.

The problem with this definition is that we do not have the means to compute this relationship for general episodes. To do this, one would have to enumerate all possible sequences that cover H and compute whether they cover G. We approach this problem by restricting ourselves to a class of episodes where this comparison can be performed efficiently.

Definition 3.2: An episode G is called *strict* if for any two nodes v and w in G sharing the same label, there exists a path either from v to w or from w to v.

Using strict episodes will also allow us to build an algorithm to efficiently discover closed episodes. In the remaining text, we consider episodes to be strict. However, as can be seen in Figure 2, this, unfortunately, means that some episodes will never be discovered.

(a) non-strict (b) strict

Figure 2. An example of a non-strict and a strict episode.

For notational simplicity, we now introduce the concept of two episodes having *identical nodes*. Given an episode G with nodes $V(G) = \{v_1, \ldots, v_N\}$, we assume from now on that the order of the nodes is always fixed such that for $i < j$ either $lab(v_i) < lab(v_j)$, or $lab(v_i) = lab(v_j)$ and v_i is an ancestor of v_j. We say that two episodes G and H, with $V(G) = \{v_1, \ldots, v_N\}$ and $V(H) = \{w_1, \ldots, w_N\}$ have *identical nodes* if $lab(v_i) = lab(w_i)$. To simplify notation, we often identify v_i and w_i.

Crucially, we can easily compute the subset relationship between two episodes, if they have identical nodes.

Theorem 3.3: For transitively closed episodes G and H with identical nodes, $E(G) \subseteq E(H)$ if and only if $G \preceq H$.

Proof: To prove the "only if" direction assume that $E(G) \subseteq E(H)$. Let $s = \{s_1, \ldots, s_N\}$ be a sequence covering H and let f be the corresponding mapping. Then f is also a valid mapping for G. Thus, $G \preceq H$.

To prove the other direction, assume that $E(G) \nsubseteq E(H)$. We therefore must have an edge $e = (x, y) \in E(G)$, such that $e \notin E(H)$. Note that this implies that $lab(x) \neq lab(y)$, otherwise H would not be strict. We know that for every node v it holds that the number of ancestors of v in G having the label $lab(v)$ is equal to the number of such ancestors of v in H. To prove that $G \npreceq H$ we will construct a sequence s that covers H but not G. We build s by first visiting every parent of y in H in a valid order with respect to H, then y itself, and then the rest of the nodes, also in a valid order. This sequence covers H. Let s_i be the event corresponding to x and let s_j be the event corresponding to y. Assume now that s covers G so that there is a map f mapping the nodes of G to indices of s. Let k be the number of ancestors of x having the same label as x in H. Let l be the number of descendants having the same label as x in H. Since s covers H and $lab(x) \neq lab(y)$ we must have k occurrences of $lab(x)$ events before s_i and l occurrences after. Let v be such that $f(v) = s_i$. We see that v must also have k ancestors with the same label so we must have $v = x$, and $f(x) = i$. Similarly, we have $f(y) = j$. This is a contradiction since $(x, y) \in E(G)$ but $j < i$. ■

Note that we do not need to define a subset relation for episodes that do not have identical nodes, as will be explained in detail in Section V.

We now define what we exactly mean when we say that two episodes are essentially the same.

Definition 3.4: Episodes G and H are said to belong to the same $class$, denoted by $G \sim H$, if each sequence that covers G also covers H, and vice versa.

Corollary 3.5 (of Theorem 3.3): For transitively closed episodes G and H, $G \sim H \Leftrightarrow E(G) = E(H)$.

Proof: This follows from the fact that $G \sim H$ is equivalent to $G \preceq H$ and $H \preceq G$, and that $E(G) = E(H)$ is equivalent to $E(G) \subseteq E(H)$ and $E(H) \subseteq E(G)$. ■

Note that by generating only transitively closed strict episodes, we have obtained an efficient way of computing the subset relationship between two episodes. At first glance, though, it may seem that we have completely omitted certain parallel episodes from consideration — namely, all non-strict parallel episodes (i.e. those containing multiple nodes with same labels). Note, however, that for each such episode G, there exists a strict episode H, such that $G \sim H$. To build such an episode H, we just need to create edges that would strictly define the order among nodes with the same labels. From now on, when we talk of parallel episodes, we actually refer to their strict equivalents.

IV. Closure

Having defined a subset relationship among episodes, we are now able to speak of an episode being more specific than another episode. However, this is only the first step towards defining the closure of an episode. We know that the closure must be more specific, but it must also be unique and well-defined. We have already seen that basing such a closure on the frequency fails, as there can be multiple more specific closed episodes that could be considered as closures. In this section, we will base the closure on another criterion, which will result in each episode having a unique closure.

We begin by associating each sequence with a corresponding serial episode.

Definition 4.1: Given a sequence $s = (s_1, \ldots, s_N)$, we define its *corresponding serial episode* G_s as the transitive closure of

$$\left(v_{m(1)} \rightarrow v_{m(2)} \rightarrow \cdots \rightarrow v_{m(N)}\right),$$

with $lab\left(v_{m(i)}\right) = s_i$. Mapping m makes sure that the nodes of G_s are ordered, that is, for $i < j$, $lab(v_i) \leq lab(v_j)$.

Based on this definition, we can now build a maximal episode that is covered by a set of sequences.

Definition 4.2: Given a set of nodes V, and a set S of sequences containing the events corresponding to labels of nodes in V, we define the *maximal episode* covered by set S as the episode H, where $V(H) = V$ and $E(H) = \bigcap_{s \in S} E(G_s)$.

We now show that we can make a Galois connection between the set of all episodes with a fixed set of nodes V and the power set $\mathcal{S}$ containing all sets of subsequences consisting only of labels of nodes V in all windows of length k in our sequence s. For all episodes G containing nodes V, we define a function f as

$$f(G) = \{\, w \mid w \in \mathcal{S},\ w \text{ covers } G \,\}.$$

For all sets of subsequences S in $\mathcal{S}$, we define a function g as $g(S) = G$, with G the maximal episode covered by S.

Theorem 4.3: Given a set of nodes V, a sequence s, and the power set $\mathcal{S}$ containing all sets of subsequences consisting only of labels of nodes V in all windows of length k in s, f and g, as defined above, satisfy the Galois connection: $S \subseteq f(G) \Leftrightarrow G \preceq g(S)$.

Proof: Assume that $S \subseteq f(G)$. This means that all sequences in S cover G. $g(S)$ gives us the maximal episode that covers all sequences in S. Clearly, G must be a subset of such an episode.

Assume now $g(S) = H$ and that $G \preceq H$. Therefore, all sequences that cover H also cover G. We know that all sequences in S cover H, and, therefore, G, so S must be a subset of $f(G)$, a set of all sequences that cover G. ■

We are now ready to define closure using f and g.

Definition 4.4: We define the *instance-closure*, or *i-closure* of an episode G, denoted $icl(G)$, as $g(f(G))$. We say an episode G is *i-closed*, if $g(f(G)) = G$.

The following example demonstrates how f and g work in practice.

Example 4.5: Consider sequence $s = abcbdxyacbcd$ and the parallel episode $G = (a, b, c, d)$, given that the chosen window length is 5. We begin with a set of nodes $V = \{\, v_1, v_2, v_3, v_4 \,\}$, labelled a, b, c and d respectively. $f(G)$ is defined as the set of all subsequences consisting of nodes V in all windows of length 5 that cover G. In our example, there are two windows of length 5 that cover G, $s_1 \cdots s_5$ and $s_8 \cdots s_{12}$, and each window contains the same two subsequences that satisfy our criteria. Therefore, $f(G) = \{\, abcd, acbd \,\}$. The serial episodes corresponding to these two subsequences are the transitive closures of $(a \to b \to c \to d)$ and $(a \to c \to b \to d)$, so $g(f(G))$, or $icl(G)$, is obtained by taking the intersection of the edges of these two serial episodes, and is given in Figure 1(d).

As we intend to allow the user to also obtain closed episodes based on frequency, we now provide a formal definition of such episodes.

Definition 4.6: An episode G is *frequency-closed*, or *f-closed*, if there exists no episode H, such that $G \prec H$ and $fr(G) = fr(H)$.

Note that, unlike the i-closure, we do not define an *f-closure* of an episode at all. As shown in Section I, such an f-closure would not necessarily be unique. For this reason, our algorithm identifies instance-closed episodes. Should the user wish to find only frequency-closed episodes, we provide this option as a post-pruning step. The following proposition proves that such a step is possible.

Proposition 4.7: A frequency-closed episode is always instance-closed.

Proof: Assume episode G is frequency-closed and not instance-closed. Note that the definition of i-closure effectively says that if G is not i-closed, then there exists an episode H, such that $G \prec H$, and for each possible mapping of G in each window of length k in the sequence s, this mapping is also a mapping of H. Clearly, in this case, $fr(G) = fr(H)$, which is a contradiction. ■

V. Algorithm for Discovering Closed Episodes

In this section we will present our mining algorithm for discovering closed episodes. First we give an overview of the algorithm. Next, we explain in detail how candidate episodes are tested before being scanned. We continue by presenting techniques for generating the candidate episodes. Finally, we discuss how closures are computed in practice.

A. Overview of the algorithm

We showed in Section IV that $icl(G)$ has a Galois connection. This allows us to use a standard level-wise approach for mining closed patterns (see, for example, [2]). The outline of the algorithm is given in Algorithm 1.

The algorithm consists of two loops. In the outer loop we discover parallel episodes. For each parallel episode we call MineDAG given in Algorithm 2. MineDAG discovers all general episodes in a level-wise fashion by adding edges. During the generation MineDAG calls GenerateCandidate (Algorithm 3) which ensures that the candidates are transitively closed. In the next step, for each candidate episode, we test whether all its subepisodes are frequent, and that the candidate episode is not contained in the closure of one of its subepisodes (see TestCandidate in Algorithm 4). Finally, we compute the frequency, and if the episode is frequent, we compute its instance-closure.

Mining episodes requires an additional step that does not occur in mining itemsets: we need to add some episodes that are not generators as candidates. This step is done on Line 13 of MineDAG and is explained in detail in Section V-D.

B. Generating Transitively Closed Candidate Episodes

Theorem 3.3 implies that if we generate only transitively closed episodes, then the subset relationship between the episodes is simply the subset relationship between the edges. In this section we define an algorithm, GenerateCandidate, which generates the candidate episodes from the episodes discovered previously. GenerateCandidate makes sure that the candidates are transitively closed. These candidates are then tested by the TestCandidate algorithm described in the next section. If the candidates pass the test they are tested for frequency.

To describe GenerateCandidate we need the following definition.

Algorithm 1: MINEEPISODES. An algorithm discovering all frequent closed episodes.

input : Sequence s. Frequency threshold.
output: f-closed frequent episodes.

$\mathcal{G} \leftarrow$ frequent episodes with one node;
while $\mathcal{G}$ is not empty **do**
  $\mathcal{H} \leftarrow$ next level of parallel frequent episodes discovered from $\mathcal{G}$;
  foreach $G \in \mathcal{H}$ **do** MINEDAG(G);
  $\mathcal{G} \leftarrow \mathcal{H}$;
return F-CLOSURE(episodes outputted by MINEDAG)

Algorithm 2: MINEDAG. An algorithm that discovers frequent episodes from a fixed parallel episode.

input : Parallel episode G.
output: i-closed frequent episodes having identical nodes as G.

$\mathcal{G}_1 \leftarrow$ frequent episodes with one edge and identical nodes as G;
$N \leftarrow |V(G)|(|V(G)|-1)/2$;
foreach $i = 1, \ldots, N$ **do**
  $\mathcal{H} \leftarrow$ GENERATECANDIDATE($\mathcal{G}_i$);
  foreach $H \in \mathcal{H}$ **do**
    if TESTCANDIDATE(H) **and** H is frequent **then**
      $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_{i+1} \cup H$;
      $icl(H) \leftarrow$ I-CLOSURE(H);
      output $icl(H)$;
      foreach $e \notin icl(H)$ **do**
        $Z \leftarrow E(tcl(H+e)) - E(H+e)$;
        if $Z \neq \emptyset$ **and** $Z \subset E(icl(H))$ **then**
          add $H + Z$ to $\mathcal{G}_{i+|Z|+1}$;

Definition 5.1: An edge (v, w) in a transitively closed episode G is called a *skeleton edge* if there is no node u such that (v, u, w) is a path in G. If v and w have different labels, we call the edge (v, w) a *proper skeleton edge*.

As pointed out in Section V-A we will first generate parallel episodes and then in a level-wise fashion add edges. Let G be a transitively closed episode. It is easy to see that if we remove a proper skeleton edge e from G, then the resulting episode $G - e$ will be transitively closed. We can reverse this property in order to generate candidates: Let G be a previously discovered transitively closed episode, add an edge e and verify that the new episode is transitively closed. However, we can improve on this naive approach with the following proposition describing the sufficient and necessary condition for an episode to be transitively closed.

Proposition 5.2: Let G be a transitively closed episode and let $e = (x, y)$ be an edge not in $E(G)$. Let $H = G + e$. Assume that H is a DAG. Then H is transitively closed if and only if there is an edge in G from x to every child of y and from every parent of x to y.

Proof: The 'only if' part follows directly from the definition of transitive closure. To prove the 'if' part, we will use induction. Let u be an ancestor node of v in H. Then there is a path from u to v in H. If the path does not use edge e, then, since G is transitively closed, $(u, v) \in E(G)$ and hence $(u, v) \in E(H)$. Assume now that the path uses e. If $v = y$, then u must be a parent of y in G, since G is transitively closed, so the condition implies that $(u, v) \in E(G) \subset E(H)$. Assume that v is a descendant of y in H. To prove the first step in the induction, assume that $v = x$, then again $(u, v) \in E(G)$. To prove the induction step, let w be the next node along the path from u to v in H. Assume that $(w, v) \in E(G)$. Then the path (u, w, v) occurs in G, so $(u, v) \in E(G)$, which completes the proof. ■

We now show when we can join two episodes to obtain a candidate episode.

Theorem 5.3: Let G_1 and G_2 be two transitively closed episodes with identical nodes and N edges. Assume that G_1 and G_2 share $N - 1$ mutual edges. Let $e_1 = (x_1, y_1) \in E(G_1) - E(G_2)$ be the unique edge for G_1 and let $e_2 = (x_2, y_2) \in E(G_2) - E(G_1)$ be the unique edge of G_2. Let $H = G_1 + e_2$. Assume that H has no cycles. Then H is transitively closed if and only if one of the following conditions is true

1) $x_1 \neq y_2$ and $x_2 \neq y_1$.
2) $x_1 \neq y_2$, $x_2 = y_1$, and (x_1, y_2) is an edge in G_1.
3) $x_1 = y_2$, $x_2 \neq y_1$, and (x_2, y_1) is an edge in G_1.

Moreover, if H is transitively closed, then e_1 is a skeleton edge in H.

Proof: We will first show that e_1 is a skeleton edge in H. If it is not, then there is a path from x_1 to y_1 in H not using e_1. The edges along this path also occur in G_2, thus forcing e_1 to be an edge in G_2, which is a contradiction.

The "only if" part is trivial so we only prove the "if" part.

Let v be a child of y_2 in G_1 and $f = (y_2, v)$ an edge in G_1.

If the first or second condition holds, then $x_1 \neq y_2$, and consequently $f \neq e_1$, so $f \in G_2$. The path (x_2, y_2, v) connects x_2 and v in G_2 so there must be an edge $h = (x_2, v)$ in G_2. Since $h \neq e_2$, h must also occur in G_1. If the third condition holds, it may be the case that $f = e_1$ (if not, then we can use the previous argument). But in such a case $v = y_1$ and the edge $h = (x_2, y_1)$ occurs in G_1.

If now u is a parent of x_2 in G_1, we can make a similar argument that u and y_2 are connected, so Proposition 5.2 now implies that H is transitively closed. ■

Theorem 5.3 provides with the means to generate transitively closed episodes in the following manner. Since our nodes are ordered, we can also order the edges using a lexicographical order. Given an episode G we define

$last(G)$ to be the last proper skeleton edge in G. Let H be a transitively closed episode. Let $e_2 = last(H)$ be its last proper skeleton edge. Define $G_1 = H - e_2$. Let $e_1 = last(G_1)$ be the last proper skeleton edge in G_1 and assume that e_1 is also a proper skeleton edge in H. Then in order for H to be transitively closed, G_1 and G_2, defined as $H - e_1$, must satisfy one of the conditions given in Theorem 5.3.

In other words, to generate a candidate, we take two previously discovered episodes with identical nodes , say G_1 and G_2, with N edges. Let e_1 and e_2 be the last proper skeleton edges in G_1 and in G_2, respectively. Assume that $e_1 < e_2$ and that G_1 and G_2 share the rest of the edges. Then if $G_1 + e_2$ satisfies one of the conditions in Theorem 5.3, we will generate it for the next stage.

This approach will *not* generate all candidates. The crucial assumption we made above is that e_1 is also a skeleton edge in H. Hence to generate all candidates we also need to generate episodes from G_1 such that e_1, the last proper skeleton edge of G_1, is no longer a skeleton edge in G_1+e_2.

Theorem 5.4: Let G be a transitively closed episode, let $e_1 = (x_1, y_1)$ be a skeleton edge of G, and let $e_2 = (x_2, y_2)$ be an edge not occuring in G and define $H = G + e_2$. Then H is a transitively closed episode such that e_1 is not a skeleton edge in H only if either $y_2 = y_1$ and (x_1, x_2) is a skeleton edge in G or $x_1 = x_2$ and (y_2, y_1) is a skeleton edge in G.

Proof: Assume that e_1 is no longer a skeleton edge in H, then there is a path of skeleton edges going from x_1 to y_1 in H not using e_1. The path must use e_2, otherwise we have a contradiction. The theorem will follow if we can show that the path must have exactly two edges. Assume otherwise. Assume, for simplicity, that the edge e_2 does not occur first in the path and let z be the node before x_2 in the path. Then we can build a new path by replacing the edges (z, x_2) and e_2 with (z, y_2). This path does not use e_2, hence it occurs in G, making e_1 a non-skeleton edge in G, which is a contradiction. If e_2 is the first edge in the path, we can select the next node after y_2 and repeat the argument. ■

We can now combine Theorem 5.3 and Theorem 5.4 into the GENERATECANDIDATE algorithm given in Algorithm 3. We will first generate candidates by combining episodes from the previous rounds using Theorem 5.3. Secondly, we use Theorem 5.4 and for each episode from the previous rounds we add edges such that the last proper skeleton edge is no longer a skeleton edge in the candidate.

C. Testing the Candidate Episode

Following the level-wise discovery, before computing the frequency of the episode, we need to test that all its subepisodes are discovered. Using transitively closed episodes has another important benefit.

Corollary 5.5 (of Theorem 3.3): Let G be a transitively closed episode. Let e be a proper skeleton edge of G. If H is an episode obtained by removing e from G, then there exists no episode H_1, such that $H \prec H_1 \prec G$.

Algorithm 3: GENERATECANDIDATE. Generates candidate episodes from the previously discovered episodes.

input : A collection of previosly discovered episodes $\mathcal{G}$. Episodes in $\mathcal{G}$ have N edges.
output: A collection of i-closed candidate episodes with $N + 1$ edges.

foreach $G_1 \in \mathcal{G}$ **do**
  $e_1 = (x_1, y_1) \leftarrow last(G_1)$;
  {Case where e_1 remains a skeleton edge.}
  $\mathcal{H} \leftarrow \left\{ H \in \mathcal{G} \;\middle|\; \begin{array}{l} |E(H) \cap E(G)| = N - 1, \\ last(H) > e_1 \end{array} \right\}$;
  foreach G_2 in $\mathcal{H}$ **do**
    $e_2 \leftarrow last(G_2)$;
    if G_1 and G_2 satisfy Thr. 5.3 **and** $e_2 \notin icl(G_1)$ **then** **output** $G_1 + e_2$;
  {Case where e_1 does not remain a skeleton edge.}
  foreach $f = (x_1, x_2)$ skeleton edge in G_1 such that $x_2 \neq y_1$ **do**
    $e_2 \leftarrow (x_2, y_1)$;
    if $e_2 \notin icl(G_1)$ **then**
      $H \leftarrow G_1 + e_2$;
      if $e_2 = last(H)$ **and** Prop. 5.2 holds **then** **output** $G_1 + e_2$;
  foreach $f = (y_2, y_1)$ skeleton edge in G_1 such that $y_2 \neq x_1$ **do**
    $e_2 \leftarrow (x_1, y_2)$;
    if $e_2 \notin icl(G_1)$ **then**
      $H \leftarrow G_1 + e_2$;
      if $e_2 = last(H)$ **and** Prop. 5.2 holds **then** **output** $G_1 + e_2$;

Corollary 5.5 implies that using transitively closed episodes will guarantee the strongest conditions for an episode to pass to the frequency computation stage.

If e is a skeleton edge of a transitively closed episode G, then $G - e$ is transitively closed. Thus, for G to be frequent, $G - e$ had to be discovered previously. This is the first test in TESTCANDIDATE (given in Algorithm 4). In addition, following the level-wise approach for mining closed patterns [2], we test that G is not a subepisode of $icl(G - e)$, and if it is, then we can discard G.

The second test involves testing whether $G - v$, where v is a node in G, has also been discovered. Note that $G - v$ has less nodes than G so, if G is frequent, we must have discovered $G-v$. Not all nodes need to be tested. If a node v has an adjacent proper skeleton edge, say e, then the episode $G - e$ has a frequency lower than or equal to that of $G - v$. Since we have already tested $G - e$ we do not need to test

$G - v$. Consequently, we need to test only those nodes that have no proper skeleton edges. This leads us to the second test in TESTCANDIDATE. Note that these nodes will either have no edges, or will have edges to the nodes having the same label. If both tests are passed we test the candidate episode for frequency.

Algorithm 4: TESTCANDIDATE. An algorithm that checks if an episode is a proper candidate.

input : An episode G.
output: Boolean value, **true** if all subepisodes of G are frequent.

```
1 foreach proper skeleton edge e in G do
2     if G − e is not discovered or e ∈ E(icl(G − e))
      then return false;
3 foreach v in G not having a proper skeleton edge do
4     if G − v is not discovered then return false;
5 return true;
```

D. Proof of Correctness

In this section we will prove that all frequent i-closed episodes are discovered. We will prove this by induction over the number of edges. To that end, we say that a skeleton edge e in an episode G is *derivable* if there is a subepisode H such that $e \in E(icl(H)) - E(H)$. Note that Lemma 2 in [2] implies that $icl(G) = icl(G - e)$. Hence, it is enough to show that either G has derivable edges or it is discovered. An episode G is discovered if an episode $G' = G - e$ is discovered for each skeleton edge e. If G' does not contain derivable edges then, by the induction assumption, it is discovered. If it has derivable edges, they turn into non-skeleton edges by adding e. Start removing derivable edges, one by one, until you reach an episode H without derivable edges. It is easy to see that all removed edges are part of $icl(H)$. H is discovered due to the induction assumption and G' is discovered due to Line 13 in MINEDAG.

E. Computing Closures

During the mining we need to compute the closure of an episode. We do this by discovering all possible valid instances of an episode in the sequence and using Definition 4.2. In order to discover the instances efficiently, we enumerate recursively all possible serial episodes H by removing sources (nodes without incoming edges) from the candidate episode G, such that $G \prec H$.

More specifically, assume that we have an episode G and that we have already removed K sources in the order $(n_1, \ldots, n_K)$. For each source v in G, we first test whether there are instances $(lab(n_1), \ldots, lab(n_K), lab(v))$ in the sequence. If there are, we set $n_{K+1} = v$ and test recursively $G - v$. Once G is empty, we have discovered a subsequence and its corresponding serial episode H such that $H \succ G$. The caveat of this approach is that the number of such serial episodes can be exponential. However, our experiments demonstrate that this is not a problem in practice. The pseudo-code is given in Algorithms 5 and 6.

Algorithm 5: I-CLOSURE. An algorithm for computing the i-closure of an episode G. The parameter ρ is the size of the window.

input : An episode G.
output: i-closure of G.

```
1 foreach v ∈ sources(G) do
2     W ← { (s_i) | s_i = lab(v) };
3     if W ≠ ∅ then
4         FINDSERIALS(G − v, W, ρ);
5 W ← all sequences outputted by FINDSERIALS;
6 V(H) ← V(G);
7 E(H) ← ⋂_{w∈W} E(G_w);
8 return H;
```

Algorithm 6: FINDSERIALS(G, W, ρ). A recursive subroutine used by I-CLOSURE. Discovers all instances of episode G.

input : An episode G. Partial candidate occurences W discovered so far. The size of the window ρ.
output: Instances of G in the sequences.

```
1  if G has no nodes then
2      output any sequence in W;
3  foreach v ∈ sources(G) do
4      R ← ∅;
5      foreach w ∈ W do
6          i ← index of the first element in w;
7          if there is s_j s.t. lab(v) = s_j and j − i < ρ then
8              Add w concatenated with s_j into R;
9      if R ≠ ∅ then
10         FINDSERIALS(G − v, R, ρ);
```

After the actual mining process we can further reduce the output by keeping only frequency-closed episodes. A naive approach would be to compare each pair of instance-closed episodes G and H and if $fr(G) = fr(H)$ and $G \prec H$, remove G from output. This approach can be considerably sped up by realizing that we need only to test episodes with identical nodes and episodes of form $G - v$. The pseudo-code is given in Algorithm 7. The algorithm can be further sped up by exploiting the subset relationship between the episodes. Our experiments demonstrate that this comparison is feasible in practice.

Algorithm 7: F-CLOSURE. Postprocessing for computing f-closed episodes from i-closures.

input : i-closed episodes $\mathcal{C}$.
output: f-closed episodes.

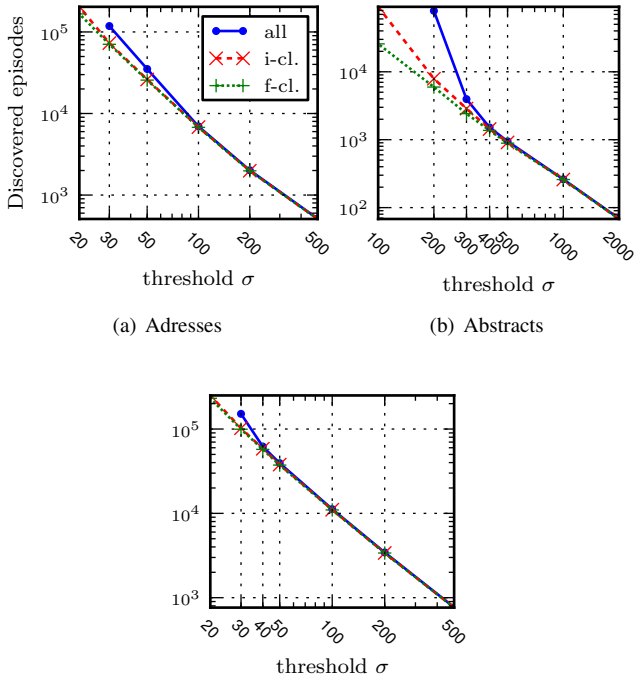
```
foreach G ∈ C do
    foreach H ∈ C with V(G) = V(H), H ≠ G do
        if G ≺ H and fr(G) = fr(H) then Mark G;
        if H ≺ G and fr(G) = fr(H) then Mark H;
    foreach v ∈ V(G) do
        F ← G − v;
        foreach H ∈ C, with V(F) = V(H) do
            if H ⪯ F and fr(G) = fr(H) then
                Mark H;
return all unmarked episodes;
```

VI. EXPERIMENTS

We tested our algorithm[1] on three text datasets, *address*, consisting of the inaugural addresses by the presidents of the United States[2], merged to form a single long sequence, *moby*, the novel Moby Dick by Herman Melville[3], and *abstract*, consisting of the first 739 NSF award abstracts from 1990[4], also merged into one long sequence. All three sequences were preprocessed using the Porter Stemmer[5] and the stop words were removed.

We used a window of size 15 for all our experiments and varied the frequency threshold σ. The main goal of our experiments was to demonstrate how we tackle the problem of pattern explosion. Figures 3(a), 3(b) and 3(c) show how the total number of frequent episodes compared with the identified i-closed and f-closed episodes we discovered in the three datasets. The results suggest that improvement is only visible at small thresholds and is less than a factor of 10. The reason for this is that the major part of the output consists of episodes with a small number of nodes. Such episodes tend to be closed.

To get a more detailed picture we examined the ratio of the number of frequent episodes and the number of f-closed episodes (Figure 4(a)) and the ratio of the number of i-closed episodes and the number of f-closed episodes (Figure 4(b)) as a function of the number of nodes. We see that while there is no improvement with small episodes, using closed episodes is essential if we are interested in large episodes. In such a case we were able to reduce the output by several orders of magnitude. For example, in the *address* dataset, with a threshold of 30, there were 1226 frequent episodes of size 7, of which only 2 were f-closed. Clearly, the number of discovered i-closed episodes remains greater than the number of f-closed episodes, but does not explode, guaranteeing the feasibility of our algorithm. For example, in the *abstract* dataset, with a threshold of 200, there were 15976 frequent episodes of size 5, of which 912 were i-closed and 250 f-closed.



(a) Adresses (b) Abstracts

(c) Moby

Figure 3. The number of frequent, i-closed and f-closed episodes with varying frequency thresholds for the *address*, *abstract* and *moby* datasets, respectively. Note that both axes are represented in log-scale.

The runtimes of our experiments varied between a few seconds and 30 minutes for the largest experiments. However, with low thresholds, our algorithm for finding closed episodes ran faster than the algorithm for finding all frequent episodes, and at the very lowest thresholds, our algorithm produced results, while the frequent-episodes algorithm ran out of memory. This demonstrates the infeasibility of approaching the problem by first generating all frequent episodes, and then pruning the non-closed ones. The i-closed episodes are the necessary intermediate step.

VII. RELATED WORK

Searching for frequent patterns in data is a very common data mining problem. The first attempt at discovering sequential patterns was made by Wang et al. [3]. There, the dataset consists of a number of sequences, and a pattern is considered interesting if it is long enough and can be found in a sufficient number of sequences. The method

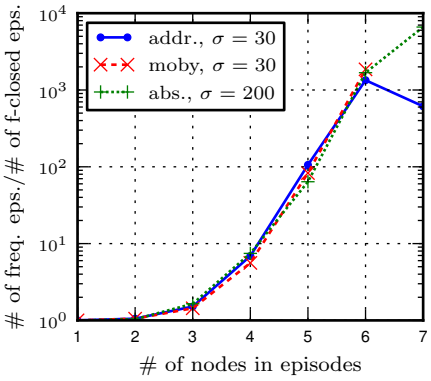
[1]The implementation of the algorithm is given at http://adrem.ua.ac.be/implementations/

[2]taken from http://www.bartleby.com/124/pres68

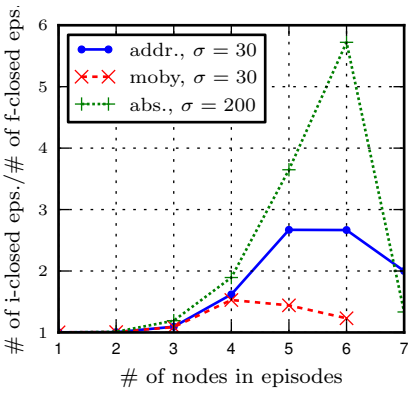
[3]taken from http://www.gutenberg.org/etext/15

[4]taken from http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html

[5]http://tartarus.org/~martin/PorterStemmer/



(a) Frequent / f-closed



(b) i-closed / f-closed

Figure 4. (a) The ratio of frequent and f-closed episodes for various episode sizes, with a fixed frequency threshold. (b) The ratio of i-closed and f-closed episodes for various episode sizes, with a fixed frequency threshold. Note that the y-axis of Figure 4(a) is in log-scale, while the y-axis of Figure 4(b) is in linear scale.

proposed in this paper, however, was not guaranteed to discover all interesting patterns, but a complete solution to a more general problem (dropping the pattern length constraint) was later provided by Agrawal and Srikant [4] using an APRIORI-style algorithm [5].

It has been argued that not all discovered patterns are of interest to the user, and some research has gone into outputting only closed sequential patterns, where a sequence is considered closed if it is not properly contained in any other sequence which has the same frequency. Yan et al. [6], Tzvetkov et al. [7], and Wang and Han [8] proposed methods for mining such closed patterns, while Garriga [9] further reduced the output by post-processing it and representing the patterns using partial orders. Despite their name, the patterns discovered by Garriga are different from the traditional episodes. A sequence covers an episode if *every* node of the DAG can be mapped to a symbol such that the order is respected, whereas a partial order discovered by Garriga is covered by a sequence if there is a subsequence corresponding to a path in the DAG from a source node to a sink node, that is, not all nodes need to be visited.

In another attempt to trim the output, Garofalakis et al. [10] proposed a family of algorithms called SPIRIT which allow the user to define regular expressions that specify the language that the discovered patterns must belong to.

Looking for frequent episodes in a single event sequence was first proposed by Mannila et al. [1]. The WINEPI algorithm finds all episodes that occur in a sufficient number of windows of fixed length. The frequency of an episode is defined as the fraction of all fixed-width sliding windows in which the episode occurs. The user is required to choose the width of the window and a frequency threshold. Specific algorithms are given for the case of parallel and serial episodes. However, no algorithm for detecting general episodes (DAGs) is provided.

The same paper proposes the MINEPI method, where the interestingness of an episode is measured by the number of minimal windows that contain it. As was shown by Tatti [11], MINEPI fails due to an error in its definition. Zhou et al. [12] proposed mining closed serial episodes based on the MINEPI method, without solving this error. Laxman et al. introduced a monotonic measure as the maximal number of non-overlapping occurrences of the episode [13].

Pei et al. [14] considered a restricted version of our problem setup. In their setup, items are allowed to occur only once in a window (string in their terminiology). This means that the discovered episodes can contain only one occurence of each item. This restriction allows them to easily construct closed episodes. Our setup is more general since we do not restrict the number of occurences of a symbol in the window and the miner introduced by Pei cannot be adapted to our problem setting since the restriction imposed by the authors plays a vital part in their algorithm.

Garriga [15] pointed out that WINEPI suffers from bias against longer episodes, and proposed solving this by increasing the window length proportionally to the episode length. However, as was pointed out by Méger and Rigotti [16], the algorithm given in this paper contained an error.

An attempt to define frequency without using any windows has been made by Calders et al. [17] where the authors define an interestingness measure of an itemset in a stream to be the frequency starting from a point in time that maximizes it. However, this method is defined only for itemsets, or parallel episodes, and not for general episodes. Cule et al. [18] proposed a method that uses neither a window of fixed size, nor minimal occurrences, and an interestingness measure is defined as a combination of the cohesion and the frequency of an episode — again, only for parallel episodes. Tatti [11] and Gwadera et al. [19], [20] define an episode as interesting if its occurrences deviate from expectations.

Finally, an extensive overview of temporal data mining has been made by Laxman and Sastry [21].

VIII. Conclusions

In this paper, we tackled the problem of pattern explosion when mining frequent episodes in an event sequence. In such a setting, much of the output is redundant, as many episodes have the same frequency as some other, more specific, episodes. We therefore output only closed episodes, for which this is not the case. Further redundancy is found in the fact that some episodes can be represented in more than one way. We solve this problem by restricting ourselves to strict, transitively closed episodes.

Defining frequency-closed episodes created new problems, as, unlike in some other settings, a non-closed frequent episode can have more than one closure. To solve this, we defined instance-closed episodes, and showed that the instance-closure of any given episode is unique. We further proved that every f-closed episode must also be i-closed. Based on this, we developed an algorithm that efficiently identifies i-closed episodes, as well as f-closed episodes, in a post-processing step. Experiments have confirmed that the reduction in output is considerable, and essential for large episodes, where we reduced the output by several orders of magnitude. Moreover, thanks to introducing i-closed episodes, we can now produce output for thresholds at which finding all frequent episodes is infeasible.

Acknowledgments

Nikolaj Tatti is funded by a FWO postdoctoral mandate.

References

[1] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 259–289, 1997.

[2] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, 1999, pp. 398–416.

[3] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, and K. Zhang, "Combinatorial pattern discovery for scientific data: some preliminary results," *ACM SIGMOD Record*, vol. 23, no. 2, pp. 115–125, 1994.

[4] R. Agrawal and R. Srikant, "Mining sequential patterns," *11th International Conference on Data Engineering (ICDE 1995)*, vol. 0, pp. 3–14, 1995.

[5] ——, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, 1994, pp. 487–499.

[6] X. Yan, J. Han, and R. Afshar, "Clospan: Mining closed sequential patterns in large datasets," in *Proceedings of the SIAM International Conference on Data Mining (SDM 2003)*, 2003, pp. 166–177.

[7] P. Tzvetkov, X. Yan, and J. Han, "Tsp: Mining top-k closed sequential patterns," in *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, 2003, pp. 347–354.

[8] J. Wang and J. Han, "Bide: Efficient mining of frequent closed sequences," *20th International Conference on Data Engineering (ICDE 2004)*, vol. 0, p. 79, 2004.

[9] G. Casas-Garriga, "Summarizing sequential data with closed partial orders," in *Proceedings of the SIAM International Conference on Data Mining (SDM 2005)*, 2005, pp. 380–391.

[10] M. Garofalakis, R. Rastogi, and K. Shim, "Mining sequential patterns with regular expression constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 3, pp. 530–552, 2002.

[11] N. Tatti, "Significance of episodes based on minimal windows," in *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM 2009)*, 2009, pp. 513–522.

[12] W. Zhou, H. Liu, and H. Cheng, "Mining closed episodes from event sequences efficiently," in *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining(1)*, 2010, pp. 310–318.

[13] S. Laxman, P. S. Sastry, and K. P. Unnikrishnan, "A fast algorithm for finding frequent episodes in event streams," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2007)*, 2007, pp. 410–419.

[14] J. Pei, H. Wang, J. Liu, K. Wang, J. Wang, and P. S. Yu, "Discovering frequent closed partial orders from strings," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 11, pp. 1467–1481, 2006.

[15] G. Casas-Garriga, "Discovering unbounded episodes in sequential data," in *Knowledge Discovery in Databases: PKDD 2003, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2003, pp. 83–94.

[16] N. Méger and C. Rigotti, "Constraint-based mining of episode rules and optimal window sizes," in *Knowledge Discovery in Databases: PKDD 2004, 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2004, pp. 313–324.

[17] T. Calders, N. Dexters, and B. Goethals, "Mining frequent itemsets in a stream," in *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007)*, 2007, pp. 83–92.

[18] B. Cule, B. Goethals, and C. Robardet, "A new constraint for mining sets in sequences," in *Proceedings of the SIAM International Conference on Data Mining (SDM 2009)*, 2009, pp. 317–328.

[19] R. Gwadera, M. J. Atallah, and W. Szpankowski, "Reliable detection of episodes in event sequences," *Knowledge and Information Systems*, vol. 7, no. 4, pp. 415–437, 2005.

[20] ——, "Markov models for identification of significant episodes," in *Proceedings of the SIAM International Conference on Data Mining (SDM 2005)*, 2005, pp. 404–414.

[21] S. Laxman and P. S. Sastry, "A survey of temporal data mining," *SADHANA, Academy Proceedings in Engineering Sciences*, vol. 31, no. 2, pp. 173–198, 2006.