Mining Closed Episodes with Simultaneous Events

Nikolaj Tatti, Boris Cule
ADReM, University of Antwerp, Antwerpen, Belgium
firstname.lastname@ua.ac.be

ABSTRACT

Sequential pattern discovery is a well-studied field in data mining. Episodes are sequential patterns describing events that often occur in the vicinity of each other. Episodes can impose restrictions to the order of the events, which makes them a versatile technique for describing complex patterns in the sequence. Most of the research on episodes deals with special cases such as serial, parallel, and injective episodes, while discovering general episodes is understudied.

In this paper we extend the definition of an episode in order to be able to represent cases where events often occur simultaneously. We present an efficient and novel miner for discovering frequent and closed general episodes. Such a task presents unique challenges. Firstly, we cannot define closure based on frequency. We solve this by computing a more conservative closure that we use to reduce the search space and discover the closed episodes as a postprocessing step. Secondly, episodes are traditionally presented as directed acyclic graphs. We argue that this representation has drawbacks leading to redundancy in the output. We solve these drawbacks by defining a subset relationship in such a way that allows us to remove the redundant episodes. We demonstrate the efficiency of our algorithm and the need for using closed episodes empirically on synthetic and real-world datasets.

Categories and Subject Descriptors

H.2.8 [**Database management**]: Database applications—Data mining; G.2.2 [**Discrete mathematics**]: Graph theory

General Terms

Algorithms, Theory

Keywords

Frequent episodes, Closed episodes, Depth-first search

1. INTRODUCTION

Discovering interesting patterns in data sequences is a popular aspect of data mining. An episode is a sequential pattern representing a set of events that reoccur in a sequence [14]. In its most general form, an episode also imposes a partial order on the events. This allows great flexibility in describing complex interactions between the events in the sequence.

Existing research in episode mining is dominated by two special cases: parallel episodes, patterns where the order of the events does not matter, and serial episodes, requiring that the events must occur in one given order. Proposals have been made (see Section 2) for discovering episodes with partial orders, but these approaches impose various limitations on the events. In fact, to our knowledge, there is no published work giving an explicit description of a miner for general episodes.

We believe that there are two main reasons why general episodes have attracted less interest: Firstly, implementing a miner is surprisingly difficult: testing whether an episode occurs in the sequence is an **NP**-complete problem. Secondly, the fact that episodes are such a rich pattern type leads to a severe pattern explosion.

Another limitation of episodes is that they do not properly address simultaneous events. However, sequences containing such events are frequently encountered, in cases such as, for example, sequential data generated by multiple sensors and then collected into one stream. In such a setting, if two events, say a and b, often occur simultaneously, existing approaches will depict this pattern as a parallel episode $\{a, b\}$, which will only tell the user that these two events often occur near each other, in no particular order. This is a major limitation, since the actual pattern contains much more information.

In this paper we propose a novel and practical algorithm for mining frequent closed episodes that properly handles simultaneous events. Such a task poses several challenges.

Firstly, we can impose four different relationships between two events a and b: (1) the order of a and b does not matter, (2) events a and b should occur at the same time, (3) b should occur after a, and (4) b should occur after or at the same time as a. We extend the definition of an episode to handle all these cases. In further text, we consider events simultaneous only if they occur exactly at the same time. However, we can easily adjust our framework to consider events simultaneous if they occur within a chosen time interval.

Secondly, a standard approach for representing a partial order of the events is by using a directed acyclic graph

(DAG). The mining algorithm would then discover episodes by adding nodes and edges. However, we point out that such a representation has drawbacks. One episode may be represented by several graphs and the subset relationship based on the graphs is not optimal. This ultimately leads to outputting redundant patterns. We will address this problem.

Thirdly, we attack the problem of pattern explosion by using closed patterns. There are two particular challenges with closed episodes. Firstly, we point out that we cannot define a unique closure for an episode, that is, an episode may have several maximal episodes with the same frequency. Secondly, the definition of a closure requires a subset relationship, and computing the subset relationship between episodes is **NP**-hard.

We mine patterns using a depth-first search. An episode is represented by a DAG and we explore the patterns by adding nodes and edges. To reduce the search space we use the *instance-closure* of episodes. While it is not guaranteed that an instance-closed episode is actually closed, using such episodes will greatly trim the pattern space. Finally, the actual filtering for closed episodes is done in a post-processing step. We introduce techniques for computing the subset relationship, distinguishing the cases where we can do a simple test from the cases where we have to resort to recursive enumeration. This filtering will remove all redundancies resulting from using DAGs for representing episodes.

The rest of the paper is organised as follows: In Section 2, we discuss the most relevant related work. In Section 3, we present the main notations and concepts. Section 4 introduces the notion of closure in the context of episodes. Our algorithm is presented in detail in Sections 5, 6 and 7. In Section 8 we present the results of our experiments, before presenting our conclusions in Section 9. The proofs of the theorems can be found in the Appendixand the code of the algorithm is available online[1].

2. RELATED WORK

The first attempt at discovering frequent subsequences, or serial episodes, was made by Wang et al. [22]. The dataset consisted of a number of sequences, and a pattern was considered interesting if it was long enough and could be found in a sufficient number of sequences. A complete solution to a more general problem was later provided by Agrawal and Srikant [2] using an Apriori-style algorithm [1].

Looking for frequent general episodes in a single event sequence was first proposed by Mannila et al. [14]. The Winepi algorithm finds all episodes that occur in a sufficient number of windows of fixed length. Specific algorithms were given for the case of parallel and serial episodes. However, no algorithm for detecting general episodes was provided.

Some research has gone into outputting only closed subsequences, where a sequence is considered closed if it is not properly contained in any other sequence which has the same frequency. Yan et al. [23], Tzvetkov et al. [20], and Wang and Han [21] proposed methods for mining such closed patterns, while Garriga [5] further reduced the output by post-processing it and representing the patterns using partial orders.[2] Harms et al. [12], meanwhile, experiment with closed serial episodes. In another attempt to trim the output, Garofalakis et al. [8] 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.

Pei et al. [17], and Tatti and Cule [19] considered restricted versions of our problem setup. The former approach assumes a dataset of sequences where the same label can occur only once. Hence, an episode can contain only unique labels. The latter pointed out the problem of defining a proper subset relationship between general episodes and tackled it by considering only episodes where two nodes having the same label had to be connected. In our work, we impose no restrictions on the labels of events making up the episodes.

In this paper we use frequency based on a sliding window as it is defined for Winepi. However, we can easily adopt our approach for other monotonically decreasing measures, as well as to a setup where the data consists of many (short) sequences instead of a single long one. Mannila et al. propose Minepi [14], an alternative interestingness measure for an episode, where the support is defined as the number of minimal windows. Unfortunately, this measure is not monotonically decreasing. However, the issue can be fixed by defining support as the maximal number of non-overlapping minimal windows [13, 18]. Zhou et al. [24] proposed mining closed serial episodes based on the Minepi method. However, the paper did not address the non-monotonicity issue of Minepi.

Alternative interestingness measures, either statistically motivated or aimed to remove bias towards smaller episodes, were made by Garriga [4], Méger and Rigotti [15], Gwadera et al. [9, 10], Calders et al. [3], Cule et al. [7], and Tatti [18].

Using episodes to discover simultaneous events has, to our knowledge, not been done yet. However, this work is somewhat related to efforts made in discovering sequential patterns in multiple streams [6, 11, 16]. Here, it is possible to discover a pattern wherein two events occur simultaneously, as long as they occur in separate streams.

3. EPISODES WITH SIMULTANEOUS EVENTS

We begin this section by introducing the basic concepts that we will use throughout the paper. First we will describe our dataset.

Definition 1 A *sequence event* $e = (id(e), lab(e), ts(e))$ is a tuple consisting of three entries, a unique id number $id(e)$, a label $lab(e)$, and a time stamp integer $ts(e)$. We will assume that if $id(e) > id(f)$, then $ts(e) \geq ts(f)$. A *sequence* is a collection of sequence events ordered by their ids.

Note that we are allowing multiple events to have the same time stamp even when their labels are equivalent. For the sake of simplicity, we will use the notation $s_1 \cdots s_N$ to mean a sequence $((1, s_1, 1), \ldots, (N, s_N, N))$. We will also write $s_1 \cdots (s_i s_{i+1}) \cdots s_N$ to mean the sequence

$$((1, s_1, 1), \ldots, (i, s_i, i), (i + 1, s_{i+1}, i), \ldots, (N, s_N, N - 1)).$$

This means that s_i and s_{i+1} have equal time stamps.

Our next step is to define patterns we are interested in.

Definition 2 An *episode event* e is a tuple consisting of two entries, a unique id number $id(e)$ and a label $lab(e)$. An *episode graph* G is a directed acyclic graph (DAG). The

[1]http://adrem.ua.ac.be/implementations
[2]Despite their name, the partial orders discovered by Garriga are different from general episodes.

graph may have two types of edges: *weak edges* $WE(G)$ and *proper edges* $PE(G)$.

An *episode* consists of a collection of episode events, an episode graph, and a surjective mapping from episode events to the nodes of the graph which we will denote by $nd(e)$. A proper edge from node v to node w in the episode graph implies that the events of v must occur before the events of w, while a weak edge from v to w implies that the events of w may occur either at the same time as those of v or later.

We will assume that the nodes of G are indexed and we will use the notation $nd(G, i)$ to refer to the ith node in G.

When there is no danger of confusion, we will use the same letter to denote an episode and its graph. Note that we are allowing multiple episode events to share the same node even if these events have the same labels.

Definition 3 Given an episode G and a node n, we define $lab(n) = \{lab(e) \mid nd(e) = n\}$ to be the multiset of labels associated with the node. Given two multisets of labels X and Y we write $X \leq_{lex} Y$ if X is lexicographically smaller than or equal to Y. We also define $lab(G)$ to be the multiset of all labels in G.

Definition 4 A node n in an episode graph is a *descendant* of a node m if there is a path from m to n. If there is a path containing a proper edge we will call n a *proper descendant* of m. We similarly define a *(proper) ancestor*. A node n is a *source* if it has no ancestors. A node n is a *proper source* if it has no proper ancestors. We denote all sources of an episode G by $src(G)$.

We are now ready to give a precise definition of an occurrence of a pattern in a sequence.

Definition 5 Given a sequence s and an episode G, we say that s covers G if there exists an *injective* mapping m from the episode events to the sequence events such that

1. labels are respected, $lab(m(e)) = lab(e)$,
2. events sharing a same node map to events with the same time stamp, in other words, $nd(e) = nd(f)$ implies $ts(m(e)) = ts(m(f))$,
3. weak edges are respected, if $nd(e)$ is a descendant of $nd(f)$, then $ts(m(e)) \geq ts(m(f))$,
4. proper edges are respected, if $nd(e)$ is a proper descendant of $nd(f)$, then $ts(m(e)) > ts(m(f))$.

Note that this definition allows us to abuse notation and map graph nodes directly to time stamps, that is, given a graph node n we define $ts(m(n)) = ts(m(e))$, where $nd(e) = n$.

Consider the first three episodes in Figure 1. A sequence $ab(cd)$ covers G_1 and G_3 but not G_2 (proper edge (c, d) is violated). A sequence $(ab)cd$ covers G_1 and G_2 but not G_3.

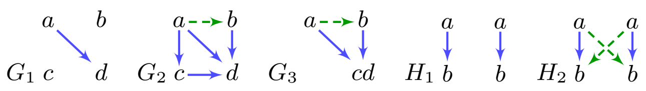


Figure 1: Toy episodes. Proper edges are drawn solid. Weak edges are drawn dashed.

Finally, we are ready to define support of an episode based on fixed windows. This definition corresponds to the definition used in Winepi [14]. The support is monotonically decreasing which allows us to do effective pruning while discovering frequent episodes.

Definition 6 Given a sequence s and two integers i and j we define a *subsequence*

$$s[i, j] = \{e \in s \mid i \leq ts(e) \leq j\}$$

containing all events occurring between i and j.

Definition 7 Given a window size ρ and an episode s, we define the *support* of an episode G in s, denoted $fr(G; s)$, to be the number of windows of size ρ in s covering the episode,

$$fr(G; s) = |\{s[i, i + \rho - 1] \mid s[i, i + \rho - 1] \text{ covers } G\}|.$$

We will use $fr(G)$ whenever s is clear from the context. An episode is *σ-frequent* (or simply *frequent*) if its support is higher or equal than some given threshold σ.

Consider a sequence $abcdacbd$ and set the window size $\rho = 4$. There are 2 windows covering episode G_1 (given in Figure 1), namely $s[1, 4]$ and $s[5, 8]$. Hence $fr(G_1) = 2$.

Theorem 8 *Testing whether a sequence s covers an episode G is an **NP**-complete problem, even if s does not contain simultaneous events.*

4. SUBEPISODE RELATIONSHIP

In practice, episodes are represented by DAGs and are mined by adding nodes and edges. However, such a representation has drawbacks [19]. To see this, consider episodes H_1 and H_2 in Figure 1. Even though these episodes have different graphs, they are essentially the same — both episodes are covered by exactly the same sequences, namely all sequences containing $abab$, $a(ab)b$, $(aa)(bb)$, $aa(bb)$, $(aa)bb$, or $aabb$. In other words, essentially the same episode may be represented by several graphs. Moreover, using the graph subset relationship to determine subset relationships between episodes will ultimately lead to less efficient algorithms and redundancy in the final output. To counter these problems we introduce a subset relationship based on coverage.

Definition 9 Given two episodes G and H, we say that G is a *subepisode* of H, denoted $G \preceq H$, if any sequence covering H also covers G. If $G \preceq H$ and $H \preceq G$, we say that G and H are *similar* in which case we will write $G \sim H$.

This definition gives us the optimal definition for a subset relationship in the following sense: if $G \not\preceq H$, then there exists a sequence s such that $fr(G; s) < fr(H; s)$.

Consider the episodes given in Figure 1. It follows from the definition that $H_1 \sim H_2$, $G_1 \preceq G_2$, and $G_1 \preceq G_3$. Episodes G_2 and G_3 are not comparable.

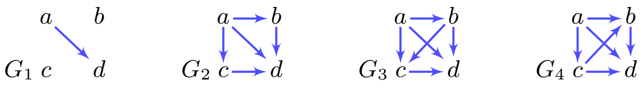
Theorem 10 *Testing $G \preceq H$ is an **NP**-hard problem.*

Proof. The hardness follows immediately from Theorem 8 as we can represent sequence s as a serial episode H. Then s covers G if and only if $G \preceq H$. $\square$

As mentioned in the introduction, pattern explosion is *the* problem with discovering general episodes. We tackle this by mining only closed episodes.

Definition 11 An episode G is closed if there is no $H \succ G$ with $fr(G) = fr(H)$.

We should point out that, unlike with itemsets, an episode may have several maximal superepisodes having the same frequency. Consider G_1, G_3, and G_4 in Figure 2, sequence *abcbdacbcd* and window size $\rho = 5$. The support of episodes G_1, G_3 and G_4 is 2. Moreover, there is no superepisode of G_3 or G_4 that has the same support. Hence, G_3 and G_4 are both maximal superepisodes having the same support as G_1. This implies that we cannot define a closure operator based on frequency. However, we will see in the next section that we can define a closure based on instances. This closure, while not removing all redundant episodes, will prune the search space dramatically. The final pruning will then be done in a post-processing step.



$$\begin{tikzcd}
 & a \arrow[dr] & b & & a \arrow[r] \arrow[d] \arrow[dr] & b \arrow[d] & & a \arrow[r] \arrow[d] \arrow[dr] & b \arrow[d] \arrow[dl] & & a \arrow[r] \arrow[d] \arrow[dr] & b \arrow[d] \\
G_1 & c & d & G_2 & c \arrow[r] & d & G_3 & c \arrow[r] & d & G_4 & c \arrow[r] \arrow[ur] & d
\end{tikzcd}$$

Figure 2: Toy episodes demonstrating closure.

Our final step is to define transitively closed episodes that we will use along with the instance-closure (defined in the next section) in order to reduce the pattern space.

Definition 12 Let G be an episode. A *transitive closure*, $tcl(G)$, is obtained by adding edges from a node to each of its descendants making the edge proper if the descendant is proper, and weak otherwise. If $G = tcl(G)$ we say that G is transitively closed.

It is trivial to see that given an episode G, we have $G \sim tcl(G)$. Thus we can safely ignore all episodes that are not transitively closed. From now on, unless we state otherwise, all episodes are assumed to be transitively closed.

5. HANDLING EPISODE INSTANCES

The reason why depth-first search is efficient for itemsets is that at each step we only need to handle the current projected dataset. In our setup we have only one sequence so we need to transport the sequence into a more efficient structure.

Definition 13 Given an input sequence s and an episode G, an *instance* i is a valid mapping from G to s such that for each $e \in range(i)$ there is no $f \in s - range(i)$ such that $lab(e) = lab(f)$, $ts(e) = ts(f)$ and $id(f) < id(e)$. We define $first(i) = \min ts(i(n))$ to be the smallest time stamp and $last(i) = \max ts(i(n))$ to be the largest time stamp in i. We require that $last(i) - first(i) \leq \rho - 1$, where ρ is the size of the sliding window. An *instance set* of an episode G, defined as $inst(G)$ is a set of all instances ordered by $first(i)$.

The condition in the definition allows us to ignore some redundant mappings whenever we have two sequence events, say e and f, with $lab(e) = lab(f)$ and $ts(e) = ts(f)$. If an instance i uses only e, then we can obtain i' from i by replacing e with f. However, i and i' are essentially the same for our purposes, so we can ignore either i or i'. We require the instance set to be ordered so that we can compute the support efficiently. This order is not necessarily unique.

Consider sequence *abcbdacbcd* and G_1 in Figure 2. Then $inst(G_1) = ((1, 2, 3, 5), (1, 4, 3, 5), (6, 7, 8, 10), (6, 9, 8, 10))$[3].

Using instances gives us several advantages. Adding new events and edges to episodes becomes easy. For example, adding a proper edge (n, m) is equivalent to keeping instances with $ts(i(n)) < ts(i(m))$. We will also compute support and closure efficiently. We should point out that $inst(G)$ may contain an exponential number of instances, otherwise Theorem 8 would imply that $\mathbf{P} = \mathbf{NP}$. However, this is not a problem in practice.

The depth-first search described in Section 6 adds events to the episodes. Our next goal is to define algorithms for computing the resulting instance set whenever we add an episode event, say f, to an episode G. Given an instance i of G and a sequence event e we will write $i + e$ to mean an expanded instance by setting $i(f) = e$.

Let G be an episode and let $I = inst(G)$ be the instance set. Assume a node $n \in V(G)$ and a label l. Let H be the episode obtained from G by adding an episode event with label l to node n. We can compute $inst(H)$ from $inst(G)$ using the AugmentEqual algorithm given in Alg. 1.

```
Algorithm 1: AugmentEqual($I, n, l$), augments $I$
  input  : $I = inst(G)$, a node $n$, a label $l$
  output: $inst(G$ with $n$ augmented with $l)$
1 return $\left\{ i + e \mid \begin{array}{l} i \in I, e \in s, lab(e) = l, \\ ts(n) = ts(e), i + e \text{ is an instance} \end{array} \right\}$;
```

The second augmentation algorithm deals with the case where we are adding a new node with an single event labelled as l to a parallel episode. Algorithm Augment, given in Alg. 2, computes the new instance set. The algorithm can be further optimised by doing augmentation with a type of merge sort so that post-sorting is not needed.

```
Algorithm 2: Augment($I, l$), augments instances
  input  : $I = inst(G)$, label $l$ of the new event
  output: $inst(G$ with a new node labelled with $l)$
1 $E \leftarrow \{e \in s \mid lab(e) = l\}$;
2 $J \leftarrow \{i + e \mid i \in I, e \in E, i + e$ is an instance$\}$;
3 Sort $J$ by $first(i)$;
4 return $J$;
```

Our next step is to compute the support of an episode G from $inst(G)$. We do this with the Support algorithm, given in Alg. 3. The algorithm is based on the observation that there are $\rho - (last(i) - first(i))$ windows that contain the instance i. However, some windows may contain more than one instance and we need to compensate for this.

Theorem 14 *Support*($inst(G)$) *computes* $fr(G)$.

Proof. Since I is ordered by $first(i)$, the first for loop of the algorithm removes any instance for which there is an instance j such that $first(i) \leq first(j) \leq last(j) \leq last(i)$. In other words, any window that contains i will also contain j. We will show that the next for loop counts the number of windows containing at least one instance from W, this will imply the theorem.

To that end, let $i_n \in J$ be the nth instance in J and define S_n to be the set of windows of size ρ containing i_n. It follows that $|S_n| = \rho - (last(i_n) - first(i_n))$ and that the first window of S_n starts at $a_n = 1 + last(i_n) - \rho$. Let $C_n = \bigcup_{m=1}^{n} S_m$. Note that because of the pruning we have $a_{n-1} < a_n$, this implies that $C_{n-1} \cap S_n = S_{n-1} \cap S_n$. We

[3]For simplicity, we write mappings as tuples

Algorithm 3: SUPPORT(I), computes support

```
    input  : $I = inst(G)$
    output: $fr(G)$
1   $J \leftarrow \emptyset$; $l \leftarrow \infty$;
2   foreach $i \in I$ in reverse order do
3       if $last(i) < l$ then
4           Add $i$ to $J$;
5           $l \leftarrow last(i)$;
6   $l \leftarrow -\infty$; $f \leftarrow 0$;
7   foreach $i \in J$ do
8       $d \leftarrow \rho - (last(i) - first(i))$;
9       $a \leftarrow 1 + last(i) - \rho$;
10      $d \leftarrow d - \max(0, 1 + l - a)$;
11      $f \leftarrow f + d$;
12      $l \leftarrow first(i)$;
13  return $f$;
```

know that $|S_{n-1} \cap S_n| = \max(0, 1 + first(i_{n-1}) - a_n)$. This implies that on Line 10 we have $d = |S_n - S_{n-1}|$ and since $|C_n| = |C_{n-1}| + d$, this proves the theorem. □

Finally, we define a closure episode of an instance set.

DEFINITION 15 Let $I = inst(G)$ be a set of instances. We define an *instance-closure*, $H = icl(I)$ to be the episode having the same nodes and events as G. We define the edges

$$PE(H) = \{(a, b) \mid ts(i(a)) < ts(i(b))\ \forall i \in I\} \text{ and}$$
$$WE(H) = \{(a, b) \mid ts(i(a)) \leq ts(i(b))\ \forall i \in I\} - PE(H).$$

If $G = icl(I)$ we say that G is *i*-closed.

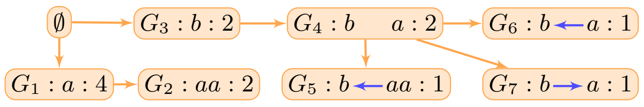
Consider sequence *abcbdacbcd* and episodes given in Figure 2. Since events b and c always occur between a and d in $inst(G_1)$, the instance closure is $icl(inst(G_1)) = G_2$. Note that G_2 is not closed because G_3 and G_4 are both superepisodes of G_2 with the same support. However, instance-closure reduces the search space dramatically because we do not have to iterate the edges implied by the closure. Note that the closure may produce cycles with weak edges. However, we will later show that we can ignore such episodes.

6. DISCOVERING EPISODES

We are now ready to describe the mining algorithm. Our approach is a straightforward depth-first search. The algorithm works on three different levels.

The first level, consisting of MINE (Alg. 4), and MINEPARALLEL (Alg. 5), adds episode events. MINE is only used for creating singleton episodes while MINEPARALLEL provides the actual search. The algorithm adds events so that the labels of the nodes are decreasing, $lab(nd(G, i+1)) \leq_{lex} lab(nd(G, i))$. The search space is traversed by either adding an event to the last node or by creating a node with a single event. Episodes $G_1, \ldots, G_5$ in Figure 3 are created by the first level. The edge in G_5 is augmented by the closure.

The second level, MINEWEAK, given in Alg. 6, adds weak edges to the episode, while the third level, MINEPROPER, given in Alg. 7, turns weak edges into proper edges. Both algorithms add only those edges which keep the episode transitively closed. The algorithms keep a list of forbidden weak edges W and forbidden proper edges P. These lists guarantee that each episode is visited only once.



∅
$G_3 : b : 2$
$G_4 : b \quad a : 2$
$G_6 : b \leftarrow a : 1$
$G_1 : a : 4$
$G_2 : aa : 2$
$G_5 : b \leftarrow aa : 1$
$G_7 : b \rightarrow a : 1$

Figure 3: Search space for a sequence $(aa)ba$, window size $\rho = 2$, and support threshold $\sigma = 2$. Each state shows the corresponding episode and its support.

Algorithm 4: MINE, discovers frequent closed episodes

```
1   for $x \in \Sigma$ do
2       $I \leftarrow inst$(singleton episode with the label $x$);
3       $G \leftarrow$ TESTEPISODE($I, \emptyset, \emptyset$);
4       if $G \neq$ null then MINEPARALLEL($I, G$);
```

Algorithm 5: MINEPARALLEL(I, G), recursive routine adding episode events

```
    input  : episode $G$, $I = inst(G)$
1   MINEWEAK($I, G, \emptyset$);
2   $M \leftarrow |V(G)|$;
3   $n \leftarrow nd(G, M)$;
4   for $x \in \Sigma, x \geq \max lab(n)$ do
5       if $M = 1$ or $lab(n) \cup \{x\} \leq_{lex} lab(nd(G, M-1))$
        then
6           $J \leftarrow$ AUGMENTEQUAL($I, n, x$);
7           $H \leftarrow$ TESTEPISODE($J, \emptyset, \emptyset$);
8           if $H \neq$ null then
9               MINEPARALLEL($J, H$);
10  for $x \in \Sigma, x \leq \min lab(n)$ do
11      $J \leftarrow$ AUGMENT($I, x$);
12      $H \leftarrow$ TESTEPISODE($J, \emptyset, \emptyset$);
13      if $H \neq$ null then
14          MINEPARALLEL($J, H$);
```

G_6 and G_7 in Figure 3 are discovered by MINEWEAK, however, the weak edges are converted into proper edges by the instance-closure.

Algorithm 6: MINEWEAK(I, G, W), recursive routine adding weak edges

```
    input  : ep. $G$, $I = inst(G)$, forbidden weak edges $W$
1   $A \leftarrow \{(a, b) \mid a, b \in V(G), (a, b) \notin E(G)\}$;
2   MINEPROPER($I, G, A, \emptyset$);
3   for $(a, b) \notin E(G) \cup W$ do
4       if $G + (a, b)$ is transitively closed then
5           $J \leftarrow \{i \in I \mid ts(i(a)) \leq ts(i(b))\}$;
6           $H \leftarrow$ TESTEPISODE($J, W, \emptyset$);
7           if $H \neq$ null then
8               MINEWEAK($J, H, W$);
9           Add $(a, b)$ to $W$;
```

At each step, we call TESTEPISODE. This routine, given a set of instances I, will compute the instance-closure $icl(I)$ and test it. If the episode passes all the tests, the algorithm will return the episode and the search is continued, otherwise the branch is terminated. There are four different tests. The

Algorithm 7: MINEPROPER(I, G, W, P), recursive routine adding proper edges

```
input : episode $G$, $I = inst(G)$, forbidden weak edges
        $W$, forbidden proper edges $P$
1  for $(a, b) \in WE(G) - P$ do
2      if $G +$ proper edge $(a, b)$ is transitively closed then
3          $J \leftarrow \{i \in I \mid ts(i(a)) < ts(i(b))\}$;
4          $H \leftarrow$ TESTEPISODE($J, W, P$);
5          if $H \neq$ null then
6              MINEPROPER($J, H, W, P$);
7          Add $(a, b)$ to $P$;
```

first test checks whether the episode is frequent. If we pass this test, we compute the instance-closure $H = icl(I)$. The second test checks whether H contains cycles. Let G be an episode such that $I = inst(G)$. In order for H to have cycles we must have two nodes, say n and m, such that $ts(i(n)) = ts(i(m))$ for all $i \in I$. Let G' be an episode obtained from G by merging nodes in n and m together. We have $G \preceq G'$ and $fr(G) = fr(G')$. This holds for any subsequent episode discovered in the branch allowing us to ignore the whole branch.

If the closure introduces into H any edge that has been explored in the previous branches, then that implies that H has already been discovered. Hence, we can reject H if any such edge is introduced. The final condition is that during MINEPROPER no weak edges should be added into H by the closure. If a weak edge is added, we can reject H because it can be reached via an alternative route, by letting MINEWEAK add the additional edges and then calling MINEPROPER.

The algorithm keeps a list $\mathcal{C}$ of all discovered episodes that are closed. If all four tests are passed, the algorithm tests whether there are subepisodes of G in $\mathcal{C}$ having the same frequency, and deletes them. On the other hand, if there is a superepisode of G in $\mathcal{C}$, then G is not added into $\mathcal{C}$.

Algorithm 8: TESTEPISODE(I, W, P), tests the episode $icl(I)$ and updates $\mathcal{C}$, the list of discovered episodes

```
input : $I = inst(G)$, forbidden weak edges $W$,
        forbidden proper edges $P$
output: $icl(I)$, if $icl(I)$ passes the tests, null otherwise
1  $f \leftarrow$ SUPPORT($I$);
2  if $f < \sigma$ then return null;
3  $G \leftarrow icl(I)$;
4  if there are cycles in $G$ then
5      return null;
6  if $WE(G) \cap W \neq \emptyset$ or $PE(G) \cap P \neq \emptyset$ then
7      return null;
8  $fr(G) \leftarrow f$;
9  foreach $H \in \mathcal{C}, lab(H) \cap lab(G) \neq \emptyset$ do
10     if $fr(G) = fr(H)$ and $G \preceq H$ then
11         return $G$;
12     if $fr(G) = fr(H)$ and $H \preceq G$ then
13         Delete $H$ from $\mathcal{C}$;
14 Add $G$ to $\mathcal{C}$;
15 return $G$;
```

7. TESTING SUBEPISODES

In this section we will describe the technique for computing $G \preceq H$. In general, this problem is difficult to solve as pointed out by Theorem 10. Fortunately, in practice, a major part of the comparisons are easy to do. The following theorem says that if the labels of G are unique in H, then we can easily compare G and H.

THEOREM 16 *Assume two episodes G and H. Assume that $lab(G) \subset lab(H)$ and for each event e in G only one event occurs in H with the same label. Let π be the unique mapping from episode events in G to episode events in H honouring the labels. Then $G \preceq H$ if and only if*

1. *$nd(e) = nd(f)$ implies that $nd(\pi(e)) = nd(\pi(f))$,*
2. *$nd(e)$ is a proper child of $nd(f)$ implies that $nd(\pi(e))$ is a proper child of $nd(\pi(f))$,*
3. *$nd(e)$ is a child of $nd(f)$ implies that $nd(\pi(e))$ is a child of $nd(\pi(f))$ or $nd(\pi(e)) = nd(\pi(f))$,*

for any two events e and f in G.

If the condition in Theorem 16 does not hold we will have to resort to enumerating the sequences covering H. In order to do that, we need to extend the definition of coverage and subset relationship to the set of episodes.

DEFINITION 17 A sequence s covers an episode set $\mathcal{G}$ if there is an episode $G \in \mathcal{G}$ such that s covers G. Given two episode sets $\mathcal{G}$ and $\mathcal{H}$ we define $\mathcal{G} \preceq \mathcal{H}$ if every sequence that covers $\mathcal{H}$ also covers $\mathcal{G}$.

We also need a definition of a prefix subgraph.

DEFINITION 18 Given a graph G, a *prefix subgraph* is a non-empty induced subgraph of G with *no* proper edges such that if a node n is included then the parents of n are also included. Given a multiset of labels L and an episode G we define $pre(G, L)$ to be the set of all maximal prefix subgraphs such that $lab(V) \subseteq L$ for each $V \in pre(G, L)$. We define $tail(G, L) = \{G - V \mid V \in pre(G, L)\}$ to be the episodes with the remaining nodes. Finally, given an episode set $\mathcal{G}$ we define $tail(\mathcal{G}, L) = \bigcup_{G \in \mathcal{G}} tail(G, L)$.

EXAMPLE 19 Consider episodes given in Figure 4. We have $tail(G, ab) = \{H_1, H_2, H_3\}$ and $tail(G, a) = \{H_1\}$.

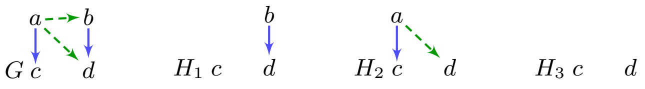


Figure 4: Toy episodes demonstrating $tail(G, ab)$.

The main motivation for our recursion is given in the following theorem.

THEOREM 20 *Given an episode set $\mathcal{G}$ and an episode H, $\mathcal{G} \preceq H$ if and only if for each prefix subgraph V of H, we have $tail(\mathcal{G}, lab(V)) \preceq H - V$.*

We focus for the rest of this section on implementing the recursion in Theorem 20. We begin by an algorithm, GENERATE, given in Alg. 9, that, given a graph without proper edges, discovers all prefix subgraphs.

Algorithm 9: GENERATE(G, V), recursive routine for iterating the nodes of all prefix subgraphs of G

```
   input : graph $G$, nodes $V$ discovered so far
   output: list of nodes of all prefix subgraphs
1  $\mathcal{O} \leftarrow \emptyset$;
2  foreach $n \in src(G)$ do
3      $\mathcal{O} \leftarrow \mathcal{O} \cup \{V + n\} \cup$ GENERATE$(G - n, V + n)$;
4      Remove $n$ and its descendants from $G$;
5  return $\mathcal{O}$;
```

Given a prefix subgraph V of H, our next step is to discover all maximal prefix subgraphs of G whose label sets are subsets of $L = lab(V)$. The algorithm, CONSUME, creating this list, is given in Alg. 10. CONSUME enumerates over all sources. For each source n such that $lab(n) \subseteq L$, the algorithm tests if there is another node sharing a label with n. If so, the algorithm creates an episode without n and its descendants and calls itself. This call produces a list of prefix graphs $\mathcal{W}$ not containing node n. The algorithm removes all graphs from $\mathcal{W}$ that can be augmented with n (since in that case they are not maximal). Finally, CONSUME adds n to the current prefix subgraph, removes n from G, and removes $lab(n)$ from L.

Algorithm 10: CONSUME(G, L, V), recursive routine for iterating the nodes of all prefix subgraphs of G whose labels are contained in L

```
    input : graph $G$, nodes $V$ discovered so far, $L$ label set
    output: list of nodes of all maximal prefix subgraphs
1   $\mathcal{O} \leftarrow \emptyset$;
2   while $src(G) \neq \emptyset$ do
3       $n \leftarrow$ a node from $src(G)$;
4       if $lab(n) \not\subseteq L$ then
5           Remove $n$ and its descendants from $G$;
6           continue;
7       if there is $m \in V(G)$ s.t. $lab(n) \cap lab(m) \neq \emptyset$ then
8           $H \leftarrow G$ with $n$ and its descendants removed;
9           $\mathcal{W} \leftarrow$ CONSUME$(H, L, V)$;
10          $\mathcal{O} \leftarrow \mathcal{O} \cup \{W \in \mathcal{W} \mid lab(W) \cup lab(n) \not\subseteq L\}$;
11      $V \leftarrow V + n$; $L \leftarrow L - lab(n)$;
12      Remove $n$ from $G$;
13  return $\mathcal{O} \cup \{V\}$;
```

We are now ready to describe the recursion step of Theorem 20 for testing the subset relationship. The algorithm, STEP, is given in Alg. 11. Given an episode set $\mathcal{G}$ and a graph H, the algorithm computes $\mathcal{G} \preceq H$. First, it tests whether we can apply Theorem 16. If this is not possible, then the algorithm first removes all nodes from H not carrying a label from $lab(G)$ for $G \in \mathcal{G}$. This is allowed because of the following lemma.

LEMMA 21 *Let G and H be two episodes. Let $n \in V(H)$ be a node such that $lab(n) \cap lab(G) = \emptyset$. Let H' be the episode obtained from H by removing n. Then $G \preceq H$ if and only if $G \preceq H'$.*

STEP continues by creating a subgraph Y of H containing only proper sources. The algorithm generates all prefix subgraphs $\mathcal{V}$ of Y and tests each one. For each subgraph $V \in \mathcal{V}$, STEP calls CONSUME and builds an episode set $\mathcal{T} = tail(\mathcal{G}, lab(V))$. The algorithm then calls itself recursively with STEP$(\mathcal{T}, H - V)$. If at least one of these calls fails, then we know that $\mathcal{G} \not\preceq H$, otherwise $\mathcal{G} \preceq H$.

Finally, to compute $G \preceq H$, we simply call STEP$(\{G\}, H)$.

Algorithm 11: STEP$(\mathcal{G}, H)$, recursion solving $\mathcal{G} \preceq H$

```
    input : episode set $\mathcal{G}$ and an episode $H$
    output: $\mathcal{G} \preceq H$
1   foreach $G \in \mathcal{G}$ do
2       if Theorem 16 guarantees that $G \preceq H$ then
3           return true;
4       if Theorem 16 states that $G \not\preceq H$ and $|\mathcal{G}| = 1$ then
5           return false;
6   $V(H) \leftarrow \{n \in V(H) \mid lab(n) \cap lab(G) \neq \emptyset, G \in \mathcal{G}\}$;
7   $Y \leftarrow$ subgraph of $H$ with only proper sources;
8   $\mathcal{V} \leftarrow$ GENERATE$(Y, \emptyset)$;
9   foreach $V \in \mathcal{V}$ do
10      $F \leftarrow H - V$; $\mathcal{T} \leftarrow \emptyset$;
11      foreach $G \in \mathcal{G}$ do
12          $X \leftarrow$ subgraph of $G$ with only proper sources;
13          $\mathcal{W} \leftarrow$ CONSUME$(X, lab(V), \emptyset)$;
14          $\mathcal{T} \leftarrow \mathcal{T} \cup \{G - W \mid W \in \mathcal{W}\}$;
15      if STEP$(\mathcal{T}, F)$ = false then
16          return false;
17  return true;
```

8. EXPERIMENTS

We begin our experiments with a synthetic dataset. Our goal is to demonstrate the need for using the closure. In order to do that we created sequences with a planted pattern $(s_1 s_2)(s_3 s_4) \cdots (s_{2N-1} s_{2N})$. We added this pattern 100 times 50 time units apart from each other. We added 500 noise events uniformly spreading over the whole sequence. We sampled the labels for the noise events uniformly from 900 different labels. The labels for the noise and the labels of the pattern were mutually exclusive. We varied N from 1 to 7. We ran our miner using a window size of $\rho = 10$ and varied the support threshold σ. The results are given in Table 1.

When we are using a support threshold of 100, the only closed frequent patterns are the planted pattern and its subpatterns of form $(s_i s_{i+1}) \cdots (s_j s_{j+1})$, since the frequency of these subpatterns is slightly higher than the frequency of the whole pattern. The number of instance-closed episodes (given in the 4th column of Table 1) grows more rapidly. The reason for this is that the instance-closure focuses on the edges and does not add any new nodes. However, this ratio becomes a bottleneck only when we are dealing with extremely large serial episodes, and for our real-world datasets this ratio is relatively small.

The need for instance-closure becomes apparent when the number of instance-closed episodes is compared to the number of all possible general subepisodes (including those that are not tranistively closed) of a planted pattern, given in the 5th column of Table 1. We see that had we not used instance-closure, a single pattern having 6 nodes and 12 events renders pattern discovery infeasible.

Table 1: Results from synthetic sequences with a planted episode P. The columns show the number of nodes in P, the support threshold, the size of the final output, the number of instance-closed episodes, the number of subepisodes of P, and the number of sequence scans, respectively.

$\|V(P)\|$	σ	$\|\mathcal{C}\|$	i-closed	frequent	scans
1	100	1	3	3	46
2	100	3	15	27	200
3	100	6	63	729	744
4	100	10	255	59 049	3 964
5	100	15	1 023	14 348 907	11 123
6	100	21	4 095	$\approx 10^{10}$	48 237
7	100	28	16 383	$\approx 2 \times 10^{13}$	191 277
7	50	29	16 384	$> 2 \times 10^{13}$	191 243
7	40	32	16 387	$> 2 \times 10^{13}$	191 298
7	30	39	16 394	$> 2 \times 10^{13}$	191 463
7	20	127	16 488	$> 2 \times 10^{13}$	197 773
7	10	684	52 297	$> 2 \times 10^{13}$	480 517

As we lower the threshold, the number of instance-closed episodes and closed episodes increases, however the ratio between instance-closed and closed episodes improves. The reason for this is that the output contains episodes other than our planted episode, and those mostly contain a small number of nodes.

We also measured the number of sequence scans, namely the number of calls made to SUPPORT, to demonstrate how fast the negative border is growing. Since our miner is a depth-first search and the computation of frequency is based on instances, a single scan is fast, since we do not have to scan the whole sequence.

Our second set of experiments was conducted on real-world data. The dataset consists of alarms generated in a factory, and contains 514 502 events of 9 595 different types, stretching over 18 months. An entry in the dataset consists of a time stamp and an event type. Once again, we tested our algorithm at various thresholds, varying the window size from 3 to 15 minutes. Here, too, as shown in Table 2, we can see that the number of i-closed episodes does not explode to the level of all frequent episodes, demonstrating the need for using the i-closure as an intermediate step. Furthermore, we see that in a realistic setting, the number of i-closed episodes stays closer to the number of closed episodes than in the above-mentioned synthetic dataset. This is no surprise, since real datasets tend to have a lot of patterns containing a small number of events.

As the discovery of all frequent episodes is infeasible, we estimated their number as follows. An episode G has $a(G) = 3^{|PE(G)|}2^{|WE(G)|}$ subepisodes (including those that are not transitively closed) with the same events and nodes. From the discovered i-closed episodes we selected a subset $\mathcal{G}$ such that each $G \in \mathcal{G}$ has a unique set of events and a maximal $a(G)$. Then the lower bound for the total number of frequent episodes is $\sum_{G \in \mathcal{G}} a(G)$. Using such a lower bound is more than enough to confirm that the number of frequent episodes explodes much faster than the number of closed and i-closed episodes, as can be seen in Table 2.

Furthermore, our output contained a considerable number of episodes with simultaneous events — patterns that no existing method would have discovered. The runtimes ranged from just under 2 seconds to 90 seconds for the lowest reported thresholds.

Table 2: Results from the alarms dataset.

win (s)	$\sigma/10^3$	$\|\mathcal{C}\|$	i-closed	freq.(est)	scans
180	500	6	6	6	194
180	400	8	8	8	220
180	300	12	12	12	282
180	240	23	26	26	792
600	2 000	4	4	4	128
600	1 000	24	27	39	374
600	500	90	137	493	1 196
600	280	422	698	2 321	8 758
900	2 000	24	26	40	350
900	1 000	52	58	94	745
900	500	280	426	1 997	4 604
900	350	1 845	9 484	190 990	63 735

Our third dataset consisted of trains delayed at a single railway station in Belgium. The dataset consists of actual departure times of delayed trains, coupled with train numbers, and contains 10 115 events involving 1 280 different train IDs, stretching over a period of one month. A window of 30 minutes was chosen by a domain expert. The time stamps were expressed in seconds, so a single train being delayed on a particular day would be found in 1800 windows. Therefore, the frequency threshold for interesting patterns had to be set relatively high. The results are shown in Table 3, and were similar to those of the alarm dataset. The runtimes ranged from a few milliseconds to 55 minutes for the lowest reported threshold. The largest discovered pattern was of size 10, and the total number of frequent episodes at the lowest threshold was at least 33 million, once again demonstrating the need for both outputting only closed episodes and using instance-closure.

Table 3: Results from the trains dataset.

σ	$\|\mathcal{C}\|$	i-closed	freq.(est)	scans
30 000	141	141	141	9 575
20 000	1 994	1 995	2 593	219 931
17 000	8 352	8 416	22 542	812 363
15 000	26 170	26 838	172 067	2 231 360
13 000	94 789	101 882	3 552 104	6 865 877
12 000	189 280	211 636	33 660 094	12 966 895

9. CONCLUSIONS

In this paper we introduce a new type of sequential pattern, a general episode that takes into account simultaneous events, and provide an efficient depth-first search algorithm for mining such patterns.

This problem setup has two major challenges. The first challenge is the pattern explosion which we tackle by discovering only closed episodes. Interestingly enough, we cannot define closure based on frequency, hence we define closure based on instances. While it holds that frequency-closed episodes are instance-closed, the opposite is not true. However, in practice, instance-closure reduces search space dramatically so we can mine all instance-closed episodes and discover frequency-closed episodes as a post-processing step.

The second challenge is to correctly compute the subset relationship between two episodes. We argue that using a subset relationship based on graphs is not optimal and will lead to redundant output. We define a subset relationship based on coverage and argue that this is the correct definition. This definition turns out to be **NP**-hard. However, this is not a problem since in practice most of the comparisons can be done efficiently.

10. ACKNOWLEDGMENTS

Nikolaj Tatti is supported by a Post-doctoral Fellowship of the Research Foundation – Flanders (FWO).

The authors wish to thank Toon Calders for providing the proof that checking whether a sequence covers an episode is **NP**-hard on a small piece of paper.

11. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, pages 487–499, 1994.

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

[3] 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)*, pages 83–92, 2007.

[4] 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*, pages 83–94, 2003.

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

[6] G. Chen, X. Wu, and X. Zhu. Sequential pattern mining in multiple streams. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 585–588, 2005.

[7] 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)*, pages 317–328, 2009.

[8] M. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):530–552, 2002.

[9] R. Gwadera, M. J. Atallah, and W. Szpankowski. Markov models for identification of significant episodes. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2005)*, pages 404–414, 2005.

[10] R. Gwadera, M. J. Atallah, and W. Szpankowski. Reliable detection of episodes in event sequences. *Knowledge and Information Systems*, 7(4):415–437, 2005.

[11] R. Gwadera and F. Crestani. Discovering significant patterns in multi-stream sequences. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008)*, pages 827–832, 2008.

[12] S. K. Harms, J. S. Deogun, J. Saquer, and T. Tadesse. Discovering representative episodal association rules from event sequences using frequent closed episode sets and event constraints. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2001)*, pages 603–606, 2001.

[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)*, pages 410–419, 2007.

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

[15] 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*, pages 313–324, 2004.

[16] T. Oates and P. R. Cohen. Searching for structure in multiple streams data. In *Proceedings of the 13th International Conference on Machine Learning (ICML 1996)*, pages 346–354, 1996.

[17] 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*, 18(11):1467–1481, 2006.

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

[19] N. Tatti and B. Cule. Mining closed strict episodes. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM 2010)*, 2010.

[20] 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)*, pages 347–354, 2003.

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

[22] 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*, 23(2):115–125, 1994.

[23] 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)*, pages 166–177, 2003.

[24] 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)*, pages 310–318, 2010.

12. PROOFS OF THEOREMS

PROOF OF THEOREM 8. If s covers G, then the map f mapping the nodes of G to indices of s provides the certificate needed for the verification. Hence, testing the coverage is in **NP**.

In order to prove the completeness we reduce 3SAT to the coverage. In order to do so, given the formula F, we will build an episode G and a sequence such that sequence s covers G if and only if F is satisfiable.

Assume that we are given a formula F with M variables and L clauses. We define the alphabet for the labels to be $\Sigma = \{\alpha_1, \ldots, \alpha_M\} \cup \{\beta_1, \ldots, \beta_L\}$, where α_i is identified with the ith variable and β_j is identified with jth clause.

We will now construct G. The nodes in the episode consist of three groups. The first group, $P = \{p_1, \ldots, p_M\}$, contains M nodes. A node p_i is labelled as $lab(p_i) = \alpha_i$. These nodes represent the positive instantiation of the variables. The second group of nodes, $N = \{n_1, \ldots, n_M\}$, also contains M nodes, and the labels are again $lab(n_i) = \alpha_i$. These nodes represent the negative instantiation of the variables. Our final group is $C = \{c_1, \ldots, c_{3L}\}$, contains $3L$ nodes, 3 nodes for each clause. The labels for these nodes are $lab(c_{3j-2}) = lab(c_{3j-1}) = lab(c_{3j}) = \beta_j$. The edges of the episode G are as follows: Let β_j be the jth clause in F and let α_i be the kth variable occurring in that clause. Note that $k = 1, 2, 3$. We connect p_i to c_{3j+k-3} if the variable is positive in the clause. Otherwise, we connect n_i to c_{3j+k-3}.

The sequence s consists of 5 consecutive subsequences $s = s_1 s_2 s_3 s_4 s_5$. We define $s_1 = s_3 = \alpha_1 \cdots \alpha_M$ and $s_2 = s_4 = s_5 = \beta_1 \cdots \beta_L$, that is, s is equal to

$$\alpha_1 \cdots \alpha_M \beta_1 \cdots \beta_L \alpha_1 \cdots \alpha_M \beta_1 \cdots \beta_L \beta_1 \cdots \beta_L .$$

Our final step is to prove that s covers G whenever F is satisfiable. First assume that F is satisfiable and let t_i be the truth assignment for the variable α_i. We need to define a mapping f. If t_i is true, then we map p_i into the first group s_1 and n_i into the third group s_3. If t_i is false, then we map n_i into s_1 and p_i into s_3. Since each clause is now satisfied, among the nodes c_{3j-2}, c_{3j-1}, and c_{3j} there is at least one node, say c_k, such that the parent of that node (p_i or n_i) is mapped to the first group s_1. We can map c_k into the second group s_2. The remaining two nodes are mapped into the fourth and the fifth group, s_4 and s_5. Clearly this mapping is valid since all the nodes are mapped and the edges are honoured.

To prove the other direction, let f be a valid mapping of G into s. Since the sequence has the same amount of symbols as there are nodes in the graph the mapping f is surjective. Define a truth mapping by setting the ith variable to true if p_i occurs in s_1, and false otherwise. Each symbol in the second group s_2 is covered. Select one symbol, say β_j from that group. The corresponding node, say c_k, has a parent (either p_i or n_i) that must be mapped into the first group. This implies that the truth value for the ith variable satisfies the jth clause. Since all clauses are satisfied, F is satisfied. This completes the proof. □

PROOF OF THEOREM 16. To prove the theorem we will use the following straightforward lemma.

LEMMA 22 *Let G be a transitively closed episode. Let e and f be two nodes. Then there exists a sequence s covering G and a mapping m such that*

1. *if $nd(e) \neq nd(f)$, then $ts(m(e)) \neq ts(m(f))$,*
2. *if $nd(e)$ is not a child of $nd(f)$, then $ts(m(e)) < ts(m(f))$,*
3. *if $nd(e)$ is not a proper child of $nd(f)$, then $ts(m(e)) \leq ts(m(f))$,*

The 'if' part is straightforward so we only prove the 'only if' part. Assume that $G \preceq H$ and let e and f be two events in G violating one of the conditions. Apply Lemma 22 with H, $\pi(e)$, and $\pi(f)$ to obtain a sequence s and a mapping m. We can safely assume that m is surjective. Define $m'(x) = m(\pi(x))$ for an event x in G. Since every label is unique in s, m' is the only mapping that will honor the labels in G. Now Lemma 22 implies that m' is not a valid mapping for G, hence $G \not\preceq H$, which proves the theorem. □

PROOF OF THEOREM 20. Assume that the condition in the theorem holds and let s be a sequence covering H and let m be the corresponding mapping. We can safely assume that m is surjective. Let t be the smallest time stamp in s and E the events in s having the time stamp t. Let V be the corresponding prefix subgraph,

$$V = \{n \in V(G) \mid ts(m(n)) = t\} .$$

By assumption, we have $tail(\mathcal{G}, lab(V)) \preceq H - V$, hence there is $G \in \mathcal{G}$ and $W \in pre(G, L)$ such that $G - W \preceq H - V$ and $lab(W) \subseteq lab(V)$. Let s' be the sequence obtained from s by removing E. Since s' covers $H - V$, there is a map m' mapping $G - W$ to s'. We can extend this map to G by mapping W to E. Thus s covers G, hence s covers $\mathcal{G}$, and by definition $\mathcal{G} \preceq H$.

To prove the other direction, assume that $\mathcal{G} \preceq H$. Let V be a prefix subgraph of H. Let s' be a sequence covering $H - V$. We can extend this sequence to s by adding the events with labels $lab(V)$ in front of s'. Let us denote these events by E. Sequence s covers H, hence it covers $\mathcal{G}$.

By assumption, there is a mapping m' from G to s for some $G \in \mathcal{G}$. Let W be the (possibly empty) prefix graph of G mapping to E. We can assume that $W \in pre(G, lab(V))$, that is, W is maximal, otherwise let $W' \in pre(G, lab(V))$ such that $W \subset W'$. We can modify m' by remapping the nodes in $W' - W$ to the events in E. This will make W equal to W'. Since s' covers $G - W$, it follows that s' covers $tail(\mathcal{G}, lab(V))$ which proves the theorem. □

PROOF OF LEMMA 21. The 'if' part is trivial. To prove the 'only if' part, assume that $G \preceq H$. Let s' be a sequence covering H' and let m' be the mapping. If we can show that s' can be extended to s by adding events with the labels $lab(n)$ such that s covers G, then the corresponding mapping m will also be a valid mapping from G to s', which will prove the lemma.

If n is a source or a sink (a node without children), then we can extend s' by adding the event with the label $lab(n)$ either at the beginning or at the end of s'.

Assume that n has parents and children. Let t_1 be the largest time stamp of the parents of n and let t_2 be the smallest time stamp of the children of n. Extend s' to s by adding an event f with the label $lab(n)$ and the time stamp $t = 1/2(t_1 + t_2)$. Let x be a child of n and y a parent of n. Since H is transitively closed, x is a child of y in H'. Since this holds for any x and y, we have $t_1 \leq t_2$. Hence $ts(y) \leq t \leq ts(x)$. If x is proper descendant of n, then it is also a proper descendant of any parent of n, hence

$t_1 < ts(x)$ and consequently $t < ts(x)$. The same holds for y. This implies that s covers G. $\square$