Data Min Knowl Disc (2012) 25:34–66
DOI 10.1007/s10618-011-0232-z

Mining closed strict episodes

Nikolaj Tatti · Boris Cule

Received: 7 December 2010 / Accepted: 3 August 2011 / Published online: 1 September 2011

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 the 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 properties 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

Responsible editor: M. J. Zaki.

The research described in this paper builds upon and extends the work appearing in Proceedings of Tenth IEEE International Conference on Data Mining (ICDM 2010), 2010 (Tatti and Cule 2010).

N. Tatti (✉) · B. Cule
University of Antwerp, Antwerp, Belgium
e-mail: nikolaj.tatti@ua.ac.be

B. Cule
e-mail: boris.cule@ua.ac.be

1 Introduction

Discovering frequent patterns in an event sequence is an important field in data mining. Episodes, as defined in Mannila et al. (1997), represent a rich class of sequential patterns, enabling us to discover events occurring in the vicinity of each other while at the same time capturing complex interactions between the events.

More specifically, a frequent episode is traditionally 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 are allowed to occur is specified by the episode. Frequency, the number of windows in which the episode occurs, is monotonically decreasing so we can use the well-known 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 Sect. 2). 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.

The advantage of episodes based on DAGs is that they allow us to capture dependencies between the events while not being too restrictive.

Example 1 As an example we will use text data, namely inaugural speeches by presidents of the US (see Sect. 8 for more details). Protocol requires the presidents to address the chief justice and the vice presidents in their speeches. Hence, a pattern

$$\text{chief} \rightarrow \text{justic} \qquad \text{vice} \rightarrow \text{president}$$

occurs in 10 disjoint windows. This pattern captures the phrases 'chief justice' and 'vice president' but because the address order varies from speech to speech, the pattern does not impose any additional restrictions.

Episodes based on DAGs have, in practice, been over-shadowed by parallel and serial episodes, despite being defined at the same time (Mannila et al. 1997). The main reason for this is the pattern explosion demonstrated in the following example.

Example 2 To illustrate the pattern explosion we will again use inaugural speeches by presidents of the US. By setting the window size to 15 and the frequency threshold to 60 we discovered a serial episode with six symbols,

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

Table 1 Illustration of the pattern explosion

Pattern	1	2	3	4	5	6	7	8	9
Itemsets	1	3	7	15	31	63	127	255	511
Episodes	1	4	16	84	652	7,742	139,387	3,730,216	145,605,024

The first row is the number of frequent itemsets produced by a single frequent itemset with n items. The second row is the number of episodes produced by a single frequent serial episode with n unique labels. These numbers were obtained by a brute force enumeration

In total, we found another 4,823 subepisodes of size 6 of this episode. However, all these episodes had only three distinct frequencies, indicating that the frequencies of most of them could be derived from the frequencies of only three episodes, so the output could be reduced by leaving out 4,821 episodes.

We illustrate the pattern explosion further in Table 1. We see from the table that if the sequence has a frequent serial episode consisting of nine labels, then mining frequent episodes will produce at least 100 million patterns.

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 superpattern with the same frequency. Mining closed patterns has been shown to reduce the output. Moreover, we can discover closed patterns efficiently. However, adopting the concept of closedness to episodes is not without problems.

Subset relationship Establishing a proper subset relationship is needed for two reasons. Firstly, to make the mining procedure more efficient by discovering all possible subpatterns before testing the actual episode, and secondly, to define a proper closure operator.

A naïve approach to define whether an episode G is a subepisode of an episode H is to compare their DAGs. This, however, leads to problems as the same episode can be represented by multiple DAGs and a graph representing G is not necessarily a subgraph of a graph representing H as demonstrated in the following example.

Example 3 Consider episodes G_1, G_2, and G_3 given in Fig. 1. Episode G_1 states that for a pattern to occur a must precede b and c. Meanwhile, G_2 and G_3 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.

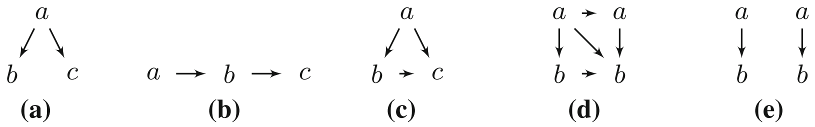

Fig. 1 Toy episodes used in Example 3. **a** Episode G_1, **b** Episode G_2, **c** Episode G_3, **d** Episode H_1 and **e** Episode H_2

We solve this particular case by considering transitive closures, graphs in which each node must be connected to all its descendants by an edge, 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. Using transitive closure does not solve all problems for episodes containing multiple nodes with the same label. For example, episodes H_1 and H_2 in Fig. 1 are the same even though their graphs are different.

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 4 Consider sequence s given in Fig. 2e and episode G_1 given in Fig. 2a. Assume that we use a sliding window of size 5. There are two windows that cover episode G_1, namely $s[1, 5]$ and $s[6, 10]$, illustrated in Fig. 2e. Hence, the frequency of G_1 is 2. There are *two* serial episodes that are more specific than G_1 and have the same frequency, namely, G_2 and G_3 given in Fig. 2. Moreover, there is no superepisode of G_2 and G_3 that has frequency equal to 2. In other words, we cannot define a unique closure for G_1 based on frequency.

The contributions of our paper address these issues:

1. We introduce *strict* episodes, a new subclass of general episodes. We say that an episode is strict if all nodes with the same label are connected. Thus all episodes in Fig. 1 are strict, except H_2. We will argue that this class is large, contains all serial and parallel episodes, as well as episodes with unique labels, yet using only strict episodes eases the computational burden.
2. We introduce a natural subset relationship between episodes based on the subset relationship of sequences covering the episodes. We will prove that for strict episodes this relationship corresponds to the subset relationship between transitively closed graphs. For strict episodes such a graph uniquely defines the episode.
3. We introduce milder versions of the closure concept, including the *instance-closure*. We will show that these closures can be used efficiently, and that a

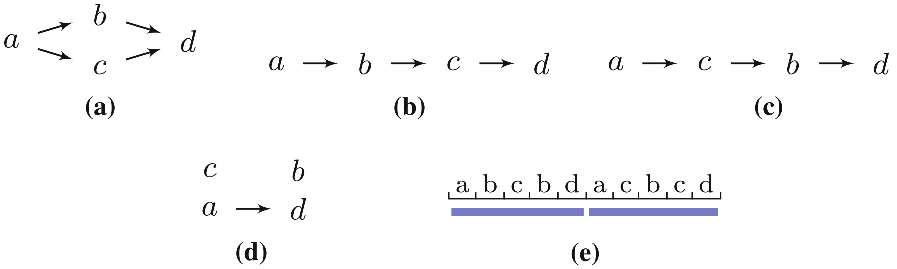

Fig. 2 Toy episodes used in Examples 4 and 5. Edges induced by transitive closure are omitted to avoid clutter. **a** Episode G_1, **b** Episode G_2, **c** Episode G_3, **d** Episode G_4 and **e** Sequence s

frequency-closed episode is always instance-closed.[1] We demonstrate that computing closure and frequency can be done in polynomial time.[2]

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.

2 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, or *events*, coming from an *alphabet* Σ, so that for each i, $s_i \in \Sigma$. Given a strictly increasing mapping $m : [1, N] \rightarrow [1, L]$ we will define s_m to be a subsequence $s_{m(1)} \ldots s_{m(N)}$. Similarly, given two integers $1 \leq a \leq b \leq L$ we define $s[a, b] = s_a \ldots s_b$.

An *episode* G is represented by a DAG with labelled nodes, that is, $G = (V, E, \mathit{lab})$, where $V = (v_1, \ldots, v_K)$ is the set of nodes, E is the set of directed edges, and lab is the function $\mathit{lab} : V \rightarrow \Sigma$, 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, or G *occurs* in s, 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, $s_{f(v_i)} = \mathit{lab}(v_i)$, 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. For an example, see Fig. 3a. If the mapping f is surjective, that is, all events in s are used, we will say that s is an *instance* of G.

2.1 Frequency

In our search for frequent episodes, we will use and compare two conceptually different definitions of frequency.

Traditionally, 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.

Definition 1 The *fixed-window frequency* of an episode G in a sequence s, denoted $\mathit{fr}_f(G)$, is defined as the number of time windows of a given size ρ within s, in which the episode G occurs. Formally,

$$\mathit{fr}_f(G) = |\{\, (a, b) \mid b = a + \rho - 1, a \leq N, b \geq 1, \text{ and } s[a, b] \text{ covers } G \,\}|.$$

[1] In Tatti and Cule (2010), the closure was based only on adding edges whereas in this version we are also adding nodes.

[2] This was not guaranteed in Tatti and Cule (2010).

See Fig. 3b for example.

The frequency of an episode is sometimes expressed using the number of minimal windows that contain it. To satisfy the downward-closed property, we say that these windows must be non-overlapping.

Definition 2 The *disjoint-window frequency* of an episode G in a sequence s, denoted $fr_d(G)$, is defined as the maximal number of non-overlapping windows within s that contain episode G. Formally,

$$fr_d(G) = \max\left\{ \left|\{(a_1, b_1), \ldots, (a_N, b_N)\}\right| \;\middle|\; \begin{array}{l} s[a_i, b_i] \text{ covers } G, b_i - a_i < \rho \text{ and} \\ [a_i, b_i] \cap [a_j, b_j] = \emptyset \text{ for } 1 \le i, j \le N \end{array} \right\}.$$

See Fig. 3c for example.

We now establish a connection between the disjoint-window frequency and actual minimal windows.

Definition 3 Given a sequence s and an episode G, a window $s[a, b]$ is called a *minimal window* of G in s, if $s[a, b]$ covers G, and if no proper subwindow of $s[a, b]$ covers G. We will also refer to the interval $[a, b]$ as a minimal window, if s is clear from the context.

It is easy to see that the maximal number of non-overlapping windows within s that contain G is equal to the maximal number of non-overlapping minimal windows within s that contain G.

Whenever it does not matter whether we are dealing with the fixed-window frequency or the disjoint-window frequency, we simply denote $fr(G)$.

3 Strict episodes

In this section we will define our mining problem and give a rough outline of the discovery algorithm.

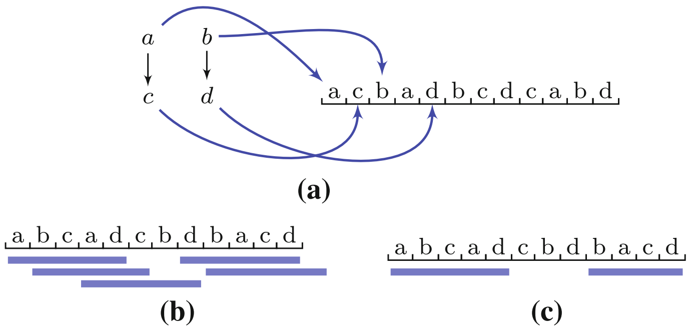

Fig. 3 A toy example illustrating different support measures. **a** contains an example of a sequence covering an episode. **b** shows all 5 sliding windows of length 5 containing the episode. **c** shows the maximal number, 2, of non-overlapping windows covering the episode

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.

Definition 4 Assume two transitively closed episodes G and H with the same number of nodes. An episode G is called a *subepisode* 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 the set of all sequences that cover H is a proper subset of the set of all sequences that cover G, we call G a *proper subepisode* of H, denoted $G \prec H$.

For a more general case, assume that $|V(G)| < |V(H)|$. We say that G is a subepisode of H, denoted $G \preceq H$, if there is a subgraph H' of H such that $G \preceq H'$. Moreover, let α be a graph homomorphism from H' to H. If we wish to emphasize α, we write $G \preceq_\alpha H$.

If $|V(G)| > |V(H)|$, then G is automatically not a subepisode of 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 5 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.

We will show later that the subset relationship can be computed efficiently for strict episodes. However, as can be seen in Fig. 4, there are episodes that are not strict. Our algorithm will not discover these types of patterns.

Having defined a subset relationship, we can now define an f-closed episode.

Definition 6 An episode G is *frequency-closed*, or *f-closed*, if there exists no episode H, such that $G \prec H$ and $fr(G) = fr(H)$.

Problem 1 Given a sequence s, a frequency measure, either fixed-window or disjoint-window, and a threshold σ, find all f-closed strict episodes from s having the frequency higher or equal than σ.

A traditional approach to discovering closed patterns is to discover generators, that is, for each closed pattern P, discover minimal patterns whose closure is equal to P (Pasquier et al. 1999). When patterns are itemsets, it holds that the collection of frequent generators are downward closed. Hence, they can be mined efficiently using a BFS-style approach.

We cannot directly apply this framework for two reasons: Firstly, unlike with itemsets, we cannot define a closure based on frequency. We solve this by defining an

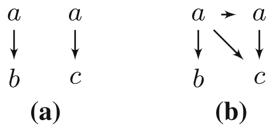

Fig. 4 An example of a non-strict and a strict episode. **a** non-strict and **b** strict

instance-closure, a more conservative closure that guarantees that all f-closed episodes are discovered. Once instance-closed episodes are discovered, the f-closed episodes are selected in a post-processing step. The second obstacle is the fact that the collection of generator episodes is not necessarily downward-closed. We solve this problem by additionally using some intermediate episodes that will guarantee the correctness of the algorithm.

A sketch of the miner is given in Algorithm 1 (the details of the algorithm are described in subsequent sections). The algorithm consists of two loops. In the outer loop, Lines 3–15, we discover parallel episodes by adding nodes. In the inner loop, Lines 5–12, we discover general episodes by adding edges. Each candidate episode is tested, and if the candidate is frequent and a generator, then the episodes is added into the collection. Finally, we discover the f-closed episodes as a last step.

Algorithm 1: Rough outline of the breath-first mining algorithm. The details of each step are given in Sects. 4–7

```
   input  : sequence $s$, threshold $\sigma$, window size $\rho$
   output : frequent $f$-closed episodes
 1 $\mathcal{C} \leftarrow$ all frequent episodes with 1 node;
 2 $\mathcal{E} \leftarrow \emptyset$; $N \leftarrow 1$;
 3 while $\mathcal{C} \neq \emptyset$ do
 4     $M \leftarrow 0$;
 5     while $\mathcal{C} \neq \emptyset$ do
 6         foreach $G \in \mathcal{C}$ do
 7             if $G$ is a frequent generator w.r.t. $i$-closure then
 8                 add $G$ to $\mathcal{E}$;
 9                 add intermediate episodes;
10         $\mathcal{P} \leftarrow$ episodes with $N$ nodes and $M$ edges from $\mathcal{E}$;
11         $\mathcal{C} \leftarrow$ candidates generated from $\mathcal{P}$ with $N$ nodes and $M + 1$ edges;
12         $M \leftarrow M + 1$;
13     $\mathcal{P} \leftarrow$ parallel episodes with $N$ nodes from $\mathcal{E}$;
14     $\mathcal{C} \leftarrow$ parallel candidates generated from $\mathcal{P}$ with $N + 1$ nodes;
15     $N \leftarrow N + 1$;
16 return $f$-closed episodes from the $i$-closures of episodes in $\mathcal{E}$;
```

To complete the algorithm we need to solve several subproblems:

1. computing the subset relationship efficiently (Sect. 4)
2. defining and computing instance-closure (Sect. 5)
3. generating candidate episodes, Line 11 (Sect. 6)
4. generating intermediate episodes and proving the correctness (Sect. 7)

4 Computing the subset relationship

In this section we will demonstrate that computing the subset relationship for strict episodes can be done efficiently. This allows us to build an algorithm to efficiently discover closed episodes.

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 7 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.

Definition 8 Let $\mathcal{S}$ be the space of all strict and transitively closed episodes.

In the remaining text, we consider episodes to be transitively closed and strict, unless stated otherwise. An episode and its transitive closure will always have the same frequency, hence by restricting ourselves to transitively closed episodes we will not lose any episodes.

For notational simplicity, we now introduce the concept of two episodes having *identical nodes*. Given an episode $G \in \mathcal{S}$ 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 $lab(v_i) < lab\left(v_j\right)$ lexicographically, or $lab(v_i) = lab\left(v_j\right)$ and v_i is an ancestor of v_j with respect to $E(G)$ (i.e. edge $(v_i, v_j) \in E(G)$). 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 $lab(v_i) = lab(w_i)$ for $i = \{1, \ldots, N\}$. To simplify notation, we often identify v_i and w_i. This convention allows us to write statements such as $E(G) \cup E(H)$, if G and H have identical nodes.

Our next step is to show how we can test the subset relationship for strict episodes.

Lemma 1 *Let $G, H \in \mathcal{S}$ be episodes with identical nodes. Let s be a valid instance of both G and H. Let g and h be the corresponding functions mapping nodes of G and H to indices of s, respectively. Then $g = h$.*

Proof Let v be a node. Function g maps v to the lth occurrence of $lab(v)$ in s. Since s is an instance, then there are $l - 1$ ancestors of v in G having the same label as v. Since G and H have identical nodes, v also has $l - 1$ ancestors in H. Since s is an instance of H, h must map v to lth occurrence of $lab(v)$. This implies that $g = h$. □

Lemma 1 implies that given an episode G and an instance s, there is only one valid function f mapping nodes of G to indices of s. Let us denote this mapping by $map(G, s) = f$. If G is a parallel episode with nodes $V = V(G)$ we write $map(V, s)$.

Crucially, we can easily compute the subset relationship between two episodes.

Theorem 1 *For episodes $G, H \in \mathcal{S}$ 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 an instance of H and let $f = map(H, s)$ be the corresponding mapping. Then f is also a valid mapping for G. Thus, $G \preceq H$.

To prove the other direction, assume that $E(G) \not\subseteq E(H)$. We therefore must have an edge $e = (x, y) \in E(G)$, such that $e \notin E(H)$. 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. Let h be the visiting order of G while constructing s, that is, $h(v) = 1$, if we visited v first, $h(v) = 2$, if we visited v second. Note that $h(y) < h(x)$. Assume now that s covers G and let $f = map(G, s)$ be the corresponding mapping. But then Lemma 1 implies that $g = h$, thus $g(y) < g(x)$, contradicting the fact that $(x, y) \in E(G)$. □

Theorem 1 essentially shows that our subset relationship is in fact a graph subset relationship which allows us to design an efficient mining algorithm.

We finish this section by defining what we exactly mean when we say that two episodes are equivalent and demonstrate that the class of strict episodes contains all parallel episodes.

Definition 9 Episodes G and H are said to be equivalent, denoted by $G \sim H$, if each sequence that covers G also covers H, and vice versa.

Corollary 1 (of Theorem 1) *For episodes $G, H \in \mathcal{S}$, $G \sim H$ if and only if $E(G) = E(H)$ and G and H have identical nodes.*

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.

5 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 establish three closure operators, based on the instances of the episode found within the sequence. The first closure adds nodes, the second one adds edges, and the third is a combination of the first two. We also show that these operators satisfy three important properties. We will use these properties to prove the correctness of our mining algorithm. The needed properties for a closure operator h are

1. *Extension:* $G \preceq h(G)$,
2. *Idempotency:* $h(G) = h(h(G))$,
3. *Monotonicity:* $G_1 \preceq G_2 \Rightarrow h(G_1) \preceq h(G_2)$.

These properties are usually shown using the Galois connection but to avoid cumbersome notation we will prove them directly.

5.1 Node closure

In this section we will define a node closure and show that it satisfies the properties. Assume that we are given a sequence s and a window size ρ.

Our first step is to define a function f_N which maps an episode $G \in \mathcal{S}$ to a set of intervals which contain all instances of G,

$$f_N(G; s) = \{ [\min m, \max m] \mid s_m \text{ covers } G,\ \max m - \min m < \rho, m \in M \},$$

where M contains all strictly increasing mappings to s.

Our next step is to define X_G to be the set of all symbols occurring in each interval,

$$X_G = \{ x \in \Sigma \mid x \text{ occurs in } s[a, b] \text{ for all } [a, b] \in f_N(G) \}.$$

Let W be the labels of the nodes of G. We define our first closure operator, $icl_N(G)$ to be G augmented with nodes having the labels $X_G - W$, that is, we add nodes to G with labels that occur inside each window that contains G.

Theorem 2 *$icl_N(G)$ is an idempotentic and monotonic extension operator.*

Proof The extension property follows immediately because we are only adding new nodes.

Assume now that $G \preceq H$. Let $[a, b] \in f_N(H)$ be an interval. Then, there is an interval $[c, d] \in f_N(G)$ such that $a \leq c \leq d \leq b$. This means that any symbol occurring in every interval in $f_N(G)$ will also occur in every interval in $f_N(H)$, that is, $X_G \subseteq X_H$.

Let α be a graph homomorphism such that $G \preceq_\alpha H$. Let $x \in X_G$ be a symbol not occurring in G and let v be the new node in $icl_N(G)$ with this label. If x does not occur in H, then $x \in X_H$ and thus a node with a label x is added into H. In any case, there is a node w with a label x in $icl_N(H)$. We can extend α by setting $\alpha(v) = w$. By doing this for each new node we have proved monotonicity, i.e. that $icl_N(G) \preceq icl_N(H)$.

To prove idempotency, let us write $H = icl_N(G)$. Since any new node in H must occur inside the instances of G, we have $f_N(G) \subseteq f_N(H)$. This implies that $X_H \subseteq X_G$ and since we saw before that $X_G \subseteq X_H$, it implies that $X_G = X_H$. Since for every label in X_H there is a node in H with the same label, it holds that $icl_N(H) = H$. □

Note that the node closure adds only events with unique labels. The reason for this is that if we add node x to an episode containing node y such that $lab(x) = lab(y)$, then we would have to connect x and y. This may reduce the instances and invalidate the proof. For the same reason, we only add a maximum of one new node with a particular label. In other words, if each window containing episode G also contains two occurrences of a, we will only add one node with label a to G (provided G does not contain a node labelled a already).

5.2 Edge closure

We begin by introducing the concept of a maximal episode that is covered by a given set of sequences.

Definition 10 Given a set of nodes V, and a set S of instances of V, interpreted as a parallel episode, we define the *maximal episode* covered by set S as the episode H, where $V(H) = V$ and

$$E(H) = \{\, (x, y) \in V \times V \mid f(x) < f(y),\, f = map(V, s) \quad \text{for all } s \in S \,\},$$

where $map(V, s)$ refers to the mapping defined in Lemma 1 and V is interpreted as a parallel episode.

To define a closure operator we first define a function mapping an episode G to all of its valid instances in a sequence s,

$$f_E(G; s) = \{\, s_m \mid s_m \ \text{ covers } G,\ \max m - \min m < \rho, m \in M \,\},$$

where M contains all strictly increasing mappings to s. When the sequence is known from the context, we denote simply $f_E(G)$

We define $icl_E(G)$ to be the maximal episode covered by $f_E(G)$. If $icl_E(G) = G$, then we call G an e-closed episode.

Theorem 3 *$icl_E(G)$ is an idempotentic and monotonic extension operator.*

Proof To prove the extension property assume an edge $(v_i, v_j) \in E(G)$. Let V be the nodes in G. Let $w \in f_E(G)$ be an instance of G and let $f = map(V, w)$ be the corresponding mapping. Lemma 1 implies that $map(V, w) = map(G, w)$. Hence $f(v_i) < f(v_j)$. Since this holds for every map, we have $(v_i, v_j) \in E(icl_E(G))$.

To prove the idempotency, let $H = icl_E(G)$. The extension property implies that $G \preceq H$ so by definition $f_E(H) \subseteq f_E(G)$. But any instance in $f_E(G)$ also covers H. Thus, $f_E(H) \subseteq f_E(G)$ and so $f_E(H) = f_E(G)$. This implies the idempotency.

Assume now that $G \preceq H$. Let α be the graph homomorphism such that $G \preceq_\alpha H$. We will show that $icl_E(G) \preceq_\alpha icl_E(H)$. Let $(x, y) \in E(icl_E(G))$. Let w be an instance of H and let $f = map(H, s)$ the corresponding mapping to w. Assume that $f(\alpha(x)) \geq f(\alpha(y))$. Let v be the subsequence of w containing only the indices in the range of $f \circ \alpha$. Note that v is a valid instance of G and $f \circ \alpha = map(V(G), v)$. This contradicts the fact that $(x, y) \in E(icl_E(G))$. Hence, $f(\alpha(x)) < f(\alpha(y))$. This implies that $(\alpha(x), \alpha(y)) \in E(icl_E(H))$ which completes the proof. □

Example 5 Consider sequence s given in Fig. 2e and episode G_4 given in Fig. 2d and assume that the window length is 5. There are four instances of G_4 in s, namely $abcd$, $acdb$, $acbd$ and $abcd$. Therefore, $f_E(G_4) = \{abcd, acbd\}$. The serial episodes corresponding to these subsequences are G_2 and G_3 given in Fig. 2. By taking the intersection of these two episodes we obtain $G_1 = icl_E(G_4)$ given in Fig. 2a.

5.3 Combining closures

We can combine the node closure and the edge closure into one operator.

Definition 11 Given an episode $G \in \mathcal{S}$, we define $icl_{EN}(G) = icl_E(icl_N(G))$. To simplify the notation, we will refer to $icl_{EN}(G)$ as $icl(G)$, the *i-closure* of G. We will say that G is *i-closed* if $G = icl(G)$.

Theorem 4 *$icl(G)$ is an idempotentic and monotonic extension operator.*

Proof The extension and monotonicity properties follow directly from the fact that both $icl_N(G)$ and $icl_E(G)$ are monotonic extension operators.

To prove idempotency let $H = icl(G)$ and $H' = icl_N(G)$. Since any instance of $H = icl_E\left(H'\right)$ is also an instance of H', and, per definition, vice versa, we see that $f_N(H) = f_N(H')$, and consequently $icl_N(H) = H$, and $icl(H) = icl_E(icl_N(H)) = icl_E(H) = H$. □

The advantage of mining i-closed episodes instead of e-closed is prominent if the sequence contains a long sequential pattern. More specifically, assume that the input sequence contains a frequent subsequence of N symbols $p_1, \ldots, p_N$, and no other permutation of this pattern occurs in the sequence. The number of e-closed subpatterns of this subsequence is $2^N - 1$, namely, all non-empty serial subepisodes. However, the number of its i-closed subpatterns is $N(N+1)/2$, namely, serial episodes of form $p_i \rightarrow \cdots \rightarrow p_j$ for $1 \leq i \leq j \leq N$.

5.4 Computing closures

During the mining process, we need to compute the closure of an episode. The definition of closures use $f_N(G)$ and $f_E(G)$ which are based on instances of G in s. However, there can be an exponential number of such instances in s.

In the following discussion we will often use the following notations. Given an episode G, we write $G + v$ to mean the episode G augmented with an additional node v. Similarly, we will use the notations $G + e$ and $G + V$, where e is an edge and V is a set of nodes. We also use $G - v$, $G - V$, and $G - e$ to mean episodes where either nodes or edges are removed.

To avoid the problem of an exponential number of instances we make two observations: Firstly, a node with a new label l is added into the closure if and only if it occurs in every *minimal window* of G. Secondly, an edge (x, y) is added into the closure if and only if there is no minimal window for $G + (y, x)$. Thus to compute the closure we need an algorithm that finds all minimal windows of G in s. Note that, unlike with instances of G, there can be only $|s|$ minimal windows.

Using strict episodes allows us to discover minimal windows in an efficient greedy fashion.

Lemma 2 *Given an episode $G \in \mathcal{S}$ and a sequence $s = s_1 \dots s_L$, let k be the smallest index such that $s_k = lab(v)$, where v is a source node in G. Then s covers G if and only if $s[k+1, L]$ covers $G - v$.*

Proof Let f be a mapping from $V(G)$ to s. If k is not already used by f, then we can remap a source node v to k. As k is the smallest entry used by f, the remaining map is a valid mapping for $G - v$ in $s[k+1, L]$. □

Lemma 2 says that to test whether a sequence s covers G, it is sufficient to greedily find entries from s corresponding to the sources of G, removing those sources as we move along. Let us denote such a mapping by $g(G; s)$. Note that if the episode is not strict, then we can have two source nodes with the same label, in which case it might be that Lemma 2 holds for one of the sources but not for the other. Since we cannot know in advance which node to choose, this ambiguity would make the greedy approach less efficient.

Lemma 3 *Let $G \in \mathcal{S}$ be an episode and let s be a sequence covering G. Let $m = g(G; s)$ and let $[a, b]$ be the first minimal window of G in s. Then $\max m = b$.*

Proof Since $s[1, b]$ covers G, there is a mapping $m' = g(G; s[1, b])$. Since $[a, b]$ is the first minimal window, we must have $\max m' = b$. Since m and m' are constructed in a greedy fashion, we must have $m = m'$. □

Lemma 3 states that if we evaluate $m = g(G; s[k, L])$ for each $k = 1, \dots, K$, store the intervals $W = [\min m, \max m]$, and for each pair of windows $[a_1, b], [a_2, b] \in W$ remove $[a_2, b]$, then W will contain the minimal windows. Evaluating $g(G; s)$ can be done in polynomial time, so this approach takes polynomial time. The algorithm for discovering minimal windows is given in Algorithm 2. To fully describe the algorithm we need the following definition.

Definition 12 An edge (v, w) in an episode $G \in \mathcal{S}$ 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*.

Theorem 5 *Algorithm* FindWindows *discovers all minimal windows.*

Proof We will prove the correctness of the algorithm in several steps. First, we will show that the loop 8–16 guarantees that f is truly a valid mapping. To see this, note that the loop upholds the following invariants: For every $v \notin Q$, we have $f(v) > b(v)$ and for each skeleton edge (v, w), we have $b(w) \neq -\infty$. Also, for every non-source node w, $b(w) = \max \{ f(v) \mid (v, w) \text{ is a skeleton edge} \}$ or $b(w) = -\infty$.

Thus when we leave the loop, that is, $Q = \emptyset$, then f is a valid mapping for G. Moreover, since we select the smallest $f(v)$ during Line 11, we can show using recursion that $f = g(G; s[k+1, L])$, where $k = \min b(v)$, where the min ranges over all source nodes. Once f is discovered, the next new mapping should be contained in $s[\min f + 1, L]$. This is exactly what is done on Line 22 by making $b(v)$ equal to $f(v)$. □

Algorithm 2: FINDWINDOWS. An algorithm for finding minimal windows of G from s. The parameter ρ is the maximal size of the window

```
   input   : an episode $G \in \mathcal{S}$
   input   : set of minimal windows $W$
1  $W \leftarrow \emptyset$;
2  $Q \leftarrow \emptyset$;
3  foreach $v \in V(G)$ do
4      $f(v) \leftarrow$ first $i$ such that $s_i = lab(v)$;
5      $b(v) \leftarrow -\infty$;
6      add $v$ into $Q$;
7  while true do
       {Make sure that $f$ honors the edges in $G$}
8      while $Q$ is not empty do
9          $v \leftarrow$ first element in $Q$;
10         remove $v$ from $Q$;
11         $f(v) \leftarrow$ the smallest $i > b(v)$ such that $s_i = lab(v)$;
12         if $f(v) =$ null then return $W$;
13         foreach skeleton edge $(v, w) \in E(G)$ do
14             $b(w) \leftarrow \max(b(w), f(v))$;
15             if $b(w) \geq f(w)$ and $w \notin Q$ then
16                 add $w$ into $Q$;
17     if $\max f - \min f < \rho$ then
18         if $\max f = b$ such that $[a, b]$ is the last entry in $W$ then
19             delete the last entry from $W$;
20         add $[\min f, \max f]$ to $W$;
21     $v \leftarrow$ node with the smallest $f(v)$;
22     $b(v) \leftarrow f(v)$;
23     add $v$ to $Q$;
```

The advantage of this approach is that we can now optimize the search on Line 11 by starting the search from the previous index $f(v)$ instead of starting from the start of the sequence. The following lemma, implied directly by the greedy procedure, allows this.

Lemma 4 *Let $G \in \mathcal{S}$ be an episode and let s be a sequence covering G. Let k be an index and assume that $s[k, L]$ covers G. Set $m_1 = g(G; s)$ and $m_2 = g(G; s[k + 1, L])$. Then $m_2(v) \geq m_1(v)$ for every $v \in V(G)$.*

Let us now analyze the complexity of FINDWINDOWS. Assume that the input episode G has N nodes and the input sequence s has L symbols. Let K be the maximal number of nodes in G sharing the same label. Let D be the maximal number of outgoing skeleton edges in a single node. Note that each event in s is visited during Line 11 K times, at maximum. Each visit may require D updates of $b(w)$, at maximum. The only remaining non-trivial step is finding the minimal source (Line 21), which can be done with a heap in $O(\log N)$ time. This brings the total evaluation time to $O((D + 1)KL + L \log N)$. Note that for parallel episodes $D = 0$ and for serial episodes $D = 1$, and for such episodes we can further optimize the algorithm such that each event is visited only twice, thus making the running time to be in $O(L \log N)$ for parallel episodes and in $O(L)$ for serial episodes. Since there can be only L minimal windows, the additional space complexity for the algorithm is in $O(L)$.

Given the set of minimal windows, we can now compute the node closure by testing which nodes occur in all minimal windows. This can be done in $O(L)$ time. To compute the edge closure we need to perform N^2 calls of FindWindows. This can be greatly optimized by sieving multiple edges simultaneously based on valid mappings f discovered during FindWindows. In addition, we can compute both frequency measures from minimal windows in $O(L)$ time.

We will now turn to discovering f-closed episodes. Note that, unlike the *i*-closure, we do not define an f-closure of an episode at all. As shown in Sect. 1, such an f-closure would not necessarily be unique. However, the set of *i*-closed episodes will contain all f-closed episodes.

Theorem 6 *An f-closed episode is always i-closed.*

Proof Let $G \in \mathcal{S}$ be an episode and define $H = icl(G)$. Let W be a set of all minimal windows of s that cover G and let V be the set of all minimal windows that cover H. Let $w = [a, b] \in W$ be a minimal window of G and let f be a valid mapping from G to $s[a, b]$. Any new node added in H must occur in $s[a, b]$ and f can be extended to H such that it honours all edges in H. Hence $s[a, b]$ covers H. It is also a minimal window for H because otherwise it would violate the minimality for G. Hence $w \in V$. Now assume that $v = [a', b'] \in V$. Obviously, $s[a', b']$ covers G, so there must be a minimal window $w = [a, b] \in W$. Using the first argument we see that $w \in V$. Since V contains only minimal windows we conclude that $v = w$. Hence $W = V$. A sequence covers an episode if and only if the sequence contains a minimal window of the episode. Thus, by definition, $V = W$ implies that $fr_f(G) = fr_f(H)$ and $fr_d(G) = fr_d(H)$.

Assume now that G is not *i*-closed, then $G \neq H$, and since $fr(G) = fr(H)$, G is not f-closed either. This completes the proof. □

A naïve approach to extract f-closed episodes 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 the output. This approach can be considerably sped up by realising that we need only to test episodes with identical nodes and episodes of form $G - V$, where V is a subset of $V(G)$. The pseudo-code is given in Algorithm 3. 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.

6 Generating transitively closed candidate episodes

In this section we define an algorithm, GenerateCandidate, which generates the candidate episodes from the episodes discovered previously. The difficulty of the algorithm is that we need to make sure that the candidates are transitively closed.

Let $G \in \mathcal{S}$ be an 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 in $\mathcal{S}$, that is, transitively closed and strict. We can reverse this property in order to generate candidates: Let $G \in \mathcal{S}$

Algorithm 3: F-Closure. Postprocessing for computing f-closed episodes from i-closures

```
input  : all $i$-closed episodes $\mathcal{G}$
output : all $f$-closed episodes
foreach $G \in \mathcal{G}$ do
    foreach $H \in \mathcal{G}$ with $V(G) = V(H)$, $H \neq G$ do
        if $G \prec H$ and $fr(G) = fr(H)$ then
            Mark $G$;
        if $H \prec G$ and $fr(G) = fr(H)$ then
            Mark $H$;
    foreach $V \subset V(G)$ do
        $F \leftarrow G - V$;
        foreach $H \in \mathcal{G}$, with $V(F) = V(H)$ do
            if $H \preceq F$ and $fr(G) = fr(H)$ then
                Mark $H$;
return all unmarked episodes from $\mathcal{G}$;
```

be a previously discovered episode, add an edge e and verify that the new episode is transitively closed. However, we can improve on this naïve approach with the following theorem describing the sufficient and necessary condition for an episode to be transitively closed.

Theorem 7 *Let $G \in \mathcal{S}$ be an episode and let $e = (x, y)$ be an edge not in $E(G)$. Let $H = G + e$. Assume that H is a DAG. Then $H \in \mathcal{S}$ 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 proper descendant of y in H. To prove the first step in the induction, assume that $u = x$, 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 inductively 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. 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. The next theorem shows the necessary conditions for the existence of the candidate episode.

Theorem 8 *Let $H \in \mathcal{S}$ be an episode with $N + 1$ edges. Then either there are two episodes, $G_1, G_2 \in \mathcal{S}$, with identical nodes to H, such that*

- *G_1 and G_2 share $N - 1$ edges,*
- *$e_1 = last(G_1) \notin E(G_2)$,*

- *$e_2 > e_1$ and $H = G_1 + e_2$, where e_2 is the unique edge in $E(G_2)$ and not in $E(G_1)$*[3]

or last(G) is no longer a skeleton edge in H, where $G \in \mathcal{S}$, $G = H - last(H)$.

We will refer to the two cases as Case A and Case B, respectively.

Proof Let $e_2 = last(H)$ and define $G_1 = H - e_2$. If e_1 is not a skeleton edge in H, then by setting $G = G_1$, the theorem holds. Assume that e_1 is a skeleton edge in H. Define $G_2 = H - e_1$. Note that G_1 and G_2 share $N - 1$ edges. Also note that $last(G_1) = e_1 \notin E(G_2)$, $\{e_2\} = E(G_2) - E(G_1)$, and $H = G_1 + e_2$. Since $e_2 = last(H)$ it must be that $e_2 > e_1$. □

Theorem 8 gives us means to generate episodes. Let us first consider Case A in Theorem 8. To generate H we simply find all pairs of episodes G_1 and G_2 such that the conditions of Case A in Theorem 8 hold. When combining G_1 and G_2 we need to test whether the resulting episode is transitively closed.

Theorem 9 *Let $G_1, G_2 \in \mathcal{S}$ be two 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 of 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 $H \in \mathcal{S}$ 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 $H \in \mathcal{S}$, then e_1 and e_2 are both skeleton edges in H.

Proof We will first show that e_1 is a skeleton edge in H if $H \in \mathcal{S}$. 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. A similar argument holds for e_2.

The "only if" part is trivial so we only prove the "if" part using Theorem 7.

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 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 Theorem 7 now implies that H is transitively closed. □

Theorem 9 allows us to handle Case A of Theorem 8. To handle Case B, we simply take an episode G and try to find all edges e_2 such that $last(G + e_2) = e_2$ and $last(G)$ is no longer a skeleton edge in $G + e_2$. The conditions for this are given in the next theorem.

[3] In Tatti and Cule (2010) we incorrectly stated that $e_2 = last(G_2)$. Generally, this is not the case.

Theorem 10 *Let $G \in \mathcal{S}$ be an 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 occurring in G and define $H = G + e_2$. Assume $H \in \mathcal{S}$ and that e_1 is not a skeleton edge in H. Then 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 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 edges (z, x_2) and e_2 with (z, y_2). This path does not use e_2, hence it occurs in G. If $z \neq x_1$ or $y_1 \neq y_2$, then the path makes 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. □

Example 6 Consider the episodes given in Fig. 5. Episodes G_1 and G_2 satisfy the second condition in Theorem 9, hence the resulting episode H_1, is transitively closed. On the other hand, combining G_1 and G_3 leads to H_1', an episode that is not transitively closed since edge (a, d) is missing. Finally, G_4 and (c, a) satisfy Theorem 10 and generate H_2.

We can now combine the preceding theorems into the GENERATECANDIDATE algorithm given in Algorithm 4. We will first generate candidates by combining episodes from the previous rounds using Theorem 9. Secondly, we use Theorem 10 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.

Note that Algorithm 4 will not generate duplicates. To see this, first note that if H is generated using Case A, then the two episodes G_1 and G_2 generating H are unique. That is, it must be that $G_1 = H - \mathit{last}(H)$ and $G_2 = H - \mathit{last}(G_1)$. In other words, G_1 and G_2 will produce a unique H. In Case B, the episode G_1 is also unique, namely, we must have $G_1 = H - \mathit{last}(H)$. Thus, each G_1 will produce a unique H. Finally, note that $\mathit{last}(G_1)$, according to Theorem 9 is always a skeleton edge in H in Case A and we demand that $\mathit{last}(G_1)$ is not a skeleton edge in Case B. Hence, Case A and Case B will never generate the same episode.

7 Algorithm for discovering closed episodes

In this section we will complete our mining algorithm for discovering closed episodes. First, we present an algorithm for testing the episodes and a more detailed version of

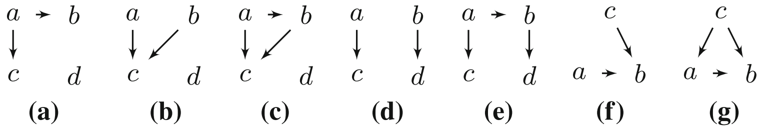

Fig. 5 Toy episodes for Example 6. **a** G_1, **b** G_2, **c** H_1, **d** G_3, **e** $H_1'2$, **f** G_4 and **g** H_2

Algorithm 4: GenerateCandidate. Generates candidate episodes from the previously discovered episodes

```
input  : a collection of previously discovered episodes $\mathcal{G} \subset \mathcal{S}$ with $N$ edges
output : $\mathcal{P} \subset \mathcal{S}$, a collection of candidate episodes with $N + 1$ edges
$\mathcal{P} \leftarrow \emptyset$;
foreach $G \in \mathcal{G}$, $G$ has no proper edges do
    foreach $x, y \in V(G)$, $lab(x) \neq lab(y)$ do
        add $G + (x, y)$ to $\mathcal{P}$;
foreach $G_1 \in \mathcal{G}$, $G_1$ has proper edges do
    $e_1 = (x_1, y_1) \leftarrow last(G_1)$;
    {Case where $e_1$ remains a skeleton edge.}
    $\mathcal{H} \leftarrow \{H \in \mathcal{G} \mid V(G) = V(H), |E(H) \cap E(G)| = N - 1, \{e_2\} = E(H) - E(G), e_2 > e_1\}$;
    foreach $G_2$ in $\mathcal{H}$ do
        $\{e_2\} \leftarrow E(H) - E(G)$;
        if $G_1$ and $G_2$ satisfy Theorem 9 and $e_2 \notin icl(G_1)$ then
            add $G_1 + e_2$ to $\mathcal{P}$;
    {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 $H$ is transitively closed (use Theorem 7) then
                add $H$ to $\mathcal{P}$;
    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 $H$ is transitively closed (use Theorem 7) then
                add $H$ to $\mathcal{P}$;
return $\mathcal{P}$;
```

Algorithm 1. Then we present an algorithm for adding the needed intermediate episodes and prove the correctness of the algorithm.

7.1 Detailed version of the algorithm

We begin by describing the test subroutine that is done for each candidate episode. Following the level-wise discovery, before computing the frequency of the episode, we need to test that all its subepisodes are discovered. It turns out that using transitively closed episodes will guarantee the strongest conditions for an episode to pass to the frequency computation stage.

Corollary 2 (of Theorem 1) *Let $G \in \mathcal{S}$ be an 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 $H_1 \in \mathcal{S}$, such that $H \prec H_1 \prec G$.*

If e is a proper skeleton edge of an episode $G \in \mathcal{S}$, then $G - e \in \mathcal{S}$. Thus, for G to be frequent, $G - e$ had to be discovered previously. This is the first test in TestCandidate (given in Algorithm 5). In addition, following the level-wise approach for mining closed patterns (Pasquier et al. 1999), 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 fewer 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.

Example 7 Consider the episodes given in Fig. 6. When testing the candidate episode H, TestCandidate will test for the existence of three episodes. Episodes G_1 and G_2 are tested when either edge is removed and G_3 is tested when node labelled d is removed.

Algorithm 5: TestCandidate. An algorithm that checks if an episode is a proper candidate

```
input  : an episode $G \in \mathcal{S}$, already discovered episodes $\mathcal{C} \subset \mathcal{S}$
output : a boolean value, true only if all subepisodes of $G$ are frequent
foreach proper skeleton edge $e$ in $G$ do
    if $G - e \notin \mathcal{C}$ or $e \in E(icl(G - e))$ then
        return false;
foreach $v$ in $G$ not having a proper skeleton edge do
    if $G - v \notin \mathcal{C}$ then
        return false;
    if $v$ has no edges and there is a node $w \in V(icl(G - v))$ s.t. $lab(w) = lab(v)$ then
        return false;
return true;
```

Finally we present a more detailed version of Algorithm 1, given in Algorithm 6. The only missing part of the algorithm is the AddIntermediate subroutine which we will give in the next section along with the proof of correctness.

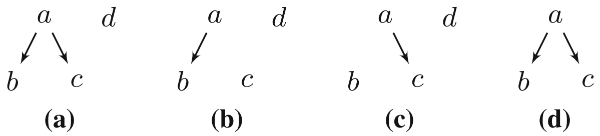

Fig. 6 Toy episodes for Example 7. **a** H, **b** G_1, **c** G_2 and **d** G_3

Algorithm 6: MINEEPISODES. An algorithm discovering all frequent closed episodes

```
input  : a sequence $s$, a frequency threshold $\sigma$, a window size $\rho$
output : $f$-closed frequent strict episodes
$\mathcal{C} \leftarrow$ all frequent episodes with one node;
$\mathcal{E} \leftarrow \emptyset$; $N \leftarrow 1$;
while there are episodes with $N$ or more nodes in $\mathcal{C} \cup \mathcal{E}$ do
    $M \leftarrow 0$;
    while there are episodes with $M$ or more edges in $\mathcal{C} \cup \mathcal{E}$ do
        foreach $G \in \mathcal{G}$ do
            if TESTCANDIDATE($G, \mathcal{E}$) and $G$ is frequent then
                add $G$ to $\mathcal{E}$;
                compute and store $icl(G)$;
                add ADDINTERMEDIATE($G$) to $\mathcal{E}$;
        $\mathcal{P} \leftarrow \{C \in \mathcal{E} \mid |E(C)| = M, |V(C)| = N\}$;
        $\mathcal{C} \leftarrow$GENERATECANDIDATE($\mathcal{P}$);
        $M \leftarrow M + 1$;
    $\mathcal{P} \leftarrow \{C \in \mathcal{E} \mid |V(G)| = N, G$ is a parallel episode$\}$;
    $\mathcal{C} \leftarrow \emptyset$;
    foreach $G \in \mathcal{P}$ do
        $x \leftarrow$ last node of $G$;
        $\mathcal{H} \leftarrow \{H \in \mathcal{G} \mid |V(G) \cap V(H)| = n - 1,\ lab(\text{last node of } H) > lab(x)\}$;
        foreach $H \in \mathcal{H}$ do
            add $G$ + last node of $H$ to $\mathcal{C}$;
        {Check episode $G$ augmented with a node carrying the label of the last node $x$}
        $y \leftarrow$ a node with the same label as $x$;
        add $G + y + (x, y)$ to $\mathcal{C}$;
    $N \leftarrow N + 1$;
return F-CLOSURE($\{icl(G) \mid G \in \mathcal{E}\}$);
```

7.2 Proof of correctness

In the original framework for mining itemsets, the algorithm discovers itemset generators, the smallest itemsets that produce the same closure. It can be shown that generators form a downward closed collection, so they can be discovered efficiently. This, however, is not true for episodes, as the following example demonstrates.

Example 8 Consider the episodes given in Fig. 7 and a sequence $s = acbxxabcxxbac$. Assume that the window size is $\rho = 3$ and the frequency threshold is $\sigma = 1$. The frequency of H is 1 and there is no subepisode that has the same frequency. Hence to discover this episode all of its maximal subepisodes need to be discovered. One of these subepisodes is $icl(G)$. However, $icl(G)$ is not added into $\mathcal{E}$ since $(a, c) \in icl(icl(G) - (a, c))$ and TESTCANDIDATE returns false.

The core of the problem is that sometimes adding some particular edge is not allowed until you have added another edge. For example, we cannot add (b, c) to G (given in Fig. 7a) until we have added (a, c).

We solve this problem by adding some additional episodes between the discovered episode G and its closure $icl(G)$. Roughly speaking, given an episode $G \in \mathcal{S}$ we need to add the episodes containing edges/nodes from $icl(G)$ such that these edges/nodes can be "hidden" by a single edge or node.

The details are given in the AddIntermediate algorithm given in Algorithm 7, and the justification for adding these episodes is given in the proof of Theorem 11.

Algorithm 7: AddIntermediate. An algorithm that given an episode adds some additional episodes between the discovered episode and its closure. This is necessary to guarantee the correctness of the algorithm

```
input  : an episode $G \in \mathcal{S}$
output : a subset of episodes between $G$ and $icl(G)$
$\mathcal{C} \leftarrow \emptyset$;
foreach $x \in V(icl(G)) - V(G)$ do
    $H \leftarrow G + x$;
    add $H$ to $\mathcal{C}$;
    foreach $y \in V(G)$ and $e = (x, y) \notin icl(G)$ 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 $\mathcal{C}$;
    foreach $y \in V(G)$ and $e = (y, x) \notin icl(G)$ 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 $\mathcal{C}$;
foreach $x, y \in V(icl(G)) - V(G)$ and $e = (x, y) \notin icl(G)$ do
    add $G + x + y$ to $\mathcal{C}$;
foreach $x, y \in V(G)$ and $e = (x, y) \notin icl(G)$ do
    $Z \leftarrow E(tcl(G + e)) - E(G + e)$;
    if $Z \neq \emptyset$ and $Z \subset E(icl(G))$ then
        add $G + Z$ to $\mathcal{C}$;
return $\mathcal{C}$;
```

We will now establish the correctness of the algorithm and justify Algorithm 7.

Definition 13 Let $G \in \mathcal{S}$ be an episode. We call a proper skeleton edge e in G *derivable* if $G \preceq icl(G - e)$. We call a solitary node v *derivable* if $G \preceq icl(G - v)$.

We will need the following straightforward lemmae to prove our main result.

Lemma 5 *Let $G \in \mathcal{S}$ be an episode. Let e be a non-derivable skeleton edge in G. Then e is a non-derivable skeleton edge in $G - f$, where $f \neq e$ is a proper skeleton edge in G. Similarly, e is a non-derivable skeleton edge in $G - v$, where v is a solitary node.*

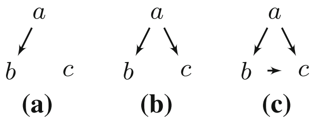

Fig. 7 Toy episodes for Example 8. **a** G, **b** $icl(G)$, **c** H

Proof Assume that e is derivable in $G - f$, then $G - f \preceq icl(G - f - e) \preceq icl(G - e)$. On the other hand, $G - e \preceq icl(G - e)$. Since the closure operator adds nodes only with unique labels, the graph homomorphisms from $G - e$ to $icl(G - e)$ and from $G - f$ to $icl(G - e)$ must be equal. This implies that $G \preceq icl(G - e)$ which is a contradiction. A similar argument holds for $G - v$. □

Lemma 6 *Let $G \in \mathcal{S}$ be an episode. Let w be a non-derivable solitary node in G. Then w is a non-derivable node in $G - v$, where v is a solitary node. Similarly, w is a non-derivable node in $G - e$, where e is a proper skeleton edge in G.*

Proof Similar to the proof of Lemma 5. □

Lemma 7 *Let $G \in \mathcal{S}$ be an episode with a derivable edge e. Then $icl(G) = icl(G - e)$. Let $G \in \mathcal{S}$ be an episode with a derivable node v. Then $icl(G) = icl(G - v)$.*

Proof The monotonicity and idempotency properties imply $icl(G) \preceq icl(icl(G-e)) = icl(G - e)$. Since $icl(G - e) \preceq icl(G)$, we have $icl(G) = icl(G - e)$.

To prove the second case, note that $icl(G) \preceq icl(icl(G - v)) = icl(G - v)$ which immediately implies that $icl(G) = icl(G - v)$. □

Theorem 11 *Assume a frequent episode $G \in \mathcal{S}$ with no derivable edges or derivable nodes. Then G is discovered.*

Proof We will prove the theorem by induction. Obviously, the theorem holds for episodes with a single node. Assume now that the theorem is true for all subepisodes of G. Episode G will be discovered if it passes the tests in TESTCANDIDATE. To pass these tests, all episodes of form $G - e$, where e is a proper skeleton edge need to be discovered. Assume that all these subepisodes are discovered but we have $e \in E(icl(G - e))$. This means, by definition, that e is a derivable edge in G which is a contradiction. The same argument holds for episodes of form $G - v$, where v is a node.

Now assume that one of the subepisodes, say, H is not discovered. The induction assumption now implies that H has either derivable nodes or edges.

Lemma 8 *There is a subepisode $F \in \mathcal{S}$, $F \prec H$ with no derivable edges and nodes such that $H \preceq icl(F)$. Any non-derivable skeleton edge in H will remain in F. Any non-derivable solitary node in H will remain in F.*

Proof Build a chain of episodes $H = H_1, H_2, \ldots, H_N = F$ such that H_{i+1} is obtained from H_i by removing either a derivable node or a derivable edge and F has no derivable edges or nodes. Note that this sequence always exists but may not necessarily be unique. Lemma 7 implies that we have $H_i \preceq icl(H_i) = icl(H_{i+1})$. Idempotency and monotonicity imply that $H \preceq icl(F)$. Lemma 5 implies that if e is a non-derivable skeleton edge in H, then e is also a non-derivable skeleton edge in each H_i. Similarly, Lemma 6 implies that any non-derivable solitary node will remain in each H_i. □

By the induction assumption F is discovered. We claim that H will be discovered by ADDINTERMEDIATE(F). Let us denote $Z = E(H) - E(F)$ and $W = V(H) - V(F)$.

The next three lemmae describe different properties of Z and W.

Lemma 9 *Assume that* $H = G - v$. *Then* $W = \{ w \}$ *such that* $lab(v) = lab(w)$ *and* $Z = \emptyset$.

Proof Lemma 5 implies that all skeleton edges in H are non-derivable. Lemma 8 implies that $Z = \emptyset$. Removing v can turn only one node, say w, into a solitary node. This happens when $lab(v) = lab(w)$ and there are no other edges adjacent to w. □

Lemma 10 *Assume that* $H = G - (x, y)$. *Then* $W \subseteq \{ x, y \}$. *If* $W = \{ x, y \}$, *then* $Z = \emptyset$.

Proof Let z be a node in G (and in H) such that $z \notin \{ x, y \}$. If z is a solitary node in G, it also a solitary node in H. Lemma 8 now implies that $z \notin W$. If z has an adjacent skeleton edge in G, say f, then $f \neq (x, y)$. Lemma 5 implies that f is also a non-derivable skeleton edge in H. Lemma 8 now implies that $z \notin W$.

If $W = \{ x, y \}$, then x (and y) cannot have any adjacent non-derivable skeleton edges in H. Hence there are no edges, except for (x, y), adjacent to x or y in G. Lemma 5 implies that all skeleton edges in H are non-derivable. Lemma 8 implies that $Z = \emptyset$. □

Lemma 11 *Assume that* $H = G - e$. *Then* $Z = E(tcl(F + W + e)) - E(F + W + e) \subset E(icl(F + W))$.

Note that $F + W + e$ is not necessarily transitively closed.

Proof Write $F' = F + W$. First note that $Z \subset E(H) \subseteq E(icl(F'))$. Lemmae 5 and 8 imply that all skeleton edges of G, except for e are in $E(F) = E(F')$. Hence, we must have $E(tcl(F' + e)) = E(G)$.

Also note that $Z \cup E(F' + e) = E(H) + e = E(G)$. Since $Z \cap E(F' + e) = \emptyset$, we have $Z = E(G) - E(F' + e) = E(tcl(F' + e)) - E(F' + e)$. □

If $H = G - v$, then Lemma 9 implies that $H = F + w$ and we discover H on Line 4 during ADDINTERMEDIATE(F). Assume that $H = G - e$. Write $(x, y) = e$. Lemma 10 now implies that $W \subseteq \{ x, y \}$ and Lemma 11 implies that $Z = E(tcl(F + W + e)) - E(F + W + e)$. We will show that there are 4 different possible cases:

1. $W = \{ x, y \}$. Lemma 10 implies that $Z = \emptyset$. This implies that $H = F + x + y$ and we discover H on Line 14 during ADDINTERMEDIATE(F).
2. $W = \{ w \}$, where w is either x or y and $Z = \emptyset$. This implies that $H = F + w$ and we discover H on Line 4 during ADDINTERMEDIATE(F).
3. $W = \emptyset$ and $Z \neq \emptyset$. This implies that $H = F + Z$ and we discover H on Line 18 during ADDINTERMEDIATE(F).
4. $W = \{ w \}$, where w is either x or y and $Z \neq \emptyset$. This implies that $H = F + w + Z$ and we discover H either on Line 8 or on Line 12 during ADDINTERMEDIATE(F).

This completes the proof of the theorem. □

Table 2 Characteristics of the sequences and the threshold values used for mining candidate episodes

Sequence	Size	$\|\Sigma\|$
moby	105,719	10,277
abstract	67,828	6,718
address	62,066	5,295

The second column is the number of symbols in the sequence

Theorem 12 *Every frequent i-closed episode will be outputted.*

Proof TESTEPISODE will output $icl(G)$ for each discovered episode G. Hence, we need to show that for each i-closed episode H there is a discovered episode G such that $H = icl(G)$.

We will prove the theorem by induction. Let H and G be episodes such that $H = icl(G)$ and H is an i-closed frequent episode. If G contains only one node, then G will be discovered. Assume that the theorem holds for any episode $icl\left(G'\right)$, where G' is a subepisode of G. If G has derivable nodes or edges, then by Lemma 7 there exists an episode $G' \prec G$ such that $H = icl(G) = icl\left(G'\right)$ and so by the induction assumption G' is discovered, and H is outputted. If G has no derivable edges or nodes, then Theorem 11 implies that G is discovered. This completes the proof. □

8 Experiments

We tested our algorithm[4] on three text datasets, *address*, consisting of the inaugural addresses by the presidents of the US,[5] merged to form a single long sequence, *moby*, the novel Moby Dick by Herman Melville,[6] and *abstract*, consisting of the first 739 NSF award abstracts from 1990,[7] also merged into one long sequence. We processed all three sequences using the Porter Stemmer[8] and removed the stop words. The characteristics of datasets are given in Table 2.

In the implementation an episode graph was implemented using sparse notation: the neighbourhood of a node was presented as a list of edges. To ensure efficient scanning, the sequence was implemented as a set of linked lists, one for each symbol. The experiments were conducted on a computer with an AMD Athlon 64 processor and 2GB memory. The code was compiled with G++ 4.3.4.

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. Tables 3 and 4 show how the total number of frequent episodes compared with the identified i-closed, e-closed and f-closed episodes we

[4] The C++ implementation is given at http://adrem.ua.ac.be/implementations/.

[5] Taken from http://www.bartleby.com/124/pres68.

[6] Taken from http://www.gutenberg.org/etext/15.

[7] Taken from http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html.

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

Table 3 The number of frequent, *i*-closed, *e*-closed and *f*-closed episodes with varying fixed-window frequency thresholds for the *address*, *abstract* and *moby* datasets, respectively

Dataset	σ	*f*-closed	*i*-closed	*e*-closed	Frequent
address	200	1,983	1,989	1,989	1,992
	100	6,774	6,817	6,820	6,880
	50	25,732	26,078	26,212	34,917
	30	70,820	72,570	73,328	119,326
	20	166,737	192,544	207,153	Out of memory
abstract	500	893	914	916	933
	400	1,374	1,436	1,440	1,499
	300	2,448	2,802	2,903	4,080
	200	6,074	7,374	8,080	98,350
	100	26,140	43,020	95,811	Out of memory
moby	200	3,389	3,394	3,394	3,395
	100	11,018	11,079	11,084	11,127
	50	37,551	38,043	38,120	39,182
	30	99,937	102,380	103,075	151,115
	20	231,563	245,683	253,208	Out of memory

discovered in the three datasets, using the fixed-window frequency and the disjoint-window frequency, respectively. The results show that while the reduction is relatively small for large thresholds, its benefits are clearly visible as we lower the threshold. The reason for this is that, because of the data characteristics, the major part of the output consists of episodes with a small number of nodes. Such episodes tend to be closed. When the threshold is lowered, episodes with a large number of nodes become frequent which leads to a pattern explosion. In the extreme case, we ran out of memory when discovering all frequent episodes for certain low thresholds while were able to compute the *f*-closed episodes.

To get a more detailed picture we examined the ratio of the number of frequent episodes and the number of *f*-closed episodes, the ratio of the number of *i*-closed episodes and the number of *f*-closed episodes, and the ratio of the number of *e*-closed episodes and the number of *i*-closed episodes, as a function of the number of nodes. The results using the fixed-window frequency are shown in Fig. 8. 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 *moby* dataset, with a threshold of 30, there were $24,131$ frequent episodes of size 6, of which only 13 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 $17,587$ frequent episodes of size 5, of which 878 were *i*-closed and 289 *f*-closed. Furthermore, we can see that while using only the *e*-closure helps reduce the output significantly, using the *i*-closure gives an even better reduction. For example, in the

Table 4 The number of frequent, *i*-closed, *e*-closed and *f*-closed episodes with varying disjoint-window frequency thresholds for the *address*, *abstract* and *moby* datasets, respectively

Dataset	σ	*f*-closed	*i*-closed	*e*-closed	Frequent
address	20	2,264	2,282	2,282	2,291
	10	9,984	10,213	10,219	10,396
	5	46,902	50,634	50,920	65,139
	4	77,853	87,935	89,076	268,675
	3	149,851	187,091	195,379	Out of memory
abstract	100	195	195	195	195
	50	932	1,000	1,002	1,020
	40	1,677	2,053	2,122	2,585
	30	3,542	5,482	5,798	20,314
	20	9,933	22,945	61,513	Out of memory
moby	20	4,076	4,119	4,121	4,121
	10	15,930	16,325	16,341	16,468
	5	67,180	72,306	72,468	77,701
	4	109,572	122,423	122,919	141,940
	3	207,031	251,757	158,303	Out of memory

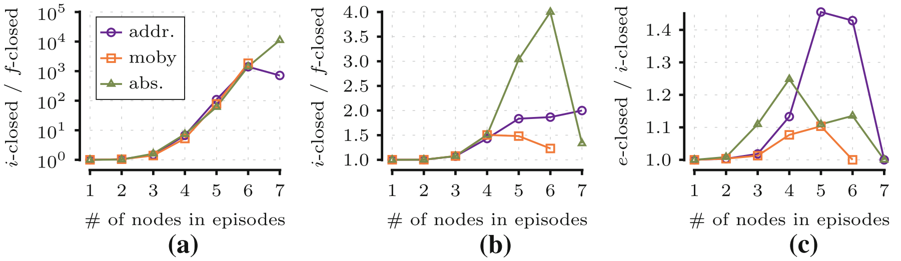

Fig. 8 Ratios of episodes as a function of the number of events. $fr_f(G)$ was used as frequency with the threshold $\sigma = 30$ for *address* and *moby*, and $\sigma = 200$ for *abstract*. Note that the *y*-axis of **a** is in log-scale. **a** Frequent / *f*-closed, **b** *i*-closed / *f*-closed and **c** *e*-closed / *i*-closed

address dataset, with a threshold of 30, there were 259 *e*-closed episodes of size 5, of which 178 were *i*-closed.

The same ratios using the disjoint-window frequency are shown in Fig. 9. Again, we can clearly see the benefits of using *i*-closure, especially on large episodes.

The difference between the number of *f*-closed and *i*-closed episodes can be explained by the fact that the *i*-closure operator looks at *all* valid mappings of the episode while the coverage requires only one valid mapping to exist. For example, consider the episode given in Example 1. This episode is *f*-closed with respect to disjoint windows, and occurs in 10 windows. A subepisode

$$\text{chief} \rightarrow \text{justic} \quad \text{vice} \quad \text{president}$$

is *i*-closed but not *f*-closed. The reason for this is that several speeches contain a line ‘vice president …chief justice …vice president’. Hence, we can construct a valid mapping for the order: president, chief, justice, vice. Consequently, we cannot add an edge from vice to president. However, for all such mappings we can construct an alternative mapping satisfying the episode in Example 1. Thus the support of both episodes will be the same and we can delete the subepisode from the output.

To complete our analysis, we present a comparison of the number of serial, parallel and general episodes we found. We compare the number of *f*-closed episodes to the overall number of frequent episodes, to illustrate how much the output has been reduced by using closed episodes. The results are shown in Tables 5 and 6 for fixed-window and disjoint-window frequency, respectively. To avoid double counting, we consider singletons to be parallel episodes, while neither serial nor parallel episodes are included in the general episodes total. As expected, the number of serial and parallel episodes does not change much, as most of them are closed. For a serial episode not to be closed, we would need to find an episode consisting of more nodes, yet having the same frequency, which is not often the case. A parallel episode is not closed if a partial order can be imposed on its nodes, without a decline in frequency—again, this is not often the case. However, as has been pointed out in Fig. 1 in the introduction, a single frequent serial episode results in an explosion of the number of discovered general episodes. The results demonstrate that the output of general episodes has indeed been greatly reduced. For example, using a fixed-window frequency threshold of 200 on the *abstract* dataset, we discovered 93,813 frequent general episodes, of which only 2,247 were *f*-closed.

The runtimes of our experiments varied between a few seconds and 3 min 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.

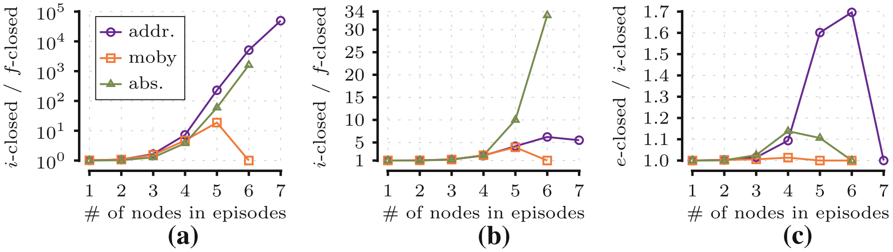

Fig. 9 Ratios of episodes as a function of the number of events. $fr_d(G)$ was used as frequency with the threshold $\sigma = 4$ for *address* and *Moby*, and $\sigma = 30$ for *abstract*. Note that the *y*-axis of **a** is in log-scale. **a** Frequent / *f*-closed, **b** *i*-closed / *f*-closed and **c** *e*-closed / *i*-closed

Table 5 The number of f-closed and frequent serial, parallel and general episodes, with varying fixed-window frequency thresholds for the *address*, *abstract* and *moby* datasets, respectively

Dataset	σ	Serial f-closed	Serial Frequent	Parallel f-closed	Parallel Frequent	General f-closed	General Frequent
address	200	293	293	1,670	1,674	20	25
	100	1,739	1,742	4,846	4,878	189	260
	50	8,436	8,494	15,526	16,068	1,770	10,355
	30	24,620	24,973	36,593	41,055	9,607	53,298
	20	61,254	N/A	67,658	N/A	37,825	N/A
abstract	500	116	116	667	669	110	148
	400	206	208	939	942	229	349
	300	433	448	1,435	1,471	580	2,161
	200	1,124	1,353	2,703	3,184	2,247	93,813
	100	4,597	N/A	7,854	N/A	13,689	N/A
moby	200	594	594	2,772	2,776	23	25
	100	2,992	2,997	7,749	7,788	277	342
	50	12,529	12,583	22,615	23,192	2,407	3,407
	30	34,469	34,915	52,481	58,021	12,987	58,179
	20	85,112	N/A	96,675	N/A	49,866	N/A

Singletons are classified as parallel episodes, and general episodes do not include the serial and parallel episodes

Table 6 The number of f-closed and frequent serial, parallel and general episodes, with varying disjoint-window frequency thresholds for the *address*, *abstract* and *moby* datasets, respectively

Dataset	σ	Serial f-closed	Serial Frequent	Parallel f-closed	Parallel Frequent	General f-closed	General Frequent
address	20	479	479	1,744	1,753	41	59
	10	3,004	3,012	6,325	6,468	655	916
	5	15,318	15,557	24,038	27,069	7,546	22,513
	4	25,532	26,372	36,116	45,108	16,205	197,195
	3	49,859	N/A	57,293	N/A	42,699	N/A
abstract	100	20	20	160	160	15	15
	50	164	166	565	573	203	281
	40	300	314	892	924	485	1,347
	30	631	708	1,479	1,660	1,432	17,946
	20	1,638	N/A	2,920	N/A	5,375	N/A
moby	20	1,010	1,014	2,983	3,004	83	103
	10	5,075	5,099	9,853	10,035	1,002	1,334
	5	22,016	22,304	34,070	37,920	11,094	17,477
	4	35,950	36,807	50,545	61,704	23,077	43,429
	3	69,048	N/A	79,944	N/A	58,039	N/A

Singletons are classified as parallel episodes, and general episodes do not include the serial and parallel episodes

9 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. (1994). 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 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 (1995) using an APRIORI-style algorithm (Agrawal and Srikant 1994).

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. (2003), Tzvetkov et al. (2003), and Wang and Han (2004) proposed methods for mining such closed patterns, while Casas-Garriga (2005) 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 all paths in the DAG occur in the sequence; however, a single event in a sequence can be mapped to multiple nodes.

In another attempt to trim the output, Garofalakis et al. (2002) 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. (1997). 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 (2009), MINEPI fails due to an error in its definition. Zhou et al. (2010) proposed mining closed serial episodes based on the MINEPI method, without solving this error. Laxman et al. (2007) introduced a monotonic measure as the maximal number of non-overlapping occurrences of the episode.

Pei et al. (2006) considered a restricted version of our problem setup. In their setup, items are allowed to occur only once in a window (string in their terminology). This means that the discovered episodes can contain only one occurrence 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 occurrences 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.

Casas-Garriga (2003) 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 (2004), 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. (2007) 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. (2009) 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 (2009) and Gwadera et al. (2005a,b) 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 (2006).

10 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 a closure operator based on instances. This closure does not suffer from the same problems that occur with the closure based on frequency. Unlike the closure based on frequency, an episode will always have only one instance-closure.

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

References

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th international conference on very large data bases (VLDB 1994), pp 487–499

Agrawal R, Srikant R (1995) Mining sequential patterns. In: 11th international conference on data engineering (ICDE 1995), pp 3–14

Calders T, Dexters N, Goethals B (2007) Mining frequent itemsets in a stream. In: Proceedings of the 7th IEEE international conference on data mining (ICDM 2007), pp 83–92

Casas-Garriga G (2003) 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, pp 83–94

Casas-Garriga G (2005) Summarizing sequential data with closed partial orders. In: Proceedings of the SIAM international conference on data mining (SDM 2005), pp 380–391

Cule B, Goethals B, Robardet C (2009) A new constraint for mining sets in sequences. In: Proceedings of the SIAM international conference on data mining (SDM 2009), pp 317–328

Garofalakis M, Rastogi R, Shim K (2002) Mining sequential patterns with regular expression constraints. IEEE Trans Knowl Data Eng 14(3):530–552

Gwadera R, Atallah MJ, Szpankowski W (2005a) Markov models for identification of significant episodes. In: Proceedings of the SIAM international conference on data mining (SDM 2005), pp 404–414

Gwadera R, Atallah MJ, Szpankowski W (2005) Reliable detection of episodes in event sequences. Knowl Inf Syst 7(4):415–437

Laxman S, Sastry PS (2006) A survey of temporal data mining. SADHANA Acad Proc Eng Sci 31(2): 173–198

Laxman S, Sastry PS, Unnikrishnan KP (2007) 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), pp 410–419

Mannila H, Toivonen H, Verkamo AI (1997) Discovery of frequent episodes in event sequences. Data Min Knowl Discov 1(3):259–289

Méger N, Rigotti C (2004) 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, pp 313–324

Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: ICDT ’99: proceedings of the 7th international conference on database theory, pp 398–416

Pei J, Wang H, Liu J, Wang K, Wang J, Yu PS (2006) Discovering frequent closed partial orders from strings. IEEE Trans Knowl Data Eng 18(11):1467–1481

Tatti N (2009) Significance of episodes based on minimal windows. In: Proceedings of the 9th IEEE international conference on data mining (ICDM 2009), pp 513–522

Tatti N, Cule B (2010) Mining closed strict episodes. In: Proceedings of the 10th IEEE international conference on data mining (ICDM 2010)

Tzvetkov P, Yan X, Han J (2003) Tsp: mining top-k closed sequential patterns. In: Proceedings of the 3rd IEEE international conference on data mining (ICDM 2003), pp 347–354

Wang J, Han J (2004) Bide: efficient mining of frequent closed sequences. In: 20th international conference on data engineering (ICDE 2004), p 79

Wang JT-L, Chirn G-W, Marr TG, Shapiro B, Shasha D, Zhang K (1994) Combinatorial pattern discovery for scientific data: some preliminary results. ACM SIGMOD Rec 23(2):115–125

Yan X, Han J, Afshar R (2003) Clospan: mining closed sequential patterns in large datasets. In: Proceedings of the SIAM international conference on data mining (SDM 2003), pp 166–177

Zhou W, Liu H, Cheng H (2010) Mining closed episodes from event sequences efficiently. In: Proceedings of the 14th Pacific-Asia conference on knowledge discovery and data mining, vol 1, pp 310–318