Knowledge and Information Systems (2019) 60:447–494
https://doi.org/10.1007/s10115-019-01349-y

REGULAR PAPER

Discovering frequent chain episodes

Avinash Achar[1] · **P. S. Sastry**[2]

Received: 25 January 2018 / Revised: 23 February 2019 / Accepted: 27 February 2019 /
Published online: 15 March 2019

Abstract
Frequent episode discovery is a popular framework in temporal data mining with many applications. An episode is a partially ordered set of nodes with each node associated with an event-type. The episodes literature has seen different notions of frequency and a variety of associated discovery algorithms under these different frequencies when the associated partial order is total (serial episode) or trivial (parallel episode). Recently an apriori-based discovery algorithm for mining episodes where the associated partial order has no restriction but the node to event-type association is one–one (general injective episodes) was proposed based on the non-overlapped frequency measure. This work pointed out that frequency alone is not a sufficient indicator of interestingness in the context of episodes with general partial orders and introduced a new measure of interestingness called bidirectional evidence (BE) to address this issue. This algorithm discovers episodes by incorporating both frequency and BE thresholds in the level-wise procedure. In this paper, we extend this BE-based algorithm to a much larger class of episodes that we call chain episodes. This class encompasses all serial and parallel episodes (injective or otherwise) and also many other non-injective episodes with unrestricted partial orders. We first discuss how the BE measure can be generalized to chain episodes and prove the monotonicity property it satisfies in this general context. We then describe our candidate generation step (with correctness proofs) which nicely exploits this new monotonicity property. We further describe the frequency counting (with correctness proofs) and BE computation steps for chain episodes. The experimental results demonstrate the effectiveness of our algorithms.

Keywords Data mining · Episode · Apriori-based · Non-overlapped frequency · Partial order

✉ Avinash Achar
achar.avinash@tcs.com

P. S. Sastry
sastry@iisc.ac.in

[1] Research, Tata Consultancy Services, IITM Research Park, Chennai 600113, India

[2] Department of Electrical Engineering, Indian Institute of Science, Bangalore 560012, India

1 Introduction

Frequent episode discovery [18] is a popular framework for mining temporal correlations in event sequences with applications in several domains like manufacturing [28], telecommunication [18], WWW [16], biology [8,20], finance [19], intrusion detection [17,29], text mining [14], composable conveyor systems [13], etc. In this framework, the data is a single long time-ordered sequence of events. The temporal patterns of interest called episodes are small, partially ordered collections of nodes, with each node associated with a symbol (called event-type). The partial order in the episode constrains the time-order in which events constituting an occurrence of the pattern appear in the data. The task is to unearth all episodes whose frequency in the data exceeds a user-defined threshold.

Over the years, in the episodes context, a variety of discovery algorithms are proposed. Most of these algorithms have restricted their attention to only *serial* episodes (where the partial order is a total order) or only *parallel* episodes (where the partial order is trivial) [1,3]. The class of methods can be broadly categorized into breadth-first search [3,14,15,18] and depth-first search [4,12] approaches based on their search strategies. Choice of frequency thresholds in pattern mining is typically arbitrary. There has been considerable work towards assessing interestingness of patterns based on sound statistical ideas under different episode frequencies [2,7,22–24]. Another important issue in pattern discovery is to be able to mine online streams. There has been some recent work in this direction in the episodes context [10,11,21].

Even though the original episodes framework is almost two decades old, algorithms for discovering episodes under general partial orders have been proposed only in the last couple of years. Apriori-based discovery algorithms to mine injective episodes[1] with general partial orders based on the non-overlapped frequency was proposed in [5]. Another level-wise algorithm for discovering frequent episodes with general partial orders under the windows-based and non-overlapped frequency is proposed in [25,27]. They consider a class of episodes with general partial orders that they call *strict* episodes. They mine for frequent closed (strict) episodes which is a compressed version of the set of all frequent episodes. Tatti and Cule [26] proposes an interesting depth-first approach for episode mining of general episodes. We elaborate more on the related work in Sect. 7 after describing our work in detail.

In this paper, we extend the apriori-based discovery algorithm for injective episodes, presented in [5], to a bigger class of episodes with general partial orders. This class is same as what was termed strict episodes in [25,27]. This class of episodes was independently proposed in [1,6] where the class has been called chain episodes. For this reason, we would call these episodes as chain episodes in this paper. However, we emphasize that the terms strict episodes and chain episodes are synonymous. The class of chain episodes includes all (injective and non-injective) serial and parallel episodes. It also includes many other non-injective episodes with general partial orders.

For instance, consider multi-neuronal spike train data from a neural tissue as an example event sequence where each event corresponds to the identity of the neuron firing and the associated time of firing. The episode pattern is useful in capturing the underlying functional circuits or dependencies present in the tissue. Injective episode graphs can be useful when the underlying functional circuits involve distinct neurons. However, when the underlying circuits have feedback relationships, injective episodes do not suffice and the need for mining beyond injective episodes with general partial orders becomes important. Consider a relationship where firing of neuron A results in two other distinct neurons B and C firing within a certain delay. Further if B and C firing almost synchronously in-turn triggers neuron A, this kind of

[1] An episode is injective if the associated node to event-type map is one to one.

a cyclic dependency is evidently not capturable by general injective episodes. It can neither be captured by serial or parallel non-injective episodes. This simple example motivates the need for mining patterns in the bigger class of chain episodes considered in this paper.

In [5], it was pointed out that, for general partial order mining, frequency alone is not a sufficient indicator of interestingness and the authors *introduced a new measure called bidirectional evidence (BE) which is used, along with frequency, to properly assess interestingness of an injective episode*. The algorithm we propose here is an apriori-style level-wise procedure for discovering in the space of chain episodes which use both frequency and bidirectional evidence to assess interestingness. We note that unlike in this paper, the other existing works like [25,27] discover chain episodes using frequency alone as a measure of interestingness. Our specific contributions in this paper are as follows:

- We introduce the class of chain episodes[2] which nicely subsumes all (injective or otherwise) serial and parallel episodes and all injective episodes with general partial orders.
- We extend the notion of BE (introduced earlier for injective episodes in [5]) to chain episodes. We identify and prove a *new* monotonicity property satisfied by the BE measure in the context of chain episodes.
- Towards the design of an apriori-style algorithm, we present a novel candidate generation method that exploits the above monotonicity property. We formally prove that the method generates all interesting episodes without any duplication. This proposed method turns out to be a neat but non-trivial extension of the injective episode candidate generation proposed earlier in [5]. On the other hand, the proposed method is very different from the existing candidate generation employed in the levelwise procedure of [25].
- The proposed candidate generation also has some intelligent modifications (compared to [5]) in a step that checks for transitive closure. These checks when applied to the injective episode candidate generation algorithm of [5] make it more efficient.
- We propose algorithms for counting non-overlapped as well as minimal occurrence-based frequencies of chain episodes which are relatively straightforward extensions of the counting schemes proposed in [5]. We also provide novel proofs of correctness of these counting schemes[3] which were not discussed even in the context of injective episodes in [5].
- We also present an algorithm for computing BE of chain episodes while frequency counting. This has some important differences when compared to the corresponding algorithm in the case of injective episodes [5].
- We demonstrate the efficacy of our method in detail on synthetic data traces.

The rest of this paper is organized as follows. Section 2 briefly reviews the formalism of episodes. We introduce the class of chain episodes in Sect. 3 and places our contribution in context of the current literature. Section 4 describes bidirectional evidence, an additional measure of interestingness for general partial order episodes, in the context of chain episodes and the monotonicity property it satisfies. Section 5 describes the candidate generation step incorporating this new monotonicity property in detail. The computational aspects of both frequency and bidirectional evidence are described in Sect. 6. We discuss related work in Sect. 7. Section 8 illustrates the effectiveness of our algorithms through simulations. In Sect. 9, we provide concluding remarks.

[2] The class was originally introduced in [6] while the term "Chain Episodes" was explicitly used for the first time in [1].

[3] The correctness proof for chain episode counting we present here is a minor extension of the correctness proofs for injective episodes first reported in [1]. An alternate correctness proof for chain episode counting is given in [25] which appeared after [1].

2 Episodes in event sequences

In the frequent episode framework [18], the data, referred to as an *event sequence*, is denoted by $\mathbf{D} = \langle (E_1, t_1), (E_2, t_2), \ldots (E_n, t_n) \rangle$, where n is the number of events in the data. In each tuple (E_i, t_i), E_i denotes the event-type and t_i the time of occurrence of the event. The E_i, take values from a finite set, $\mathcal{E}$. The sequence is ordered so that $t_i \leq t_{i+1}$ for $i = 1, 2, \ldots$. Each event here is instantaneous and we make a further assumption that two events of the same event-type do not occur at the same instant. As this is the case in most applications encountered in practice, we assume that at any given time t, an event of a given type can occur only once. However, at a given time there can be more than one event (of different types) which is why we allow for $t_{i+1} = t_i$.

The event-types denote some information regarding nature of each event and they are application-specific. For example, event sequence could contain information about spikes or firing of action potentials by individual neurons in a neural tissue [9,22]. Each event is now represented by the identity of a neuron and its time of firing. Another example of an event sequence could be a sequence of fault alarms in an assembly line in a manufacturing plant [28] and the event-types represent some codes that characterize each such fault-reporting event. The objective is to analyse such sequential data to unearth interesting temporal patterns that are useful in the context of applications. In the above two applications, we may be interested in temporal patterns that enable us to unearth the functional dependencies between interacting neurons or to diagnose the root-cause for some fault alarm that is currently seen. The temporal patterns that we may wish to represent and discover are called *episodes* which we formally define below.

Definition 1 [18] An N-node episode α, is a tuple, $(V_\alpha, <_\alpha, g_\alpha)$, where $V_\alpha = \{v_1, v_2, \ldots, v_N\}$ is a collection of nodes, $<_\alpha$ is a (strict) partial order[4] on V_α and $g_\alpha : V_\alpha \rightarrow \mathcal{E}$ is a map that associates each node with an event-type from $\mathcal{E}$.

In other words, an episode is a multiset of event-types with a partial order on it. When $<_\alpha$ is a total order, α is referred to as a *serial episode* and when $<_\alpha$ is empty, α is referred to as a *parallel episode*. An example of a 3-node episode could be $\alpha = (V_\alpha, <_\alpha, g_\alpha)$, where $v_1 <_\alpha v_2$ and $v_1 <_\alpha v_3$, and $g_\alpha(v_1) = B, g_\alpha(v_2) = A, g_\alpha(v_3) = C$. This is shown in Fig. 1a. We denote this using a simple graphical notation as $(B \rightarrow (A\ C))$, because it captures the essence of the temporal pattern represented by this episode, namely B is followed by A and C in any order. Similarly, we represent a serial episode capturing the pattern A followed by B followed by C as $(A \rightarrow B \rightarrow C)$. A parallel episode involving the event-types A, B and C is represented as $(A\ B\ C)$. Figure 1 gives a variety of example episodes with their compact graphical notation. This compact graphical notation omits transitively closed edges and is used throughout the paper to refer to episodes succinctly.

The episode patterns defined above represent some kind of temporal dependencies involving a set of event-types. The same pattern can mean different things depending on the application. For example, in the manufacturing plant context discussed before, suppose an event-type C represents a major type of fault. If the underlying causative chain resulting in C's occurrence is such that faults of type B and A occurring in either order sufficiently close together trigger C, then the episode pattern $((B\ A) \rightarrow C)$ aptly captures this correlation.

[4] Given any set V, a *relation* R over V (which is a subset of $V \times V$) is said to be a *strict partial order* if it is *irreflexive* (i.e. for all $v \in V$, $(v, v) \notin R$), *asymmetric* (i.e. $(v_1, v_2) \in R$ implies that $(v_2, v_1) \notin R$, for all distinct $v_1, v_2 \in V$) and *transitive* (i.e. $\forall v_1, v_2, v_3 \in V$, $(v_1, v_2) \in R$ and $(v_2, v_3) \in R$ implies that $(v_1, v_3) \in R$).

(a) $(B \rightarrow (A\,C))$ **(b)** $(B \rightarrow (A(B \rightarrow C)))$ **(c)** $((B \rightarrow A)(B \rightarrow C))$ **(d)** $((E\,F) \rightarrow (F\,D) \rightarrow E)$

Fig. 1 Example episodes

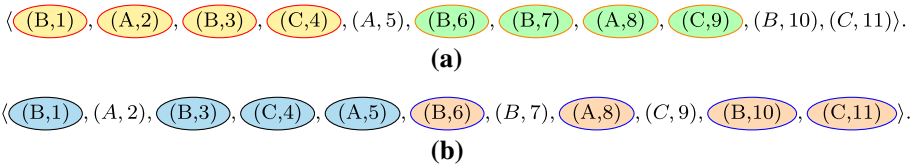


Fig. 2 Some occurrences of $(B \rightarrow (A(B \rightarrow C)))$

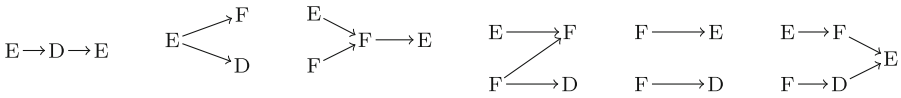


Fig. 3 Example subepisodes of $((E\,F) \rightarrow (F\,D) \rightarrow E)$ (Fig. 1d)

In the multi-neuronal context, the episode pattern $((B\,A) \rightarrow C)$ can capture the functional connectivity among 3 neurons, namely neurons B and A when firing close together in any order causes C to fire.

Definition 2 [18] Given an event sequence, $\langle (E_1, t_1), \ldots, (E_n, t_n)\rangle$ and an episode $\alpha = (V_\alpha, <_\alpha, g_\alpha)$, an occurrence of α in the event sequence is a one–one map $h : V_\alpha \rightarrow \{1, \ldots, n\}$ such that $g_\alpha(v) = E_{h(v)}\ \forall v \in V_\alpha$, and $\forall v, w \in V_\alpha$, $v <_\alpha w$, we have $t_{h(v)} < t_{h(w)}$.

An occurrence of an N-node episode α is basically a subset of N events from the event sequence which conform to the underlying partial order $<_\alpha$. From the h-map point of view, this subset of N events corresponds to the range of the h function. Figure 2a, b shows four occurrences of $(B \rightarrow (A(B \rightarrow C)))$, with all events constituting each occurrence marked with the same colour. This episode is shown in Fig. 1b.

Definition 3 [18] Episode $\beta = (V_\beta, <_\beta, g_\beta)$ is said to be a *subepisode* of $\alpha = (V_\alpha, <_\alpha, g_\alpha)$ (denoted $\beta \preceq \alpha$) if there exists a $1-1$ map $f_{\beta\alpha} : V_\beta \rightarrow V_\alpha$ such that (i) $g_\beta(v) = g_\alpha(f_{\beta\alpha}(v))$ $\forall v \in V_\beta$, (ii) $\forall v, w \in V_\beta$ with $v <_\beta w$, we have $f_{\beta\alpha}(v) <_\alpha f_{\beta\alpha}(w)$ in V_α.

Thus, $(B \rightarrow A)$, $(B \rightarrow C)$ and (AC) are 2-node subepisodes of $(B \rightarrow (AC))$ while (BAC) is 3-node subepisode of it. Figure 3 gives a number of subepisodes of $((E\,F) \rightarrow (F\,D) \rightarrow E)$, which is the episode shown in Fig. 1d. The importance of the notion of subepisode is that if $\beta \preceq \alpha$, then every occurrence of α contains an occurrence of β [18]. We say β is a strict subepisode of α if $\beta \preceq \alpha$ and $\alpha \neq \beta$.

Given an event sequence, the data mining task is to discover all interesting episodes. In most of the episode discovery algorithms, this is same as discovering episodes whose frequencies exceed a given threshold. Frequency is some measure of how often an episode occurs. As mentioned earlier, the frequency of episodes can be defined in many ways [3]. In this paper, we concentrate on the non-overlapped frequency only. While describing frequency

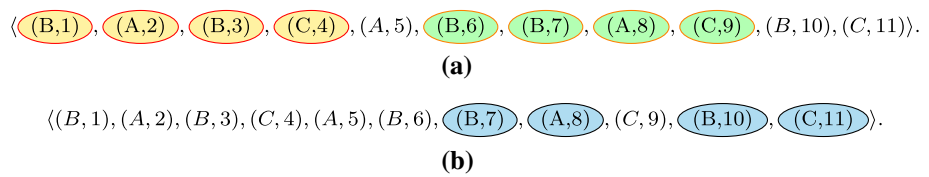


Fig. 4 Minimal occurrences of $(B \rightarrow (A(B \rightarrow C)))$: two are shown in **a** and one in **b**

counting, we will also touch upon the closely related frequency measure based on the minimal occurrences [18]. We now explain both these measures.

Definition 4 [18] A *minimal window* of α in an event sequence $\mathbf{D}$ is a time-window which contains an occurrence of α, such that no proper sub-window of it contains an occurrence of α. An occurrence in a minimal window is called a minimal occurrence. The **minimal occurrence-based frequency** of α in $\mathbf{D}$ (denoted f_{mi}) is defined as the number of minimal windows of α in $\mathbf{D}$.

In the event sequence of Fig. 4, there exists 3 minimal windows of $(B \rightarrow (A(B \rightarrow C)))$, namely [1, 4], [6, 9] and [7, 11] and the occurrences indicated are minimal occurrences from each of these windows.

Definition 5 [15] Two occurrences h_1 and h_2 of α are said to be non-overlapped in $\mathbf{D}$ if either $\max_i t_{h_1(v_i)} < \min_j t_{h_2(v_j)}$ or $\max_i t_{h_2(v_i)} < \min_j t_{h_1(v_j)}$. A set of occurrences is said to be non-overlapped if every pair of occurrences in the set is non-overlapped. A set H, of non-overlapped occurrences of α in $\mathbf{D}$, is *maximal* if $|H| \geq |H'|$, where H' is any other set of non-overlapped occurrences of α in $\mathbf{D}$. The **non-overlapped frequency** of α in $\mathbf{D}$ (denoted as f_{no}) is defined as the cardinality of a maximal non-overlapped set of occurrences of α in $\mathbf{D}$.

For example, occurrences marked in Fig. 4a form a maximal set of non-overlapped occurrences of $(B \rightarrow (A(B \rightarrow C)))$. This means its non-overlapped frequency is 2. This paper primarily concerns discovery algorithms under non-overlapped frequency. Minimal occurrence-based frequency is considered only in Sect. 6. Hence, in the rest of the paper, by frequency, we mean non-overlapped (unless otherwise specified).

3 Chain episodes

We first define injective episodes and then consider chain episodes. Injective episodes exactly capture the first-cut intuitive notion of an episode being a set of event-types with a partial order on it.

Definition 6 An episode $\alpha = (V_\alpha, <_\alpha, g_\alpha)$ is an injective episode if the corresponding g_α map is one to one. An injective episode α can also be viewed as a partially ordered set of event-types (X^α, R^α) where X^α is the range of the g_α map and R^α is the partial order induced on X^α by $<_\alpha$.

Basically, injective episodes are a set of non-repeated event-types with a partial order on it. Injective episodes represent a fairly rich class of episodes with general partial orders and there are efficient algorithms to discover frequent injective episodes [5]. The method reported

in [5] is one of the first methods that can discover episodes with unrestricted partial orders. The assumption of injectiveness on the g_α map implies that no event-types can repeat in an episode. For example, consider the multi-neuronal spike train data when episodes can capture functional circuits in the neuronal tissue. In such a scenario, many types of functional circuit graphs can be represented by injective episodes as long as all the participating neurons are distinct. However, when there is feedback or cyclic relationships, then the underlying temporal dependencies cannot be captured by injective episodes. Such cyclic dependencies exist in many other application domains also, e.g. fluctuations in share prices. In this paper, we introduce a class of episodes that we call chain episodes which is a generalization of injective episodes where certain kinds of repetition of event-types would be possible. Before defining chain episodes, we define the standard notion of a chain.

Definition 7 Given a partial order $(V_\alpha, <_\alpha)$, a chain is a totally ordered subset of V_α under the partial order $<_\alpha$.

Definition 8 An episode $\alpha = (V_\alpha, <_\alpha, g_\alpha)$ is a chain episode if any set of nodes which map to the same event-type under the map g_α forms a chain under $<_\alpha$.

One can immediately see that any injective episode is (vacuously) a chain episode. When $<_\alpha$ is a total order, any g_α map satisfies Definition 8. Thus all serial episodes (injective or otherwise) are chain. We define notion of *equivalent* episodes, which will be useful in the rest of section.

Definition 9 We define two episodes β and β' to be equivalent if they share the same set of occurrences in any event sequence.

We now show how a non-injective parallel episode too can be cast as a chain episode. Consider, for example, $\alpha = (AABBB)$, a non-injective parallel episode. By the definition of a parallel episode, $\alpha = (V_\alpha, <_\alpha, g_\alpha)$ has a representation where $V_\alpha = \{v_1, v_2, v_3, v_4, v_5\}$, $<_\alpha = \{\}$, $g_\alpha(v_1) = g_\alpha(v_2) = A$ and $g_\alpha(v_3) = g_\alpha(v_4) = g_\alpha(v_5) = B$, which does not satisfy Definition 8 and hence is not a chain episode. But consider the following episode $\alpha' = ((A \rightarrow A)(B \rightarrow B \rightarrow B))$. That is, $\alpha' = (V_{\alpha'}, <_{\alpha'}, g_{\alpha'})$, where $V_\alpha = \{v_1, v_2, v_3, v_4, v_5\}$, $<_\alpha = \{(v_1, v_2), (v_3, v_4), (v_4, v_5), (v_3, v_5)\}$ and $g_\alpha(v_1) = g_\alpha(v_2) = A$ and $g_\alpha(v_3) = g_\alpha(v_4) = g_\alpha(v_5) = B$. One can see that this is a chain episode. Recall from Sect. 2 that event sequences considered here contain instantaneous events. Also, at any given time tick t, a specific event-type can occur at most once (though multiple events of different event-types can occur at the same time instant). In such event sequences, every occurrence of α is also an occurrence of α' and conversely and hence are equivalent as per Definition 9. Even though α and α' are different discrete structures, they are indistinguishable episodes in terms of their occurrences. *In general, an episode β having an equivalent representation β' satisfying Definition 8 is also a chain episode*. Hence, in this sense, α is a chain episode. From this example, one can see that every non-injective parallel episode will have an equivalent representation in the class of chain episodes, and hence every (non-injective) parallel episode is indeed a chain episode.

Remark 1 Generalizing from the above non-injective parallel episode case, we can conclude that episodes that apparently do not satisfy Definition 8 can still be chain episodes as long as they have an equivalent representation (as per Definition 9) that satisfies Definition 8. To summarize, an episode (as per Definition 1) which also satisfies Definition 8 is certainly a chain episode. However, an episode β (as per Definition 1) which does not satisfy Definition 8 may still be a chain episode as long as it has an equivalent representation β' (as per Definition 9) that satisfies Definition 8.

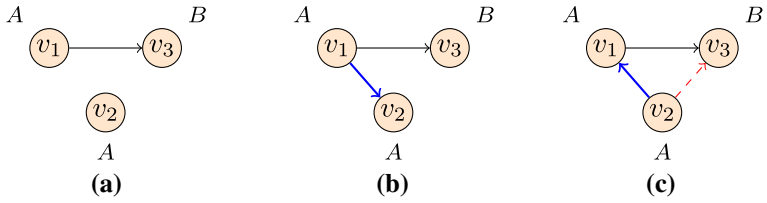


Fig. 5 Illustration of why $((A \rightarrow B)(A))$ is a chain episode

Based on this crucial observation, we briefly explore the space of episodes in the order of increasing size (number of nodes) for genuine non-chain episodes. By genuine, we mean episodes which don't have an equivalent representation satisfying Definition 8. At the 2-node level, we find only serial or parallel episodes. Hence every episode of size 2 is a chain episode.

One can show that every 3-node episode is also chain. For example, consider $\beta = ((A \rightarrow B)(A))$ (Fig. 5a), which does not satisfy Definition 8. Nevertheless, $\beta' = (A \rightarrow (AB))$ (Fig. 5b) which satisfies Definition 8 is an equivalent representation of β. Hence, β is a chain episode.

We got an equivalent chain representation $\beta' = (A \rightarrow (AB))$ by adding the edge $(v1, v2)$ to $<_\beta$ as indicated in Fig. 5b. On the other hand, if we had tried to generate a chain representation by adding the edge $(v2, v1)$ to $<_\beta$, this would have induced an edge $(v2, v3)$ to maintain transitivity as shown in Fig. 5c. Let β'' denote this new discrete structure. The important thing to observe is that v_2 and v_3 are associated with different event-types under the g_β map. Hence, β and β'' do not share the same set of occurrences and are not equivalent. Specifically, A followed by B followed by A would be an occurrence of β but not of β''. From this example, we make a crucial observation.

Remark 2 If by adding an edge between two nodes mapped to the same event-type of an episode α, an edge is induced between two nodes mapped to different event-types (as per α) to maintain transitivity in the newer episode α', then α and α' are clearly not equivalent as α' is more constrained. In general, given an episode that does not satisfy Definition 8, to ascertain if its still a genuine chain episode OR if it has an equivalent representation satisfying Definition 8, one can do the following. We introduce edges between nodes mapping to the same event-type and eventually impose a total order on every group of nodes (cluster) mapped to the same event-type. This in principle can be done in multiple ways. For each such (total order based) addition of edges within a cluster, we ask if the transitive closure operation to maintain transitivity introduces any new edges between nodes mapped to distinct event-types. If it introduces such an edge for every combination of total orders possible on every cluster of nodes mapped to the same event-type, then the episode cannot be a chain episode. For example $\alpha = ((B \rightarrow A)(B \rightarrow C))$, as discussed further cannot be a chain episode. If there exists a total order imposition on each cluster of nodes such that the subsequent transitive closure does not introduce any edge across clusters, then we have found an equivalent chain episode representation and hence the original episode is genuinely chain. The episode $\beta = ((A \rightarrow B)(A))$ discussed above is a case in point here.

Proposition 1 *Every* 3*-node episode is chain.*

Proof At the 3-node level, for an episode α, the underlying $<_\alpha$ can have either 0, 1, 2 or 3 edges. If the number of edges is 0, it is a parallel episode and if the number of edges is 3, it is a serial episode. For both of these cases, we already know that α is a chain episode. We now show for the remaining two cases. (i) 1 edge in $<_\alpha$: $<_\alpha = \{(v_1, v_2)\}$ is graph

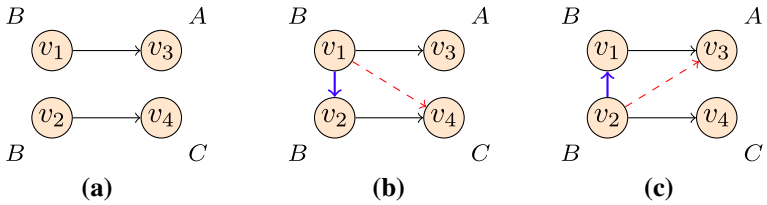


Fig. 6 Illustration of why $((B \rightarrow A)(B \rightarrow C))$ is not a chain episode

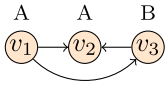
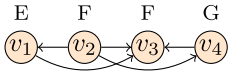
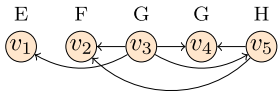
isomorphic to the other possibilities of $<_\alpha$. For this $<_\alpha$, α can be non-chain in two ways, (a) $g_\alpha(v_1) = g_\alpha(v_3)$ and (b) $g_\alpha(v_2) = g_\alpha(v_3)$. For (a), addition of (v_1, v_3) shows that α has an equivalent chain episode representation as explained in the example of Fig. 5. Similarly for (b), addition of (v_3, v_2) makes α a chain episode. (ii) 2 edges in $<_\alpha$: There are essentially two possibilities for $<_\alpha$ here (up to graph isomorphisms), (a) $<_\alpha = \{(v_1, v_2), (v_1, v_3)\}$ and (b) $<_\alpha = \{(v_1, v_3), (v_2, v_3)\}$. Under (a), α can be non-chain if $g_\alpha(v_2) = g_\alpha(v_3)$. By adding either (v_3, v_2) or (v_2, v_3) to $<_\alpha$, we have an equivalent chain episode representation. Under (b), α can be non-chain if $g_\alpha(v_1) = g_\alpha(v_2)$. By adding either (v_1, v_2) or (v_2, v_1) to $<_\alpha$, we have an equivalent chain episode representation. This completes the proof that every 3-node episode is chain. □

Next, consider the following 4-node episode $\alpha = ((B \rightarrow A)(B \rightarrow C))$, shown in Fig. 6a, which does not satisfy Definition 8. We now show that there does not exist an equivalent representation for α in the class of chain episodes. There are two ways in which one can try to generate a chain representation for α. We add either (v_1, v_2) or (v_2, v_1) to $<_\alpha$ to make α a chain episode. If we add $(v1, v2)$ to $<_\alpha$, then to maintain transitivity an extra edge (v_1, v_4) has to be added to $<_\alpha$ (see Fig. 6b). Since v_1 and v_4 are associated with different event-types under the g_α map, from Remark 2, we would not get an equivalent episode. Similarly, adding the edge $(v2, v1)$ to $<_\alpha$ induces the edge $(v2, v3)$ (see Fig. 6c). Again, v_2 and v_3 are associated with A and B (distinct event-types), respectively, in α because of which the new episode will not share the same occurrences as that of α. This shows that this is a 4-node episode that does not have an equivalent representation in the class of chain episodes.

3.1 Representation of chain episodes

Even if one restricts to episodes satisfying Definition 8 (chain episodes), there exists an inherent ambiguity in the representation of the episode pattern as given by Definition 1. To tackle this issue, we first assume a lexicographic ordering on $\mathcal{E}$ and restrict the g-map such that $g(v_1), g(v_2), \ldots g(v_\ell)$ obey this lexicographic order. For example, suppose we have a 5-node episode with 3 of the nodes mapped to A and the remaining 2 mapped to B. Then, $g(v_i)$ must be A for $i = 1, 2, 3$ and B for $i = 4, 5$. Further, since chain episodes are such that the nodes mapped to the same event form a chain, we further impose a special restriction on $<_\alpha$ to avoid further ambiguity. Suppose $v_i, v_{i+1}, \ldots v_{i+m}$ are mapped to the same event-type E. There are $(m + 1)!$ total orders possible among these nodes, each of which would represent the same episode pattern. To avoid this redundancy, we restrict $<_\alpha$ to be such that $v_i <_\alpha v_{i+1} <_\alpha \cdots v_{i+m}$. Table 1 gives 3 examples to illustrate this unique representation. In actual implementations, a chain episode is stored using an array $\alpha.g$ and the adjacency matrix $\alpha.e$. $\alpha.g[i]$ is assigned the value $g_\alpha(v_i)$. The partial order $<_\alpha$, associated with the episode

Table 1 Some example episodes: first row—$(V_\alpha, <_\alpha, g_\alpha)$ notation

$V_\alpha = \{v_1, v_2, v_3\}$ $g_\alpha(v_1) = A, g_\alpha(v_2) = A,$ $g_\alpha(v_3) = A.$ $<_\alpha = \{(v_1, v_2),$ $(v_1, v_3)(v_3, v_2)\}$	$V_\alpha = \{v_1, v_2, v_3, v_4\}$ $g_\alpha(v_1) = E, g_\alpha(v_2) = F,$ $g_\alpha(v_3) = F, g_\alpha(v_4) = G$ $<_\alpha = \{(v_1, v_3), (v_2, v_1)$ $(v_2, v_3), (v_2, v_4), (v_4, v_3)\}$	$V_\alpha = \{v_1, v_2, v_3, v_4, v_5\}$ $g_\alpha(v_1) = E, g_\alpha(v_2) = F, g_\alpha(v_3) =$ $G, g_\alpha(v_4) = G, g_\alpha(v_5) = H$ $<_\alpha = \{(v_1, v_2), (v_1, v_3), (v_1, v_4)$ $(v_1, v_5), (v_2, v_4), (v_2, v_5)\}$
$(A \to B \to A)$	$(F \to (E\, G) \to F)$	$(G \to ((H \to (F\, G))\, E))$
A A B	E F F G	E F G G H
$\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
		

Second row—graphical notation. Third row—array of event-types, $\alpha.g$, which can have repeated event-types. Fourth row—adjacency matrix, $\alpha.e$ corresponding to $<_\alpha$. Last row shows the episode graphically with nodes placed in order. This representation is very useful in explaining candidate generation as we see next

is stored as a binary adjacency matrix, where $\alpha.e[i][j] = 1$ iff $v_i <_\alpha v_j$. Table 1 also gives details of the array $\alpha.g$ and matrix $\alpha.e$ for the episodes considered.

4 Bidirectional evidence

The notion of bidirectional evidence was introduced in [5] for injective episodes. Recall from Definition 6, an injective episode α can be viewed as a partially ordered set of event-types (X^α, R^α). Given this simplified representation, an episode pattern specifies two kinds of pairs of event-types: (a) related under R^α (b) unrelated under the R^α. The occurrence of any episode α by definition (and hence its frequency) captures evidence for only pairs of event-types from X^α which are related under R^α. This is in the sense that for any pair of event-types $E_i, E_j \in X^\alpha$ such that $(E_i, E_j) \in R^\alpha$, any occurrence of α assures that the E_i precedes E_j in time. The time order between pairs of event-types that are unrelated could be anything. Hence, frequency alone does not capture any evidence in the data for unrelated pairs of event-types in an injective episode. This aspect of frequency also manifests itself as a combinatorially explosive number of (partial order) patterns being frequent in spite of being uninteresting as explained in the beginning of Sect. 7.

The notion of bidirectional evidence tackles this issue. The BE-based threshold for injective episodes not only filters such explosive number of uninteresting patterns but also makes mining more efficient. It captures evidence in the data for pairs of unrelated event-types from the episode at hand. The way it does so is as follows. For every pair of unrelated event-types, among the episode occurrences contributing to the frequency it demands the unrelated event-types to occur in either order sufficiently often, as a mark of evidence for the absence of any edge between the two event-types. For instance, for a 3-node injective episode like

$((AB) \rightarrow C)$, BE demands that the event-types A and B occur in either order sufficiently often in the occurrences of $((AB) \rightarrow C)$ for the episode to be flagged as interesting in addition to being frequent. Specifically, the measure is designed such that BE is high when A and B occur in either order sufficiently often. On the other hand, if the data has mostly occurrences of event-types A, B and C in the restricted order of A followed by B followed by C, then even though the frequency of $((AB) \rightarrow C)$ may be high in the data, there is little evidence for an absence of an edge between A and B. In such a case, the BE measure captures this appropriately by flagging a low BE value.

4.1 Bidirectional evidence for chain episodes

A similar issue exists in the class of chain episodes as well which encompasses the class of injective episodes, and the notion of bidirectional evidence can be immediately extended to chain episodes. The idea is for pairs of nodes that are unrelated in the episode, can BE capture evidence in the data for an absence of edge in the pattern. It would do this by checking if the associated event-types occur in either order sufficiently often. Note the associated event-types have to be distinct (by chain episode definition) and hence this check is unambiguous. More formally, we note that working with the (X^α, R^α) notation is not possible for general chain episodes, which can be non-injective in general. In the context of chain episodes, given any episode $\alpha = (V_\alpha, <_\alpha, g_\alpha)$, frequency only captures evidence (in the data) for pairs of nodes of V_α that are related as per $<_\alpha$. Support for pairs of nodes that are unrelated as per $<_\alpha$ is not captured by frequency. For any such pair of nodes (v_i, v_j), we consider it evidence in the data for not constraining (v_i, v_j) under $<_\alpha$ if among the occurrences (h) tracked by the algorithm, $t_{h(v_i)}$ is both less than and greater than $t_{h(v_j)}$ sufficiently often. In other words, we ask for $g_\alpha(v_i)$ and $g_\alpha(v_j)$ to occur in either order sufficiently often among the occurrences tracked by the algorithm. This is not ambiguous, as under chain episodes, $g_\alpha(v_i)$ and $g_\alpha(v_j)$ would be distinct event-types by definition, because v_i and v_j are not related under $<_\alpha$. For the sake of completeness, we discuss the notion of bidirectional evidence for general chain episodes now for the case of data with one event per time tick.

Let $\mathcal{G}^\alpha = \{(i, j) : i, j \in \{1, 2 \ldots N\}, i \neq j, \big((v_i, v_j), (v_j, v_i) \notin <_\alpha\big)\}$ for an N-node episode α. Let f^α denote the number of occurrences (i.e. frequency) of α counted by our algorithm and let f^α_{ij} denote the number of these occurrences where $t_{h(v_i)}$ is less than $t_{h(v_j)}$. Let $p^\alpha_{ij} = f^\alpha_{ij}/f^\alpha$. Note that $p^\alpha_{ji} = 1 - p^\alpha_{ij},\ \forall\ (i, j) \in \mathcal{G}^\alpha$. We would want p^α_{ij} to be close to p^α_{ji} for all $(i, j) \in \mathcal{G}^\alpha$. It is intuitive to expect that closer both p^α_{ij} and p^α_{ji} are to 0.5, the more the evidence for no edge between v_i and v_j. The more this holds for every $i, j \in \mathcal{G}^\alpha$, higher the evidence for the interestingness of the entire partial order pattern. As in [5], to obtain such a figure of merit, (essentially a function of p^α_{ij} which peaks when p_{ij} is close to $1/2$), we choose the entropy of the distribution given by $(p^\alpha_{ij},\ 1 - p^\alpha_{ij})$. Let

$$H^\alpha_{ij} = -p^\alpha_{ij} \log(p^\alpha_{ij}) - (1 - p^\alpha_{ij}) \log(1 - p^\alpha_{ij}). \tag{1}$$

The *bidirectional evidence* of a chain episode α, denoted by $H(\alpha)$, is defined as follows.

$$H(\alpha) = \min_{(i,j) \in \mathcal{G}^\alpha} H^\alpha_{ij}. \tag{2}$$

If $\mathcal{G}^\alpha$ is empty (which will be the case for serial episodes) then, by convention, we take $H(\alpha) = 1$. This notion of bidirectional evidence for chain episodes nicely generalizes the existing notion for injective episodes. For event sequences with multiple event-types at a given time, there would be occurrences of a chain episode where, for a pair of unconstrained nodes (v_i, v_j), $t_{h(v_j)} = t_{h(v_i)}$. We divide the count of such occurrences equally between p_{ij}

and p_{ji} for calculating H^{α}_{ij}. The idea is that if all occurrences h contributing to the frequency are such that $t_{h(v_j)} = t_{h(v_i)}$, then there is maximum evidence in the data for the nodes v_i and v_j to be unrelated and the corresponding H^{α}_{ij} value must be close to the maximum. In such a case, our strategy calculates p_{ij} as almost $1/2$ and hence H^{α}_{ij} is maximum.

4.2 Incorporating H_{th} level-wise

We begin by defining the notion of a maximal subepisode for a general episode. The notion is key in understanding the monotonicity property satisfied by the BE measure exploited further by our candidate generation.

Definition 10 Let $\alpha = (V_\alpha, <_\alpha, g_\alpha)$ be an ℓ-node episode. If β is an episode obtained by first restricting $<_\alpha$ and g_α to a k-node subset of V_α and then suitably renumbering the nodes from 1 to k, β is called a k-node maximal subepisode of α.

For instance, if α is as in Fig. 1d, then its subepisodes in Fig. 3a–c are its maximal subepisodes whereas its subepisodes in Fig. 3d–f are non-maximal. It is easy to verify that *maximal subepisodes of a chain episode will be chain, whereas its non-maximal subepisodes need not be chain episodes* with the subepisode in Fig. 3f as an example.

For injective episodes, it was shown that if an episode α has a BE of $H(\alpha)$ in an event sequence, then at least in the occurrences of α the BE of any of its maximal subepisodes will be at least $H(\alpha)$. Further, in the same set of occurrences the BE of each of its non-maximal subepisodes will be close to zero. This crucial observation was utilized in using BE-based threshold level-wise. We generated an injective episode α as a candidate at level $(\ell + 1)$ only if all its ℓ-node maximal subepisodes were also found to satisfy the frequency and BE thresholds. While extending this idea to chain episodes, one needs to be slightly careful.

Consider the following non-injective chain episode $\alpha = (A \rightarrow (A\ B))$. Each occurrence of this episode is basically an A followed by A and B in either order. The 2-node maximal subepisodes of α are $(A \rightarrow B)$, $(A \rightarrow A)$ and $(A\ B)$. Consider an event sequence consisting only of occurrences of α. Let f be its frequency and say $f/2$ of them comes from A followed by A followed by B and the remaining $f/2$ come from A followed by B followed by A. On such a data, consider a simple algorithm which counts occurrences in a non-overlapped fashion. Such an algorithm would follow a greedy strategy of picking or tracking the earliest occurrences in a non-overlapped fashion.[5] This simple strategy would compute the frequency of $(A\ B)$ as f but its BE would be computed to be zero because each occurrence it tracks would be an A followed by a B. This is because, the algorithm would look for the earliest A and earliest B, but in the process tracks the A corresponding to v_1 in α which precedes the occurrence of A and B happening in either order. We note that $(A\ B)$ was obtained by dropping v_1 from α and this ended up in the algorithm aliasing the A corresponding to v_1 of α to the A in $(A\ B)$. We would have ideally liked the algorithm to track the A (in the event sequence) corresponding to v_2 in α while tracking occurrences of $(A\ B)$. This problem mainly occurred because v_1 was not the last node in α mapped to A. Generalizing this observation, we have the following important property.

Property 1 *For a general $(\ell + 1)$-node chain episode α, one can only guarantee that every ℓ-node maximal subepisode of α, namely β, obtained by dropping a node v_i which is the last node among all nodes in α mapping to $g_\alpha(v_i)$, will at least have a BE of $H(\alpha)$* in the

[5] We will elaborate later in Sect. 6 on how finite state automata can be used to track occurrences of episodes and a strategy for counting with expiry time constraints.

occurrences of α. *In any occurrence of α, in case of any ambiguity in the choice of the occurrence of β, the earliest occurrence of β is considered towards BE computation.*

Proof Consider any occurrence h_α (a set of events from the input event sequence) of α. Consider two nodes $v_{i'}$ and $v_{j'}$ in V_β with no edge between them as per $<_\beta$. Since β is also a chain episode, note that $g_\beta(v_{i'})$ and $g_\beta(v_{j'})$ are distinct event-types. Since β is a maximal subepisode of α, there exist two associated nodes v_i and v_j in α (images of $v_{i'}$ and $v_{j'}$ under the $f_{\beta\alpha}(.)$-map as per Definition 3), which map to the same two event-types under g_α-map with no edge between them in $<_\alpha$. If there were an edge between them in $<_\alpha$, β wouldn't be a maximal subepisode of α. Consider the earliest occurrence of β from h_α, the set of event-types constituting the occurrence of α. Let us denote it by h_β. The image of $v_{i'}$ and $v_{j'}$ under the h_β map would be the same two events obtained by applying h_α on v_i and v_j. This is guaranteed only because β is a maximal subepisode obtained by dropping the last node among nodes mapping to the same event-type from α. Further, this would mean the contribution of the constructed occurrence h_β to $H^\beta_{i'j'}$ would be exactly identical to the contribution of h_α towards H^α_{ij}. Considering all occurrences of α now, for every $(i', j') \in \mathcal{G}^\beta$, there exists an $(i, j) \in \mathcal{G}^\alpha$, such that $H^\beta_{i'j'} = H^\alpha_{ij}$. Using Eq. 2, we finally have $H(\beta) \geq H(\alpha)$. □

Any non-maximal chain subepisode β of α obtained by dropping an edge between nodes v_i and v_j which are mapped to two different nodes. The BE of such an episode would be typically low when computed in the occurrences of α. This is because (i, j) now belongs to $\mathcal{G}^\beta$ even though is absent from $\mathcal{G}^\alpha$. Suppose the dropped edge is from v_i to v_j. Then in the occurrences of α, $g(v_i)$ always precedes $g(v_j)$. This would imply that H^β_{ij} is close to zero and hence $H(\beta)$ would be close to zero. This would mean such non-maximal subepisodes on account of having low BE would get eliminated right from the lower levels when BE is also applied at each level to assess interestingness in addition to frequency.

We note that Property 1 is not a strict but rather a restrictive monotonicity property. In situations where most of the maximal subepisode occurrences of a significant frequent pattern come from occurrences of the significant parent pattern, this property can be very useful to employ during candidate generation in the level-wise procedure. Also, from a computational perspective to combat the inherent combinatorial explosion in partial order mining employing a BE-based threshold can help us prune a lot of uninteresting patterns right from the lower levels. Also in more general real situations, many of the occurrences of lower-sized maximal subepisodes of an interesting pattern may come up from occurrences outside the interesting pattern's occurrences. This could for instance be due to the presence of random occurrences of events involving the episodes, in which case these small-sized subepisodes tend to have a high BE and hence will contribute to the potential generation of an interesting pattern as a candidate. On the other hand, *under tight expiry time constraints*, the occurrences of larger sized maximal subepisodes may mostly come from the significant interesting pattern and hence will also have a high BE (by Property 1) and contribute to the generation of the significant interesting pattern. Hence, employing BE thresholds in the levelwise search can be a meaningful and useful strategy. We next describe our novel and non-trivial candidate generation which fully exploits this monotonicity property.

5 Candidate generation

The candidate generation, at level $(\ell + 1)$ takes as input $\mathcal{F}_\ell$, the set of ℓ-node frequent chain episodes and outputs $\mathcal{C}_{\ell+1}$, a set of $(\ell + 1)$-node candidate chain episodes.

5.1 Steps in candidate generation

Each $(\ell+1)$-node candidate in $\mathcal{C}_{\ell+1}$ is generated by combining two suitable ℓ-node frequent chain episodes (out of $\mathcal{F}_\ell$). The method involves three main steps:

1. Picking suitable pairs of episodes from $\mathcal{F}_\ell$.
2. Combining each such pair to generate up to three episodes of size $\ell+1$ which we call *potential* candidates.
3. Finally constructing $\mathcal{C}_{\ell+1}$ by retaining only those potential candidates for which each of their ℓ-node subepisodes are frequent.

The steps in candidate generation for chain episodes resemble the ones proposed in the context of injective episodes [5] and can be viewed as nice generalizations of the injective episode case. We comment on similarities and differences between chain episode and injective episode case at the end of each of the three subsections to follow.

5.1.1 Pairs of ℓ-node episodes that can be combined

Each episode $\alpha_1 = (\{v_1, v_2, \ldots v_\ell\}, <_{\alpha_1}, g_{\alpha_1})$ from $\mathcal{F}_\ell$ is combined with two types of episodes. The first of this type of episodes $(\alpha_2 = (\{v_1, v_2, \ldots v_\ell\}, <_{\alpha_2}, g_{\alpha_2}))$ are such that the following hold:

1. $g_{\alpha_1}(v_i) = g_{\alpha_2}(v_i)\ \forall i = 1, \ldots (\ell-1)$,
2. $<_{\alpha_1} |_{\{v_1, v_2, \ldots v_{\ell-1}\}} = <_{\alpha_2} |_{\{v_1, v_2, \ldots v_{\ell-1}\}}$, that is, the restriction of $<_{\alpha_1}$ on the first $(\ell-1)$ nodes of α_1 is same as the restriction of $<_{\alpha_2}$ on the same set. In other words, $v_i <_{\alpha_1} v_j$ if and only if $v_i <_{\alpha_2} v_j$ for $i, j = 1, \ldots, (\ell-1)$.
3. $g_{\alpha_1}(v_\ell) < g_{\alpha_2}(v_\ell)$.

Both α_1 and α_2 are in their respective unambiguous representations. Thus, we combine α_1 and α_2 if the subepisodes obtained by dropping v_ℓ (their last node) from α_1 and α_2 are the same. For example, $\alpha_1 = (B \to A \to C \to B)$ and $\alpha_2 = (B \to A \to D \to B)$ as shown in Fig. 7 share the same 3-node subepisode $(B \to A \to B)$ on dropping their last node v_4. Similarly, the two episodes $\alpha_1 = (((B \to A)D) \to B)$ and $\alpha_2 = (B \to A \to (B\,E))$ shown in Fig. 8 share the same 3-node subepisode $(B \to A \to B)$ on dropping their last node v_4.

Before describing the second type of combinable episodes, we need the following definition.

Definition 11 The rth node of an ℓ-node episode α is the last node in V_α which maps to an event-type different from $g_\alpha(v_\ell)$.

As an example, for the 5-node serial episode $(A \to B \to B \to C \to C)$, the rth node is v_3 (i.e. $r = 3$). Also note that for an ℓ-node episode r is at most $(\ell-1)$.

Given an episode α_1, the second type of episodes $(\alpha_2 = (\{v_1, v_2, \ldots v_\ell\}, <_{\alpha_2}, g_{\alpha_2}))$ that can be combined with it are such that the following hold:

1. $g_{\alpha_1}(v_i) = g_{\alpha_2}(v_i)\ \forall i = 1, \ldots (r-1)$, $g_{\alpha_1}(v_i) = g_{\alpha_2}(v_{i-1})\ \forall i = (r+1), \ldots \ell$, where **r refers to the appropriate node (as per Definition 11) of 1**.
2. Consider the restriction of $<_{\alpha_1}$ to $\{v_1, v_2, \ldots v_r, v_{r+1}, \ldots v_\ell\}$ and renumber the nodes $v_{r+1}, \ldots v_\ell$ to $v_r, \ldots v_{\ell-1}$ without affecting the order among the nodes. This ordered set must be identical to $<_{\alpha_2} |_{\{v_1, v_2, \ldots v_{\ell-1}\}}$. This means the restriction of $<_{\alpha_1}$ on the $(\ell-1)$ nodes of α_1 by dropping its rth node is same as the restriction of $<_{\alpha_2}$ on the first $(\ell-1)$ nodes of α_2.

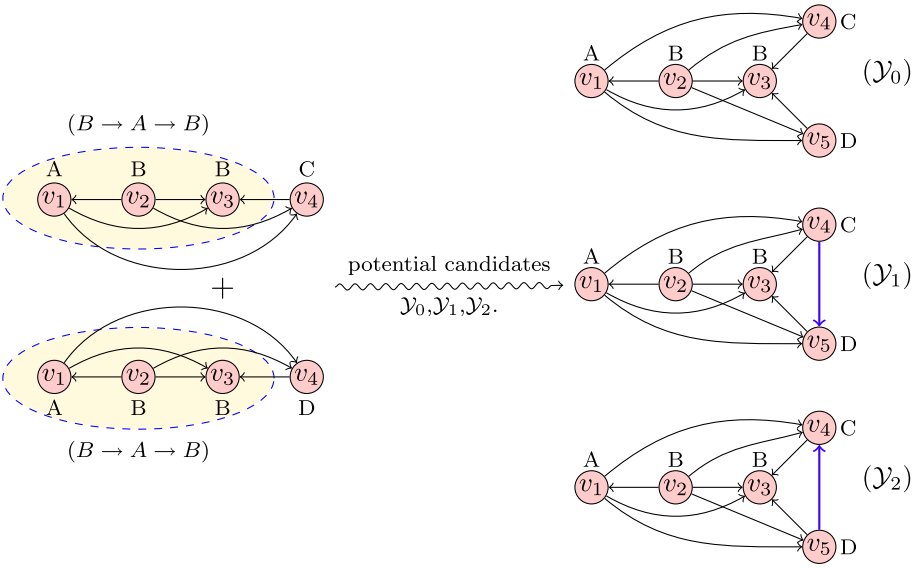


Fig. 7 α_2 is of first type: illustration where all 3 combinations come up. $\alpha_1 = (B \to A \to C \to B)$ and $\alpha_2 = (B \to A \to D \to B)$

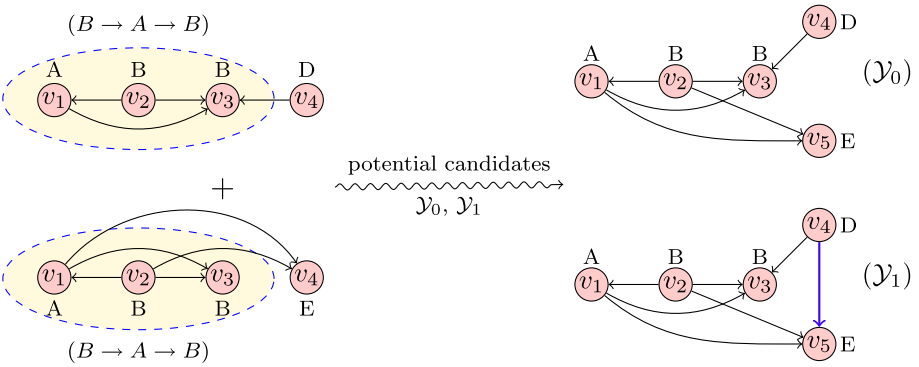


Fig. 8 α_2 is of first type: illustration where 2 combinations come up. $\alpha_1 = (((B \to A)D) \to B)$ and $\alpha_2 = (B \to A \to (B\,E))$

3. $g_{\alpha_2}(v_\ell) = g_{\alpha_2}(v_{\ell-1})$, $(= g_{\alpha_1}(v_\ell)$ as well, as per condition 1 above).

The first two conditions above basically means that the subepisode obtained by dropping the rth node of α_1 is identical to the subepisode of α_2 obtained by dropping the last node of α_2. As an example, consider $\alpha_1 = (A(C \to C)(E \to E))$ and $\alpha_2 = (AC(E \to E \to E))$ as in Fig. 9. Observe that the rth node of α_1 is v_3 and on dropping v_3 from α_1 and v_5 from α_2, we obtain the same subepisode, namely $(AC(E \to E))$, which is shown highlighted.

Note that if an episode α_1 is combined with only the first type of episodes (α_1), the candidate generation exactly boils down to that of injective episodes discussed in [5].

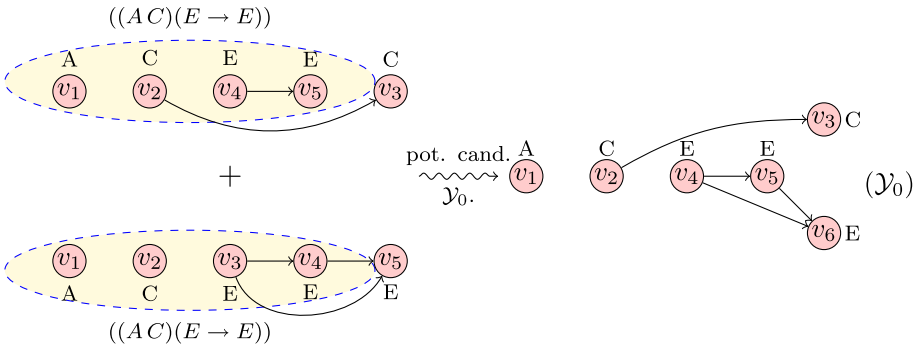


Fig. 9 α_2 is of second type: Illustration where only $\mathcal{Y}_0$ combination is valid. $\alpha_1 = (A(C \to C)(E \to E))$ and $\alpha_2 = (AC(E \to E \to E))$

5.1.2 Finding potential candidates

Now we explain how to combine α_1 with both types of episodes (α_2). For the first type of episodes, we first build an $(\ell+1)$-node episode $\mathcal{Y}_0 = (V_\mathcal{Y}, <_{\mathcal{Y}_0}, g_\mathcal{Y})$ from α_1 and α_2. We take $V_\mathcal{Y} = \{v_1, \ldots, v_\ell, v_{\ell+1}\}$. The partial order relation, $<_{\mathcal{Y}_0}$, on $V_\mathcal{Y}$ is defined as follows: $v_i <_{\mathcal{Y}_0} v_j$ iff $v_i <_{\alpha_1} v_j$ for $i, j = 1, \ldots \ell$. Also, for $i = 1, 2 \ldots (\ell-1)$, we have $v_i <_{\mathcal{Y}_0} v_{\ell+1}$ iff $v_i <_{\alpha_2} v_\ell$, and $v_{\ell+1} <_{\mathcal{Y}_0} v_i$ iff $v_\ell <_{\alpha_2} v_i$. The $g_\mathcal{Y}$ map from $V_\mathcal{Y}$ to $\mathcal{E}$ is such that $g_\mathcal{Y}(v_i) = g_{\alpha_1}(v_i)$ for $i = 1, \ldots \ell$ and $g_\mathcal{Y}(v_{\ell+1}) = g_{\alpha_2}(v_\ell)$. As an example of this construction, again consider the two 4-node episodes of Fig. 7. Their $\mathcal{Y}_0$ combination is the 5-node episode $(B \to A \to (C\,D) \to B)$ as indicated in Fig. 7.

We first construct 3 possible episodes from α_1 and α_2: $\mathcal{Y}_0$ (as explained above), $\mathcal{Y}_1 = (V_\mathcal{Y}, <_{\mathcal{Y}_1}, g_\mathcal{Y})$ and $\mathcal{Y}_2 = (V_\mathcal{Y}, <_{\mathcal{Y}_2}, g_\mathcal{Y})$. Here $<_{\mathcal{Y}_1} = <_{\mathcal{Y}_0} \cup (v_\ell, v_{\ell+1})$ and $<_{\mathcal{Y}_2} = <_{\mathcal{Y}_0} \cup (v_{\ell+1}, v_\ell)$. We note here that the three possible episodes $\mathcal{Y}_0$, $\mathcal{Y}_1$ and $\mathcal{Y}_2$ differ only in the respective partial orders: $<_{\mathcal{Y}_1}$ and $<_{\mathcal{Y}_2}$ are obtained by adding one new edge each to $<_{\mathcal{Y}_0}$. An episode $(V_\mathcal{Y}, <_{\mathcal{Y}_i}, g_\mathcal{Y})$ is generated as a *potential candidate* iff $<_{\mathcal{Y}_i}$ is a partial order. (For the remainder of the section, we refer to $(V_\mathcal{Y}, <_{\mathcal{Y}_i}, g_\mathcal{Y})$ as the $\mathcal{Y}_i$ combination of two combinable chain episodes α_1 and α_2). Figure 7 demonstrates a case where all the three combinations are potential candidates. On the other hand, Fig. 8 illustrates an example where exactly two combinations are potential candidates.

For the second type of episodes, to start off, we again build an $(\ell+1)$-node episode $\mathcal{Y}_0 = (V_\mathcal{Y}, <_{\mathcal{Y}_0}, g_\mathcal{Y})$ from α_1 and α_2. We take $V_\mathcal{Y} = \{v_1, \ldots, v_\ell, v_{\ell+1}\}$. The partial order relation, $<_{\mathcal{Y}_0}$, on $V_\mathcal{Y}$ is defined as follows: $v_i <_{\mathcal{Y}_0} v_j$ iff $v_i <_{\alpha_1} v_j$ for $i, j = 1, \ldots \ell$. Also, for $i = 1, 2 \ldots (r-1)$, we have $v_i <_{\mathcal{Y}_0} v_{\ell+1}$ iff $v_i <_{\alpha_2} v_\ell$, and $v_{\ell+1} <_{\mathcal{Y}_0} v_i$ iff $v_\ell <_{\alpha_2} v_i$. Also, for $i = (r+1), \ldots \ell$, we have $v_i <_{\mathcal{Y}_0} v_{\ell+1}$ iff $v_{i-1} <_{\alpha_2} v_\ell$, and $v_{\ell+1} <_{\mathcal{Y}_0} v_i$ iff $v_\ell <_{\alpha_2} v_{i-1}$. The $g_\mathcal{Y}$ map from $V_\mathcal{Y}$ to $\mathcal{E}$ is such that $g_\mathcal{Y}(v_i) = g_{\alpha_1}(v_i)$ for $i = 1, \ldots \ell$ and $g_\mathcal{Y}(v_{\ell+1}) = g_{\alpha_2}(v_\ell)$. As an example of this construction, again consider the two 5-node episodes of Fig. 9. Their $\mathcal{Y}_0$ combination is the 5-node episode $(A(C \to C)(E \to E))$ as indicated in Fig. 9.

We again construct 3 possible episodes from α_1 and α_2: $\mathcal{Y}_0$ (as explained above), $\mathcal{Y}_1 = (V_\mathcal{Y}, <_{\mathcal{Y}_1}, g_\mathcal{Y})$ and $\mathcal{Y}_2 = (V_\mathcal{Y}, <_{\mathcal{Y}_2}, g_\mathcal{Y})$. Here $<_{\mathcal{Y}_1} = <_{\mathcal{Y}_0} \cup (v_r, v_{\ell+1})$ and $<_{\mathcal{Y}_2} = <_{\mathcal{Y}_0} \cup (v_{\ell+1}, v_r)$. We note here that the three possible episodes $\mathcal{Y}_0$, $\mathcal{Y}_1$ and $\mathcal{Y}_2$ differ only in the respective partial orders: $<_{\mathcal{Y}_1}$ and $<_{\mathcal{Y}_2}$ are obtained by adding one new edge each to $<_{\mathcal{Y}_0}$. An episode $(V_\mathcal{Y}, <_{\mathcal{Y}_i}, g_\mathcal{Y})$ is generated as a *potential candidate* iff $<_{\mathcal{Y}_i}$ is a partial order. (For the remainder of the section, we refer to $(V_\mathcal{Y}, <_{\mathcal{Y}_i}, g_\mathcal{Y})$ as the $\mathcal{Y}_i$ combination of two combinable

chain episodes α_1 and α_2). Figure 9 demonstrates a case where only $\mathcal{Y}_0$ combination is a potential candidate.

To verify that $<_{\mathcal{Y}_i}$ is a valid partial order, for $i = 0, 1, 2$ we need to check the antisymmetry and transitivity of each $<_{\mathcal{Y}_i}$. For the first type of combination, α_1 and α_2 share the same $(\ell-1)$-node subepisode by dropping their last node v_ℓ and because of the way $\mathcal{Y}_0$ is constructed, antisymmetry of each $<_{\mathcal{Y}_i}$ is immediate. Recall that in the second type of combination, the same $(\ell - 1)$-node subepisode obtained by dropping the rth-node of α_1 and last node of α_2. This fact and the way $\mathcal{Y}_0$ is constructed renders each $<_{\mathcal{Y}_i}$ antisymmetric. For the same reasons, to check transitivity of each $<_{\mathcal{Y}_i}$, it is enough to check transitivity for all size 3-subsets of $V_{\mathcal{Y}}$ of the form $\{v_\ell, v_{\ell+1}, v_i \ : \ 1 \leq i \leq (\ell - 1)\}$ for the first type of combination and $\{v_r, v_{\ell+1}, v_i \ : \ i = 1 \ldots (r-1), (r+1), \ldots, \ell\}$ for the second type of combination. Hence, the transitivity check is $\mathcal{O}(\ell)$. Since each $<_{\mathcal{Y}_i}$ differs in at most one edge, one can check for transitivity of $<_{\mathcal{Y}_i}$ in a more intelligent way as described in Sect. 5.3.

The 3 combinations proposed here are similar to the 3 combinations considered for candidate generation in the case of injective episodes [5]. In the above described procedure, if we generate potential candidates by only combining an α_1 with the first type of episodes (α_2), (at all levels), the candidate generation would be specialized to generate only injective episodes.

5.1.3 Forming the final candidate episodes

The last step is to decide which of the potential candidates are actual candidates and hence can be placed in $\mathcal{C}_{\ell+1}$. Recall from Sect. 4.2 that we generate an $(\ell + 1)$-node episode as a candidate episode if all ℓ-node maximal subepisodes of α obtained by dropping a node v_i which is the last node among all nodes in α mapping to $g_\alpha(v_i)$ are also found to be interesting at level ℓ. The way we have formed a potential candidate guarantees that the two such maximal ℓ-node subepisodes of α are already found in $\mathcal{F}_\ell$. Specifically, these two subepisodes are obtained by dropping a node from the last two set of nodes of α which map to the same event-type. For example, if $\{(A\,(C \rightarrow C)\,(E \rightarrow E \rightarrow E)\}$ is the potential candidate as on the right-hand side of Fig. 9, then we already know that the maximal subepisodes obtained by dropping v_6 and v_3 are already frequent. Hence in this step, we check for the existence of the remaining such maximal subepisodes in $\mathcal{F}_\ell$ to finally place a potential candidate in $\mathcal{C}_{\ell+1}$. For $\{(A\,(C \rightarrow C)\,(E \rightarrow E \rightarrow E)\}$, we only need to check for the existence of the subepisode obtained by dropping v_1 in $\mathcal{F}_4$.

For the injective episodes case, one blindly checks for the existence of all maximal subepisodes of a potential candidate. The above specialized maximal subepisodes check when applied to injective episode checks the existence of all maximal subepisodes as every node maps to a unique event-type in this case. Hence, the checks for forming the final candidate episode for chain episodes goes through as it is for injective episodes.

As emphasized earlier, the above proposed candidate generation is very different from that of the existing apriori-based method [25,27]. *Please refer to* Appendix A *for a detailed comparison with the existing method.* A detailed correctness proof of the proposed candidate generation is provided next.

5.2 Correctness proof of candidate generation

In this section, we show that: (i)every frequent chain episode is generated by our candidate generation algorithm. (ii) a given chain episode is generated only once in the algorithm.

Theorem 1 *Every frequent*[6] *chain episode would belong to the set of candidates generated.*

Proof We show this by induction on the size of the episode. At level one, the set of candidates contain all the one node episodes and hence contains all the frequent one node episodes. Now suppose at level ℓ, all frequent chain episodes of size ℓ are indeed generated as candidates. If an $(\ell+1)$-node chain episode $\alpha = (V_\alpha, <_\alpha, g_\alpha))$ is frequent, then all its maximal subepisodes (obtained by dropping the last node among all nodes mapped to the same event-type) are frequent. We consider two cases here.

Case (i) $g_\alpha(v_\ell) < g_\alpha(v_{\ell+1})$: The maximal ℓ-node subepisodes α_1 and α_2 obtained by dropping the nodes $v_{\ell+1}$ and v_ℓ, respectively, are also chain episodes and are frequent and hence generated at level ℓ (as per the induction hypothesis). The important point to note is that the $(\ell-1)$-node subepisodes obtained by dropping the last nodes of these two episodes are the same. Specifically for α_1 here, α_2 is an episode of type 1 (as explained earlier in Sect. 5.1.1). Hence, the candidate generation method would combine these two frequent chain episodes. *Any chain episode α with $g_\alpha(v_\ell) < g_\alpha(v_{\ell+1})$ would be either a $\mathcal{Y}_0$, $\mathcal{Y}_1$ or $\mathcal{Y}_2$ combination of its two maximal subepisodes α_1 and α_2 obtained by dropping the last and last but-one nodes, respectively.*

Case (ii) $g_\alpha(v_\ell) = g_\alpha(v_{\ell+1})$: The maximal ℓ-node subepisodes α_1 and α_2 obtained by dropping the nodes $v_{\ell+1}$ and the rth node of α, respectively, are also chain episodes and are frequent and hence generated at level ℓ (as per the induction hypothesis). Note that the $(\ell-1)$-node subepisode obtained by dropping the last node of α_2 and the rth node of α_1 are the same. Further in this case since $g_{\alpha_2}(v_\ell) = g_{\alpha_2}(v_{\ell-1})$ (condition 3 for episode α_2 of type 2 as per Sect. 5.1.1) is satisfied. Hence, in this case for episode α_1, α_2 is of type 2 (as explained in Sect. 5.1.1), which means the candidate generation method would combine these two frequent episodes. Even in this case, the chain episode α would be either a $\mathcal{Y}_0$, $\mathcal{Y}_1$ or $\mathcal{Y}_2$ combination of its two maximal subepisodes α_1 and α_2 obtained by dropping the last and the rth node of α, respectively.

In both the above cases, $<_\alpha$ is also a valid partial order.

Hence in either case α would be a potential candidate. Further, since all its appropriate ℓ-node maximal subepisodes are frequent and chain, they would all be generated at level ℓ by induction hypothesis. Hence α would be finally output in the set of final candidates generated by our method. □

Theorem 2 *The candidate generation algorithm does not generate any duplicate discrete structures.*

Proof It is easy to see from our candidate generation method that episodes generated from a given pair (α_1, α_2) of ℓ-node episodes are all different. This is because under the case when $g_\alpha(v_\ell) < g_\alpha(v_{\ell+1})$ the three possible combinations differ with respect to the way v_ℓ and $v_{\ell+1}$ are related and hence are different. When $g_\alpha(v_\ell) = g_\alpha(v_{\ell+1})$, the three possible combinations differ with respect to the way v_r and $v_{\ell+1}$ are related and hence are different. Hence we need to consider the case when the same candidate is generated from two different pairs of episodes.

Let α and α' represent the two generated episodes from two distinct pairs (α_1, α_2) and (α'_1, α'_2), respectively. Suppose α and α' generate the same candidate. We have four possibilities here depending on whether α_2 and α'_2 are of type 1 or type 2. If one of α_2 and α'_2 is of type 1 and the other of type 2, then we now show the two generated episodes α and α' are distinct. Suppose they are same, then we have $g_{\mathcal{Y}} = g'_{\mathcal{Y}}$. Without loss of generality, let us

[6] By frequent here, we mean episodes which satisfy both the frequency and BE thresholds.

assume, α_2 is of type 1 and α'_2 is of type 2. Then, we have $g_{\mathcal{Y}}(v_\ell < g_{\mathcal{Y}}(v_{\ell+1})$ ($\because \alpha_2$ is of type 1). Also, because α'_2 is of type 2, we have many equalities coming ahead. Since r is at most $(\ell-1)$, we have $g_{\alpha'_1}(v_\ell) = g_{\alpha'_2}(v_{\ell-1})$. Further, condition 3 for a type 2 combination says $g_{\alpha'_2}(v_{\ell-1}) = g_{\alpha'_2}(v_\ell)$. We also have $g_{\alpha'_2}(v_\ell) = g'_{\mathcal{Y}}(v_{\ell+1})$ and $g_{\alpha'_1}(v_\ell) = g'_{\mathcal{Y}}(v_\ell)$. Combining the preceding 4 inequalities, we have $g'_{\mathcal{Y}}(v_\ell = g'_{\mathcal{Y}}(v_{\ell+1})$. But this contradicts the equality of $g_{\mathcal{Y}}$ and $g'_{\mathcal{Y}}$ and hence the generated episodes must be distinct.

Let us consider the case of both α'_2 and α_2 being of type 2. Let r_1 and r'_1 denote the rth node of α_1 and α'_1, respectively. We now show that if r_1 and r'_1 are distinct then the generated episodes will also be distinct. Suppose r_1 and r'_1 are different and α and α' are same. This means $g_{\mathcal{Y}} = g'_{\mathcal{Y}}$. By the way episodes are combined, we have $g_{\alpha_1} = g_{\alpha'_1}$. Without loss of generality, let us assume $r_1 < r'_1$. By the defn. of r_1, we have $\forall i > r_1$, $g_{\alpha_1}(v_i)$ maps to the same event-type. On the other hand, the defn of r'_1 implies that $g_{\alpha'_1}(v_i)$ maps to the same event-type, say E, $\forall i > r'_1$. For i s.t. $r_1 < i \leq r'_1$, each $g_{\alpha'_1}(v_i)$ maps to an event-type different from E. This clearly contradicts $g_{\alpha_1} = g_{\alpha'_1}$ and hence if r_1 and r'_1 are different, α and α' must be distinct. Hence now we are left with the proof of unique candidates being generated under that subcase of $r_1 = r'_1$. However, the proof of this subcase is very similar to the last case of both α'_2 and α_2 being of type 1. Therefore, we next present the proof of only the last case.

For the generated episodes to be the same, both of them should come up as some $\mathcal{Y}_i$ combination. Without loss of generality, we consider the case when both these candidates come up as $\mathcal{Y}_0$ combination. We have $\alpha = (V_{\mathcal{Y}}, <_{\mathcal{Y}_0}, g_{\mathcal{Y}})$ and $\alpha' = (V_{\mathcal{Y}}, <'_{\mathcal{Y}_0}, g'_{\mathcal{Y}})$, where $V_{\mathcal{Y}} = \{v_1, v_2, \ldots v_{\ell+1}\}$, $<_{\mathcal{Y}_0}$, $<'_{\mathcal{Y}_0}$, $g_{\mathcal{Y}}$ and $g'_{\mathcal{Y}}$ are as explained in Sect. 5.1.2. Since the generated candidates α and α' are the same, we have (i)$g_{\mathcal{Y}} = g'_{\mathcal{Y}}$ and (ii)$<_{\mathcal{Y}_0} = <'_{\mathcal{Y}_0}$.

Recall from the conditions for forming candidates that for $i = 1, \ldots, (\ell-1)$, $g_{\mathcal{Y}}(v_i) = g_{\alpha_1}(v_i) = g_{\alpha_2}(v_i)$. The second equality is because the restriction of g_{α_1} and g_{α_2} on their first $(\ell-1)$ nodes are identical. Also, $g_{\mathcal{Y}}(v_\ell) = g_{\alpha_1}(v_\ell)$ and $g_{\mathcal{Y}}(v_{\ell+1}) = g_{\alpha_2}(v_\ell)$. An analogous thing holds for $g'_{\mathcal{Y}}$, $g_{\alpha'_1}$, $g_{\alpha'_2}$. This, along with $g_{\mathcal{Y}}(v_i) = g'_{\mathcal{Y}}(v_i)$ for $i = 1, \ldots, \ell+1$ ((i) above) would mean $g_{\alpha_1} = g_{\alpha'_1}$ and $g_{\alpha_2} = g_{\alpha'_2}$. Thus if the pairs (α_1, α_2) and (α'_1, α'_2) are to be different, then the partial orders have to be different.

We have $v_i <_{\alpha_1} v_j \iff v_i <_{\mathcal{Y}_0} v_j \iff v_i <'_{\mathcal{Y}_0} v_j \iff v_i <_{\alpha'_1} v_j$ for $i = 1, \ldots, \ell$. The first and last equivalence come from the conditions for forming $\mathcal{Y}_0$. The second equivalence is because $<_{\mathcal{Y}_0} = <'_{\mathcal{Y}_0}$ ((ii) above). This implies that $<_{\alpha_1} = <_{\alpha'_1}$. For $i = 1, \ldots, (\ell-1)$, we have $v_i <_{\alpha_2} v_j \iff v_i <_{\alpha_1} v_j \iff v_i <_{\alpha'_1} v_j \iff v_i <_{\alpha'_2} v_j$. The first and last equivalence is because the restriction of the partial orders of two combinable ℓ-node episodes on their first $(\ell-1)$ nodes are same. The second equivalence is from what we have just concluded, that $<_{\alpha_1} = <_{\alpha'_1}$. Also for $i = 1, \ldots, (\ell-1)$, we have $v_i <_{\alpha_2} v_\ell \iff v_i <_{\mathcal{Y}_0} v_{\ell+1} \iff v_i <'_{\mathcal{Y}_0} v_{\ell+1} \iff v_i <_{\alpha'_2} v_\ell$. The first and last equivalences are from the conditions for forming $\mathcal{Y}_0$. The second equivalence is because of (ii) above. We can similarly show that $v_\ell <_{\alpha_2} v_i \iff v_\ell <_{\alpha'_2} v_i$ for $i = 1, \ldots, (\ell-1)$. This altogether would now imply that $<_{\alpha_2} = <_{\alpha'_2}$.

From the preceding two paragraphs, we have come to a point where $\alpha_1 = \alpha'_1$ and $\alpha_2 = \alpha'_2$. This means the pairs of episodes we started off with are not distinct which is a contradiction. Using similar arguments, we can show that no $\mathcal{Y}_1$ (or $\mathcal{Y}_2$) combination of two distinct pairs of combinable chain episodes can give the same episodes. The case of both α_2 and α'_2 being to type 2 with r_1 and r'_1 being the same can be handled on very similar lines.

This completes the proof that every candidate chain episode is uniquely generated. Thus we can see that our algorithm does not generate any candidate twice. □

Table 2 The naive checks for transitivity

Check id	Type of transitivity check	$\mathcal{Y}_0$	$\mathcal{Y}_1$	$\mathcal{Y}_2$
(a)	$(v_\ell, v_{\ell+1}), (v_{\ell+1}, z) \in <_{\mathcal{Y}_i} \implies (v_\ell, z) \in <_{\mathcal{Y}_i}$	No	Yes	No
(b)	$(v_\ell, z), (z, v_{\ell+1}) \in <_{\mathcal{Y}_i} \implies (v_\ell, v_{\ell+1}) \in <_{\mathcal{Y}_i}$	Yes	No	Yes
(c)	$(v_{\ell+1}, v_\ell), (v_\ell, z) \in <_{\mathcal{Y}_i} \implies (v_{\ell+1}, z) \in <_{\mathcal{Y}_i}$	No	No	Yes
(d)	$(v_{\ell+1}, z), (z, v_\ell) \in <_{\mathcal{Y}_i} \implies (v_{\ell+1}, v_\ell) \in <_{\mathcal{Y}_i}$	Yes	Yes	No
(e)	$(z, v_\ell), (v_\ell, v_{\ell+1}) \in <_{\mathcal{Y}_i} \implies (z, v_{\ell+1}) \in <_{\mathcal{Y}_i}$	No	Yes	No
(f)	$(z, v_{\ell+1}), (v_{\ell+1}, v_\ell) \in <_{\mathcal{Y}_i} \implies (z, v_\ell) \in <_{\mathcal{Y}_i}$	No	No	Yes

Some of them are redundant which is indicated by 'no' in the appropriate column

Table 3 Classification of the nodes in α

Sl. no.	Node type for z	Relation with v_ℓ and $v_{\ell+1}$
1	(1)	(v_ℓ, z) and $(z, v_{\ell+1})$ belong to $<_{\mathcal{Y}_0}$
2	$(1')$	$(v_{\ell+1}, z)$ and (z, v_ℓ) belong to $<_{\mathcal{Y}_0}$
3	(2)	$(v_\ell, z) \in <_{\mathcal{Y}_0}$, no edge between z and $v_{\ell+1}$
4	$(2')$	$(z, v_\ell) \in <_{\mathcal{Y}_0}$, no edge between z and $v_{\ell+1}$
5	(3)	$(v_{\ell+1}, z) \in <_{\mathcal{Y}_0}$, no edge between z and v_ℓ
6	$(3')$	$(z, v_{\ell+1}) \in <_{\mathcal{Y}_0}$, no edge between z and v_ℓ
7	(4)	$(z, v_{\ell+1})$ and (z, v_ℓ) belong to $<_{\mathcal{Y}_0}$
8	$(4')$	$(v_{\ell+1}, z)$ and (v_ℓ, z) belong to $<_{\mathcal{Y}_0}$
9	$(4'')$	Neither connected to v_ℓ nor $v_{\ell+1}$

5.3 Efficient checks for transitivity

We will describe these efficient checks for the first type of combinations where α_1 and α_2 share the same subepisode on dropping their last nodes. The checks for the second type of combination is just a minor modification of the first type which will be indicated at the end of this subsection. As seen in the previous section, to check for the transitivity of $\mathcal{Y}_0$, $\mathcal{Y}_1$ and $\mathcal{Y}_2$ combinations of two combinable frequent episodes, we need to check only for all size-3 subsets of $V_\mathcal{Y}$ that are of the form $\{v_\ell, v_{\ell+1}, v_i \ : \ 1 \leq i \leq (\ell-1)\}$. This would mean performing 6 checks for every tuple $(v_\ell, v_{\ell+1}, v_i), i \in \{1 \ldots (\ell-1)\}$ as listed in Table 2 and as adopted in [5]. One can check for transitivity of all the three combinations, $\mathcal{Y}_0$, $\mathcal{Y}_1$ and $\mathcal{Y}_2$ more efficiently mainly because *these combinations differ with respect to only one edge among themselves*. The more efficient algorithm for transitivity check to be presented now also has worst case complexity $\mathcal{O}(\ell)$. However, the actual number of checks would be less thus contributing to the efficiency of candidate generation.

Recall that α_1 and α_2 share the same subepisode on dropping their respective last nodes. We denote this common $(\ell-1)$-node episode as α. We note that α is the subepisode obtained by dropping v_ℓ and $v_{\ell+1}$ from $\mathcal{Y}_0$ combination of α_1 and α_2. Our efficient procedure constructs a $\mathcal{Y}_0$ combination and does some special checks on the first $(\ell-1)$ nodes in $\mathcal{Y}_0$ (or the nodes in α) based on their edge relationships with v_ℓ and $v_{\ell+1}$ and outputs all the potential candidates (among the possible three) that can be generated from α_1 and α_2.

Algorithm 1: GetPotentialCandidates(α_1, α_2)

Input: Patterns, α_1 and α_2, both of size ℓ
Output: $\mathcal{P}$, potential candidates from α_1 and α_2
1 Initialize $\mathcal{P} \leftarrow \phi$;
2 **if** $\exists$ *a node of type* 1 *in* α **then** $\mathcal{P} = \{\mathcal{Y}_1\}$; return;
3 **if** $\exists$ *a node of type* $1'$ *in* α **then** $\mathcal{P} = \{\mathcal{Y}_2\}$; return;
4 **else**
5 Add $\mathcal{Y}_0$ to $\mathcal{P}$;
6 **if** $\nexists$ *nodes of type* $2'$ *and* 3 *in* α **then** Add $\mathcal{Y}_1$ to $\mathcal{P}$; ;
7 **if** $\nexists$ *nodes of type* 2 *and* $3'$ *in* α **then** Add $\mathcal{Y}_2$ to $\mathcal{P}$; ;
8 return;

For purposes of easier understanding and illustration of this algorithm, we classify the nodes in α based on its relation with v_ℓ and $v_{\ell+1}$. A node $z \in \alpha$ (and hence $z \neq v_\ell, z \neq v_{\ell+1}$) is one of the 9 types described in Table 3.

`GetPotentialCandidates()` function (listed as Algorithm 1) describes our more efficient procedure (in comparison with a naive procedure which performs the 6 checks enlisted in Table 2 for every node $z = v_i$, where $i = 1, 2, \ldots (\ell - 1)$) based on the node type in α, as explained in Table 3. It takes two combinable episodes α_1 and α_2 as input and returns $\mathcal{P}$, the set of potential candidates obtained by combining them. We can summarize the working of Algorithm 1 as follows. If a node of type (1) exists in α, then $\mathcal{Y}_1$ is the only generated candidate (line 2). Similarly, if a node of type $(1')$ exists , then $\mathcal{Y}_2$ is the only generated candidate (lines 3). Suppose neither nodes of the type (1) nor $(1')$ exist, then $\mathcal{Y}_0$ is a sure candidate (line 5). Further, $\mathcal{Y}_1$ is generated iff nodes of type $(2')$ and (3) do not exist in α. If the algorithm finds a node of one of these types, it decides against adding $\mathcal{Y}_1$ to $\mathcal{P}$ (line 6). Similarly, $\mathcal{Y}_2$ is generated iff nodes of type (2) and $(3')$ do not exist in α (line 7). Even though nodes of type (4), $(4')$ and $(4'')$ are not used in the algorithm, we provide them in Table 3 for a complete classification of the nodes in α.

Illustration via an example The $\mathcal{Y}_0$ combination of the episodes in Fig. 10 has a node v_3 of type (1). Transitivity demands the existence of the edge (v_4, v_5) which is absent in the $\mathcal{Y}_0$ and $\mathcal{Y}_2$ combinations. Hence $\mathcal{Y}_0$ and $\mathcal{Y}_2$ violating transitivity here is immediate. Our procedure concludes, without any more checks, that $\mathcal{Y}_1$ is a potential candidate. Theorem 3 shows the correctness of this step. Analogously, $\mathcal{Y}_2$ is the only potential candidate when a node of type $(1')$ exists in α. In our efficient procedure, we have considered the cases of nodes of type (1) and $(1')$ existing separately. The procedure in this sense is unambiguous as nodes of type (1) and $(1')$ cannot coexist as shown later in Lemma 1. Continuing our illustration, Fig. 11 gives an example of a $\mathcal{Y}_0$ combination where no nodes of type (1) or $(1')$ exist. Accordingly, $\mathcal{Y}_0$ is a potential candidate which will be shown below. Also nodes of type (2) (v_3 in Fig. 11) and $(3')$ (v_2 in Fig. 11) exist in α. Transitivity of $<_{\mathcal{Y}_2}$ demands the existence of edges (v_5, v_3) and (v_2, v_4) which are absent in $<_{\mathcal{Y}_2}$ (indicated as dashed lines in the figure) and hence $<_{\mathcal{Y}_2}$ violates transitivity. Finally α in this example does not contain nodes of type $(2')$ or (3) and hence $\mathcal{Y}_1$ is a potential candidate which will be proved in Theorem 3.

Computational savings We now explain a few immediate computational savings that our efficient procedure achieves over the procedure which performs the 6 checks listed in Table 2 on every node in α for each of the three $\mathcal{Y}_i$ combinations. From Table 2, the first thing to notice is that for any given node, not all of these six checks are actually necessary to check transitivity of a particular $\mathcal{Y}_i$ combination. Specifically, it is easy to see that checks (b) and (d) are sufficient to check transitivity of a $\mathcal{Y}_0$ combination. Similarly it is not hard to see that

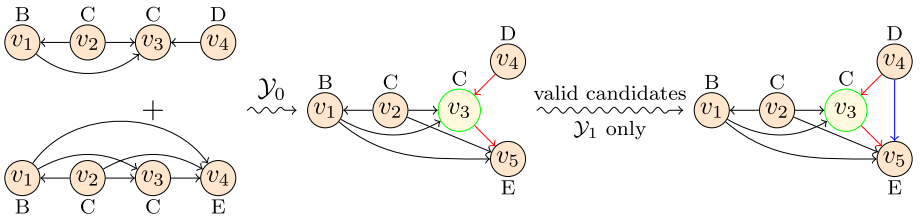


Fig. 10 Illustration of a case when node of type (1) exists

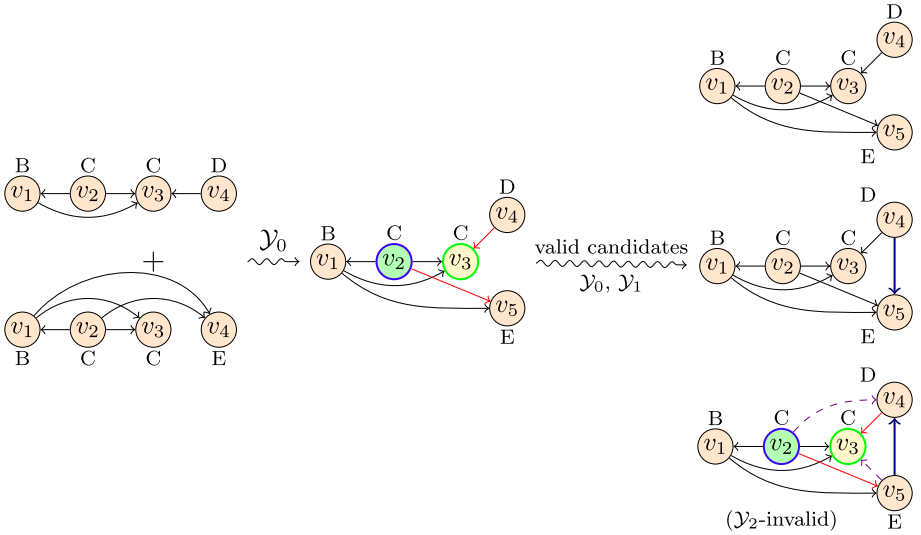


Fig. 11 Illustration of a case when neither nodes of type (1) and (1′) exists

checks (a), (d) and (e) checks are sufficient for transitivity check of a $\mathcal{Y}_1$ combination while (b), (c) and (f) checks are sufficient for transitivity check of a $\mathcal{Y}_2$ combination. As we show below, our efficient procedure performs even lesser checks than this first-cut optimization.

For $<_{\mathcal{Y}_0}$, performing the necessary checks (b) and (d) is equivalent to asking for the absence of nodes of type (1) and (1′), respectively. This is exactly the first set of checks carried out by our efficient procedure. As per this, $\mathcal{Y}_0$ is generated iff neither of the nodes of type (1) and (1′) exist in α. Also, as per this algorithm, if a node of type (1) exists, then it is immediately seen that $\mathcal{Y}_2$ violates transitivity. So one does not have to check for (b), (c) and (f) separately (as given in Table 2). Further, the algorithm also concludes that $\mathcal{Y}_1$ is a potential candidate and hence we don't have to perform checks (a), (d) and (e) on each node in α. An analogous computational saving happens when a node of type (1′) exists in α.

Similarly in the last part of the algorithm, the two checks are enough to ascertain the validity of $\mathcal{Y}_1$ or $\mathcal{Y}_2$ instead of three as per the first-cut optimization procedure explained earlier. We summarize the computational saving that our efficient procedure provides while combining two ℓ-node episodes. The naive procedure would need to carry out $3 * 6 * (\ell - 1)$ checks in the worst case to ascertain the validity of each $\mathcal{Y}_i$. Therefore, it would need a total of $54(\ell - 1)$ checks in the worst case. On the other hand, our intelligent procedure would need only up to $2(\ell - 1)$ checks if a node of type (1) exists in α, which is the best case scenario for the algorithm. If α contains a node of type (1′), it would need up to $4(\ell - 1)$

checks. If neither of (1) or (1′) exist, then it would need up to $12(\ell - 1)$ checks which is the worst case scenario for the procedure.

To show the correctness, we first make an important observation regarding the kind of nodes that are allowed to coexist in α.

Lemma 1 *In $\mathcal{Y}_0$, if a node of type* (1) *exists, there cannot exist nodes of type* (1′), (2′) *and* (3). *Similarly, if a node of type* (1′) *exists, there cannot exist nodes of type* (1), (2) *and* (3′).

Proof Given that a node z_0 of type (1) exists in α, we will show by contradiction that no nodes of type (1′), (2′) and (3) can exist. Suppose a node z_1 of type (1′) exists. Then $(z_1, v_\ell) \in <_{\mathcal{Y}_0}$ and hence $(z_1, v_\ell) \in <_{\alpha_1}$. Since z_0 is of type (1), $(v_\ell, z_0) \in <_{\mathcal{Y}_0}$ and hence $(v_\ell, z_0) \in <_{\alpha_1}$. By the transitivity of $<_{\alpha_1}$, it follows that $(z_1, z_0) \in <_{\alpha_1}$. Also, since z_0 is of type (1), we have $(z_0, v_{\ell+1}) \in <_{\mathcal{Y}_0} \implies (z_0, v_\ell) \in <_{\alpha_2}$. Likewise, since z_1 is of type (1′), $(v_{\ell+1}, z_1) \in <_{\mathcal{Y}_0} \implies (v_\ell, z_1) \in <_{\alpha_2}$. Hence, the transitivity of $<_{\alpha_2}$ tells us that $(z_0, z_1) \in <_{\alpha_2}$. This means we have two nodes z_0 and z_1 (neither of these being v_ℓ or $v_{\ell+1}$ of $\mathcal{Y}_0$) both belonging to α_1 and α_2, but related in opposite ways. This contradicts the condition that α_1 and α_2 share the same maximal subepisode on dropping their last nodes.

Suppose a node z_2 of type (2′) exists, then $(z_2, v_\ell) \in <_{\mathcal{Y}_0} \implies (z_2, v_\ell) \in <_{\alpha_1}$. Also $(v_\ell, z_0) \in <_{\alpha_1}$ since z_0, a node of type (1) also exists. Transitivity of $<_{\alpha_1}$ tells us $(z_2, z_0) \in <_{\mathcal{Y}_0}$. Since both z_0 and z_2 belong to α, $(z_2, z_0) \in <_{\alpha_2}$. We also have $(z_0, v_{\ell+1}) \in <_{\mathcal{Y}_0} \implies (z_0, v_\ell) \in <_{\alpha_2}$. Transitivity in $<_{\alpha_2}$ now implies $(z_2, v_\ell) \in <_{\alpha_2}$ and hence is in $<_{\mathcal{Y}_0}$. But this edge must be absent as z_2 is of type (2′). A similar contradiction arises for a node of type (3).

On similar lines, we can show that if a node of type (1′) exists in $<_{\mathcal{Y}_0}$, there cannot exist nodes of type (1), (2) and (3′). □

Since $<_{\mathcal{Y}_1}$ or $<_{\mathcal{Y}_2}$ differ from $<_{\mathcal{Y}_0}$ in only one edge involving v_ℓ and $v_{\ell+1}$, these coexistence results hold good for $<_{\mathcal{Y}_1}$ and $<_{\mathcal{Y}_2}$ also. We will now show that this efficient procedure generates all potential candidates.

Theorem 3 *The procedure described in Algorithm 1 generates only those combinations (out of the three possible combinations $\mathcal{Y}_0$, $\mathcal{Y}_1$ and $\mathcal{Y}_2$) that satisfy transitivity.*

Proof To find potential candidates, it is enough to (efficiently) perform the transitivity checks as described earlier in Table 2. We consider the various conditions under which the algorithm operates.

Condition(i) A node z of type (1) exists in α : We have already shown that $<_{\mathcal{Y}_0}$ and $<_{\mathcal{Y}_2}$ are not transitively closed here. We need to prove the transitivity of $<_{\mathcal{Y}_1}$.

To prove this, we need to check (a), (d) and (e) in Table 2. If hypothesis of (a) is true, and $(v_\ell, z) \notin <_{\mathcal{Y}_1}$, then either there exists an edge $(z, v_\ell) \in <_{\mathcal{Y}_1}$ or there exists no edge between z and v_ℓ. In the first case, z must be of type (1′) which cannot exist from Lemma 1. In the second case z must be of type (3) which also cannot exist from Lemma 1. This proves (a). The hypothesis of (d) indicates the existence of a type (1′) node in $\mathcal{Y}_1$ which is not possible from Lemma 1. Correctness of (e) is similar to that of (a). If hypothesis of (e) is true, and $(z, v_{\ell+1}) \notin <_{\mathcal{Y}_1}$, then either there exists an edge $(v_{\ell+1}, z) \in <_{\mathcal{Y}_1}$ or there exists no edge between z and $v_{\ell+1}$. In the first case, z must be of type (1′) which cannot exist from Lemma 1. In the second case, z must be of type (2′) which also cannot exist from Lemma 1. This proves (e).

Condition(ii) A node of type (1′) exists in α: This is analogous to condition(i).

Condition(iii) neither a node of type(1) nor type (1′) exists : First we need to show that $<_{\mathcal{Y}_0}$ satisfies transitivity, for which showing (b) and (d) (Table 2) is enough. We have already seen that this is same as the absence of nodes of type (1) and (1′).

Further, we show that $<_{\mathcal{Y}_1}$ is transitive iff no nodes of type $(2')$ and (3) exist in α. We prove the contra-positive of the forward implication. If a node z of type $(2')$ exists, then we have $(z, v_\ell), (v_\ell, v_{\ell+1}) \in <_{\mathcal{Y}_1}$ but there is no edge between z and $v_{\ell+1}$. This violates transitivity of $<_{\mathcal{Y}_1}$. similarly, if a node z of type (3) exists, then we have $(v_\ell, v_{\ell+1}), (v_{\ell+1}, z) \in <_{\mathcal{Y}_1}$, but there is no edge between z and v_ℓ. This violates transitivity of $<_{\mathcal{Y}_1}$.

For the converse, suppose no nodes of type $(2')$ and (3) exist. To show the transitivity of $<_{\mathcal{Y}_1}$, it is enough to show (a), (d) and (e). If hypothesis of (a) is true, and $(v_\ell, z) \notin <_{\mathcal{Y}_1}$, then either there exists an edge $(z, v_\ell) \in <_{\mathcal{Y}_1}$ or there exists no edge between z and v_ℓ. In the first case, z must be of type $(1')$ which cannot exist here (condition (iii)). In the second case, z must be of type (3) which also cannot exist from the hypothesis. This proves (a). The hypothesis of (d) demands the existence of nodes of type(1) and $(1')$ which cannot exist here (condition (iii)). If hypothesis of (e) is true, and suppose $(z, v_{\ell+1}) \notin <_{\mathcal{Y}_1}$), then either there exists an edge $(v_{\ell+1}, z) \in <_{\mathcal{Y}_1}$ or there exists no edge between z and $v_{\ell+1}$. In the first case z must be of type $(1')$ which cannot exist here (condition (iii)). In the second case, z must be of type $(2')$ which also cannot exist from the hypothesis. This proves (e).

Further, we show that $<_{\mathcal{Y}_2}$ is transitively closed iff no nodes of type (2) and $(3')$ exist in α. The proof of this is analogous to that of $<_{\mathcal{Y}_1}$. This completes the proof of Theorem 3. □

Remark 3 Till now we considered the case of α_1 being combined with the first type of episodes. For the second type of combination, we have the same subepisode obtained by dropping v_r and $v_{\ell+1}$ from the $\mathcal{Y}_0$ combination. For this case, we just need to work with this common subepisode α and Algorithm 1 goes through as it is for this case.

Remark 4 The transitivity checks we propose here can also be applied to the injective episode candidate generation algorithm of [5]. As one can easily see, these checks will enhance the efficiency of the candidate generation algorithm of [5].

5.4 Implementation issues in candidate generation

In this section, we explain how for a given episode, one can efficiently search for combinable episodes. Similar in spirit to the procedure adopted for injective episodes [5], the candidate generation procedure for chain episodes is such that the episodes which share the same subepisode on dropping their last nodes appear consecutively in the generated list of candidates, at each level. Episodes which share the same subepisode by dropping their respective last nodes are referred to as a *block*. Let $\mathcal{F}_\ell[i]$ denote the ith episode of $\mathcal{F}_\ell$, the set of all ℓ-node frequent episodes. At level 1 (i.e. $\ell = 1$), $\mathcal{F}_1$ is ordered according to the lexicographic ordering on the set of event-types $\mathcal{E}$. Suppose $\mathcal{F}_1$ consists of the frequent episodes B and C, then we have $\mathcal{F}_1[1] = B$ and $\mathcal{F}_1[2] = C$. At level 1, we combine an episode with itself and with all other episodes below it in $\mathcal{F}_1$. Accordingly, we first combine B with itself to form $(B \to B)$. When two distinct episodes are combined, there are three possibilities. For example, B would be combined with C to form $(B\,C)$, $(B \to C)$ and $(C \to B)$. Finally, C would be combined with itself to form $(C \to C)$. Observe that the first four candidates in $\mathcal{C}_2$ here share the same 1-node episode, namely B on dropping their last node. In fact they were obtained by combining a particular episode in $\mathcal{F}_1$ namely B with all combinable episodes below it in $\mathcal{F}_1$. Generalizing this observation, the block information of $\mathcal{C}_{\ell+1}$ can be naturally obtained during its construction itself. Also our procedure is such that at each level, episodes in every block are ordered lexicographically with respect to the array of event-types $\alpha.g$.

The pseudocode for the chain episode candidate generation, `GenerateCandidates()`, is listed in Algorithm 2. The input to Algorithm 2 is $\mathcal{F}_\ell$, a set of ℓ-node frequent episodes

Algorithm 2: GenerateCandidates($\mathcal{F}_\ell$)

Input: Sorted array, $\mathcal{F}_\ell$, of frequent episodes of size ℓ
Output: Sorted array, $\mathcal{C}_{\ell+1}$, of candidates of size $(\ell+1)$

1 Initialize $\mathcal{C}_{\ell+1} \leftarrow \phi$ and $k \leftarrow 0$;
2 **if** $\ell = 1$ **then**
3 **for** $h \leftarrow 1$ **to** $|\mathcal{F}_\ell|$ **do** $\mathcal{F}_\ell[h].blockstart \leftarrow 1$;
4 **for** $i \leftarrow 1$ **to** $|\mathcal{F}_\ell|$ **do**
5 $currentblockstart \leftarrow k+1$;
6 **if** *rth node of* $\mathcal{F}_\ell[i]$ *exists* **then**
7 Search for the block of episodes $\mathcal{B}$ in $\mathcal{F}_\ell$ which matches the subepisode obtained by dropping the rth node of $\mathcal{F}_\ell[i]$;
8 **foreach** $\beta \in \mathcal{B}$ *s.t.* $\beta.g[\ell] = \beta.g[\ell-1] = \mathcal{F}_\ell[i].g[\ell]$ **do**
9 $\mathcal{P} \leftarrow$ `GetPotentialCandidates` $(\mathcal{F}_\ell[i], \beta)$;
10 $\mathcal{P}' \leftarrow$ MaxSubepisodeCheck($\mathcal{P}$);
11 **foreach** $\alpha \in \mathcal{P}'$ **do**
12 $k \leftarrow k+1$;
13 Add α to $\mathcal{C}_{\ell+1}$;
14 $\mathcal{C}_{\ell+1}[k].blockstart \leftarrow currentblockstart$;
15 **else**
16 $\alpha \leftarrow (\ell+1)$-node serial episode with $\alpha.g[j] = \mathcal{F}_\ell[i].g[1]\ \forall j$;
17 Add α to $\mathcal{C}_{\ell+1}$;
18 $\mathcal{C}_{\ell+1}[k].blockstart \leftarrow currentblockstart$;
19 **for** $(j \leftarrow i+1;\ \mathcal{F}_\ell[j].blockstart = \mathcal{F}_\ell[i].blockstart;\ j \leftarrow j+1)$ **do**
20 **if** $\mathcal{F}_\ell[i].g[\ell] \neq \mathcal{F}_\ell[j].g[\ell]$ **then**
21 $\mathcal{P} \leftarrow$ `GetPotentialCandidates` $(\mathcal{F}_\ell[i], \mathcal{F}_\ell[j])$;
22 $\mathcal{P}' \leftarrow$ MaxSubepisodeCheck($\mathcal{P}$);
23 **foreach** $\alpha \in \mathcal{P}'$ **do**
24 $k \leftarrow k+1$;
25 Add α to $\mathcal{C}_{\ell+1}$;
26 $\mathcal{C}_{\ell+1}[k].blockstart \leftarrow currentblockstart$;
27 **return** $\mathcal{C}_{\ell+1}$

(where, $\mathcal{F}_\ell[i]$ denotes the ith episode in the collection). In $\mathcal{F}_\ell$, the episodes are organized as blocks. To store the block information of every episode, we use an array $\mathcal{F}_\ell.blockstart$. $\mathcal{F}_\ell.blockstart[i]$ essentially points to the first element of the block to which $\mathcal{F}_\ell[i]$ belongs to. It holds a value k such that $\mathcal{F}_\ell[k]$ is the first element of the block to which $\mathcal{F}_\ell[i]$ belongs to. The algorithm output is $\mathcal{C}_{\ell+1}$, the set of candidate episodes of size $(\ell+1)$. Initially, $\mathcal{C}_{\ell+1}$ is empty and, when $\ell = 1$, all (1-node) episodes are assigned to the same block (lines 1–3, Algorithm 2). The main loop runs over all the episodes in $\mathcal{F}_\ell$ (starting on line 4, Algorithm 2). Recall from Sect. 5.1.1 that the algorithm tries to combine an episode, $\mathcal{F}_\ell[i]$, with two types of episodes. In the pseudocode, $\mathcal{F}_\ell[i]$ and $\mathcal{F}_\ell[j]$ correspond to α_1 and α_2 that was used to describe the procedure earlier. We first try to combine $\mathcal{F}_\ell[i]$ with second type of episodes (lines 7–14). Episodes of this type would necessarily not belong to the block in which $\mathcal{F}_\ell[i]$ resides. We would need to search for such episodes in blocks further down in $\mathcal{F}_\ell$. The notion of an rth node does not exist for all 1-node episodes. Among episodes with more than 1 node, this can happen only when the episode is a serial episode with all ℓ nodes mapped to the same event. In this scenario, we form an $(\ell+1)$ node serial episode with all nodes mapped to the same event (lines 16–18). We note that this corner case was not mentioned in Sect. 5.1.1. We next combine $\mathcal{F}_\ell[i]$ with episodes of the first type which would all be stored in the same block below it (lines 19–26).

It is very important to combine a given episode $\mathcal{F}_\ell[i]$ with the second type of episodes first followed by the first type. This order of combination ensures episodes within the same block

are lexicographically ordered with respect to the array of event-types at each level. We can see this via induction. The way we combine episodes at level 1, this property holds for level 2 episodes. For instance, episodes $(A \rightarrow A)$, $(A\ B)$, $(A \rightarrow B)$, $(A \rightarrow B)$, (AC) and so on will be stacked one below the other. They all belong to the same block corresponding to the common 1-node subepisode (A). Note that they are lexicographically ordered with respect to the array of event-types. At any higher level say ℓ, an episode $\mathcal{F}_\ell[i]$ contributes to a block of episodes in $\mathcal{C}_{\ell+1}$ all of which on dropping their last node share the same subepisode namely $\mathcal{F}_\ell[i]$. The first combination of $\mathcal{F}_\ell[i]$ is with an episode of type 2, which means all $(\ell+1)$-node episodes constructed with this combination will have their last $(\ell+1)$th event-type as $\mathcal{F}_\ell[i].g[\ell]$. This follows from the third condition that defines combinable episodes of type 2. Further, combination with episodes of first type will yield $(\ell+1)$-node episodes with their last $((\ell+1)$th) nodes progressively greater than $\mathcal{F}_\ell[i].g[\ell]$ as per the lexicographic order on $\mathcal{E}$. This ensures that episodes within the same block in $\mathcal{C}_{\ell+1}$ are lexicographically ordered.

The strategy of combining with the second type of episodes first has further advantages. It further ensures that given an episode $\mathcal{F}_\ell[i]$, the associated first type of episodes are all below it in the same block and the associated second type of episodes are in a block further down and in turn makes the search more efficient. This also ensures that while searching for existence of certain maximal subepisodes to retain a potential candidate formed by $\mathcal{F}_\ell[i]$ and $\mathcal{F}_\ell[j]$, one could search for all these subepisodes in at most one pass over all the frequent episodes below $\mathcal{F}_\ell[j]$.

`GetPotentialCandidates()` function takes $\mathcal{F}_\ell[i]$ and $\mathcal{F}_\ell[j]$ as input and returns the set, $\mathcal{P}$, of *potential* candidates corresponding to them as described in Algorithm 1 in Sect. 5.3. The `MaxSubepisodeCheck()` function (listed in Algorithm 3) takes as input a set of potential candidates $\mathcal{P}$ and returns those candidates (set denoted as $\mathcal{P}'$), all whose maximal subepisodes of the type described earlier in the section are also in $\mathcal{F}_\ell$. For each potential candidate, $\alpha \in \mathcal{P}$, this function constructs an ℓ-node (maximal) subepisode (denoted as β in the pseudocode) by dropping a node (which is the last node among nodes mapped to the same event-type) at a time from α. If all such ℓ-node maximal subepisodes of α are found to be frequent, then α is added to $\mathcal{P}'$.

Algorithm 3: MaxSubepisodeCheck($\mathcal{P}$)

Input: $\mathcal{P}$, a set of 1 to 3 potential candidates of size $\ell+1$
Output: $\mathcal{P}'$, candidates from $\mathcal{P}$ all whose suitable maximal subepisodes are in $\mathcal{F}_\ell$.

```
1  Initialize P' = φ.;
2  foreach α ∈ P do
3      flg ← TRUE;
4      for (i ← ℓ − 1; i ≥ 1 and flg =TRUE; i − −) do
5          if α.g[i] ≠ α.g[i + 1] then
6              for x ← 1 to i − 1 do
7                  Set β.g[x] = α.g[x];
8                  for z ← 1 to i − 1 do β.e[x][z] ← α.e[x][z];
9                  for z ← i to ℓ do β.e[x][z] ← α.e[x][z + 1];
10             for x ← i to ℓ do
11                 β.g[x] ← α.g[x + 1];
12                 for z ← 1 to i − 1 do β.e[x][z] ← α.e[x + 1][z];
13                 for z ← i to ℓ do β.e[x][z] ← α.e[x + 1][z + 1];
14             if β ∉ F_ℓ then flg ← FALSE;
15     if flg = TRUE then
16         Add α to P';
17 return P'
```

6 Counting

In this section, we present algorithms for counting minimal windows and non-overlapped frequency of a set of candidate episodes through one pass over the data. For counting, we use finite state automata (FSA) to track occurrences. The FSA construction procedure for injective episodes [5] can be generalized to chain episodes as follows. For ease of exposition, we consider event sequences with at most one event-type per time tick. The ideas presented can be readily extended to general event sequences.

Definition 12 FSA $\mathcal{A}_\alpha$, used to track occurrences of episode $\alpha = (V_\alpha, <_\alpha, g_\alpha)$ in the event sequence is defined as follows. Each state, i, in $\mathcal{A}_\alpha$, is represented by a pair of subsets of V_α, namely $(\mathcal{Q}_i^\alpha, \mathcal{W}_i^\alpha)$. $\mathcal{Q}_i^\alpha$ contains those nodes in V_α whose associated event-types (under g_α) are accepted by the time FSA came to state i and $\mathcal{W}_i^\alpha$ contains those nodes in V_α whose associated event-types can be accepted by FSA in state i. The initial state, namely *state 0*, is associated with the subsets pair, $(\mathcal{Q}_0^\alpha, \mathcal{W}_0^\alpha)$, where $\mathcal{Q}_0^\alpha = \phi$ and $\mathcal{W}_0^\alpha$ is the collection of *minimal*[7] elements in V_α with respect to $<_\alpha$. Let i be the current state of $\mathcal{A}_\alpha$. $\mathcal{A}_\alpha$ remains in *state i* on seeing any event from $(\mathcal{E} \setminus g_\alpha(\mathcal{W}_i^\alpha))$. If the next event is of type $g_\alpha(v)$ for some $v \in \mathcal{W}_i^\alpha$, then $\mathcal{A}_\alpha$ accepts it and transits into a state j, with:

$$\mathcal{Q}_j^\alpha = \mathcal{Q}_i^\alpha \cup \{v\} \tag{3}$$

$$\mathcal{W}_j^\alpha = \{v' \in (V_\alpha \setminus \mathcal{Q}_j^\alpha) : \pi_\alpha(v') \subseteq \mathcal{Q}_j^\alpha\} \tag{4}$$

where $\pi_\alpha(v)$ is the subset of nodes in V_α that are less than E (with respect to $<_\alpha$). When $\mathcal{Q}_j^\alpha = V_\alpha$, (and $\mathcal{W}_j^\alpha = \phi$), j is the *final state* of $\mathcal{A}_\alpha$.

From (4), it is clear that no two elements v_k and v_m in any $\mathcal{W}_j^\alpha$ are related (under $<_\alpha$). Since we are dealing with chain episodes, the contra-positive of the chain episode definition implies that $g_\alpha(v_k)$ and $g_\alpha(v_i)$ are distinct. This would mean no two state transitions from any given state happen on seeing the same event-type. Hence, the FSA as per Definition 12 is deterministic for chain episodes. Figure 12 illustrates the FSA for the episode $(F \rightarrow (E\ G) \rightarrow F)$. We note here that, in view of (4), $\mathcal{Q}_j^\alpha$ alone is sufficient to characterize state j. However, maintaining the redundant information in the form of $\mathcal{W}_j^\alpha$ in the state makes the counting algorithm simpler to describe.

The algorithm description is conceptual and involves manipulation of automata described above. We then discuss an important issue in BE computation for chain episodes. We discuss in Appendix C the implementation aspects of counting.

6.1 Algorithm description

In this section we describe the counting schemes for tracking minimal windows and a maximal set of non-overlapped occurrences. We also introduce the important notion of an earliest transiting occurrence for chain episodes useful in the algorithm illustration and the correctness proofs.

The span of an occurrence h is defined as the time difference between the first and last events constituting the occurrence. In many applications, one would be interested in only those occurrences whose span is below some user-defined threshold. We call such a constraint on span as an *expiry-time constraint* which is specified by a threshold, T_X. Such a time constraint

[7] An element in V_α is minimal if there is no other element less than it as per $<_\alpha$. Note that a poset can in general have multiple minimal elements.

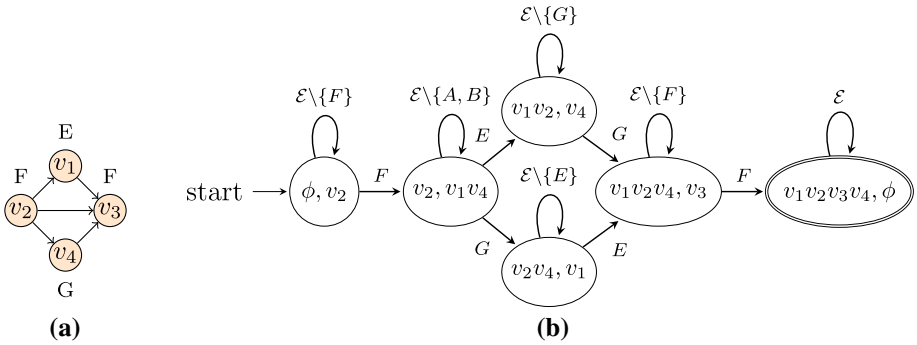


Fig. 12 The episode $(F \rightarrow (E\,G) \rightarrow F)$ and its associated finite state automaton

can make discovery more efficient by reducing the search space. Also, events widely spaced in time may actually not represent any correlated activity and hence expiry-time constraint would be natural in many applications. Our discovery algorithm presented in this paper can handle such an expiry-time constraint.

6.1.1 Algorithm for counting minimal occurrences

The algorithm for counting minimal occurrences (denoted as MO-algorithm) uses the general deterministic FSA described in Definition 12, for tracking chain episodes. The algorithm idea is similar to the minimal occurrences counting algorithm for serial episodes [3]. The algorithm takes as input a set of candidate episodes and returns the count of minimal windows of all the episodes. To start with, for each episode we will have one automaton of that episode (as per Definition 12) waiting in the start state. Then we move on the event sequence and for each event in the data, affect state transitions for all automata that can make a state transition on that event-type. Whenever an automaton moves out of its start state on seeing an appropriate event in the event sequence, we initialize a new automaton (of that episode) in the start state. In this process two automata (of an episode) can come to the same state. In this eventuality, we only retain the latest initialized automaton among the two automata. This is mainly because from now on, these multiple automaton will make identical state transitions, and hence, the earlier initialized automata cannot contribute to the minimal window frequency. Following this process, the automata that reach final state track all minimal windows. With an expiry-time constraint of T_X, one needs to only count minimal windows tracked by the MO algorithm which satisfy the expiry-time constraint.

6.1.2 Algorithm for counting non-overlapped occurrences

To obtain the algorithm for computing the non-overlapped frequency under expiry constraint T_X (denoted as NO-X), we need to modify the MO algorithm as follows. Whenever an automaton reaches final state, we check whether the occurrence tracked satisfies the expiry constraint, that is, span of the occurrence is less than T_X. If it does not, we continue the algorithm on the lines of MO. If it does, then we increment the frequency and retire all the existing automata except the one in the start state and continue. All such automata would have tracked some partial occurrences overlapped with the current occurrence. Since we are tracking non-overlapped occurrences, we retire these automata.

$\mathbf{D}_1 = \langle (F,1),(G,2),(F,3),(E,4),(G,6),$ (F,8), (G,9), (E,10), (F,11),
$(F,12),(E,13),$ (F,14), (G,15), (E,16), (F,17)$.\rangle$.

Fig. 13 Maximal set of non-overlapped occurrences of $(F \rightarrow (E\,G) \rightarrow F)$ with $T_X = 4$

6.1.3 Illustration

We illustrate the algorithm on an example event sequence. For this, and the proof of correctness to follow, we need the important notion of earliest transiting (ET) occurrences of chain episodes. Before introducing this, we discuss an unambiguous and simple representation for an occurrence of an episode that we use here. Given any N-node episode, α, one can also represent an occurrence, h, by the range set of the map h (which is one–one), namely $h(V_\alpha) = \{h(v_1), h(v_2), \ldots h(v_N)\}$ consisting of exactly N integers. This means h can be unambiguously represented as a vector of integers ordered in increasing order, $[\bar{h}(1)\ \bar{h}(2)\ldots\bar{h}(N)]$, where $\bar{h}(i) < \bar{h}(i+1)$, $i = 1,\ldots,(N-1)$. Consider the non-injective chain episode $(F \rightarrow (EG) \rightarrow F)$. Consider its occurrence $\langle (F,3),(G,6),(E,10),(F,11)\rangle$ in event sequence $\mathbf{D}_1$ of Fig. 13. It can also be represented as a vector of integers [3 5 8 9] (since $(F,3)$, $(G,6)$, $(E,10)$, $(F,11)$ are the third, fifth, eighth and the ninth events in $\mathbf{D}_1$. Since our illustrations are using event sequences with at most one event per time-tick here, we can use another more intuitive representation, which is, the vector of times representation. That is, we can use $[t_{\bar{h}(1)}\ \ldots t_{\bar{h}(N)}]$ to represent an occurrence. For example, $\langle (F,3),(G,6),(E,10),(F,11)\rangle$ would be represented as [3 6 10 11] in this representation.

We denote by $\mathcal{H}$ the set of **all** occurrences of an episode α in a event sequence $\mathbb{D}$. On this set, there is a 'natural' lexicographic order (to be denoted as $<_\star$) which is formally defined below.

Definition 13 The lexicographic ordering on $\mathcal{H}$, the set of all occurrences of α, is defined as: for any two different occurrences h_1 and h_2, of α, $h_1 <_\star h_2$ if the least i for which $\bar{h}_1(i) \neq \bar{h}_2(i)$ is such that $\bar{h}_1(i) < \bar{h}_2(i)$. This is a total order on the set $\mathcal{H}$.

6.1.4 Earliest transiting occurrences

The basic idea of an ET occurrence is that once an occurrence starts, it tries to include the earliest possible events into its fold without violating the definition of an occurrence. Once it starts, it essentially performs earliest possible transitions. We define this formally below.

Definition 14 Given an occurrence h of α, let v_1^h denote that node in V_α such that $\bar{h}(1) = h(v_1^h)$. An occurrence h of a chain episode α in an event sequence $\mathbf{D}$ (where at most one event-type occurs at a time-tick) is said to be an **ET** occurrence if $\forall v_i \neq v_1^h$ the following hold. (a) if $\pi_\alpha(v_i) = \phi$, then $t_{h(v_i)}$ is the time of the first occurrence of the event-type $E_{h(v_i)}$ after $t_{\bar{h}(1)}$. (b) if $\pi_\alpha(v_i) \neq \phi$, $t_{\bar{h}(i)}$ is the first occurrence time of the event-type $E_{\bar{h}(i)}$ after the occurrence of all events associated with $\pi_\alpha(v_i)$ (a subset of V_α) as per the h-map.

We denote by $\mathcal{H}^e$ the set of all earliest transiting occurrences of a given chain episode α. We denote the ith occurrence (as per the lexicographic ordering of occurrences) in $\mathcal{H}^e$ as h_i^e. Recall from Definition 12 that $\mathcal{W}_0^\alpha$ represents the set of minimal elements of V_α. From the above definition, one can easily check that starting from each event in the event sequence whose corresponding event-type belongs to $g_\alpha(\mathcal{W}_0^\alpha)$, there exists a unique ET occurrence.

There are 6 ET occurrences of the chain episode $(F \rightarrow (EG) \rightarrow F)$ (shown in Fig. 12a) in $\mathbf{D}_1$ which is the event sequence in Fig. 13. The 6 ET occurrences are: h_1^e = [1 2 4 8],

$h_2^e = [3\ 4\ 6\ 8]$, $h_3^e = [8\ 9\ 10\ 11]$, $h_4^e = [11\ 13\ 15\ 17]$, $h_5^e = [12\ 13\ 15\ 17]$ and $h_6^e = [14\ 15\ 16\ 17]$. As a negative example, the occurrence $\langle (F, 3), (E, 4), (G, 6), (F, 11)\rangle$ of $(F \rightarrow (EG) \rightarrow F)$, (that is, the occurrence [3 4 6 11]) in $\mathbf{D}_1$, is not ET. This is because $t_{h(v_3)} = t_9 = 11$ is not the time of the first occurrence of the event-type $E_{h(v_3)} = E_9 = F$ after the occurrence of all events associated with $\pi_\alpha(v_3) = \{v_1, v_2, v_4\}$, namely $\{(F, 3), (E, 4), (G, 6)\}$. The first such event is $(F, 8)$.

For any chain episode α, the associated deterministic FSA as per Definition 12 (suitably initialized) can be made to track any ET occurrence. Specifically, given any ET occurrence h_i^e of an episode, an automaton for the episode initialized in the start state just before processing the event at $t_{h_i^e(1)}$, would exactly track h_i^e by undergoing state transitions on seeing events constituting h_i^e.

ET occurrences tracked by MO In the MO algorithm described earlier, the first initialized automaton would exactly track h_1^e. On seeing the first relevant event, this automaton moves out of its start state and the MO algorithm would accordingly initialize a new automaton in the start state which would exactly track h_2^e and so on. Hence, the MO algorithm for chain episodes searches in the space of ET occurrences only. However, as explained earlier, when two automaton come to the same state we retain only the newer automaton as, from now on, both these automaton make the same state transitions. For example, after processing $(G, 6)$ in $\mathbf{D}_1$, both the first and second initialized automaton associated with $\beta = (F \rightarrow (E\ G) \rightarrow F)$ are in the same state, namely $(\{v_1, v_2, v_4\}, \{v_3\})$, for the first time. We drop the first automaton and continue. On processing the next event $(F, 8)$, the second initialized automaton reaches final state and tracks h_2^e completely. If we define the window of an occurrence h of an N-node episode as $[t_{\bar{h}(1)}, t_{\bar{h}(N)}]$, then it is easy to see that the window of h_2^e in $\mathbf{D}_1$ is minimal. Continuing like this, it is easy to see that the MO algorithm ultimately tracks h_2^e, h_3^e and h_6^e whose windows exactly correspond to the minimal windows of β in $\mathbf{D}_1$.

ET occurrences tracked by NO-X Figure 13 shows the occurrences tracked by the NO-X algorithm for a $T_X = 4$. The first automaton that reaches final state as per NO-X tracks h_2^e completely in $\mathbf{D}_1$. This is because it exactly mimics the MO-algorithm until it finds an occurrence satisfying expiry-time. As h_2^e violates expiry for $T_X = 4$, we continue and the next ET occurrence completely tracked is h_3^e which satisfies expiry. We accordingly increment the frequency. We now retire all the existing automata except the one in start state and continue. The next minimal occurrence tracked is h_6^e which satisfies expiry and hence, frequency would be incremented. Please refer to Appendix C for details on implementation issues of counting.

6.2 Correctness proofs

We start off by showing the correctness of the MO algorithm on event sequences where at most one event-type occurs per time-tick. The idea of the correctness proof is broadly along the lines of that of serial episodes [3]. We need two important properties of ET occurrences of chain episodes for further analysis. The proof of the second follows immediately from the first. We state the two properties here and give the proof of the first property in Appendix D.

Property 2 *Given a chain episode α and data stream $\mathbf{D}$, consider an ET occurrence h and another occurrence h' of α in $\mathbf{D}$ such that h' starts on or after $t_{\bar{h}(1)}$. Let $\mathbf{D}_j$ denote the first j events of $\mathbf{D}$. For every j, the set of all nodes in V_α whose associated events under h occur in $\mathbf{D}_j$ is a superset of the set of all nodes in V_α whose associated events under h' occur in $\mathbf{D}_j$.*

Property 3 *Suppose h is an ET occurrence of an N-node chain episode α. If h' is any other occurrence such that $t_{\bar{h}(1)} \leq t_{\bar{h}'(1)}$, then $\bar{h}(i) \leq \bar{h}'(i)\ \forall i = 1, 2, \ldots N$.*

Another interesting consequence of Property 2 useful in the proofs is as follows.

Remark 5 If for $j > i$, $\bar{h}_i^e(k) = \bar{h}_j^e(k)$ for some k, $1 < k \leq N$, then $\bar{h}_i^e(k') = \bar{h}_j^e(k')$ for every $k' \geq k$. This is because, from Property 2, on the event sequence till $t_{\bar{h}_i^e(k)}$ $(\mathbf{D}_{\bar{h}_i^e(k)})$, the set of all nodes in V_α whose associated events occurring in $\mathbf{D}_{\bar{h}_i^e(k)}$ under $\bar{h}_i^e$ and $\bar{h}_j^e$ are identical. This would in turn mean that the associated automata which can track h_i^e and h_j^e would be in the same state just after $t_{\bar{h}_i^e(k)}$, as a state here encodes all nodes of V_α whose associated event-types have been seen till now. From now on, since the two automata make identical transitions, we have $\bar{h}_i^e(k') = \bar{h}_j^e(k')$ for every $k' \geq k$. Also, from Property 2, we can see that, if two automata tracking ET occurrences have accepted the same number of event-types, then they must be in the same state.

6.2.1 Proof of correctness of MO-algorithm

After showing the two relevant properties, we now try to characterize all minimal windows in terms of ET occurrences. Any minimal window of a chain episode is also a window of some ET occurrence of the episode. Specifically, the earliest occurrence of the episode in the minimal window would be the concerned ET occurrence. Hence, it is enough to search for minimal windows in the space of ET occurrences. The following lemma characterizes the set of ET occurrences whose windows are minimal.

Lemma 2 *The window w of an earliest transiting (ET) occurrence h_i^e, of an N-node chain episode α, is not a minimal window if and only if $t_{\bar{h}_i^e(N)} = t_{\bar{h}_{i+1}^e(N)}$.*

Proof Let $w = [t_s, t_e]$[8]$= [t_{\bar{h}_i^e(1)}, t_{\bar{h}_i^e(N)}]$ denote the time-window of h_i^e. Since there can exist at most one ET occurrence starting at a time-tick, we have $t_{\bar{h}_i^e(1)} < t_{\bar{h}_{i+1}^e(1)}$. If $t_{\bar{h}_i^e(N)} = t_{\bar{h}_{i+1}^e(N)}$, then the time window of $\bar{h}_{i+1}^e$ is a proper sub-window of w. Hence w is not a minimal window.

For the converse, if w is not a minimal window, we need to show that $\bar{h}_i^e(N) = \bar{h}_{i+1}^e(N)$. Since w is not a minimal window, one of its proper sub-windows contains an occurrence, say, h, of α. If the time window of h starts at t_s, it has to end strictly before t_e because the time window of h is a strict sub-window of w. This means we have an occurrence h starting at t_s and ending before t_e. This contradicts the fact that h_i^e is an ET occurrence (cf. Property 3). Thus, the window of h has to start beyond $t_{\bar{h}_i^e(1)}$ and hence we have $t_{\bar{h}(1)} > t_{\bar{h}_i^e(1)}$. This means, by Property 3, since h_i^e is ET, we have $\bar{h}_i^e(N) \leq \bar{h}(N)$. Since the window of h has to be contained in w (the window of h_i^e), we thus have $t_{\bar{h}_i^e(N)} = t_{\bar{h}(N)}$. By definition, h_{i+1}^e will start at the earliest possible position after $t_{h_i^e(1)}$. Since there is an occurrence starting with $t_{\bar{h}(1)}$ $(> t_{\bar{h}_i^e(1)})$, we must have $t_{\bar{h}_{i+1}^e(1)} \leq t_{\bar{h}(1)}$. Now, again from Property 3, since h_{i+1}^e is ET, we have $\bar{h}_{i+1}^e(N) \leq \bar{h}(N)$. Since h_{i+1}^e starts beyond $t_{\bar{h}_i^e(1)}$ and since h_i^e is ET, from Property 3, we have $\bar{h}_i^e(N) \leq \bar{h}_{i+1}^e(N)$. Therefore combining the last three deductions, we have (since $\bar{h}_1(N) < \bar{h}_2(N)$ implies $t_{\bar{h}_1(N)} < t_{\bar{h}_2(N)}$)

$$t_{\bar{h}_i^e(N)} \leq t_{\bar{h}_{i+1}^e(N)} \leq t_{\bar{h}(N)} = t_{\bar{h}_i^e(N)}. \tag{5}$$

Thus, we have $t_{\bar{h}_i^e(N)} = t_{\bar{h}_{i+1}^e(N)}$. This completes proof of lemma.

[8] In t_e, subscript e denotes the end time of the window. In h_i^e, superscript e refers to earliest transiting.

Remark 6 This lemma shows that any ET occurrence h_i^e such that $t_{\bar{h}_i^e(N)} < t_{\bar{h}_{i+1}^e(N)}$ is a minimal occurrence (or the window of h_i^e is a minimal window) and conversely. Thus we can track all minimal windows if we track all ET occurrences h_i^e such that $t_{\bar{h}_i^e(N)} < t_{\bar{h}_{i+1}^e(N)}$.

The MO algorithm initializes a new automaton (of the type described in definition 12) once an existing automaton moves out of its start state. In the process, the ith initialized automaton $\mathcal{A}_i^\alpha$ would track h_i^e, the ith ET occurrence. However, not every automaton results in increase in frequency; when an automaton comes into a state already occupied by an older automaton, the older one is removed. If we can prove the automaton $\mathcal{A}_i^\alpha$ results in increment of frequency if and only if h_i^e, the occurrence tracked by it is such that $t_{\bar{h}_i^e(N)} < t_{\bar{h}_{i+1}^e(N)}$, then, the proof of correctness of MO algorithm is complete. This is done in the lemma below.

Lemma 3 *In the MO algorithm, the ith automaton that was initialized for α, referred to as $\mathcal{A}_i^\alpha$, contributes to the frequency count iff $t_{\bar{h}_i^e(N)} < t_{\bar{h}_{i+1}^e(N)}$.*

Proof

$\mathcal{A}_i^\alpha$ does not contribute to the frequency.
$\implies \mathcal{A}_i^\alpha$ is removed by a more recently initialized automaton.
$\implies \exists\ \mathcal{A}_k^\alpha, k > i,$ which transits into a state already occupied by $\mathcal{A}_i^\alpha$.
$\implies \exists k, j\ s.t.\ k > i,\ 1 < j \leq N$ and $\bar{h}_i^e(j) = \bar{h}_k^e(j)$.
$\implies \exists j\ 1 < j \leq N\ s.t.\ \bar{h}_i^e(j) = \bar{h}_{i+1}^e(j)$.
follows from Property 3 applied on (h_i, h_k), (h_{i+1}, h_k) and (h_i, h_{i+1})
$\implies \bar{h}_i^e(N) = \bar{h}_{i+1}^e(N)$.
$\implies t_{\bar{h}_i^e(N)} = t_{\bar{h}_{i+1}^e(N)}$.

The last but one step in the forward argument follows from Remark 5.

Conversely, we have

$\mathcal{A}_i^\alpha$ contributes to the frequency.
$\implies$ no automata initialized later than $\mathcal{A}_i^\alpha$ comes into a state occupied by $\mathcal{A}_i^\alpha$.
$\implies$ for $1 < j \leq N$, $\bar{h}_i^e(j) < \bar{h}_{i+1}^e(j)$.
$\implies \bar{h}_i^e(N) < \bar{h}_{i+1}^e(N)$.
$\implies t_{\bar{h}_i^e(N)} < t_{\bar{h}_{i+1}^e(N)}$.

The second deduction can be shown by contradiction. Suppose, $\exists j,\ 1 < j \leq N$ such that $\bar{h}_i^e(j) = \bar{h}_{i+1}^e(j)$. The MO algorithm dynamics is such that there exists some automata $\mathcal{A}_k^\alpha$ with $k \geq (i+1)$ which accepts $(E_{\bar{h}_{i+1}^e(j)}, t_{\bar{h}_{i+1}^e(j)})$ as its jth event. Since $\mathcal{A}_i^\alpha$ contributes to the frequency, $\mathcal{A}_i^\alpha$ accepts $(E_{\bar{h}_i^e(j)}, t_{\bar{h}_i^e(j)})$ which is equal to $(E_{\bar{h}_{i+1}^e(j)}, t_{\bar{h}_{i+1}^e(j)})$. Hence, we have a situation where both $\mathcal{A}_i^\alpha$ and $\mathcal{A}_k^\alpha$ accept $(E_{\bar{h}_i^e(j)}, t_{\bar{h}_i^e(j)})$ as their jth event. From Remark 5, both $\mathcal{A}_i^\alpha$ and $\mathcal{A}_k^\alpha$ must be in the same state after seeing $(E_{\bar{h}_i^e(j)}, t_{\bar{h}_i^e(j)})$. By the MO algorithm, $\mathcal{A}_i^\alpha$ must be knocked off now, which contradicts the fact that $\mathcal{A}_i^\alpha$ contributes to the frequency. This completes proof of the lemma. □

With the completion of the proof of Lemma 3, correctness proof of MO is also complete. Once we prove the correctness of the MO algorithm the correctness proof of NO-X follows by viewing it as a minor modification of the MO.

6.2.2 Proof of correctness of NO-X algorithm

Since all automata in NO-X algorithm also make state transitions as soon as they are possible, till the first time an automata reaches its final state, the NO-X and MO algorithm are identical. Hence, the first occurrence tracked by NO-X is its first minimal occurrence. If this satisfies expiry-time constraint, then the NO-X algorithm retires all automata and starts afresh. If not, continues in the MO-mode. Thus NO-X initially searches for the first minimal occurrence satisfying expiry-time constraints. After this, it looks for the next immediate minimal occurrence non-overlapped with the first one (satisfying expiry constraint) and also satisfying expiry constraint and so on.

Let $H_{nX} = \{h_1^{nX}, h_2^{nX} \ldots h_f^{nX}\}$ denote the sequence of occurrences of an N-node episode tracked by the NO-X algorithm. Then the following property of H_{nX} is obvious.

Property 4 *h_1^{nX} is the earliest minimal occurrence satisfying expiry time constraints. For any i, h_i^{nX} is the first minimal occurrence (of the N-node episode) after $\bar{h}_{i-1}^{nX}(N)$ satisfying expiry time constraint. There is no minimal occurrence satisfying expiry time constraints which starts after $\bar{h}_f^{nX}(N)$.*

Theorem 4 *H_{nX} is a maximal non-overlapped sequence satisfying expiry time constraints.*

Proof Consider any other set of non-overlapped occurrences satisfying expiry constraints: $H' = \{h'_1, h'_2 \ldots h'_l\}$ ordered such that $h'_i <_\star h'_{i+1}$. Let $m = min\{f, l\}$. To show the maximality of H_{nX}, we first show the following.

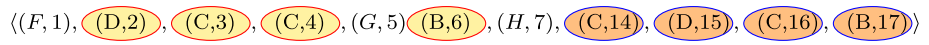
$$\bar{h}_i^{nX}(N) \leq \bar{h}'_i(N) \qquad \forall i = 1, 2, \ldots m. \tag{6}$$

This will be shown by induction on i. We first show it for $i = 1$. Suppose $\bar{h}'_1(N) < \bar{h}_1^{nX}(N)$. Then there exists a minimal occurrence within the window of h'_1. Since h'_1 satisfies expiry, we have found a minimal occurrence satisfying expiry constraints ending before h_1^{nX} which contradicts the first statement of Property 4. Hence $\bar{h}_1^{nX}(N) \leq \bar{h}'_1(N)$. Suppose $\bar{h}_i^{nX}(N) \leq \bar{h}'_i(N)$ is true for some $i < m$. We show that $\bar{h}_{i+1}^{nX}(N) \leq \bar{h}'_{i+1}(N)$. By Property 4, h_{i+1}^{nX} is the first minimal occurrence of α satisfying expiry time constraints in the data stream beyond $\bar{h}_i^{nX}(N)$. Suppose $\bar{h}'_{i+1}(N) < \bar{h}_{i+1}^{nX}(N)$. Then, very similar to the $i = 1$ case, there exists a minimal occurrence of α whose window is contained in that of h'_{i+1}. h'_{i+1} is non-overlapped with h_i^{nX} from the inductive hypothesis. Hence, we have found a minimal occurrence satisfying constraints starting after $\bar{h}_i^{nX}(N)$ ending before h_{i+1}^{nX} which contradicts the second statement of Property 4.

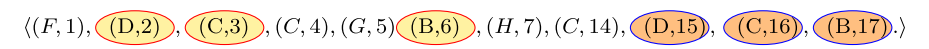
Now from Eq. (6), we can conclude that $l \leq f$, i.e. any sequence of non-overlapped occurrences can at most have f occurrences. This is because if H' is such that $l > f$, then from Eq. (6), h'_{f+1} is an occurrence that starts beyond $\bar{h}_f^{nX}(N)$. As before we can construct a minimal occurrence of α satisfying expiry constraints in the window of h'_{f+1}, which contradicts the last statement of Property 4 that there is no minimal occurrence satisfying expiry beyond $\bar{h}_f^{nX}(N)$. Hence $|H_{nX}| \geq |H'|$ for every non-overlapped sequence H' satisfying expiry constraints. Hence, H_{nX} is maximal and $f = f_{nX}$. □

6.3 Bidirectional evidence computation

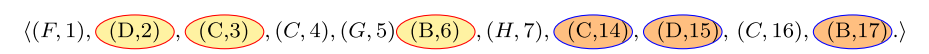
For a given candidate episode, it is very convenient if one can also compute BE along with frequency when going down the event sequence. In fact, this was the strategy used in [5] for



(a) Non-overlapped occurrences of $((D\,C) \rightarrow C \rightarrow B)$.



(b) Non-overlapped innermost occurrences of $((D\,C) \rightarrow B)$.



(c) Non-overlapped leftmost occurrences of $((D\,C) \rightarrow B)$.

Fig. 14 Illustration of why BE computation with innermost (minimal) occurrences would fail

BE computation and we stick to it here. The idea was to create a binary matrix (initialized to zero) for each automaton that is spawned and update the CountMatrix suitably as events constituting the tracked ET occurrence are encountered. If h is the occurrence tracked, then the (i, j)-entry in the binary matrix should be 1 if and only if $t_{h(v_i)} < t_{h(v_j)}$. If a spawned automaton reaches the final state, then the associated (completely updated) binary matrix contributes to the BE of the episode. Many of the spawned automata get knocked off (when two automata reach the same state) and its only the automata that reach the final state which influence the final BE. Recall from Fig. 13, that such automata basically track the innermost ET occurrence among a set of ET occurrences that end together.

However, while discovering non-injective chain episodes, this strategy of computing BE using innermost ET occurrences causes problems. Let us illustrate this with an example. Consider the episode $\alpha = ((D\,C) \rightarrow C \rightarrow B)$. Consider the data sequence in Fig. 14a which has two non-overlapped occurrences of α. As is easy to see, events in any occurrence of α can happen in one of two ways captured by the two highlighted occurrences of Fig. 14a. The bidirectional evidence of $\alpha = ((D\,C) \rightarrow C \rightarrow B)$ here is 1 and on running discovery with a frequency threshold of 2, it is natural to expect α to be output. For α to be output, we need 3 of its maximal subepisodes to be frequent at level 3. Of these, let us concentrate on $\beta = ((D\,C) \rightarrow B)$ obtained by dropping v_3 in α. It has a count of 2 but if one uses its innermost (or minimal)) occurrences (highlighted in Fig. 14b) to compute its BE, we obtain an unexpected value of 0. This would ultimately result in algorithm totally missing an interesting pattern like α. Note this problem does not arise in the context of injective episodes and is happening again because of some form of aliasing (discussed earlier in Sect. 4.2). For instance, the innermost occurrence tracking strategy for β resulted in the aliasing of the event $(C, 15)$ corresponding to the node v_3 of α, by the C in β, which essentially corresponds to v_2 of α.

Given this crucial issue, we propose to circumvent it by looking at the leftmost ET occurrence ending at the same event as the innermost occurrence (tracked by NO-X) and also satisfying the expiry constraint. With a T_X of about 5, the leftmost non-overlapped occurrences of β are shown in Fig. 14c. In this set of occurrences, the BE value of β is 1. Hence to incorporate this, we slightly modify NO-X (the counting strategy described in the previous subsection) as follows. Whenever two or more automata come to the same state, we don't blindly retire the older one as before. *We thus allow multiple automata in the same state and retain only those automata which can potentially still track an occurrence satisfying T_X. This would lead to a slight increase in the space complexity but is not a problem in practice for reasonable expiry thresholds. When one or more automata reach the final state, we increment the frequency count as before. However, for BE computation, we use the binary matrix associated with the oldest automaton only whose tracked occurrence satisfies expiry.* Our

simulations indicate that this strategy is indeed effective. *A pseudo-code providing details of this procedure is given in Appendix* B.

7 Related work

There is an inherent combinatorial explosion in partial order mining as pointed out in all existing works pertaining to mining episodes with general partial orders. For instance, consider an event sequence with non-overlapped occurrences of say a 10-node serial episode. It is reasonable to expect that this 10-node episode is representative of this event sequence. However, a frequency-based discovery would report all 10-node subepisodes of the serial episode as also frequent. As one can see, there are also a combinatorially explosive number of such redundant subepisodes. Tatti and Cule [25–27] tackle this issue by mining closed[9] frequent episodes which dramatically compresses the episode output. Unlike itemsets, defining closure based on frequency is not well defined because in the episodes context one can have multiple maximal superepisodes with the same frequency. This makes mining for closed frequent episodes directly infeasible. To tackle this issue, all these works consider the notion of what they call instance closure of an episode. The instance closure of an episode is essentially defined as the unique maximal episode which covers (or occurs in) all valid instances or occurrences of the episode in question. All the closed episode works efficiently mine for instance closed episodes and finally obtain frequency closed episodes by post-filtering the set of instance closed episodes. This is feasible because any frequency closed episode is also instance closed.

As described in the introduction, Tatti and Cule [25,27] propose apriori-based discovery algorithms for mining chain (or 'strict' as they call it) episodes by performing a breadth-first search of the space of all chain episodes. The two mainly differ in the way instance closure is defined. Tatti and Cule [27] considers instance closure by the addition of edges alone without addition of new event-types, where as [25] considers instance closure based on both edges and event-types. In this sense, the algorithm in [25] is a refined version of that of [27]. The algorithm tries to mimic closed itemset mining idea of mining frequent generators[10] which first discovers all frequent generators and then taking their closure to obtain frequent closed episodes. At each step of candidate generation, a potential candidate is generated by combining two subepisodes of the same size. It further checks for two other conditions: (a) if all its subepisodes are frequent (including episodes of the same size) and (b) if it does not lie in the closure of any of its subepisodes (essentially making sure its not a generator). The frequency of each such generated candidate is now obtained by one pass of the data. The monotonicity property, and hence, the candidate generation step in these algorithms is very different from the current proposed method. *We will elaborate more on this further in Appendix* A.

Tatti and Cule [26] considers mining in the space of all episodes under the windows-based frequency [18] even outside the class of chain episodes considered in this paper. Their approach can be readily extended to the non-overlapped frequency. However, the search approach in [26] is a depth-first approach. The idea here is to carry the list of all occurrences satisfying the expiry-time constraints for a given episode. Recursively, the algorithm traverses the lattice of all episodes in a DFS fashion by making a current episode more specific by either adding edges or nodes. In case of an addition of an edge, the occurrence list of the

[9] An episode is said to be frequency closed if every superepisode has a strictly lower frequency.

[10] A generator is an episode whose every subepisode has a strictly greater frequency.

new episode can be obtained by just dropping some of the invalid occurrences. In the event of an addition of a node, one needs to suitably combine the current occurrence list with all the occurrences of the added node (1-node episode) to obtain the occurrence list of the more specific episode. The frequency computation here is immediate as one is actually carrying the occurrence list itself. Importantly, one performs an instance-closure of all instances (or occurrences) which helps bypass redundant counting of many intermediate episodes. It also makes sure the currently arrived instance closed episode is not already explored via previous branches and discards the episode if so. This strategy outputs all instance closed episodes which is a superset of the set of all frequency closed episodes. Consequently, frequency closed episodes are obtained from a final post-processing step on instance closed episodes. One possible limitation of the DFS approach for general partial order episodes would be that one needs to carry all the occurrences of an episode and this can be exponential in general.

As discussed in the introduction, in the context of general injective episodes, Achar et al. [5] showed there that frequency alone is not a sufficient indicator of interestingness when dealing with general partial order episodes. To tackle this issue, a new measure of evaluating interestingness called bidirectional evidence (BE) was also introduced. The final discovery algorithm incorporated the new measure BE into the level-wise procedure in addition to frequency. This strategy was found to be extremely effective in not only pruning uninteresting patterns but also making the discovery efficient. For these reasons, we follow a similar approach for mining in the larger space of chain episodes.

It was argued in [5] that if an injective episode α has a BE of $H(\alpha)$ in the data, then it is guaranteed that all its *maximal subepisodes*[11] have a BE of at least $H(\alpha)$ among the occurrences of α. This crucial property was exploited for the design of the candidate generation step for injective episodes. Since BE is important both conceptually and for algorithm design in the context of partial order episodes, we will first discuss BE in detail in the context of chain episodes before getting into the algorithm details. Specifically, we will discuss how BE can be extended to chain episodes and the monotonicity property it satisfies in the context of chain episodes.

8 Experimental results

We present results of our chain episode-based discovery algorithm on synthetic data. One advantage of working with synthetic data in general is that one has access to the underlying ground truth. In our setting, it gives us information of the underlying embedded patterns that are representative of the generated data. The proposed algorithm is demonstrated to be effective in unearthing the embedded episodes while keeping a check on the number of spurious patterns reported. It is also robust enough to scale well with parameters like noise, data length and number of patterns. We also briefly demonstrate via simulation how our BE-based breadth-first search (BFS) algorithm can be more effective in pruning spurious patterns over the BFS-based closed episode miner [25]. The process of our synthetic data generation is presented next.

[11] Recall from Definition 6, an injective episode α can be viewed as a partially ordered set of event-types (X^α, R^α). (X^β, R^β) is a maximal subepisode of an injective episode α if $X^\beta \subseteq X^\alpha$ and R^β is the restriction of R^α on to X^β. The notion of a maximal subepisode of a general episode is discussed in the next section.

8.1 Synthetic data generation

The set of episodes that we want to embed in the synthetic data is the input to the data generator. For convenience of illustration, each t_i is chosen from the set of positive integers in this section. For each episode in the set, we generate an *episode event sequence* which contains just non-overlapped occurrences of the episode (and no other events). An episode event sequence for $(A \rightarrow (A\ B))$ would look like $\langle (A, t_1), (B, t_2), (A, t_3), (A, t_4), (A, t_5), (B, t_6), \ldots \rangle$ for example. Separately, we generate a noise event sequence $\langle (X_1, \tau_1), (X_2, \tau_2), \ldots \rangle$ where X_i's take values from the entire alphabet of event-types. All the episode event sequences and the noise event sequence are merged to generate the final event sequence (by stringing together all events in all the streams in a time-ordered fashion). There are three important user-specified parameters associated with the data generation process: η (span parameter), p (inter-occurrence parameter) and ρ (noise parameter), whose roles are explained below.

To generate an episode event sequence, we generate several occurrences of the episode successively in a non-overlapped way. For each occurrence (of the episode to be embedded), we randomly choose one of its *serial extensions*[12] and this fixes the sequence of event-types that will appear in the occurrence being embedded. The time difference $(t_{i+1} - t_i)$ between successive events in an occurrence is chosen to be a geometric distribution with parameter η $(0 < \eta \leq 1)$. The time between end of an occurrence and the start of the next is also distributed geometrically with (a different) parameter p $(0 < p \leq 1)$. As one can see, using serial extensions to embed an episode with geometric inter event times (governed by η) in the above fashion gives us control over the expected time span of an embedded occurrence. This information can be useful in readily setting reasonable thresholds on expiry time. η in conjunction with p (which governs the time between end and start of successive occurrences) gives an immediate approximate estimate of the mean frequency of an embedded pattern for a given length of data. This in turn aids us in readily choosing reasonable frequency thresholds.

We generate the noise event sequence as follows. For each event-type in the alphabet we generate a separate sequence of its occurrences with inter-event times distributed geometrically. For all *noise* event-types, namely event-types that are not in any of the embedded episodes, the geometric parameter is ρ $(0 < \rho \leq 1)$ and for all other event-types this parameter is set to $\rho/5$. This way, we introduce some random occurrences of the event-types associated with the embedded partial orders. All these streams are merged to form a single noise event sequence. Noise stream is generated in this way so that there may be multiple events (constituting noise) at the same time instant. We note here that the value of ρ alone does not indicate any percentage of noise. For example, with $\rho = 0.05$ we expect each noise event-type to appear once every 20 time-ticks and if there are 40 noise event-types, then (on the average) there would be two noise events at every time tick. Thus, even small values of ρ can insert substantial levels of noise in the data owing to the presence of a sufficient number of noise event-types.

Overall our method for synthetic data generation allows us to control the expected spans/frequency of embedded episodes independently of the level of noise. The merging of the various episode event sequences makes sure that the occurrences of different embedded episodes are sufficiently overlapped and possibly sharing some time ticks too. This in addition to introducing noise especially via some random occurrences of the event-types associated with the embedded partial orders makes the synthetic data sufficiently challenging for mining.

[12] A serial extension of a chain episode $(V_\alpha, <_\alpha, g_\alpha)$ is a serial episode $\beta = (V_\beta, <_\beta, g_\beta)$ where $V_\beta = V_\alpha$ and $g_\alpha = g_\beta$ such that $<_\alpha \subseteq <_\beta$.

While presenting our results, in all our tables, we give the values of different parameters in the table caption. In addition to ρ, p and η, the other parameters are as follows: M denotes the total number of event-types or the cardinality of $\mathcal{E}$, T represents the number of time ticks for which data is generated, T_X is the expiry-time threshold, f_{th} and H_{th} are the thresholds on frequency and bidirectional evidence.

8.2 Effectiveness of mining

To demonstrate the effectiveness of bidirectional evidence-based chain episode mining, we consider an event sequence with 2 embedded chain episodes (one injective and one non-injective) $\alpha_1 = (A \rightarrow (BCD) \rightarrow E \rightarrow F)$ and $\alpha_2 = ((J \rightarrow H) \rightarrow I \rightarrow (H(G \rightarrow J)))$. The event sequence generated consisted of about 25 000 events (using an alphabet of 50 event-types) with 10 000 time-ticks or distinct event times. One can easily see that the data has about 2.5 event-types on an average per time-tick. The caption of Table 4 gives the other parameters of data generation. Table 4 shows the results of our chain episode mining algorithm.[13] We show the number of candidates (*#Cand*) and the number of frequent episodes (*#Freq*) at different levels. We show results for three cases: (A) when only a frequency threshold, f_{th} is applied at each level, to determine the output, (B) when we apply f_{th} as usual, but also use a threshold, H_{th}, on bi-directional evidence, $H(\alpha)$, to post-filter the output, and (C) when we apply f_{th} and H_{th} *at each level* (during the level-wise procedure) as explained in Sect. 4.2 to generate candidates at the next level.

In all three scenarios, both the embedded patterns are reported as frequent. (To keep the terminology simple, we generally refer to the output as 'frequent episodes' even in cases that use H_{th}). A variety of patterns are reported frequent when only a frequency threshold is used (case A). At lower levels (2–4), in addition to the subepisodes of the embedded patterns, we observe (i) a substantial number of episodes involving one or more noise event-types (or event-types not part of the embedded episodes) and (ii) *Mixed* episodes which are purely a mix of the event-types from the two embedded episodes. At the higher levels (5 and 6), we do not observe either of these (noisy or the mixed variety) type of episodes reported frequent. Frequent episodes reported at higher levels involve episodes whose (multi) set of event-types exclusively match one of the maximal subepisodes of the embedded patterns. For instance at level 6, in addition to the embedded patterns, the frequent episodes reported *typically* include (i) huge number of non-maximal subepisodes of either of the embedded patterns, (ii) a few superepisodes of either of the embedded patterns and (iii) a considerable number of episodes which are neither subepisodes or superepisodes of the embedded patterns inspite of sharing the same g-map, which we refer to in short as *Neither*.

BE-based level-wise mining is effective in not only pruning most of the non-maximal subepisodes but also most of the episodes of type Neither. For example, at level 6, out of the 1989 reported frequent episodes, 2 are the interesting embedded patterns whereas the remaining 1987 patterns are uninteresting. These uninteresting episodes predominantly include non-maximal subepisodes of either of the embedded patterns and episodes of type neither. On filtering these frequent episodes based on their bidirectional evidence (BE) or $H(\alpha)$ value (case B), almost all such uninteresting patterns get pruned. *This is mainly possible because for both non-maximal subepisodes and episodes of type neither, there exists some edge present in the corresponding embedded episode but absent in these episodes.* This results in their BE being generally low as per the BE definition. In fact, BE-based thresholds are

[13] The source codes have all been written in C++. The experiments have been run on a 2.5GHz Pentium PC under a Linux operating system.

Table 4 Results obtained in three cases: (A) frequency threshold (f_{th}) only, (B) bidirectional evidence threshold (H_{th}) as a post filter, (C) both f_{th} and H_{th} level-wise

Level	(A)		(B)		(C)	
	#Cand	#Freq	#Cand	#Freq	#Cand	#Freq
1	50	50	50	50	50	50
2	3725	458	3725	458	3725	458
3	10,958	258	10,958	159	10,958	159
4	2610	592	2610	73	1595	66
5	3323	1453	3323	25	223	19
6	7766	1989	7766	3	47	3
7	10,143	0	10,143	0	5	0
Run time	890 s		920 s		205 s	

Patterns: α_1 and α_2, $\eta = 0.7$, $\rho = 0.05$, $p = 0.05$, $M = 50$, $f_{\text{th}} = 300$, $T_X = 15$, $H_{\text{th}} = 0.5$

generally effective in pruning non-maximal subepisodes at all levels. In this case, at level 6, the only non-embedded episode that is reported frequent is a non-maximal subepisode of α_1 having a BE close to 0.5. Note the run-time marginally increases because of the additional $H(\alpha)$ computation and post filtering.

When we additionally use $H(\alpha)$ also in the level-wise procedure, the frequent pattern output almost remains the same with considerable improvements in run-times. The run-time improvements are significantly more pronounced as size of the episodes increases similar to the case of injective episode mining [5]. We wish to reiterate that in case (B), at each level, the episodes used to generate candidates are the ones that only satisfy the frequency threshold even though under (# Freq) we report episodes that satisfy both f_{th} and H_{th}. The results in Table 4 show that using a level-wise threshold on $H(\alpha)$ makes the mining substantially efficient without missing important patterns present in the data as in the case of injective episodes. Hence, this strategy of mining (as also explained earlier in Sect. 4) is adopted in our subsequent experiments.

8.3 Scaling

The algorithm scales well with noise level, number of embedded patterns, and data length Tables 5, 6 and 7. In these experiments, the frequency thresholds have been chosen to be roughly 75% of the expected frequency of the embedded patterns. Further, expiry-time thresholds have been roughly chosen to be twice the expected span of the embedded patterns. The data in the noise level variation experiments use two 6-node embedded patterns namely $\beta_1 = (B \rightarrow ((C \rightarrow B)(D \rightarrow A)) \rightarrow C)$ and $\beta_2 = ((D\,C) \rightarrow C \rightarrow ((A \rightarrow D)B))$. While varying the number of embedded patterns, we use a variety of 8-node episodes inclusive of serial, parallel and general episodes (both injective and non-injective) for embedding in the data. The data in the data length variation experiments use two 8-node patterns for embedding, one of them being injective and the other non-injective with neither serial nor parallel. The run-times given are average values obtained over 10 different runs. In these tables, the column titled Avg. #Freq gives the number of frequent episodes averaged over the 10 runs. Column titled Avg. #FN denotes the number of embedded patterns missed by the algorithm, averaged over 10 trials. Avg. #FP denotes the average (over 10 runs) number of false positives, i.e. the number of spurious frequent patterns. Both Avg. #FP and Avg.

Table 5 Run-time as noise level is increased by varying ρ

ρ	Noise level (L_{ns})	Run-time (s)	Avg. #Freq	Avg. #FN	Avg. #FP
0.0	0.0	< 5	4.0	0	2.0
0.005	0.33	< 5	4.1	0	2.1
0.01	0.66	< 5	4.0	0	2.0
0.015	1.0	< 5	4.2	0	2.2
0.02	1.33	12	6.3	0	4.3
0.025	1.66	20	9.3	0.1	7.4
0.03	2.00	137	11.1	0.1	9.2

Patterns embedded: β_1 and β_2, $p = 0.033$, $\eta = 0.5$, $M = 50$, $f_{\text{th}} = 375$, $T_X = 20$, $H_{\text{th}} = 0.4$, $T = 20\,000$

Table 6 Run-time as the number of 8-node embedded patterns is increased

N_{emb}	Run-time (s)	Avg. #Freq	Avg. #FN	Avg. #FP
4	102	4.5	0	0.5
8	670	9.9	0.8	2.7
12	1810	18.5	1.4	7.9
16	4500	20.2	1.4	5.6

$\rho = 0.02$, $p = 0.05$, $\eta = 0.7$, $M = 200$, $f_{\text{th}} = 290$, $T_X = 17$, $H_{\text{th}} = 0.35$, $T = 10{,}000$

#FN indicate the number of frequent episodes at a level equal to the size of the respective embedded episodes (namely 8 for Table 6 and 6 for the other two tables).

Table 5 describes increase in run-times with noise level L_{ns}, which is the ratio of the number of noise events to the number of (embedded) episode events in the data. The run times are pretty reasonable for noise levels as high as about 2.0. Also, note the steady but tolerable increase in the number of false positives with noise level. Similarly, Table 7 describes the run-time variations with data length (number of events in the event sequence, denoted as n). We observe that the run-times increase almost linearly with data length. As the data length is increased, the ratio of f_{th}/T is kept constant, where T denotes the number of time ticks up to which we carry out the simulation. Table 6 shows the run-time variations with the number of embedded partial orders (N_{emb}). We observe that the algorithm scales reasonably with the density of the embedded patterns. The increase in run-times with the number of embedded patterns is because of increased number of candidates. We observe that the number of false negatives are tolerable across tables. We also infer from the Avg. #FP (false positives) column that there is no blow-up in the number of spurious patterns reported. Thus the mining algorithm reported here is quite effective in unearthing the embedded patterns.

At low noise the false positives reported were few and were mainly superepisodes of one of the embedded patterns. Specifically, Fig. 15 shows the reported superepisodes of β_1, namely β_1' and β_1'' . These superepisodes will have a lower frequency but higher BE (by the definition of BE) than the corresponding embedded pattern (β_1 in this case). They are reported as their frequencies lie in between f_{th} and the embedded pattern's frequency. Superepisodes of embedded patterns typically obtained by an addition of a very few edges tend to get mined because they satisfy the frequency threshold. As noise level increases, the false positives increase and are contributed to by episodes of type Neither and Mixed (involving event-types from both β_1 and β_2). Figure 16d illustrates an episode of the Neither type associated with

Table 7 Run-time as the data length is increased

T	Data length (n)	Run-time (s)	Avg. #Freq	Avg. #FN	Avg. #FP
10 000	20, 800	55	2.1	0	0.1
40 000	83, 000	278	5.4	0	3.4
70 000	145, 000	478	4.7	0	2.7
100 000	208, 000	671	2.6	0	0.6

$f_{\text{th}}/T = 0.028$, Patterns Embedded: two 8-node episodes, $\rho = 0.04$, $p = 0.05$, $\eta = 0.7$, $M = 50$, $T_X = 20$, $H_{\text{th}} = 0.5$

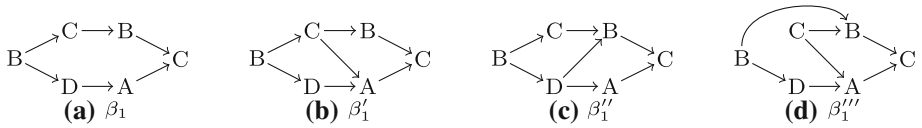


Fig. 15 β_1 and its associated false positives

Table 8 False positives comparison as noise level is increased by varying ρ

ρ	Noise level (L_{ns})	BEminer #FP	Closedminer #FP
0.0	0.0	2	32
0.005	0.33	2	24
0.01	0.66	2	61
0.015	1.0	2	153
0.02	1.33	2	140
0.025	1.66	2	127

Patterns Embedded: β_1 and β_2, $p = 0.033$, $\eta = 0.5$, $M = 50$, $f_{\text{th}} = 375$, $T_X = 20$, $H_{\text{th}} = 0.4$, $T = 20{,}000$

β. Note there is an edge from C to A in β_1''' that is missing in β_1 and similarly an edge from B to A in β_1 that is missing in β_1'''.

Given our algorithm's effectiveness and scaling abilities, we now briefly compare its pattern output with the other apriori approach of [25]. As discussed in the introduction, Tatti and Cule [25] proposes an algorithm (indicated as Closedminer in the table) for mining all frequent and closed chain episodes. Table 8 compares the false positive output of both the algorithms as the noise level is slowly increased. Figure 16 gives the run-time comparison of both the algorithms. Even though the Closedminer runs much faster than BEminer, the runtimes of our method are pretty reasonable for noise levels upto as high as 2, where the number of noise events is twice the no. of episode events. We note that there were no false negatives reported by our method (indicated as BEminer in the table) in this experiment. The false positives here are numbers obtained on a single trial and not averaged over independent trials as in the previous 3 tables. The 2 false positives reported by the BE miner here are the superepisodes of β_1. On the other hand, false positives reported by the closed miner belong to the Neither category mostly. They also include the two superepisodes of β_1 output by the BEminer. *Table 8 and Fig. 16 overall clearly demonstrate that our method reports significantly lower false positives compared to closed episode miner while maintaining reasonable run times.*

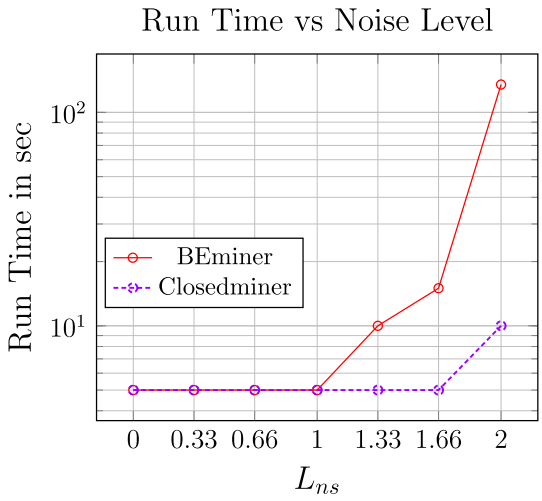


Fig. 16 Runtime comparison with increase in noise level

9 Conclusion

In this paper, we considered algorithms for discovering episodes with general partial orders but with a mild restriction on the partial order; the episodes that satisfy this restriction may be called strict episodes or chain episodes. The notion of bidirectional evidence introduced in the context of general injective episodes was extended here to chain episodes easily. We pointed out and proved a new restrictive monotonicity property that this measure satisfies in the context of chain episodes. This new monotonicity property also nicely generalizes the existing property known for injective episodes [5]. The overall main contribution of the paper is a bidirectional evidence-based level-wise discovery algorithm for mining in the space of chain episodes under the non-overlapped frequency count. Specifically, the candidate generation step that exploits this new monotonicity property is a non-trivial extension of that of [5]. This step is also very different from the candidate generation adopted by the apriori-based closed episode mining approach of [25]. We further gave correctness proofs showing that the proposed candidate generation generates all frequent chain episodes without any duplicates. We also presented an intelligent way of performing the transitivity check of a potential candidate, a key step in the candidate generation. We note that this efficient algorithm for transitivity check can also be used in candidate generation of injective episodes and it can improve the efficiency of the algorithm reported in [5]. The counting step employed an automata-based algorithm which was a natural extension of the counting employed in [5]. We introduced the notion of Earliest Transiting occurrences for chain episodes, using which novel correctness proofs for counting chain episodes were provided. We also pointed out some issues in extending our algorithm to non-chain episodes (*please refer to Appendix* E). Our simulation results finally shows the effectiveness and scalability of our discovery algorithm in detail. We also pointed out the superiority of our method in filtering out more uninteresting patterns (or false positives) in comparison with the other breadth-first search method for partial order episode mining in [25].

We extended the BE measure introduced earlier in [5] for injective episodes to chain episodes. Even though the measure was pretty effective in weeding out lots of uninteresting patterns in the level-wise procedure, it is not guaranteed to be exactly monotonic. This means there are instances when the embedded interesting pattern might actually be missed by the

algorithm. An interesting future direction of work can be in trying to modify this measure to make it exactly monotonic while retaining its fundamental characteristic of capturing evidence between pairs of unrelated event-types in a general episode.

A Comparison with the apriori-based closed episode miner

As stated earlier in Sect. 7, monotonicity property exploited by [27] (or its refined version [25]) and the one exploited here are different. This makes the candidate generation step proposed here substantially different from that of [27] or [25]. The algorithm in [25] produces candidate episodes that are generators of ultimately closed episodes. One needs to ultimately perform a closure operation on the generators to obtain what are called instance-closed episodes. The final set of closed episodes are obtained from post-filtering the set of instance-closed episodes. The main point to note is that [25] generates a potential candidate if all its subepisodes (including that of the same size) are also frequent. In other words, it exploits the subepisode structure that exists within episodes of the same size sharing the same g-map.

In this paper, we are using both frequency and BE to prune candidates. The monotonicity property satisfied by BE is a much weaker condition as compared to that of frequency alone. An ℓ-node episode is generated as a candidate if and only if all its $(\ell - 1)$-node maximal subepisodes obtained by dropping the last node among all nodes mapped to the same event-type are found frequent. In fact the BE-based measure does not demand the check for the existence of subepisodes of the same size as subepisodes of the same size are not guaranteed to have high BE in spite of the given episode's high BE. Continuing with the same serial episode event sequence example, the serial episode has a high BE in this data; however, all its subepisodes of the same size will have zero BE as they are obtained by dropping one or more edges from the parent serial episode. For instance, suppose there is an edge from node i to node j in α. If the edge between node i and j is dropped from α to obtain a β, then H_{ij}^{β} will be zero because in the occurrences tracked i precedes j always.

More specifically, Tatti and Cule [25] at each level ℓ, first mines for all frequent parallel episodes of size ℓ. It then starts generating potential candidates by progressively adding one edge at a time, doing subsequent necessary subepisode existence and closure checks before counting its frequency and mining for frequent generators. An episode with an ℓ-node episode and N edges is constructed as a potential candidate by combining two ℓ-node subepisodes of $(N-1)$ edges which share $(N-2)$ edges in common. In other words, the ℓ-node subepisode obtained by dropping an edge from the both the combinable episodes is the same. Note that the g_α map is assumed to be the same among all the above involved episodes. For each such generated episode, certain intelligent checks for transitive closure are first carried out. This is followed by checking for the existence of subepisodes (as frequent) obtained by dropping either an edge or a node. The last check before computing its frequency would be if its a generator too by making sure its not contained in the closure of any of its subepisodes.

In contrast to this, in the current approach we are constructing a potential candidate of size $(\ell + 1)$ by combining two ℓ-node episodes. This is because the BE-based monotonicity we are exploiting does not guarantee subepisodes of size $(\ell + 1)$ obtained by dropping edges alone to also have a high enough BE. The $(\ell - 1)$-node subepisode obtained by dropping an appropriate node from the combining ℓ-node episodes is the same. This is what makes the candidate generation steps fundamentally different in our approach from that of [25] or [27].

B Computation of BE

Algorithm 4 describes the pseudocode for computing the BE of a given episode. Maintaining multiple automata is easily done by maintaining two lists in addition to the state information consisting of : (i) $\mathcal{Q}$, the set of currently accepted nodes (ii) $\mathcal{W}$, the set of nodes an automaton is waiting for. The first is a list of first state transition times of each automata and the second is a list of associated binary matrices. Recall that if h is the occurrence tracked by an automaton, then by the time the automaton reaches its final state, the (i, j)-entry in the binary matrix would be 1 if and only if $t_{h(v_i)} < t_{h(v_j)}$. Both these lists are stored together in TimeMatrixList. The pseudocode assumes that $\mathcal{Q}$ and $\mathcal{W}$ store the integer indices of the associated episode nodes. Lines 6–10 consider the case when the automaton is in its start state. If an automaton is not in its start state, we first delete all those automaton whose associated occurrences evidently violate the expiry time constraint (Line 13). After this filtering, if there still exist automaton (TimeMatrixList being non-empty), we compute the next state, update the binary matrix of each of these automata. If the next state also happens to be the final state, then we use the binary matrix of the oldest automaton to update the CountMatrix. By the end of processing the entire event sequence, the (i, j)th element of the CountMatrix would contain f^{α}_{ij}, which can be further utilized to compute $H(\alpha)$ as explained in Sect. 4.

Algorithm 4: CountBE

Input: Episode α of size ℓ ($\ell > 1$) and the event sequence **D**.
Output: BE of episode α.

1 Initialize ListAutomata $\leftarrow \{(\phi, \mathcal{W}^{\alpha}_0, \phi)\}$;
2 Initialize $CountMatrix = \mathbf{0}$ ($\ell \times \ell$ matrix) ;
3 **foreach** $(E_i, t_i) \in$ **D do**
4 **foreach** $(\mathcal{Q}, \mathcal{W}$, TimeMatrixList$) \in$ ListAutomata **do**
5 **if** TimeMatrixList $== \phi$ **then**
6 **if** $\exists j \in \mathcal{W}\, s.t.\, E_i = g_{\alpha}(j)$ **then**
7 Compute next state $\mathcal{Q}_{nxt}$ using Eq. (3);
8 Compute next state $\mathcal{W}_{nxt}$ using Eq. (4);
9 Modify current $(\mathcal{Q}, \mathcal{W}$, TimeMatrixList$)$ to $(\mathcal{Q}_{nxt}, \mathcal{W}_{nxt}, (t_i, \mathbf{0}))$;
10 break;
11 **else**
12 **foreach** (StartTime, BinaryMatrix) $\in$ TimeMatrixList **do**
13 **if** $(t_i -$ StartTime$) > T_X$ **then** Delete (StartTime,BinaryMatrix) from TimeMatrixList;
14 **if** TimeMatrixList $\neq \phi$ **then**
15 **if** $\exists j \in \mathcal{W}\, s.t.\, E_i = g_{\alpha}(j)$ **then**
16 Compute next state $\mathcal{Q}_{nxt}$ using Eq. (3);
17 Compute next state $\mathcal{W}_{nxt}$ using Eq. (4);
18 Update the current $(\mathcal{Q}, \mathcal{W})$ by $(\mathcal{Q}_{nxt}, \mathcal{W}_{nxt})$;
19 **foreach** (StartTime, BinaryMatrix) $\in$ TimeMatrixList **do**
20 $B_{nxt} =$ BinaryMatrix;
21 **foreach** $(k, j)\, s.t.\, k \in \mathcal{Q}$ **do** $B_{nxt}(k, j) = 1$;
22 Update the current BinaryMatrix by B_{nxt};
23 **if** $|\mathcal{Q}_{nxt}| == \ell$ **then**
24 Choose (StartTimeMax, BinaryMatrixMax) from TimeMatrixList such that StartTimeMax is maximum start time;
25 Add BinaryMatrixMax to $CountMatrix$;
26 Remove $(\mathcal{Q}_{nxt}, \mathcal{W}_{nxt}$, TimeMatrixList$)$ from ListAutomata;
27 Use $CountMatrix$ to compute the BE of α using Eqs. (1) and (2);

C Implementation issues in counting

As explained earlier in Sect. 3.1, an ℓ-node episode α is represented using two data structures: an array $\alpha.g$ such that $\alpha.g[i] = g_\alpha(v_i)$, $i = 1, \ldots, \ell$ and a binary adjacency matrix, $\alpha.e$ storing the partial order ($<_\alpha$) information. As in the injective episodes case, to efficiently count a set of ℓ-node candidates, we use a collection of lists $waits()$, indexed by the set of all event-types. Each element in these various lists stores information about the currently active automata corresponding to the various candidates. A typical element in each of these lists is of the form $(\alpha, \mathbf{q}, \mathbf{w}, j)$, where, α is a candidate, $\mathbf{q}$ and $\mathbf{w}$ essentially represent the state of an automaton and j is an integer. $\mathbf{q}$ and $\mathbf{w}$ are ℓ-length binary vectors encoding the two sets $(\mathcal{Q}^\alpha, \mathcal{W}^\alpha)$, which represent a state in the FSA associated with α. For example, $\mathbf{q}[j] = 1$ iff $v_j \in \mathcal{Q}^\alpha$. For an event-type E, if $(\alpha, \mathbf{q}, \mathbf{w}, j) \in waits(E)$, it denotes that an automaton of the episode α is currently in state $(\mathbf{q}, \mathbf{w})$ and is waiting for an event-type $E = \alpha.g[j] = g_\alpha(v_j)$ to make a state transition (with $\mathbf{w}[j] = 1$). As an example, consider the automaton (Fig. 12) corresponding to $(F \rightarrow (E\ G) \rightarrow F)$ in a state with $\mathcal{Q}^\alpha = \{v_2\}$ and $\mathcal{W}^\alpha = \{v_1, v_4\}$. Here we would have $(\beta, \mathbf{q}, \mathbf{w}, 1) \in waits(E)$ and $(\beta, \mathbf{q}, \mathbf{w}, 4) \in waits(G)$ where $\mathbf{q} = [0\,1\,0\,0]$ and $\mathbf{w} = [1\,0\,0\,1]$.

In the injective episode case [5], since the g_α-map is injective, it was convenient to work with the set $X^\alpha = \{g_\alpha(v_1), g_\alpha(v_2) \ldots g_\alpha(v_N)\}$ while defining states of the associated automaton. Consequently, the binary vectors $\mathbf{q}$ and $\mathbf{w}$ coded for certain subsets of X^α as states there. If instead, $\mathbf{q}$ and $\mathbf{w}$ coded for subsets of V_α as states, the algorithm (with pseudocode) presented for injective episodes [5] would still go through (for injective episodes). Generalizing further, since the resultant FSA for general chain episodes turns out to be deterministic always, the counting algorithm for general injective episodes with all the implementation details of [5], would similarly go through for chain episodes also. *As explained in Sect. 6.3, the only addition would be that for each state one maintains multiple automata (unlike injective episodes where one needs to maintain at most one automata per state). This is easily done by maintaining the first state transition times of each automata (in a given state) in a list.* Hence, for all the implementation details of the counting step for chain episodes, refer [5].

D Property of ET occurrences

We now prove Property 2 introduced in Sect. 6.2. We restate it here for convenience.

Property 5 *Given a chain episode α and data stream $\mathbf{D}$, consider an ET occurrence h and another occurrence h' of α in $\mathbf{D}$ such that h' starts on or after $t_{\bar{h}(1)}$. Let $\mathbf{D}_j$ denote the first j events of $\mathbf{D}$. For every j, the set of all nodes in V_α whose associated events under h occur in $\mathbf{D}_j$ is a superset of the set of all nodes in V_α whose associated events under h' occur in $\mathbf{D}_j$.*

Proof We show this by induction of j. For any $j < \bar{h}(1)$, the property is obviously true. For $j = \bar{h}(1)$, where $\bar{h}(1) = h(v_1^h)$, $v_1^h \in V_\alpha$, if h' starts strictly after $t_{\bar{h}(1)}$, then the property is immediate. If h' also starts at $t_{\bar{h}(1)}$, then $h'(v_1^h)$ must be equal to j as we are dealing with chain episodes. (Recall that all nodes in $\mathcal{W}_0^\alpha$ are unrelated and hence must be mapped to distinct event-types under g_α for a chain episode α.) Let us assume Property 2 is true for some $j > \bar{h}(1)$. Let $\mathcal{Q}$ and $\mathcal{Q}'$ denote the set of all nodes in V_α whose associated events under h and h', respectively, occur in $\mathbf{D}_j$. By hypothesis, $\mathcal{Q}' \subseteq \mathcal{Q}$. If (E_{j+1}, t_{j+1}) is not a part of h', then the property is immediate for $j+1$. Suppose (E_{j+1}, t_{j+1}) is a part of h'. For convenience, we denote $h'^{-1}(j+1)$ by v_k. We now claim that $h(v_k)$ is between $\bar{h}(1)$ and

$(j+1)$ (both inclusive). Since h' is a valid occurrence, all parents of v_k belong to $\mathcal{Q}'$. Since $\mathcal{Q}' \subseteq \mathcal{Q}$, we also have seen events associated with all parents of v_k (in $<_\alpha$) under h in $\mathbf{D}_j$. Since h is ET (Definition 14), the event associated with v_k under h must be (E_{j+1}, t_{j+1}) or some event in $\mathbf{D}_j$ before it. Hence, the property continues to hold on $\mathbf{D}_{j+1}$ too. □

E Problems in handling non-chain episodes

The first point we want to make here is that non-chain episodes suffer from the problem of ambiguity in representation. For example, the 4-node episode $((A \rightarrow C)(A \rightarrow B))$ is not a chain episode. This episode has two representations in spite of constraining the g-map such that $(g_\alpha(v_1), \ldots, g_\alpha(v_N))$ is ordered as per the lexicographic ordering on $\mathcal{E}$ as shown in Fig. 17a, d. One can verify that both α (Fig. 17a) and α' (Fig. 17d) share the same set of occurrences on any event sequence. *This ambiguity creeps in mainly because the nodes which map to the event-type A are unrelated under* $<_\alpha$. This ambiguity also reflects in the equivalent array of event-types and adjacency matrix notation. As discussed earlier in Sect. 7, Tatti and Cule [26] considers discovery algorithms to output the most general episodes which includes $((A \rightarrow C)(A \rightarrow B))$. It also recognizes this issue of inherent ambiguity in representation for general episodes. The algorithm in [26] does not resolve this ambiguity in representation for most general episodes. It tackles it by actually comparing every currently generated (instance) closed episode during the DFS traversal of the space of all episodes, with the remaining currently discovered set of closed episodes. The comparison actually tests for a subepisode relationship whose computation can be very involved for non-chain episodes in general. In fact, it is shown to be NP-hard in general.

There would also be difficulties in counting occurrences of non-chain episodes. Consider the above non-chain episode α (Fig. 17a). To track an occurrence of such an episode, we would initially wait for two As and on seeing an A, we would need to accept the A associated with both v_1 and v_2. This means on seeing A there is more than one next state possible as per Definition 12. Generalizing this, one can show that the construction of an FSA for tracking occurrences of a non-chain episode α as per Definition 12 always leads to a non-deterministic finite state automaton (NFA). To track occurrences of such an α, one would first need to convert this NFA into an equivalent DFA. In the process of this conversion, the number of states in the equivalent DFA would be larger. In fact it is shown in [26] that checking if an event sequence contains an occurrence of an episode is an NP-complete problem. Thus, counting the occurrences is also not straight forward for non-chain episodes in addition to problems of ambiguous representation.

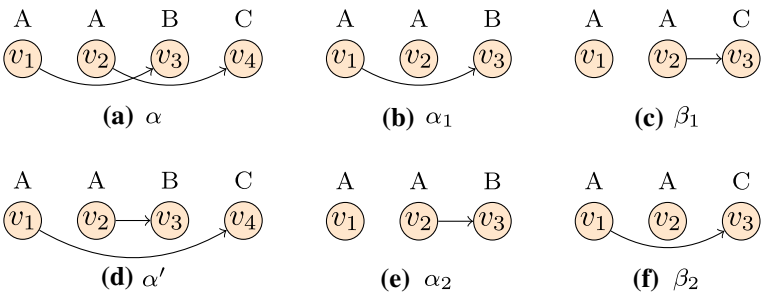


Fig. 17 Illustrates multiple representation problem of non-chain episode $((A \rightarrow B)(A \rightarrow C))$

Given these issues, it looks non-trivial to extend apriori-based discovery algorithms to the class of all episodes.

References

1. Achar A (2010) Discovering Frequent episodes with general partial orders. PhD thesis, Department of Electrical Engineering, Indian Institute of Science, Bangalore, India
2. Achar A, Sastry PS (2015) Statistical significance of general partial orders. Inf Sci 296:175–200
3. Achar A, Laxman S, Sastry PS (2012) A unified view of the apriori-based algorithms for frequent episode discovery. Knowl Inf Syst 31(2):223–250
4. Achar A, Ibrahim A, Sastry PS (2013) Pattern-growth based frequent serial episode discovery. Data Knowl Eng 87:91–108
5. Achar A, Laxman S, Raajay V, Sastry PS (2012) Discovering injective episodes with general partial orders. Data Min Knowl Discov 25(1):67–108
6. Achar A, Laxman S, Raajay V, Sastry PS (2009) Discovering general partial orders from event streams. In: Technical report arXiv: 0902.1227v2 [cs.AI]
7. Atallah MJ, Gwadera R, Szpankowski W (2004) Detection of significant sets of episodes in event sequences. In: Proceedings of the 4th IEEE international conference on data mining (ICDM), Brighton, UK, pp 3–10
8. Bouqata B, Caraothers CD, Szymanski BK, Zaki MJ (2006) Vogue: a novel variable order-gap state machine for modeling sequences. In: Proceedings of the 10th European conference on principles and practice of knowledge discovery in databases, vol 4213. Springer, Berlin, pp 42–54
9. Brown EN, Kass RE, Mitra PP (2004) Multiple neuronal spike train data analysis: state of art and future challenges. Nat Neurosci 7(5):456–461
10. Gan M, Dai H (2012) An efficient one-pass method for discovering bases of recently frequent episodes over online data streams. Int J Innov Comput Inf Control 8(7(A)):4675–4690
11. Gan M, Dai H (2014) Detecting and monitoring abrupt emergences and submergences of episodes over data streams. Inf Syst 39:277–289
12. Huang K, Chang C (2008) Efficient mining of frequent episodes from complex sequences. Inf Syst 33(1):96–114
13. Ibrahim A, Sastry S, Sastry PS (2016) Discovering compressing serial episodes from event sequences. Knowl Inf Syst 47(2):405–432
14. Iwanuma K, Takano Y, Nabeshima H (2004) On anti-monotone frequency measures for extracting sequential patterns from a single very-long sequence. In: Proceedings of the 2004 IEEE conference on cybernetics and intelligent systems, vol 1, pp 213–217
15. Laxman S, Sastry PS, Unnikrishnan KP (2005) Discovering frequent episodes and learning hidden Markov models: a formal connection. IEEE Trans Knowl Data Eng 17:1505–1517
16. Laxman S, Tankasali V, White RW (2008) Stream prediction using a generative model based on frequent episodes in event sequences. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'08), pp 453–461
17. Luo J, Bridges SM (2000) Mining fuzzy association rules and fuzzy frequent episodes for intrusion detection. Int J Intell Syst 15:687–703
18. Mannila H, Toivonen H, Verkamo AI (1997) Discovery of frequent episodes in event sequences. Data Min Knowl Discov 1(3):259–289
19. Nag A, Fu AW (2003) Mining frequent episodes for relating financial events and stock trends. In: Proceedings of 7th Pacific-Asia conference on knowledge discovery and data mining (PAKDD 2003). Springer, Berlin, pp 27–39
20. Patnaik D, Sastry PS, Unnikrishnan KP (2008) Inferring neuronal network connectivity from spike data: a temporal data mining approach. Sci Program 16:49–77
21. Patnaik D, Laxman S, Chandramouli B, Ramakrishnan N (2012) Efficient episode mining of dynamic event streams. In: IEEE international conference on data mining, pp 605–614
22. Sastry PS, Unnikrishnan KP (2010) Conditional probability based significance tests for sequential patterns in multi-neuronal spike trains. Neural Comput 22(4):1025–1059
23. Tatti N (2009) Significance of episodes based on minimal windows. In: Proceedings of 2009 IEEE international conference on data mining
24. Tatti N (2015) Ranking episodes using a partition model. Data Min Knowl Discov 29(5):1312–1342
25. Tatti N, Cule B (2012) Mining closed strict episodes. Data Min Knowl Discov 25(3):34–66

26. Tatti N, Cule B (2011) Mining closed episodes with simultaneous events. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining, pp 1172–1180
27. Tatti N, Cule B (2010) Mining closed strict episodes. In: Proceedings of 2010 IEEE international conference on data mining, pp 501–510
28. Unnikrishnan KP, Shadid BQ, Sastry PS, Laxman S (2009) Root cause diagnostics using temporal datamining, U.S.Patent no. 7509234, 24 Mar
29. Wang MF, Wu YC, Tsai MF (2008) Exploiting frequent episodes in weighted suffix tree to improve intrusion detection system. In: Proceedings of the 22nd international conference on advanced information networking and applications-workshops. IEEE Computer Society, Washington, pp 1246–1252

Avinash Achar got his B.E. degree from National Institute of Technology, Karnataka. He has a masters and Ph.D. (data mining) from the Indian Institute of Science, Bangalore. He was a post-doctoral fellow at NTNU, Norway. He is currently a Research Scientist at Tata Consultancy Services, Chennai. His research interests broadly span the areas of data mining and machine learning and their applications in different domains.

P.S. Sastry received his B.Sc. in Physics from Indian Institute of Technology, Kharagpur, and B.E. in Electrical Communication Engineering and Ph.D. from the Department of Electrical Engineering, both from Indian Institute of Science (IISc), Bangalore. Since 1986 he is a faculty member at the department of Electrical Engineering, IISc, where currently he is a professor. He was the chairman of the department during 2010–2015. He has held visiting positions at University of Massachusetts, Amherst, USA; University of Michigan, Ann Arbor, USA; General Motors Research Laboratories, Warren, USA; and Texas A&M University, College Station, USA. His research interests include Pattern Recognition, Machine Learning, Data Mining and Computational Neuroscience. Prof. Sastry received C.V. Raman Award for Young Scientists from Government of Karnataka, Hari Om Ashram Dr. Vikram Sarabhai Research Award from PRL, Ahmadabad, Most Valued Colleague Award from General Motors Corporation, and the Alumni Award for Excellence in Research from IISc. He is a Fellow of the Indian National Academy of Engineering and the National Academy of Sciences, India.