Knowl Inf Syst (2012) 31:223–250
DOI 10.1007/s10115-011-0408-2

REGULAR PAPER

A unified view of the apriori-based algorithms for frequent episode discovery

Avinash Achar · Srivatsan Laxman · P. S. Sastry

Received: 28 October 2010 / Revised: 7 March 2011 / Accepted: 20 March 2011 /
Published online: 27 May 2011

Abstract Frequent episode discovery framework is a popular framework in temporal data mining with many applications. Over the years, many different notions of frequencies of episodes have been proposed along with different algorithms for episode discovery. In this paper, we present a unified view of all the apriori-based discovery methods for serial episodes under these different notions of frequencies. Specifically, we present a unified view of the various frequency counting algorithms. We propose a generic counting algorithm such that all current algorithms are special cases of it. This unified view allows one to gain insights into different frequencies, and we present quantitative relationships among different frequencies. Our unified view also helps in obtaining correctness proofs for various counting algorithms as we show here. It also aids in understanding and obtaining the anti-monotonicity properties satisfied by the various frequencies, the properties exploited by the candidate generation step of any apriori-based method. We also point out how our unified view of counting helps to consider generalization of the algorithm to count episodes with general partial orders.

Keywords Frequent episode mining · Serial episodes · Apriori-based · Frequency notions

1 Introduction

Temporal data mining is concerned with finding useful patterns in sequential (and often symbolic) data streams [19]. Frequent episode discovery, first introduced in [17], is a popular framework for mining useful/interesting temporal patterns from sequential data.

A. Achar (✉) · P. S. Sastry
Indian Institute of Science, Bangalore, Karnataka, India
e-mail: achar.avinash@gmail.com

P. S. Sastry
e-mail: sastry@ee.iisc.ernet.in

S. Laxman
Microsoft Research Labs, Bangalore, Karnataka, India
e-mail: slaxman@microsoft.com

The framework has been successfully used in many application domains, e.g., analysis of alarm sequences in telecommunication networks [17], root cause diagnostics from faults log data in manufacturing [29], user-behavior prediction from web interaction logs [13], inferring functional connectivity from multineuronal spike train data [23–25], relating financial events and stock trends [20], protein sequence classification [2], intrusion detection [14,30], text mining [8], seismic data analysis [18] etc.

In the frequent episodes framework, the data is viewed as a single long sequence of events. Each event is characterized by a symbolic event-type (from a finite alphabet of symbols) and a time of occurrence. The patterns of interest are termed episodes. Informally, an episode is a short-ordered sequence of event-types. A *frequent* episode is one that occurs often enough in the given data sequence. Discovering frequent episodes is a good way to unearth temporal correlations in the data. Given a user-defined frequency threshold, the task is to efficiently obtain all frequent episodes in the data sequence.

An important design choice in frequent episode discovery is the definition of frequency of episodes. Intuitively, any frequency should capture the notion of the episode occurring many times in the data and, at the same time, should have an efficient algorithm for computing the same. There are many ways to define frequency, and this has given rise to different algorithms for frequent episode discovery [3,7–9,16–18]. In the original framework by Mannila et al. [17], frequency was defined as the number of fixed-width sliding windows over the data that contain at least one occurrence of the episode. Another notion for frequency is based on the number of *minimal* occurrences [16,17]. Two frequency definitions called *head frequency* and *total frequency* are proposed in [8] in order to overcome some limitations of the window-based frequency of [17]. In [9], two more frequency definitions for episodes were proposed, based on certain specialized sets of occurrences of episodes in the data.

Many of the algorithms, such as the WINEPI of [17] and the occurrence-based frequency counting algorithms of [10,12], use apriori-based discovery methods which use finite state automata as the basic building blocks for recognizing occurrences of episodes in the data sequence. An automata-based counting scheme for minimal occurrences has also been proposed in [4].

The multiplicity of frequency definitions and the associated algorithms for frequent episode discovery makes it difficult to compare the different methods. In this paper, we present a unified view of the apriori-based algorithms for frequent episode discovery under all the various frequency definitions. We present a generic automata-based counting algorithm for obtaining frequencies of a set of episodes and show that all the currently available counting algorithms can be obtained as special cases of this method. This viewpoint helps in obtaining useful insights regarding the kinds of occurrences tracked by the different algorithms. The framework also aids in deriving proofs of correctness for the various counting algorithms, many of which are not currently available in literature. Apart from counting, another crucial step in any apriori-based discovery method is that of candidate generation. Our framework also helps in understanding the anti-monotonicity condition satisfied by different frequencies, which is utilized in the respective candidate generation steps. Our general view can also help in generalizing current counting algorithms, which can discover only serial or parallel episodes, to the case of episodes with general partial orders, and we briefly comment on this in our conclusions. In addition to the unified view of all frequent episode discovery algorithms, the novel contributions of this paper are proofs of correctness for different algorithms and some quantitative relationships among different frequency measures.

The paper is organized as follows. Section 2 gives an overview of the episode framework and explains all the currently used frequency definitions in the literature. Section 3 presents our generic counting algorithm and shows that all current counting techniques for these

various frequencies can be derived as special cases. Section 4 gives proofs of correctness for the various counting algorithms utilizing this unified framework. Section 5 describes the quantitative relationships between the various frequencies. Section 6 discusses the candidate generation step for all these frequencies. In Sect. 7, we provide some discussion and concluding remarks. Section A is an appendix where some subepisode properties of the minimal window and non-interleaved based frequencies are proved.

2 Frequent episode discovery

In this section, we briefly review the framework of frequent episode discovery [17]. The data, referred to as an *event sequence*, is denoted by $\mathbb{D} = \langle (E_1, t_1), (E_2, t_2), \ldots, (E_n, t_n) \rangle$, where each pair (E_i, t_i) represents an *event*, and the number of events in the event sequence is n. Each E_i is a symbol (or *event-type*) from a finite alphabet, $\mathcal{E}$, and t_i is a positive integer representing the time of occurrence of the ith event. The sequence is ordered so that, $t_i \leq t_{i+1}$ for all $i = 1, 2, \ldots$. The following is an example event sequence with 10 events:

$$(A, 1), (A, 2), (A, 3), (B, 3), (A, 6), (A, 7), (C, 8), (B, 9), (D, 11), (C, 12), (A, 13), (B, 14), (C, 15). \quad (1)$$

We note that there can be events with different event-types occurring at the same time instant. There are formalisms that handle the case of events having finite but non-zero time duration [11,22]. We do not consider this in this paper.

Definition 1 An N-node episode, α, is defined as a triple, $(V_\alpha, \leq_\alpha, g_\alpha)$, where $V_\alpha = \{v_1, v_2, \ldots v_N\}$ is a collection of N nodes, $\leq_\alpha$ is a partial order on V_α and $g_\alpha : V_\alpha \rightarrow \mathcal{E}$ is a map that associates each node in α with an event-type from $\mathcal{E}$.

Thus, an episode is a (typically small) collection of event-types along with an associated partial order. When the order $\leq_\alpha$ is total, α is called a serial episode, and when the order is empty, α is called a parallel episode. In this paper, we restrict our attention to serial episodes.[1] Without loss of generality, we can now assume that the total order on the nodes of α is given by $v_1 \leq_\alpha v_2 \leq_\alpha \ldots \leq_\alpha v_N$. For example, consider a 3-node episode $V_\alpha = \{v_1, v_2, v_3\}$, $g_\alpha(v_1) = A$, $g_\alpha(v_2) = B$, $g_\alpha(v_3) = C$, with $v_1 \leq_\alpha v_2 \leq_\alpha v_3$. We denote such an episode by $(A \rightarrow B \rightarrow C)$.

Definition 2 An occurrence of episode α in an event sequence $\mathbb{D}$ is a map $h : V_\alpha \rightarrow \{1, \ldots, n\}$ such that $g_\alpha(v) = E_{h(v)}$ for all $v \in V_\alpha$, and for all $v, w \in V_\alpha$ with $v <_\alpha w$ we have $t_{h(v)} < t_{h(w)}$.

In the example event sequence (1), the events $(A, 2)$, $(B, 3)$, and $(C, 8)$ constitute an occurrence of $(A \rightarrow B \rightarrow C)$ while $(B, 3)$, $(A, 7)$, and $(C, 8)$ do not. Note that the events (in the data stream) constituting an occurrence of an episode need not be contiguous in the data stream. We use $\alpha[i]$ to refer to the ith in α. As per the above notation, $\alpha[i]$ is $g_\alpha(v_i)$. This way, an N-node (serial) episode α can be represented as $(\alpha[1] \rightarrow \alpha[2] \rightarrow \cdots \rightarrow \alpha[N])$.

In the above description, the data is viewed at an abstract level. What constitutes an event-type depends on the application. For example, consider the application of analyzing data in the form of fault-report-logs from an assembly line [12,29]. Here, each event is a fault report. The event-type would contain information relating to the station and subsystem where the

[1] From now on, we will simply use 'episode' to refer to a serial episode.

fault has occurred along with some fault codes to specify the type of fault. The episodes could capture some causative influences among different faults and hence aid in root-cause diagnostics. For example, if $A \rightarrow B \rightarrow C$ is a frequent episode then, when one sees fault C the actual cause may be A. As another example, consider analyzing data in the form of user-action-logs in a web browsing session [13]. Here, the data would be a sequence of user actions (clicking of different buttons in the web pages) in a browsing session. Event-types here capture some relevant description of the links that user can click. Frequent episodes here can capture some useful behavioral patterns of the user. Like in frequent itemset mining, one can use frequent episodes to generate rules. For example, if $A \rightarrow B \rightarrow C$ is a frequent episode, then we can generate a rule to say that if B follows A then soon C may follow. In this application, such rules can be used to predict user behavior which in turn can be used, e.g., for prefetching of pages by the browser, etc. In this paper, our interest is in a unified view of different frequent episode mining algorithms. Hence, we would be using the above abstract view of the data.

We next define the notion of a subepisode of an episode. This is very useful in designing algorithms for frequent episode discovery.

Definition 3 An 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) $g_\beta(v) = g_\alpha(f_{\beta\alpha}(v))$ for all $v \in V_\beta$, and (ii) for all $v, w \in V_\beta$ with $v <_\beta w$, we have $f_{\beta\alpha}(v) <_\alpha f_{\beta\alpha}(w)$ in V_α.

In other words, an episode β is said to be a *subepisode* of α if all the event-types in β also appear in α, and if their order in β is same as that in α. For example, $(A \rightarrow C)$ is a 2-node subepisode of the episode $(A \rightarrow B \rightarrow C)$, while $(B \rightarrow A)$ is not. If β is a subepisode of α then it is easy to see that every occurrence of α contains an occurrence of β [17].

Like in frequent itemset mining, the goal in episode mining is to discover all frequent episodes. A frequent episode is one whose *frequency* exceeds a user-defined threshold. The *frequency* of an episode is some measure of how often it occurs in the event sequence. As mentioned earlier, there are many different frequency measures proposed in literature. We discuss all these in the next subsection.

Given an occurrence h of an N-node (serial) episode α, $(t_{h(v_N)} - t_{h(v_1)})$ is called the *span* of the occurrence. We call any such constraint on span as an *expiry-time constraint*. The constraint we consider is specified by a threshold, T_X, such that occurrences of episodes whose span is greater than T_X are not considered while counting the frequency. In many applications, one may want to consider only such occurrences whose span is below some user-chosen limit. (This is because, occurrences constituted by events that are widely separated in time may not represent any underlying causative influences). Thresholds based on such constraints can also improve efficiency of the discovery process by reducing the search space.

One popular approach to frequent episode discovery is to use an Apriori-style level-wise procedure. At level k of the procedure, a "candidate generation" step combines frequent episodes of size[2] $(k-1)$ to build candidates (or potential frequent episodes) of size k using some kind of anti-monotonicity property (e.g., frequency of an episode cannot exceed frequency of any of its subepisodes). The second step at level k is called 'frequency counting' in which, the algorithm counts or computes the frequencies of all the candidates (through one pass over the data) and determines which of them are frequent.

[2] The size of an episode α is the number of nodes in V_α.

2.1 Various frequency definitions

There are many ways to define the frequency of an episode [8,9,15,17]. Intuitively, any definition must capture some notion of how often the episode occurs in the data. It must also admit an efficient algorithm to obtain the frequencies for a set of episodes. Further, to be able to apply a level-wise procedure, we need the frequency definition to satisfy some anti-monotonicity criterion. Additionally, we would also like the frequency definition to be conducive to statistical significance analysis [5,27,28].

In this section, we discuss various frequency definitions proposed in literature. (Recall that the data is an event sequence, $\mathbb{D} = \langle (E_1, t_1), \ldots (E_n, t_n) \rangle$).

Definition 4 [17] A window on an event sequence, $\mathbb{D}$, is a time interval $[t_s, t_e]$, where t_s and t_e are integers such that $t_s \leq t_n$ and $t_e \geq t_1$. The *window width* of $[t_s, t_e]$ is given by $(t_e - t_s)$. Given a user-defined window width T_X, the **window-based frequency** of α is the number of windows of width T_X which contain at least one occurrence of α.

For example, in the event sequence (1), there are 5 windows with window width 5 which contain an occurrence of $(A \rightarrow B \rightarrow C)$. These are: [7, 12], [10, 15], [11, 16], [12, 17] and [13, 18].

Definition 5 [17] The time-window of an occurrence, h, of α is given by $[t_{h(v_1)}, t_{h(v_N)}]$. A *minimal window* of α is a time-window which contains an occurrence of α, such that no proper subwindow 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 $\mathbb{D}$ (denoted f_{mi}) is defined as the number of minimal windows of α in $\mathbb{D}$.

In the example sequence (1) there are 3 minimal windows of $(A \rightarrow B \rightarrow C)$: [2, 8], [7, 12] and [13, 15].

Definition 6 [8] Given a window width k, the **head frequency** of α is the number of windows of width k which contain an occurrence of α starting at the left end of the window and is denoted as $f_{\text{h}}(\alpha, k)$.

Definition 7 [8] Given a window width k, the **total frequency** of α, denoted as $f_{\text{tot}}(\alpha, k)$, is defined as follows.

$$f_{\text{tot}}(\alpha, k) = \min_{\beta \preceq \alpha} f_{\text{h}}(\beta, k) \tag{2}$$

For a window width of 6, the head frequency $f_{\text{h}}(\gamma, 6)$ of $\gamma = (A \rightarrow B \rightarrow C)$ in (1) is 4. The total frequency of γ, $f_{tot}(\gamma, k)$, in (1) is 3 because the head frequency of $(B \rightarrow C)$ in (1) is 3.

Definition 8 [10] Two occurrences h_1 and h_2 of α are said to be *non-overlapped* if either $t_{h_1(v_N)} < t_{h_2(v_1)}$ or $t_{h_2(v_N)} < t_{h_1(v_1)}$. 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 $\mathbb{D}$ is *maximal* if $|H| \geq |H'|$, where H' is any other set of non-overlapped occurrences of α in $\mathbb{D}$. The **non-overlapped frequency** of α in $\mathbb{D}$ (denoted as f_{no}) is defined as the cardinality of a maximal non-overlapped set of occurrences of α in $\mathbb{D}$.

Two occurrences are non-overlapped if no event of one occurrence appears in between events of the other. The notion of a maximal non-overlapped set is needed since there can be many sets of non-overlapped occurrences of an episode with different cardinality [9]. The non-overlapped frequency of γ in (1) is 2. A maximal set of non-overlapped occurrences is $\langle (A, 2), (B, 3), (C, 8) \rangle$ and $\langle (A, 13), (B, 14), (C, 15) \rangle$.

Definition 9 [9] Two occurrences h_1 and h_2 of α are said to be *non-interleaved* if either $t_{h_2(v_j)} \geq t_{h_1(v_{j+1})}, \quad j = 1, 2, \ldots N-1$ or $t_{h_1(v_j)} \geq t_{h_2(v_{j+1})}, \quad j = 1, 2, \ldots N-1$. A set of occurrences H of α in $\mathbb{D}$ is *non-interleaved* if every pair of occurrences in the set is non-interleaved. A set H of non-interleaved occurrences of α in $\mathbb{D}$ is **maximal** if $|H| \geq |H'|$, where H' is any other set of non-interleaved occurrences of α in $\mathbb{D}$. The **non-interleaved frequency** of α in $\mathbb{D}$ (denoted as f_{ni}) is defined as the cardinality of a maximal non-interleaved set of occurrences of α in $\mathbb{D}$.

For example, the occurrences $\langle (A, 2), (B, 3), (C, 8)\rangle$ and $\langle (A, 3), (B, 9)(C, 12)\rangle$ are non-interleaved (though overlapped) occurrences of $(A \rightarrow B \rightarrow C)$ in $\mathbb{D}$. Together with $\langle (A, 13), (B, 14), (C, 15)\rangle$, these two occurrences form a set of maximal non-interleaved occurrences of $(A \rightarrow B \rightarrow C)$ in (1) and thus $f_{\text{ni}} = 3$.

Definition 10 [15] Two occurrences h_1 and h_2 of an N-node episode α are said to be *distinct* if $h_1(v_i) \neq h_2(v_j) \, \forall \, i, j = 1, 2 \ldots N$, i.e., the range of their corresponding maps do not intersect. (Informally, two occurrences of an episode are distinct if they do not share any event in the data stream). A set of occurrences is distinct if every pair of occurrences in it is distinct. A set H of distinct occurrences of α in $\mathbb{D}$ is **maximal** if $|H| \geq |H'|$, where H' is any other set of distinct occurrences of α in $\mathbb{D}$. The **distinct occurrence-based frequency** of α in $\mathbb{D}$ (denoted as f_d) is the cardinality of a maximal set of distinct occurrences of α in $\mathbb{D}$.

The three occurrences that constituted the maximal non-interleaved occurrences of $(A \rightarrow B \rightarrow C)$ in (1) also form a maximal set of distinct occurrences in (1).

The first frequency proposed in the literature was the window-based count [17] and was originally applied for analyzing alarms in a telecommunication network. It uses an automata-based algorithm called WINEPI for counting. Candidate generation exploits the anti-monotonicity property that all subepisodes are at least as frequent as the parent episode. A statistical significance test for frequent episodes based on the window-based count was proposed in [6]. There is also an algorithm for discovering frequent episodes with a maximum-gap constraint under the window-based count [3].

The minimal window-based frequency and a level-wise procedure called MINEPI to discover episodes based on minimal windows were also proposed in [17]. This algorithm has high space complexity since the exact locations of all the minimal windows of the various episodes of a given size are kept in memory. Nevertheless, it is useful in rule generation. An efficient automata-based scheme for counting the number of minimal windows (along with a proof of correctness) was proposed in [4]. The problem of statistical significance of minimal windows was recently addressed in [28]. An algorithm for discovering episodes and extracting rules based on minimal occurrences under a maximal gap constraint has been proposed in [18]. Unlike the apriori-based methods which employ a breadth-first search of the lattice of episodes, it employs a depth-first search strategy of this lattice.

In the window-based frequency, the window width is essentially an expiry-time constraint (an upper bound on the span of the episodes). However, if the span of an occurrence is much smaller than the window width, then its frequency is artificially inflated because the same occurrence will be found in several successive sliding windows. The head frequency measure, proposed in [8], is a variant of the window-based count intended to overcome this problem. Based on the notion of head frequency, Huang and Chang [7] presents two algorithms MINEPI+ and EMMA. Both these algorithms employ a depth-first-based traversal of the episode lattice similar to the algorithm in [18]. Huang and Chang [7] also points out how head frequency can be a better choice for rule generation compared with the window-based or the minimal window-based counts. Under the head frequency count, however, there can be

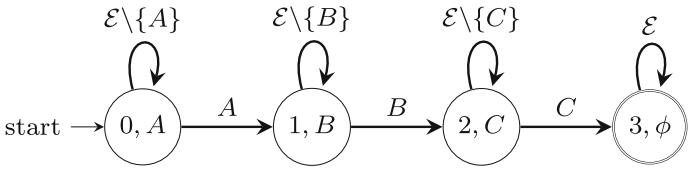

Fig. 1 Automaton for tracking occurrences of $\alpha = (A \rightarrow B \rightarrow C)$

episodes whose frequency is substantially higher than some of their subepisodes (see [8] for details). To circumvent this, Iwanuma et al. [8] propose the idea of total frequency. Currently, there is no statistical significance analysis based on head frequency or total frequency.

An efficient automata-based counting algorithm under the non-overlapped frequency measure (along with a proof of correctness) can be found in [12]. A statistical significance test for the same is proposed in [10]. However, the algorithm in [12] does not handle any expiry-time constraints. An efficient automata-based algorithm for counting non-overlapped occurrences under expiry-time constraint was proposed in [9,10] though this has higher time and space complexity than the algorithm in [12]. No proofs of correctness are available for non-overlapped occurrences under an expiry-time constraint. Algorithms for frequent episode discovery under the non-interleaved frequency can be found in [9]. No proofs of correctness are available for these algorithms.

Another frequency measure we discuss in this paper is based on the idea of distinct occurrences proposed in [15]. To the best of our knowledge, no algorithms are available for counting frequency under this measure. The unified view of automata-based counting that we will present in this paper can be readily used to design algorithms for counting distinct occurrences of episodes.

3 Unified view of all the automata-based algorithms

In this section, we present a generic algorithm for obtaining frequencies of episodes under the different frequency definitions listed in Sect. 2.1. The basic ingredient in all the algorithms is a simple finite state automaton (FSA) that is used to recognize (or track) an episode's occurrences in the event sequence.

The FSA for recognizing occurrences of $(A \rightarrow B \rightarrow C)$ is illustrated in Fig. 1. In general, an FSA for an N-node serial episode $\alpha = (\alpha[1] \rightarrow \alpha[2] \rightarrow \cdots \rightarrow \alpha[N])$ has $(N+1)$ states. The first N states are represented by a pair $(i, \alpha[i+1]), i = 0, \ldots N-1$. The $(N+1)$th state is (N, ϕ) where ϕ is a null symbol. Intuitively, if the FSA is in state $(j, \alpha[j+1])$, it means that the FSA has already seen the first j event-types of this episode and is now waiting for $\alpha[j+1]$; if we now encounter an event of type $\alpha[j+1]$ in the data, it can accept it (that is, it can transit to its next state). The start (first) state of the FSA is $(0, \alpha[1])$. The $(N+1)$th state is the accepting state because when an automaton reaches this state, a full occurrence of the episode is tracked. As we shall see later, all counting algorithms are obtained by different ways of managing a set of such automata.

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

Definition 11 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 $t_{h_1(v_i)} \neq t_{h_2(v_i)}$ is such that $t_{h_1(v_i)} < t_{h_2(v_i)}$. This is a total order on the set $\mathcal{H}$.

We next introduce a class of occurrences called *earliest transiting* (ET) occurrence. Many of the current algorithms for episode counting restrict their search to this smaller class of occurrences (instead of tracking all occurrences). This notion of ET occurrences, proposed in this paper, is very useful in the analysis of different frequency counting algorithms. We formally define an earliest transiting (ET) occurrence as follows.

Definition 12 An occurrence h of a serial episode α is called earliest transiting if $t_{h(v_i)}$ is the first occurrence time of event-type $\alpha[i]$ after $t_{h(v_{i-1})} \forall i = 2, 3 \ldots N$.

We denote by $\mathcal{H}^e$ the set of all earliest transiting occurrences of a given episode. We denote the ith occurrence (as per the lexicographic ordering of occurrences) in $\mathcal{H}^e$ as h_i^e.

To illustrate ET occurrences and discuss all algorithms in this section, we consider the episode $\alpha = (A \rightarrow B \rightarrow C \rightarrow D)$ and its occurrences in the data stream $\mathbb{D}_1$ given by

$$\mathbb{D}_1 = (A, 1)(B, 3)(A, 4)(A, 5)(C, 7)(B, 9)(C, 11)(A, 14)(D, 15)(C, 16)(B, 17)$$
$$(D, 18)(A, 19)(C, 20)(B, 21)(A, 22)(D, 23)(B, 24)(C, 25)(D, 29)(C, 30)(D, 31)$$

Also, while discussing various algorithms in this section, we represent any occurrence h by $[t_{h(v_1)} \ t_{h(v_2)} \ldots t_{h(v_N)}]$, which is the vector of times of the events that constitute the occurrence. In the above stream, consider the occurrence of α constituted by the events $(A, 4)$, $(B, 9)$, $(C, 11)$ and $(D, 15)$. This occurrence would be represented by the vector of times [4 9 11 15]. There are 6 earliest transiting occurrences of α in $\mathbb{D}_1$. As per the time vector notation, they are h_1^e = [1 3 7 15], h_2^e = [4 9 11 15], h_3^e = [5 9 11 15], h_4^e = [14 17 20 23], h_5^e = [19 21 25 29] and h_6^e = [22 24 25 29].

The simplest of all automata-based frequency counting algorithms is the one for counting non-overlapped occurrences [12], which uses only 1-automata per episode. (We call it algorithm NO here). At the start, one automaton for each of the candidate episodes is initialized in its start state. Each of the automata makes a state transition as soon as a relevant event-type appears in the data stream. Whenever an automaton reaches its final state, frequency of the corresponding episode is incremented, the automaton is removed from the system, and a fresh automaton for the episode is initialized in the start state. As is easy to see, this method will count non-overlapped occurrences of episodes. Under the NO algorithm, we denote the occurrence tracked by the ith automaton initialized for α as h_i^{no}.

In our example, algorithm NO tracks the following two occurrences of the episode α: (i) h_1^{no} = [1 3 7 15] and (ii) h_2^{no} = [19 21 25 29], and the corresponding non-overlapped frequency is 2. It is easy to see that all occurrences tracked by algorithm NO are earliest transiting. The ET occurrences tracked by the NO algorithm are $h_1^{no} = h_1^e$ and $h_2^{no} = h_5^e$.

While the algorithm NO is very simple and efficient, it cannot handle any expiry-time constraint. Recall that the expiry-time constraint specifies an upper bound, T_X, on the span of any occurrence that is counted. Suppose we want to count with $T_X = 9$. Both the occurrences tracked by NO have spans greater than 9, and hence, the resulting frequency count would be zero. However, h_4^e is an occurrence which satisfies the expiry-time constraint. Algorithm NO cannot track h_4^e because it uses only one automaton per episode, and the automaton has to make a state transition as soon as the relevant event-type appears in the data.

To overcome this limitation, the algorithm can be modified so that a new automaton is initialized in the start state, whenever an existing automaton moves out of its start state. All automata make state transitions as soon as they are possible. Each such automaton would track an earliest transiting occurrence. In this process, two automata may reach the same state. In our example, after seeing $(A, 5)$, the second and third automata to be initialized

for α would be waiting in the same state (ready to accept the next B in the data). Clearly, both automata will make state transitions on the same events from now on, and so, we need to keep only one of them. We retain the newer or most recently initialized automaton (in this case, the third automaton) since the span of the occurrence tracked by it would be smaller. When an automaton reaches its final state, if the span of the occurrence tracked by it is less than T_X, then the corresponding frequency is incremented and all automata of the episode except the one waiting in the start state are retired. (This ensures we are tracking only non-overlapped occurrences). When the occurrence tracked by the automaton that reaches the final state fails the expiry constraint, we just retire the current automaton; any other automata for the episode will continue to accept events. Under this modified algorithm, in $\mathbb{D}_1$, the first automaton that reaches its final state tracks h^e_3, which violates the expiry-time constraint of $T_X = 9$. So, we drop only this automaton. The next automaton that reaches its final state tracks h^e_4. This occurrence has span less than $T_X = 9$. Hence, we increment the corresponding frequency count and retire all current automata for this episode. Since there are no other occurrences non-overlapped with h^e_4, the final frequency would be 1. We denote this algorithm for counting the non-overlapped occurrences under an expiry-time constraint as NO-X. The occurrences tracked by both NO and NO-X would be earliest transiting.

Note that several earliest transiting occurrences may end simultaneously. For example, in $\mathbb{D}_1$, h^e_1, h^e_2, and h^e_3 all end together at $(D, 15)$. Apart from $\{h^e_1, h^e_5\}$, the sets of occurrences $\{h^e_1, h^e_6\}$, $\{h^e_2, h^e_5\}$, $\{h^e_2, h^e_6\}$ and $\{h^e_3, h^e_5\}$, $\{h^e_3, h^e_6\}$ also form maximal sets of non-overlapped occurrences. Sometimes (e.g., when determining the distribution of spans of occurrences for an episode) we would like to track the *innermost* one among the occurrences that are ending together. In this example, this means we want to track the set of occurrences $\{h^e_3, h^e_6\}$. This can be done by simply omitting the expiry-time check in the NO-X algorithm. (That is, whenever an automaton reaches final state, irrespective of the span of the occurrence tracked by it, we increment frequency and retire all other automata except for the one in start state). We denote this as the NO-I algorithm, and this is the algorithm proposed in [10].

In NO-I, if we only retire automata that reached their final states (rather than retire all automata except the one in the start state), we have an algorithm for counting minimal occurrences (denoted MO). In our example, the automata tracking h^e_3, h^e_4, and h^e_6 are the ones that reach their final states in this algorithm. The time-windows of these occurrences constitute the set of all minimal windows of α in $\mathbb{D}_1$. Expiry-time constraints can be incorporated by incrementing frequency only when the occurrence tracked has span less than the expiry-time threshold. The corresponding expiry-time algorithm is referred to as MO-X.

The window-based counting algorithm (which we refer to as WB) is also based on tracking earliest transiting occurrences. WB also uses multiple automata per episode to track minimal occurrences of episodes like in MO. The only difference lies in the way frequency is incremented. The algorithm essentially remembers, for each candidate episode, the last minimal window in which the candidate was observed. Then, at each time tick, effectively, if this last minimal window lies within the current sliding window of width T_X, frequency is incremented by one. This is because, an occurrence of episode α exists in a given window w if and only if w contains a minimal window of α.

It is easy to see that head frequency with a window width of T_X is simply the number of earliest transiting occurrences whose span is less than T_X. Thus, we can have a head frequency counting algorithm (referred to here as HD) that is similar to MO-X except that when two automata reach the same state simultaneously we do not remove the older automaton. This way, HD will track all earliest transiting occurrences which satisfy an expiry-time constraint of T_X. For $T_X = 10$ and for episode α, HD tracks h^e_3, h^e_4, h^e_5, and h^e_6 and returns a frequency count of 4. The total frequency count for an episode α is the minimum of the head

frequencies of all its subepisodes (including itself). This can be efficiently computed as the minimum of the head frequency of α and the total frequency of its $(N-1)$-suffix subepisode $(\alpha[2] \rightarrow \alpha[3] \rightarrow \cdots \rightarrow \alpha[N])$, which would have been computed in the previous pass over the data. (See [8] for details). The head frequency counting algorithm can have high space complexity as all the time instants at which automata make their first state transition need to be remembered.

The non-interleaved frequency counting algorithm (which we refer to as NI) differs from the minimal occurrence algorithm in that, an automaton makes a state transition only if there is no other automaton of the same episode in the destination state. Unlike the other frequency counting algorithms discussed so far, such an FSA transition policy will track occurrences which are not necessarily earliest transiting. In our example, until the event $(A, 4)$ in the data sequence, both the minimal and non-interleaved algorithms make identical state transitions. However, on $(A, 5)$, NI will not allow the automaton in state $(0, A)$ to make a state transition as there is already an active automaton for α in state $(1, B)$ which had accepted $(A, 4)$ earlier. Eventually, NI tracks the occurrences $h_1^{ni} = [1\ 3\ 7\ 15]$, $h_2^{ni} = [4\ 9\ 16\ 18]$, $h_3^{ni} = [14\ 17\ 20\ 23]$ and $h_4^{ni} = [19\ 21\ 25\ 29]$.

While there are no algorithms reported for counting distinct occurrences, we can construct one using the same ideas. Such an algorithm (to be called as DO) differs from the one for counting minimal occurrences, in allowing multiple automata for an episode to reach the same state. However, on seeing an event (E_i, t_i) which multiple automata can accept, only one of the automata (the oldest among those in the same state) is allowed to make a state transition; the others continue to wait for future events with the same event-type as E_i to make their state transitions. The set of maximal distinct occurrences of α in $\mathbb{D}_1$ are $h_1^d = h_1^e, h_2^d = [4\ 9\ 11\ 18], h_3^d = [5\ 17\ 20\ 23], h_4^d = [14\ 21\ 25\ 29]$ and $h_5^d = [19\ 24\ 30\ 31]$ which are the ones tracked by this algorithm.

We can also consider counting *all* occurrences of an episode even though it may be inefficient. The algorithm for counting *all* occurrences (referred to as the AO) allows all automata to make transitions whenever the appropriate events appear in the data sequence. However, at each state transition, a copy of the automaton in the earlier state is added to the set of active automata for the episode.

3.1 A unified algorithm for frequency counting

From the above discussion, it is clear that by manipulating the FSA (that recognize occurrences) in different ways we get counting schemes for different frequencies. The choices to be made in different algorithms essentially concern when to initiate a new automaton in the start state, when to retire an existing automaton, when to effect a possible state transition, and when (and by how much) to increment the frequency. We now present a unified scheme incorporating all this in *Algorithm* 1 for obtaining frequencies of a set of serial episodes. The algorithm takes as input a set of candidate N-node serial episodes and an event stream $\mathbb{D}$. The form of the data stream is such that event-types sharing the same time of occurrence $\bar{t}_i$ are all put together as a set $\mathcal{E}_i$. This form of the data stream helps us illustrate the algorithm more lucidly. This algorithm has five boolean variables, namely, TRANSIT, COPY-AUTOMATON, JOIN-AUTOMATON, INCREMENT-FREQ, and RETIRE-AUTOMATON. The counting algorithms for all the different frequencies are obtained from this general algorithm by suitably setting the values of these boolean variables (either by some constants or by values calculated using the current context in the algorithm). Tables 2, 3, 4, 5, 6 specify the choices needed to obtain the algorithms for different frequencies. (A list of all algorithms is given in Table 1).

Algorithm 1 Unified Algorithm for counting serial episodes

Input: Set $\mathcal{C}_N$ of N-node serial episodes, event stream $\mathbb{D} = \langle(\mathcal{E}_1, \bar{t}_1), \ldots, (\mathcal{E}_n, \bar{t}_m))\rangle$.
Output: Frequencies of episodes in $\mathcal{C}_N$

```
1: for all α ∈ C_N do
2:    Initialize an automaton of α waiting in the start state.
3:    Initialize frequency of α to ZERO.
4: for i = 1 to m do
5:    for each automaton, A, ready to accept event-type E ∈ E_i do
6:       α := candidate associated with A;
7:       j := state which A is ready to transit into;
8:       if TRANSIT then
9:          if COPYAUTOMATON then
10:            Add Copy of A to collection of automata.
11:         Transit A to state j
12:         if ∃ an earlier automaton of α already in state j (before t̄_i) but not waiting for any E ∈ E_i then
13:            if JOIN-AUTOMATON then
14:               Retain A and retire earlier automaton
15:         if A reached final state then
16:            Retire A.
17:            if INCREMENT-FREQ then
18:               Increment frequency of α by INC.
19:               if RETIRE-AUTOMATON then
20:                  Retire all automaton of α and create a state '0' automaton.
```

Table 1 Various frequency counts

WB	Windows based
MO	Minimal occurrences based
MO-X	Minimal occurrence with expiry-time constraints
NO	Non-overlapped
NO-I	Non-overlapped innermost
NO-X	Non-overlapped with expiry-time constraints
NI	Non-interleaved
DO	Distinct occurrences based
AO	All occurrences based
HD	Head frequency

Table 2 Conditions for TRANSIT=TRUE

WB, MO, MO-X, HD NO, NO-X, NO-I AO	Always
NI	If there does not exist an earlier automaton of α already in target state j (before current time $\bar{t}_i$) but not waiting for any $E \in \mathcal{E}_i$
DO	No other earlier automaton for α waiting in same state can transit on an event-type $E \in \mathcal{E}_i$

Table 3 Conditions for COPY-AUTOMATON = TRUE

WB, MO, MO-X, HD, NI, NO-X, NO-I, DO	Only if $\mathcal{A}$ is in start state
NO	Never
AO	Always

Table 4 Conditions for JOIN-AUTOMATON = TRUE

WB, MO, MO-X, NO-X, NO-I	Always
DO, AO, HD, NO, NI	Never

Table 5 Conditions for INCREMENT-FREQ = TRUE

MO, NO, NI, DO, AO, NO-I	Always
WB, NO-X MO-X, HD	If time difference between first and last state transitions is less than T_X (window width for WB, expiry time for others)

Table 6 Conditions for RETIRE-AUTOMATA = TRUE

NO, NO-X, NO-I	Always
WB, MO, MO-X HD, NI, DO, AO MO-X	Never

As can be seen from our general algorithm, when an event-type for which an automaton is waiting is encountered in the data, the automaton can accept it only if the variable TRANSIT is true. Hence, for all algorithms that track earliest transiting occurrences, TRANSIT will be set to true as can be seen from Table 2. For algorithms NI and DO where we allow the state transition only if some condition is satisfied.

The condition COPY-AUTOMATON (Table 3) is for deciding whether or not to leave another automaton in the current state when an automaton is transiting to the next state. Except for NO and AO, we create such a copy only when the currently transiting automaton is moving out of its start state. In NO, we never make such a copy (because this algorithm uses only one automaton per episode), while in AO, we need to do it for every state transition.

As we have seen earlier, in some of the algorithms, when two automata for an episode reach the same state, the older automaton is removed. This is controlled by JOIN-AUTOMATON, as given by Table 4. The JOIN-AUTOMATON condition is set to true for algorithms which retain only the newer automaton when two automata come to the same state. Also, for algorithms like AO, DO, and HD which allow multiple automata to remain in the same state, the JOIN-AUTOMATON is set to false so that no automata are retired. Note that this condition is not relevant for NO and NI because in these algorithms we will never have two automata reaching the same state.

INCREMENT-FREQUENCY (Table 5) is the condition under which the frequency of an episode is incremented when an automaton reaches its final state. This increment is always done for algorithms that have no expiry-time constraint or window width. For the others, we increment the frequency only if the occurrence tracked satisfies the constraint.

Table 7 Values taken by INC

INC = 1 for all counts except WB.
For Window-Based count (WB),
If (first window which contains current minimal occurrence also contains the previous minimal occurrence), then
INC = Time diff. between start of last window containing the current minimal occurrence and the start of last window which contains previous minimal occurrence.
else
INC = Time difference between the first and last window containing the current occurrence +1.

RETIRE-AUTOMATA condition (Table 6) is concerned with the removal of all automata of an episode when a complete occurrence has been tracked. This condition is true only for the non-overlapped occurrence-based counting algorithms.

Apart from the five boolean variables explained above, our general algorithm contains one more variable, namely, INC, which decides the amount by which frequency is incremented when an automaton reaches the final state. Its values for different frequency counts are listed in Table 7. For all algorithms except WB, we set $INC = 1$. We now explain how frequency is incremented in WB. To count the number of sliding windows that contain at least one occurrence of the episode, whenever a new minimal occurrence enters a sliding window, we can calculate the number of consecutive windows in which this new minimal occurrence will be found in. For example, in $\mathbb{D}_1$, with a window width of $T_X = 16$, consider the first minimal occurrence of $(A \rightarrow B \rightarrow C \rightarrow D)$, namely, the occurrence constituted by events $(A, 5)$, $(B, 9)$, $(C, 11)$, and $(D, 15)$. The first sliding window in which this occurrence can be found is $[-1, 15]$. The occurrence stays in consecutive sliding windows, until the sliding window $[5, 21]$. When this first minimal occurrence enters the sliding window $[-1, 15]$, we observe that there is no other "older" minimal occurrence in $[-1, 15]$, and hence, as per the *else* condition in *Table* 7, the INC is incremented by $(5 - (-1) + 1) = 7$. Similarly, when the second minimal occurrence enters the sliding window $[7, 23]$, we increment INC by $(14 - 7 + 1 = 8)$. The third minimal occurrence (constituted by the events $(A, 22)$, $(B, 24)$, $(C, 25)$, and $(D, 29)$) first enters the sliding window $[13, 29]$, with the second minimal window still occurring within this window. This third minimal occurrence remains in consecutive sliding windows until $[22, 38]$. As per the *if* condition of Table 7, INC is incremented by $22 - 14 = 8$. Hence, the window-based frequency is $f_{\text{wi}} = 23$ in $\mathbb{D}_1$.

Remark 1 Even though we included AO (for counting all occurrences of an episode) for sake of completeness, this is not a good frequency measure. This is mainly because it does not seem to satisfy any anti-monotonicity condition. For example, consider the data sequence $<AABBCC>$. There are 8 occurrences of $(A \rightarrow B \rightarrow C)$ but only 4 occurrences of each of its 2-node subepisodes. Also, its space complexity can be high when one desires to count with a large expiry-time constraint.

4 Proofs of correctness

In this section, we present correctness proofs of the various frequency counting algorithms presented in Sect. 3 (all of which are specific instances of *Algorithm* 1).

4.1 Minimal window counting algorithm

We first present a proof of correctness for the Minimal Occurrence counting algorithm (MO). Our proof methodology is different from the one presented in [4]. We briefly explain the proof in [4] at the end of this section. Our analysis of the MO algorithm leads to a better understanding of the ET occurrences in general and the occurrences tracked by NO, NO-I, and NO-X algorithms in particular. Also, it leads us to the relationship between minimal and non-interleaved occurrences as explained in Sect. 5. Another advantage of our proof of correctness is that it can be generalized to the case of episodes with general partial orders. For this, instead of using the automaton for serial episodes (as described in the previous section), we would need to employ a more general finite state automaton to track occurrences of episodes with unrestricted partial orders. This is discussed briefly in Sect. 7.

Lemma 1 *Suppose h is an earliest transiting occurrence of an N-node episode α. If h' is any general occurrence such that $t_{h(v_1)} \leq t_{h'(v_1)}$, then $h(v_i) \leq h'(v_i) \forall i = 1, 2, \ldots, N$.*

This lemma follows easily from the definition of ET occurrence.

Remark 2 Recall that h_i^e is the ith earliest transiting (ET) occurrence of an episode. Thus, by definition, $h_i^e(v_1) < h_j^e(v_1)$ and $h_i^e <_\star h_j^e$ whenever $i < j$. Hence, from the above lemma, we have $h_i^e(v_k) \leq h_j^e(v_k)$ for all k and $i < j$. In particular, we have, $h_i^e(v_1) < h_{i+1}^e(v_1)$ and $h_i^e(v_N) \leq h_{i+1}^e(v_N)$, for an N-node episode.

The main idea of our proof is that, to find all minimal windows of an episode, it is enough to capture a certain subset of earliest transiting occurrences. This is because any minimal window is also a window of some ET occurrence. Hence, if we can exactly characterize the set of ET occurrences whose windows are minimal, we also have a complete characterization for all minimal windows.

Lemma 2 *An earliest transiting (ET) occurrence h_i^e, of an N-node episode, is not a minimal occurrence if and only if $h_i^e(v_N) = h_{i+1}^e(v_N)$.*

Proof The "if" part follows easily from Remark 2. For the "only if" part, let us denote by $w = [n_s, n_e] = [h_i^e(v_1), h_i^e(v_N)]$ the window of h_i^e. Given that w is not a minimal window, we need to show that $h_i^e(v_N) = h_{i+1}^e(v_N)$. Since w is not a minimal window, one of its proper subwindows contains an occurrence, say, h, of this episode. That means if h starts at n_s then it must end before n_e. But, since h_i^e is earliest transiting, any occurrence starting at the same event as h_i^e cannot end before h_i^e. Thus, we must have $h(v_1) > h_i^e(v_1)$. This means, by Lemma 1, since h_i^e is earliest transiting, we cannot have $h_i^e(v_N) > h(v_N)$. Since the window of h has to be contained in the window of h_i^e, we thus have $h_i^e(v_N) = h(v_N)$. By definition, h_{i+1}^e will start at the earliest possible position after h_i^e. Since there is an occurrence starting with $h(v_1)$, we must have $h_{i+1}^e(v_1) \leq h(v_1)$. Now, since h_{i+1}^e is earliest transiting, it cannot end after h. Thus, we must have $h_{i+1}^e(v_N) \leq h(v_N)$. Also, h_{i+1}^e cannot end earlier than h_i^e because both are earliest transiting. Thus, we must have $h_i^e(v_N) = h_{i+1}^e(v_N)$. This completes proof of lemma. □

Remark 3 This lemma shows that any ET occurrence h_i^e such that $h_i^e(v_N) < h_{i+1}^e(v_N)$ 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 $h_i^e(v_N) < h_{i+1}^e(v_N)$.

Now, we are ready to prove correctness of the MO algorithm. Consider *Algorithm* 1 operating in the MO(minimal occurrence) mode for tracking occurrences of an N-node episode

α. Since TRANSIT is always true in the MO mode, all automata would be tracking ET occurrences. Since COPY-AUTOMATON is true in MO mode whenever an automaton transits out of start state, we will always have an automaton in the start state. *This, along with the fact that TRANSIT is always true, implies that the ith initialized automaton would be tracking h_i^e, the ith ET occurrence.* Let us denote by $\mathcal{A}_i^\alpha$ the ith initialized automaton. However, since JOIN-AUTOMATON is also always true, not all automata (initialized for this episode) would result in incrementing the frequency; some of them would be removed when one automaton transits into a state already occupied by some other automaton. In view of Lemma 2 and Remark 3, if we show that 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 $h_i^e(v_N) < h_{i+1}^e(v_N)$, then the proof of correctness of MO algorithm is complete.

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 if $h_i^e(v_N) < h_{i+1}^e(v_N)$.*

Proof

$$
\begin{aligned}
&\mathcal{A}_i^\alpha \text{ does not contribute to the frequency} \\
&\quad \Longrightarrow \mathcal{A}_i^\alpha \text{ is removed by a more recently initialized automaton} \\
&\quad \Longrightarrow \exists\ \mathcal{A}_k^\alpha, k > i, \quad \text{which transits into a state already occupied by} \mathcal{A}_i^\alpha. \\
&\quad \Longrightarrow \exists k, j \text{ s.t. } k > i,\ 1 < j \leq N \text{ and } h_i^e(v_j) = h_k^e(v_j). \\
&\quad \Longrightarrow \exists j \text{ s.t. } 1 < j \leq N \text{ and } h_i^e(v_j) = h_{i+1}^e(v_j). \\
&\qquad \text{because, by Remark 2, for } k > i,\ \forall j,\ h_i^e(v_j) \leq h_{i+1}^e(v_j) \leq h_k^e(v_j). \\
&\quad \Longrightarrow h_i^e(v_N) = h_{i+1}^e(v_N)
\end{aligned}
$$

The last step follows because both h_i^e and h_{i+1}^e are ET occurrences, and hence, $h_i^e(v_j) = h_{i+1}^e(v_j)$ implies $h_i^e(v_{j'}) = h_{i+1}^e(v_{j'}),\ \forall j' > j$.

Conversely, we have

$$
\begin{aligned}
&\mathcal{A}_i^\alpha \text{ contributes to the frequency} \\
&\quad \Longrightarrow \forall j,\ 1 < j \leq N,\ h_i^e(v_j) < h_{i+1}^e(v_j) \\
&\quad \Longrightarrow h_i^e(v_N) < h_{i+1}^e(v_N).
\end{aligned}
$$

The first step follows because, if $\mathcal{A}_i^\alpha$ contributes to the frequency then no automaton initialized after it would ever come to the same state occupied by it and since all occurrences tracked are earliest transiting, this must mean $h_i^e(v_j) < h_{i+1}^e(v_j), \forall j$. This completes proof of the lemma. □

Remark 4 As stated in the proof of Lemma 3, if $h_i^e(v_j) = h_{i+1}^e(v_j)$, then $h_i^e(v_{j'}) = h_{i+1}^e(v_{j'}),\ \forall j' > j$. A consequence of this is that $h_i^e(v_N) < h_{i+1}^e(v_N)$ is equivalent to $h_i^e(v_k) < h_{i+1}^e(v_k)\ \forall\, 1 \leq k \leq N$. From Lemma 2, this is another equivalent condition for h_i^e to be minimal.

Remark 5 There already exists an earlier proof of correctness for the MO algorithm [4]. The proof methodology presented here is different from the existing proof by Das et al. [4]. The proof in [4] makes explicit use of one of the data structures used for implementation of the MO algorithm. It specifically uses an array that stores the first transition time of all the existing automata. This is particularly useful when handling expiry constraints. The proof in [4] uses this array and views its update as recursively computing a matrix $S[0 \ldots n, 0 \ldots N]$ using dynamic programming, where the ith row of this matrix corresponds to the state of this array after processing E_i from the data stream. One can show that $S[i, j]$ is the largest value $k \leq i$ such that $E_k \ldots E_i$ contains an occurrence of $\alpha[1] \rightarrow \ldots \alpha[j]$. Whenever

$S[i, N] > S[i-1, N]$, the count is incremented since a new minimal occurrence is recognized. Our proof of correctness does not need this array for the proof arguments. It utilizes the concept of ET occurrences introduced in this paper. Unlike our proof, this proof does not give us any insights about the geometry of the minimal occurrences and ET occurrences in general. Also, it is not clear how one can extend this proof idea to minimal occurrences of unrestricted partial order episodes.

4.2 Other ET occurrence-based algorithms

4.2.1 Proofs of correctness for NO-X and NO-I

The NO-X algorithm can be viewed as a slight modification to the MO algorithm. As in the MO algorithm, we always have an automaton in the start state and all automata make transitions as soon as possible, and when an automaton transits into a state occupied by another, the older one is removed. However, in the NO-X algorithm, the INCREMENT-FREQ variable is true only when we have an occurrence satisfying T_X constraint. Hence, to start with, we look for the first minimal occurrence which satisfies the expiry-time constraint and increment frequency. At this point, (unlike in the MO algorithm) we terminate all automata except the one in the start state since we are trying to construct a non-overlapped set of occurrences. Then, we look for the next earliest minimal occurrence (which will be non-overlapped with the first one) satisfying expiry-time constraint and so on. Since minimal occurrences locally have the least time span, this strategy of searching for minimal occurrences satisfying expiry-time constraint in a non-overlapped fashion is quite intuitive. Let $H_{nX} = \{h_1^{nX}, h_2^{nX} \ldots h_{f'}^{nX}\}$ denote the sequence of occurrences tracked by the NO-X algorithm (for an N-node episode). Then, the following property of H_{nX} is obvious.

Property 1 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 $t_{h_{i-1}^{nX}(v_N)}$ satisfying expiry-time constraint. There is no minimal occurrence satisfying expiry-time constraints which starts after $t_{h_f^{nX}(v_N)}$.

Theorem 1 *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.

$$t_{h_i^{nX}(v_N)} \leq t_{h'_i(v_N)} \qquad \forall i = 1, 2, \ldots, m. \tag{3}$$

This will be shown by induction on i. We first show it for $i = 1$. Suppose $t_{h'_1(v_N)} < t_{h_1^{nX}(v_N)}$. Consider the earliest transiting occurrence h'' starting from $h'_1(v_1)$. This ends on or before $t_{h'_1(v_N)}$ by Lemma 1. Further, the last ET occurrence ending with h'' is a minimal occurrence by Lemma 2. Its window is contained in the window of h'_1, which satisfies the expiry constraints. Hence, we have found a minimal occurrence satisfying expiry constraints ending before h_1^{nX} which contradicts the first statement of Property 1. Hence, $t_{h_1^{nX}(v_N)} \leq t_{h'_1(v_N)}$. Suppose $t_{h_i^{nX}(v_N)} \leq t_{h'_i(v_N)}$ is true for some $i < m$. We show that $t_{h_{i+1}^{nX}(v_N)} \leq t_{h'_{i+1}(v_N)}$. By Property 1, h_{i+1}^{nX} is the first minimal occurrence of α satisfying expiry-time constraints in the data stream beyond $t_{h_i^{nX}(v_N)}$. Suppose $t_{h'_{i+1}(v_N)} < t_{h_{i+1}^{nX}(v_N)}$. Then, very similar to the $i = 1$ case, we can construct 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 $t_{h_i^{nX}(v_N)}$ ending before h_{i+1}^{nX} which contradicts the second statement of Property 1.

Now from eqn. (3), 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 eqn. (3), h'_{f+1} is an occurrence beyond $t_{h_f(v_N)}$. 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 1 that there is no minimal occurrence satisfying T_X beyond $t_{h_f^{nX}(v_N)}$. Hence, $|H_{nX}| \geq |H'|$ for every non-overlapped sequence H' satisfying expiry constraints. Hence, H_{nX} is maximal, and $f = f_{nX}$. □

If we choose T_X equal to the time span of the data stream, the NO-X algorithm reduces to the NO-I algorithm because every occurrence satisfies expiry constraint. Hence, proof of correctness of NO-I algorithm is immediate.

4.2.2 Relation between NO-I and NO algorithms

We now explain the relation between the sets of occurrences tracked by the NO and NO-I algorithms. As proved in [12], the NO algorithm (which uses one automaton per episode) tracks a maximal non-overlapped sequence of occurrences, say, $H_{\text{no}} = \{h_1^{no}, h_2^{no} \ldots h_{f_{\text{no}}}^{no}\}$. Since the NO-I algorithm has no expiry-time constraint, it also tracks a maximal set of non-overlapped occurrences. Among all the ET occurrences that end at $h_i^{no}(v_N)$, let h_i^{in} be the last one (as per the lexicographic ordering). Then, the ith occurrence tracked by the NO-I algorithm would be h_i^{in} as we show now. Since h_1^{no} would be the first ET occurrence, it is clear from our discussion in the previous subsection that the first occurrence tracked by the MO algorithm would be h_1^{in}. As is easy to see, the MO and NO-I algorithms would be identical till the first time an automaton reaches the accepting state. Hence, h_1^{in} would be the first occurrence tracked by the NO-I algorithm. Now the NO-I algorithm would remove all automata except for the one in the start state. Hence, it is as if we start the algorithm with data starting with the first event after $t_{h_1^{no}(v_N)} = t_{h_1^{in}(v_N)}$. Now, by the property of NO algorithm, h_2^{no} would be the first ET occurrence in this data stream and hence h_2^{in} would be the first minimal window here. Hence,it is the second occurrence tracked by NO-I and so on.

The above also shows that each occurrence tracked by the NO-I algorithm is also tracked by the MO algorithm, and hence, we have $f_{\text{no}} \leq f_{\text{mi}}$, stated as one of the relationships in Theorem (2). H_{in} is also a maximal set of non-overlapping minimal windows as discussed in [28].

4.3 Non-interleaved and distinct occurrence-based algorithms

The algorithm NI that counts non-interleaved occurrences is different from all the ones discussed so far because it does not track ET occurrences. Here also we always have an automaton waiting in the start state. However, the transitions are conditional in the sense that the ith-created automaton makes a transition from state $(j-1)$ to j provided the $(i-1)$th-created automaton is past state j after processing the current event. This is because we want the ith automata to track an occurrence non-interleaved with the occurrence tracked by $(i-1)$th automaton. Let $\mathcal{H}_{\text{ni}} = \{h_1^{ni}, h_2^{ni}, \ldots h_{f'}^{ni}\}$ be the sequence of occurrences tracked by NI. From the above discussion, it is clear that it has the following property (while counting occurrences of α).

Property 2 h_1^{ni} is the first or earliest occurrence (of α). For all $i > 1$ and $\forall j = 1, \ldots, N-1$, $h_i^{ni}(v_j)$ is the first occurrence of $\alpha[j]$ at or after $t_{h_{i-1}^{ni}(v_{j+1})}$, and $h_i^{ni}(v_N)$ is the earliest occurrence of $\alpha[N]$ after $t_{h_i^{ni}(v_{N-1})}$. There is no occurrence of α beyond $h_{f'}^{ni}$ which is non-interleaved with it.

The proof that H_{ni} is a maximal non-interleaved sequence is very similar in spirit to that of the NO-X algorithm. As earlier, we can show that given an arbitrary sequence of non-interleaved occurrences $H' = \{h'_1, h'_2 \ldots h'_l\}$, we have $h_i^{ni}(v_k) \leq h'_i(v_k)$, $\forall i, k$ and hence get the correctness proof of NI algorithm. It is easy to verify the correctness of the DO algorithm also along similar lines.

4.3.1 Extending with expiry constraints

To obtain a maximal set of non-overlapped, non-interleaved or distinct set of occurrences, the NO, NI, and DO algorithms just described adopt a greedy strategy which works as follows. We look for the first occurrence of α say h_1. Now, among all occurrences that are greater than h_1 (as per $<_\star$) and non-overlapped with or non-interleaved with/distinct from h_1, we choose the earliest occurrence of α and call it h_2. We continue this process until we cannot find any more occurrences in the event stream. One can show that the resulting set of occurrences will be a maximal set in the non-overlapped, non-interleaved, or distinct sense. One can also show that this greedy strategy extracts a maximal set of occurrences under these counts even when expiry-time constraints are incorporated.

We first look at how to implement this strategy for the non-overlapped count with expiry constraints and compare it with the NO-X algorithm. To track the earliest occurrence of α satisfying expiry constraints, it is enough to search in the space of ET occurrences. So, to start with we need to follow a strategy similar to the HD algorithm to track h_1 (which is the first ET occurrence satisfying expiry constraints). Once h_1 is located, we drop all the existing automata and find the earliest occurrence satisfying expiry in the remaining portion of the data and so on. This strategy suffers from the problem of needing large memory as multiple automata occupying the same state need to be tracked. We point out that the NO-X algorithm which also tracks a maximal set of non-overlapped occurrences with expiry constraints follows a more intelligent strategy. Instead of searching in the space of all ET occurrences, it searches only in the space of minimal windows (essentially a specialized set of ET occurrences). This strategy of NO-X works mainly because of the non-overlapped nature of the frequency measure. For ensuring maximality of the set of occurrences tracked, it is enough if the first occurrence tracked satisfies T_X and ends along with h_1, the first ET occurrence satisfying expiry constraints. Similarly, the next occurrence need not have to be the earliest occurrence non-overlapped with h_1. It is enough if it ends with h_2. By searching in the space of minimal ET occurrences, the ith occurrence tracked by NO-X would be the last ET occurrence ending with h_i, the ith occurrence tracked by the greedy strategy with expiry constraint T_X. And more importantly, the temporary memory needed per episode is bounded above by the size of the episode.

Suppose, we intend to implement the greedy strategy to count non-interleaved or distinct occurrences under expiry-time constraint. Firstly, we have to find the earliest occurrence of α satisfying T_X, for which we need to locate the first ET occurrence satisfying T_X. To search for earliest transiting occurrences, we need to spawn automata very similar to the HD algorithm. Hence, we would have multiple automata in the same state along with the time information of their first state transition, due to which the temporary storage can go unbounded. Also, for each such potential ET occurrence that the HD algorithm strategy would track, we need

to track the earliest occurrence non-interleaved with (distinct from) it and satisfying T_X. For example, consider the following data stream.

$$\mathbb{D}' = (A, 1), (A, 6), (B, 9), (A, 10), (C, 11), (B, 12), (A, 13), (D, 14), (B, 15), (C, 16), (B, 17), (A, 18), (D, 19), (C, 20), (D, 21), (B, 22), (C, 23), (D, 24)$$

In $\mathbb{D}'$, the set {[10 12 16 19], [13 17 20 21], [18 22 23 24]} forms a maximal set of non-interleaved occurrences for $\alpha = (A \rightarrow B \rightarrow C \rightarrow D)$ and $T_X = 10$. Hence, we have $f_{\text{ni}} = 3$. The first two ET occurrences here namely [1 9 11 14] and [3 9 11 14] do not satisfy T_X constraint. h_3^e = [10 12 16 19] is the first ET occurrence in $\mathbb{D}'$ satisfying T_X. Here, till h_1^e and h_2^e end, we would need to keep track of the earliest occurrences beyond and non-interleaved with these. In other words, additional automata have to be spawned to track the prospective second non-interleaved/distinct occurrence satisfying time constraints. For example, h' = [13 17 20 21] is one such occurrence which is non-interleaved with h_3^e. Also, h'' = [18 22 23 24] is the third occurrence of the maximal set in the event sequence considered. Note that it starts before h_3^e ends. And to be able to track this occurrence, one has to track occurrences beyond and non-interleaved with second level of occurrences like h', which at the outset looks very complicated. Hence, with the greedy approach, it looks infeasible to devise NI/DO counting algorithms with expiry-time constraints under the generic scheme presented earlier.

Also, any relaxation on this general greedy strategy does not guarantee maximality for the non-interleaved/distinct occurrence-based counts (even though the NO-X algorithm described before, instead of picking the first ET occurrence satisfying T_X, relaxes this strategy and chooses the last ET occurrence ending with it). h_3^e = [10 12 16 19] is the first ET occurrences satisfying $T_X = 10$, and h_4^e = [13 15 16 19] is an ET occurrences satisfying T_X and ending with h_3^e. We cannot choose the fourth ET occurrence as the first relevant occurrence in the set of maximal non-interleaved occurrences. This is because the occurrence h' = [13 17 20 21] even though non-interleaved with h_3^e is not so with h_4^e. Hence, we would miss h' and count one less if we adopt the strategy of choosing h_4^e as the first relevant occurrence. Note that h' is the first occurrence beyond h_3^e that is non-interleaved with it and satisfies T_X. Hence, we have to look for the first ET occurrence which satisfies T_X (in $\mathbb{D}'$, it is h_3^e), as advocated by the greedy strategy.

5 Relationships between different frequencies

We now point out some quantitative relationships between the different frequency counts for a given episode. It is immediate from the definitions of head and window-based frequency that $f_{wo} \geq f_{\text{h}}$ for any window width. Except the window-based count, all the other frequencies count some subset of the set of all occurrences. For example, suppose there is a data stream which has exactly one occurrence of a particular episode. The window-based count for any window width greater than the span of this occurrence is greater than one. Window-based count, not being an occurrence-based count, is not explicitly considered in the next result.

Theorem 2 (a) *For any serial episode, the various frequencies obey the following in any given event sequence.*

$$f_{all} \geq f_{\text{h}} \geq f_{tot} \geq f_{\text{d}} \tag{4}$$

$$f_{tot} \geq f_{\text{ni}} \geq f_{\text{mi}} \geq f_{\text{no}} \tag{5}$$

where f_{all} denotes the total number of occurrences of an episode, while f_h and f_{tot} denote the corresponding head and total frequencies defined with a window width exceeding the total time span of the event sequence.

(b) *If the serial episode is injective, the inequality $f_d \geq f_{\text{ni}}$ is also true (An episode α is injective if it does not contain any repeated event-types).*

As an illustration, note that these relationships indeed hold for the episode $(A \rightarrow B \rightarrow C \rightarrow D)$ in the example sequence $\mathbb{D}_1$ discussed in Sect. 3. For a large sliding window width (width exceeding the time span of any earliest transiting occurrence), the head frequency f_h is same as the number of earliest transiting occurrences of an episode. This is because a window contains an occurrence starting at its left end if and only if it contains an ET occurrence starting at its left end.

Proof We prove (a) first. Let us start with (4). The first inequality in (4) is obvious. From Eq. (2) in Definition (7), it is obvious that $f_h \geq f_{\text{tot}}$. The second inequality follows directly from Eq. (2) in Definition (7). Given a set of f maximal distinct occurrences of an episode α in a data stream $\mathbb{D}$, one can extract that many earliest transiting occurrences of not only α but also of all its subepisodes in $\mathbb{D}$. Hence, the total frequency of α, which is the minimum of head frequency (number of earliest transiting occurrences) of all its subepisodes, is at least equal to the number of distinct occurrences. Hence, we have $f_{tot} \geq f_d$.

We now prove the relations in (5). To prove the first inequality, consider a maximal set of f_{ni} non-interleaved occurrences $\{h_1, h_2, \ldots h_{f_{\text{ni}}}\}$ of an episode α. Consider one of its subepisodes, say β. For each h_i, extract those event-types constituting β and call it h_i^{β}. Now $t_{h_i^{\beta}(v_1)} = t_{h_i(v_k)}$ for some $k = 1, 2, \ldots N$. If $k = N$, then β must be the 1-node subepisode $\alpha[N]$. This means $t_{h_i^{\beta}(v_i)} = t_{h_i(v_N)} > t_{h_i(v_{N-1})} \geq t_{h_{i-1}(v_N)} = t_{h_{i-1}^{\beta}(v_1)}$. (The first inequality is because we are dealing with serial episodes. The second inequality is because h_i is non-interleaved with h_{i-1}.) Hence, we have $t_{h_i^{\beta}(v_1)} > t_{h_{i-1}^{\beta}(v_1)}$. This means we have f_{ni} distinct occurrences of the 1-node episode $\alpha[N]$, and therefore,head frequency of $\beta = (\alpha[N])$ must be at least f_{ni}.

Now, we consider the case for $k = 1, 2, \ldots (N-1)$. Here, we have $t_{h_i^{\beta}(v_1)} = t_{h_i(v_k)} \geq t_{h_{i-1}(v_{k+1})} > t_{h_{i-1}(v_k)} = t_{h_{i-1}^{\beta}(v_1)}$. The first inequality is because h_i is non-interleaved with h_{i-1}. The second inequality is because we are dealing with serial episodes. Hence, we have $t_{h_i^{\beta}(v_1)} > t_{h_{i-1}^{\beta}(v_1)}$. This means h_i^{β}, for different i start at different time instants. Consider the earliest occurrence of β (which is also ET) starting at each of these time instants. Hence, we have found f_{ni} number of ET occurrences of β too in the data stream. Thus the head frequency of β is at least f_{ni}. Since this is true for any subepisode β, the total frequency of $\alpha(f_{tot})$ will be at least f_{ni}.

To show the second inequality in (5), we first note that if an ET occurrence h_i^e is minimal, then it is non-interleaved with h_{i+1}^e. We can prove this by contradiction. Suppose h_i^e is minimal and h_i^e is not non-interleaved with h_{i+1}^e. Since h_i^e is minimal, we have $h_i^e(v_{j'}) < h_{i+1}^e(v_{j'})$, $\forall j'$, from Remark 4. Since we are dealing with serial episodes, this is same as $t_{h_i^e(v_{j'})} < t_{h_{i+1}^e(v_{j'})}$. If h_i^e is not non-interleaved with h_{i+1}^e, there exists a $j < N$ such that $t_{h_{i+1}^e(v_j)} < t_{h_i^e(v_{j+1})}$. Thus, we must have $t_{h_i^e(v_j)} < t_{h_{i+1}^e(v_j)} < t_{h_i^e(v_{j+1})} < t_{h_{i+1}^e(v_{j+1})}$. But this cannot be because $E_{h_i^e(v_{j+1})}$ is the earliest $\alpha[j+1]$ after $t_{h_i^e(v_j)}$ and if it is also after $t_{h_{i+1}^e(v_j)}$ then the fact that both h_i^e and h_{i+1}^e are ET occurrences should mean $h_i^e(v_{j+1}) = h_{i+1}^e(v_{j+1})$ which is a contradiction. Hence, h_i^e and h_{i+1}^e are non-interleaved. Thus, given the sequence of minimal windows, the earliest transiting occurrences from each of these minimal windows gives us a sequence of (same number of) non-interleaved occurrences. This leads to $f_{\text{mi}} \leq f_{\text{ni}}$.

For the last inequality in (5), we have already seen that the algorithm NO-I tracks a maximal set of non-overlapped occurrences. Also, each occurrence tracked by the NO-I algorithm is also tracked by the MO algorithm,and hence, we have $f_{\text{no}} \leq f_{\text{mi}}$.

We now prove (b) by showing that any set of non-interleaved occurrences of an injective episode is also distinct. Suppose h_1 and h_2 are two occurrences (with $h_1 <_\star h_2$), which are non-interleaved. Then, $t_{h_2(j)} \geq t_{h_1(j+1)}, \quad j = 1, 2, \ldots N-1$. Also, $t_{h_1(j)} < t_{h_1(j+1)}$ for all j, since we are dealing with serial episodes only. Hence, we have $t_{h_2(j)} > t_{h_1(j)}$ for all $j = 1, 2, \ldots (N-1)$. Hence, $h_2(j)$ can be equal to some $h_1(i)$ for $i > j$ only. This cannot happen if the episode under consideration is injective, because then $h_2(j) = h_1(i)$ for an $i > j$ implies that $E_{h_2(j)} = E_{h_1(i)}$ for an $i > j$. This is not possible for injective episodes. Hence, h_1 and h_2 are distinct. To illustrate why the inequality fails for non-injective serial episodes, consider an event stream $\langle (A, 1), (A, 2), (A, 3) \rangle$ and an episode $(A \rightarrow A)$. This stream has $f_{\text{ni}} = 2$ and $f_{\text{d}} = 1$. □

6 Candidate generation

The counting part of the apriori-based discovery was explained in great detail in the previous sections. We now elaborate on the candidate generation step here. As explained, the candidate generation step for either serial or parallel episodes (irrespective of the frequency) exploits some necessary condition for an l-node episode to be frequent in terms of its $(l-1)$-node subepisodes. In this section, we discuss such anti-monotonicity properties of the various frequency counts, which in turn are exploited by their respective candidate generation steps in the Apriori-style level-wise procedure for frequent episode discovery.

It is well known that the *window-based* [17], *non-overlapped* [10] and *total* [8] frequency measures satisfy the anti-monotonicity property that *all subepisodes of a frequent episode are frequent*. One can verify that the same holds for the distinct occurrence-based frequency too. Accordingly, the candidate generation step in discovering episodes based on these frequencies generates an l-node episode as a probable candidate for counting at level l, only if all its subepisodes of size $(l-1)$ are already found to be frequent. On the other hand, the head, minimal, and non-interleaved frequencies do not satisfy the property that all subepisodes are as frequent as the parent episode. In spite of this, they satisfy a more restricted anti-monotonicity property which is exploited for discovery based on these counts.

As pointed out in [8], for an episode α, in general, only the subepisodes involving $\alpha[1]$ are as frequent as α under the head frequency. In a level-wise apriori-based episode discovery, the candidate generation for the head frequency count would exploit the condition that if an N-node episode is frequent, then all $(N-1)$-node subepisodes that include $\alpha[1]$ have to be frequent. The head frequency definition has some limitations in the sense that the frequency of the $(N-1)$-node suffix[3] subepisode an be arbitrarily low. Consider the event stream with $100A$'s followed by a B and C. Suppose all occurrences of $A \rightarrow B \rightarrow C$ satisfy the expiry constraint T_X. Even though there are 100 occurrences of $A \rightarrow B \rightarrow C$, there is only one occurrence of $B \rightarrow C$. This can be a problem when one desires that the frequent episodes capture repetitive causative influences.

Like the head frequency, the minimal occurrences (windows) and the non-interleaved occurrences also do not satisfy the property that all subepisodes are at least as frequent as the corresponding episode. However, the $(N-1)$-node prefix and suffix subepisodes are at

[3] Given an N-node episode $\alpha[1] \rightarrow \alpha[2] \rightarrow \cdots \rightarrow \alpha[N]$, its K-node *prefix subepisode* is $\alpha[1] \rightarrow \alpha[2] \rightarrow \cdots \rightarrow \alpha[K]$ and its $(N-K)$-node suffix subepisode is $\alpha[K+1] \rightarrow \alpha[K+2] \rightarrow \cdots \rightarrow \alpha[N]$ for $K = 1, 2, \ldots, (N-1)$.

least as frequent as the episode as we show below. This aspect of the minimal occurrence count has not been reported in literature to the best of our knowledge including [17] which introduces the minimal window measure. This aspect of the non-interleaved count has been reported in [9] without a proof. For an example, consider a data stream where successive events are given by $ABACBDCD$. Even though there are two minimal windows (and two non-interleaved occurrences) of $A \rightarrow B \rightarrow C \rightarrow D$, there is only one minimal window (and one non-interleaved occurrence) of each of the non-prefix and non-suffix subepisodes $A \rightarrow B \rightarrow D$ and $A \rightarrow C \rightarrow D$. We now formally prove the anti-monotonicity property for minimal and non-interleaved occurrence-based frequencies.

Theorem 3 *If an N-node serial episode α has a frequency f in the minimal window or the non-interleaved sense, then its $(N-1)$-node prefix subepisode (α_p) and suffix subepisode (α_s) have a frequency of at least f.*

Proof Consider a minimal window of the episode α namely $w = [t_s, t_e]$. Consider the ET occurrence h_p of the $(N-1)$-node prefix subepisode starting from t_s. Its window $w' = [t_s, t'_e]$ would be such that $t'_e < t_e$ because α is a serial episode. It is clear that w' forms a minimal window for the prefix subepisode α_p. This is because any proper subwindow of w' starting at t_s cannot contain an occurrence of α_p as there can be no occurrence of α_p ending before h_p (as per Lemma 1). A proper subwindow of w' containing an occurrence of α_p starting after t_s would contradict the minimality of w itself. Hence, w' is a minimal window of α_p starting at t_s. Hence, for each minimal window of α, we can find a minimal window of α_p which starts with w and ends strictly before it. Since each minimal window of α has a unique starting point, each of the minimal windows of its $(N-1)$-node prefix subepisode (α_p) considered above has different starting points. Therefore, these minimal windows of α_p are distinct. We hence conclude that α_p has a frequency of at least f.

A similar proof works for the suffix subepisode by considering the window of the last occurrence h_s of the suffix subepisode ending at t_e.

Consider $\mathcal{H}_{\text{ni}} = \{h_1, h_2, \ldots h_f\}$, a maximal set of non-interleaved occurrences of α. From each occurrence h_k of α, we choose a suboccurrence $h^p_k = [h_k(v_1)\ h_k(v_2) \ldots h_k(v_{N-1})]$, which is an occurrence of α_p. One can verify from the non-interleaved definition that this new set of occurrences $h^p_1, h^p_2, \ldots h^p_f$ is non-interleaved. Hence, the frequency of α_p is at least f. A similar thing can be shown for α_s by considering $h^s_k = [h_k(v_2)\ h_k(v_3) \ldots h_k(v_N)]$. □

From the above theorem, we can further conclude that if an N-node serial episode has a frequency f in the minimal window or non-interleaved sense, then all its contiguous subepisodes[4] will have a frequency of at least f. However with respect to the non-contiguous episodes, the situation here is not as bad as that for head frequency because all non-contiguous (in fact all) subepisodes will have at least as many distinct occurrences as the number of minimal or non-interleaved occurrences of the episode, at least in case of injective episodes. This is because, in case of injective episodes, the number of distinct occurrences is always greater than the non-interleaved count, which in turn is greater than the minimal windows count (Theorem 2). Hence, given that there are f non-interleaved or minimal occurrences of an injective episode α, there are at least f distinct occurrences of α too. Since the distinct occurrence-based frequency satisfies the original anti-monotonicity property (of all subepisodes being at least as frequent as the episode), all subepisodes of α

[4] A K-node subepisode α_1, of α is a contiguous subepisode if $\alpha_1 = (\alpha[i] \rightarrow \alpha[i+1] \rightarrow \cdots \alpha[i+(K-1)]$ for some $i = 1, 2, \ldots (N-K+1)$

too will have at least f distinct occurrences. If the episode is non-injective, then we cannot say anything about the frequencies of its subepisodes in the distinct occurrences sense. For the case of general episodes, we can show that the minimal/non-interleaved frequencies of at least all the $(N-1)$-node subepisodes of an N-node episode having minimal/non-interleaved frequency f cannot be less than $f/2$. We have just seen that all the prefix and suffix subepisodes will be at least as frequent as the episode. We show in Sect. A that the frequency of those $(N-1)$-node subepisodes which are neither prefix nor the suffix subepisodes is lower bounded by $f/2$. We will also see that this bound is in fact tight for injective serial episodes.

Accordingly, for frequent episode discovery under minimal and non-interleaved occurrence-based frequency, an $(\ell+1)$-node episode is generated as a potential candidate if and only if its ℓ-node suffix and prefix subepisodes are frequent. The candidate generation step here combines two ℓ-node episodes α_1 and α_2 from the set of frequent serial episodes at level ℓ if the $(\ell-1)$-node suffix subepisode of α_1 matches the $(\ell-1)$-node prefix subepisode of α_2. The combination results in an episode $(\alpha_1[1] \rightarrow \alpha_1[2] \rightarrow \cdots \rightarrow \alpha_1[\ell] \rightarrow \alpha_2[\ell])$. As an example, if $(A \rightarrow C \rightarrow D)$ and $(C \rightarrow D \rightarrow B)$ are frequent 3-node episodes, the candidate generation step would combine these two episode to generate $(A \rightarrow C \rightarrow D \rightarrow B)$ as a potential candidate at level 4. Candidates at a given level when stored in lexicographic order results in episodes sharing the same $(N-1)$-node prefix subepisode naturally coming together as a block [17]. For every episode α, we extract its $(N-1)$ suffix, and search for a block of episodes whose $(N-1)$ prefix matches this suffix and form a candidate for each episode in this block. This kind of candidate generation has already been reported in the literature in [21,26] and [24] in the context of mining under gap/inter-event time constraints.

7 Discussion and conclusions

The framework of frequent episodes in event streams is a very useful data mining technique for unearthing temporal dependencies from data streams in many applications. The framework is about a decade old, and many different frequency measures and associated algorithms have been proposed over the last ten years. In this paper, we have presented a coherent view of all apriori-based discovery algorithms for frequent serial episode discovery. In particular, we presented a generic automata-based algorithm for obtaining frequencies of a set of candidate episodes. This method unifies all the known algorithms in the sense that we can particularize our algorithm (by setting values for a set of variables) for counting frequent episodes under any of the frequency measures proposed in literature.

As we showed here, this unified view of counting gives useful insights into the kind of occurrences counted under different frequency definitions and thus also allows us to prove relations between frequencies of an episode under different frequency definitions. Our view also allows us to get correctness proofs for all algorithms. We introduced the notion of earliest transiting occurrences and, using this concept, are able to get simple proofs of correctness for most algorithms. This has also allowed us to understand the kind of anti-monotonicity properties satisfied by different frequency measures, which is needed for the design of the candidate generation step.

While the main contribution of this paper is this unified view of all frequency counting algorithms, some of the specific results presented here are also new. The relationships between different frequencies of an episode (cf. Theorem 2) are proved here for the first time. The distinct occurrence-based algorithm (automata-based) described is also novel. The specific proof of correctness presented here for minimal occurrences is also novel. Also, the

correctness proofs for non-overlapped occurrence-based frequency counting under expiry-time constraint has been provided here for the first time.

In this paper, we have considered only the case of serial episodes. This is because, at present, there are no algorithms for discovering general partial orders under the various frequency definitions. However, all counting algorithms explained here for serial episodes can be extended to episodes with a general partial order structure. We can come up with a similar finite state automata (FSA), which track the earliest transiting occurrences of an episode with a general partial order structure [1]. For example, consider a partial order episode $(A\ B) \rightarrow C$ which represents A and B occurring in any order followed by a C. In order to track an occurrence of such a pattern, the initial state has to wait for either of A and B. On seeing an A, it goes to state-1 where it waits only for a B; on the other hand, on seeing a B first it moves to state-2 where it waits only for an A. Then on seeing a B in state-1 or seeing a A in state-2 it moves into state-3 where it waits for a C and so on. Thus, in each state in such a FSA, in general, we wait for any of a set of event-types (instead of a single event for serial episodes) and a given state will now branch out into different states on different event-types. With such a FSA technique, it is possible to generalize the method presented here so that we have algorithms for counting frequencies of general partial order episodes under different frequencies. The proofs presented here for serial episodes can also be extended for general partial order episodes. While it seems possible, as explained above, to generalize the counting schemes to handle general partial order episodes, it is not obvious what would be an appropriate candidate generation scheme for general partial order episodes under different frequency definitions. This is an important direction for future work.

In this paper, we have considered only expiry-time constraint which prescribes an upper bound on the span of the occurrence. It would be interesting to see under what other time constraints (e.g., gap constraints), design of counting algorithms under this generic framework is possible. Also, some unexplored choice of the boolean conditions in the proposed generic algorithm may give rise to algorithms for new useful frequency measures. This is also a useful direction of research to explore.

We have only considered the apriori-based discovery algorithms in this paper, which essentially do a breadth-first search in the space of all episodes. As discussed in Sect. 2.1, discovery algorithms based on a depth-first search strategy [7,18] of the space of all episodes have also been proposed. These algorithms explicitly carry the occurrence lists of the episodes being counted. The occurrence list of a given N-node episode is formed recursively by a temporal join of a list of occurrences of its $(N-1)$-node prefix subepisode and an occurrences list of its 1-node suffix subepisode. Such depth-first algorithms currently exist under the head and minimal occurrences (with gap constraints) based frequencies. Currently, there are no algorithms employing such strategies under the remaining frequencies discussed here. Under the non-overlapped or non-interleaved frequencies, which count a maximal set of occurrences by definition, the temporal join idea is not straightforward. This is because any temporal join of the occurrence lists of its suitable subepisodes yields an occurrence list whose occurrences need not form a maximal set in general. Designing such algorithms for the remaining frequencies and viewing all of them in a unified light is an interesting future direction to explore.

A Appendix

In order to show the bound discussed at the end of Sect. 6, we first make an observation regarding the minimal windows of a $(N-1)$-node *non-(prefix/suffix)* subepisode, which is

neither a prefix nor a suffix subepisode. Given an episode α and its minimal window w, no subwindow w' of w with both its endpoints strictly within w can contain an occurrence of either its prefix or suffix $(N-1)$-node subepisodes. This is because it would immediately contradict the minimality of w. Interestingly, a similar thing holds good for the non-(prefix/suffix) subepisodes too which is shown below.

Lemma 4 *Given an episode α and one of its minimal windows $w = [t_s, t_e]$, there does not exist a subwindow $w' = [t'_s, t'_e]$ of w with both its endpoints strictly within w that contains an occurrence of some non-(prefix/suffix) $(N-1)$-node subepisode of α.*

Proof Suppose there exists such a subwindow w' containing an occurrence of some non-(prefix/suffix) subepisode, say α_n. Let α_n be obtained by dropping the ith node of α. Note that i can be anywhere from 2 to $(N-1)$. Consider the last occurrence of α in w, say h. Let the earliest occurrence of α_n in w', say h_n. Since α_n is obtained by dropping the ith node of α, we have $\forall j < i$, $E_{h_n(v_j)} = \alpha[j]$ and $\forall j \geq i$, $E_{h_n(v_j)} = \alpha[j+1]$. We now make an observation that $t_{h_n(v_j)} \geq t_{h(v_{j+1})}$ for all $j = 1, 2 \ldots (i-1)$. We show this by contradiction. Suppose for some j_0, $t_{h_n(v_{j_0})} < t_{h(v_{j_0+1})}$, then $[t'_s, t_e]$ contains an occurrence of α, namely $[h_n(v_1) \ \ldots h_n(v_{j_0})\ h(v_{j_0+1}) \ldots\ h(v_N)]$, which contradicts the minimality of w. Hence, we now have $t_{h_n(v_i)} > t_{h_n(v_{i-1})} \geq t_{h(v_i)}$. This would mean that the window $[t_s, t'_e]$ contains an occurrence $[h(v_1) \ldots h(v_i) h_n(v_i) \ldots h_n(v_{N-1})]$ of α which would contradict the minimality of w. □

We now prove the lower bound on the frequency f of such $(N-1)$-node subepisodes under both minimal window and non-interleaved occurrence-based counts when f is even. On exactly similar lines as the proof of Theorem 4, we can show that this lower bound translates to $\lceil f/2 \rceil$ when f is odd. Later, we will illustrate with examples how this lower bound can actually be achieved for injective episodes.

Theorem 4 *In any event stream $\mathbb{D} = \langle (\mathcal{E}_1, \bar{t}_1), (\mathcal{E}_2, \bar{t}_2), \ldots (\mathcal{E}_m, \bar{t}_m) \rangle$, (where $\bar{t}_i < \bar{t}_{i+1}$ and $\mathcal{E}_i$ are a set of event-types) having an episode α with frequency $(f = 2k)$ in the minimal window (non-interleaved) sense, every non-(prefix/suffix) $(N-1)$-node subepisode α_n will have a frequency of at least $f/2 = k$ in the minimal window (or non-interleaved) sense.*

Proof For the minimal window case, we will prove this by induction on k. Existence of two minimal windows certainly guarantees at least one occurrence of each of its subepisodes. Hence, the statement is obviously true for $k = 1$. Suppose it is true for some $k = r$, we need to show that it is also true for a data sequence s containing $2(r+1)$ minimal windows of α. Let $w_{2r} = [t_s^{2r}, t_e^{2r}]$ denote the $2r$th minimal window of α in $\mathbb{D}$. Let us consider the portion of $\mathbb{D}$ from $\bar{t}_1$ till t_e^{2r}, and denote it as $\mathbb{D}_p$. $\mathbb{D}_p$ is an event stream with $2r$ minimal windows of α and hence satisfies the inductive hypothesis for $k = r$. Hence, each non-(prefix/suffix) episode has a minimal window frequency of at least r in $\mathbb{D}_p$. If we can show that extending $\mathbb{D}_p$ to $\mathbb{D}$ results in at least one new minimal window of α_n in $\mathbb{D}$, we are done. We shall show this by contradiction.

Suppose by extending $\mathbb{D}_p$ to $\mathbb{D}$, there is no new minimal window of α_n in $\mathbb{D}$. Let us focus on the last three minimal windows of α namely w_{2r}, w_{2r+1} and w_{2r+2}. Any minimal window of α contains an occurrence of α_n. This occurrence window of α_n must contain some subwindow which is a minimal window of α_n. Hence, every minimal window of α contains a minimal window of α_n. In particular, w_{2r+1} and w_{2r+2} contain minimal windows of α_n. If no new minimal window of α_n would come up when data stream is extended from $\mathbb{D}_p$ to $\mathbb{D}$, it should be the case that the minimal windows of α_n contained in w_{2r+1} and w_{2r+2} must

coincide and occur in $\mathbb{D}_p$. Let us call this minimal window $w_n = [t'_s, t'_e]$. Since $\mathbb{D}_p$ ends at t_e^{2r}, we have $t'_e \leq t_e^{2r} < t_e^{2r+1}$. Also, since w_n is a part of w_{2r+2}, we have $t'_s \geq t_s^{2r+2} > t_s^{2r+1}$. Hence, we have both the end points of w_n strictly within w_{2r+1}. This contradicts Lemma 4 applied on w_{2r+1}. This ultimately proves that there must be at least one minimal window of α_n in $\mathbb{D}$ which lies partly or fully outside $\mathbb{D}_p$.

Considering the non-interleaved case, let $H_{\text{ni}} = \{h_1, h_2, \ldots h_{2f}\}$ be a maximal set of non-interleaved occurrences of α, where $h_i <_\star h_{i+1}$. Suppose α_n is obtained by dropping the ith element of α where i ranges from 2 to $N-1$. From each h_k where k is odd, let us choose a suboccurrence of α_n by dropping $h_k(v_i)$. We will show that this sequence of occurrences $H'_{\text{ni}} = \{h'_1, h'_2 \ldots h'_f\}$ forms a set of non-interleaved occurrences. We start with showing that h'_1 and h'_2 are non-interleaved i.e., $t_{h'_2(v_j)} \geq t_{h'_1(v_{j+1})}$ for $j = 1, 2, \ldots (N-2)$. We will show this separately for different ranges of j. First, for $j = 1, \ldots (i-2)$, we have $h'_2(v_j) = h_3(v_j) \geq h_2(v_{j+1}) \geq h_1(v_{j+2}) > h_1(v_{j+1}) = h'_1(v_{j+1})$ which finally implies $h'_2(v_j) > h'_1(v_{j+1})$. Here, the first equality follows from the definition of h'_2. The next two inequalities follow because h_i is a non-interleaved sequence. The last equality is again by the definition of h'_1. Next, for $j = (i-1)$, we have $h'_2(v_{i-1}) = h_3(v_{i-1}) \geq h_2(v_i) \geq h_1(v_{i+1}) = h'_1(v_i)$. As before, the first equality follows from the definition of h'_2. The next two inequalities follow because h_i is a non-interleaved sequence. The last equality is again by the definition of h'_1. Hence, we have $h'_2(v_{i-1}) \geq h'_1(v_i)$. Next, we show the non-interleaved property for $j \geq i$. Again, for reasons similar to the ones outlined above, we have $h'_2(v_{i+k}) = h_3(v_{i+k+1}) \geq h_2(v_{i+k+2}) \geq h_1(v_{i+k+3}) > h_1(v_{i+k+2}) = h'_1(v_{i+k+1}) \forall 0 \leq k \leq (N-2)-i$. This completes the proof for h'_1 and h'_2 being non-interleaved. On exactly similar lines we can show that h'_i is non-interleaved with h'_{i+1} for every i. □

We now show the tightness of the bound through some examples. Consider the episode $\alpha = (A \rightarrow B \rightarrow C \rightarrow D)$ and the data stream where successive events are given by $q = ABACBDCD$. Recall that this was the same data stream we used to illustrate that the subepisodes that are neither prefix nor suffix can have a lesser frequency than the given episode. This data stream has 2 minimal windows (or non-interleaved occurrences) of α and only one minimal window (non-interleaved occurrence) of $(A \rightarrow C \rightarrow D)$ and $(A \rightarrow B \rightarrow D)$. This illustrates the tightness of the bound for $f = 2$ and $k = 1$. We can show the tightness of the bound for any k by considering the data stream q^k, which is the concatenation of q to itself k times. This string contains $2k$ minimal windows (non-interleaved occurrences) of α and only k minimal windows (non-interleaved occurrences) of each of its non-(prefix/suffix) subepisodes. This example can be generalized to any injective episode α by considering the data stream $q = \alpha[1]\alpha[2]\alpha[1]\alpha[3]\alpha[2] \ldots \alpha[N]\alpha[N-1]\alpha[N]$ and its k self-concatenations.

References

1. Achar A, Laxman S, Raajay V, Sastry PS (2009) Discovering general partial orders from event streams. Technical Report arXiv: 0902.1227v2 [cs.AI] http://arxiv.org, Dec 2009
2. Bouqata Bouchra, Caraothers Christopher D, Szymanski Boleslaw K, Zaki Mohammed J. (2006) Vogue: a novel variable order-gap state machine for modeling sequences. In: Proceedings of European conference principles and practice of knowledge discovery in databases (PKDD'06), pp 42–54, Sep 2006
3. Casas-Garriga G (2003) Discovering unbounded episodes in sequential data. In: Proceedings of European conference principles and practice of knowledge discovery in databases (PKDD'03), pp 83–94, Sep 2003
4. Das G, Fleischer R, Gąsieniec L, Gunopulos D, Kärkkäinen J (1997) Episode matching. In: Proceedings of annual symposium combinatorial pattern matching(CPM'97), pp 12–27, June–July 1997
5. Gwadera R, Atallah MJ, Szpankowski W (2003) Reliable detection of episodes in event sequences. In: Proceedings of IEEE international conference data mining (ICDM'03), pp 67–74, Nov 2003

6. Gwadera R, Atallah MJ, Szpankowski W (2005) Reliable detection of episodes in event sequences. Knowl Inf Syst 7(4):415–437
7. Huang K-Y, Chang C-H (2008) Efficient mining of frequent episodes from complex sequences. Inf Syst 33(1):96–114
8. 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 IEEE conference cybernetics and intelligent systems, pp 213–217, Dec 2004
9. Laxman S (2006) Discovering frequent episodes: fast algorithms, connections with HMMs and generalizations. PhD thesis, Bangalore, India, Sep 2006
10. Laxman S, Sastry PS, Unnikrishnan KP (2005) Discovering frequent episodes and learning hidden Markov models: a formal connection. IEEE Trans Knowl Data Eng 17(11):1505–1517
11. Laxman S, Sastry PS, Unnikrishnan KP (2007) Discovering frequent generalized episodes when events persist for different durations. IEEE Trans Knowl Data Eng 19(9):1188–1201
12. Laxman S, Sastry PS, Unnikrishnan KP (2007) A fast algorithm for finding frequent episodes in event streams. In: Proceedings of ACM SIGKDD international conference knowledge discovery and data mining (KDD'07), pp 410–419, Aug 2007
13. Laxman S, Tankasali V, White RW (2008) Stream prediction using a generative model based on frequent episodes in event sequences. In: Proceedings of ACM SIGKDD international conference knowledge discovery and data mining (KDD'08), pp 453–461, Jul 2008
14. Luo J, Bridges SM (2000) Mining fuzzy association rules and fuzzy frequent episodes for intrusion detection. Int J Intell Syst 15(8):687–703
15. Karypis G, Joshi MV, Kumar V (1999) Universal formulation of sequential patterns. Technical Report TR 99-021, Department of Computer Science, University of Minnesota, Minneapolis
16. Mannila H, Toivonen H (1996) Discovering generalized episodes using minimal occurrences. In: Proceedings of international conference knowledge discovery and data mining (KDD'96), pp 146–151, Aug 1996
17. Mannila H, Toivonen H, Verkamo AI (1997) Discovery of frequent episodes in event sequences. Data Mining Knowl Discov 1(3):259–289
18. Meger N, Rigotti C (2004) Constraint-based mining of episode rules and optimal window sizes. In: Proceedings of European conference principles and practice of knowledge discovery in databases (PKDD'04), Sep 2004
19. Morchen F (2007) Unsupervised pattern mining from symbolic temporal data. In: SIGKDD explorations, vol 9, pp 41–55, Jun 2007
20. Nag A, A Wai-Chee Fu (2003) Mining freqeunt episodes for relating financial events and stock trends. In: Proceedings of Pacific-Asia conference knowledge discovery and data mining, (PAKDD 2003), pp 27–39
21. Orlando S, Perego R, Silvestri C (2004) A new algorithm for gap constrained sequence mining. In: Proceedings of ACM symposium applied computing, pp 540–547, Mar 2004
22. Papapetrou P, Kollios G, Sclaroff S, Gunopulos D (2009) Mining frequent arrangements of temporal intervals. Knowl Inf Syst 21:133–171
23. Patnaik D, Laxman S, Ramakrishnan N (2010) Discovering excitatory relationships using dynamic bayesian networks. Knowl Inf Syst, pp 1–31. doi:10.1007/s10115-010-0344-6
24. Patnaik D, Sastry PS, Unnikrishnan KP (2008) Inferring neuronal network connectivity from spike data: a temporal data mining approach. Sci Programm 16(1):49–77
25. Sastry PS, Unnikrishnan KP (2010) Conditinal probability based significance tests for sequential patterns in multi-neuronal spike trains. Neural Comput 22(4):1025–1059
26. Srikanth R, Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In: Proceedings of international conference extending database technology (EDBT), Mar 1996
27. Tatti N (2008) Maximum entropy based significance of itemsets. Knowl Inf Syst 17(1):57–77
28. Tatti N (2009) Significance of episodes based on minimal windows. In: Proceedings of IEEE international conference data mining (ICDM'09), Dec 2009
29. Unnikrishnan KP, Shadid BQ, Sastry PS, Laxman S (2009) Root cause diagnostics using temporal datamining. U.S.Patent no. 7509234, 24 Mar 2009
30. Wang M-F, Wu Y-C, Tsai M-F (2008) Exploiting frequent episodes in weighted suffix tree to improve intrusion detection system. In: Proceedings of international conference advanced information networking and applications(AINA'08), pp 1246–1252, Mar 2008

Author Biographies

Avinash Achar obtained his B.E. from Electrical Engg. in NIT, Suratkal. He obtained his M.Sc.(Engg.) from Electrical Communications Engg department, IISc, Bangalore and is currently a Ph.D. student in the Electrical Engg. Department of IISc. His research interests are broadly in the areas of Machine Learning, Signal Processing and Data Mining.

Srivatsan Laxman is a Researcher in the CSAM Group (Cryptography, Security and Applied Mathematics) at Microsoft Research India, Bangalore. He works broadly in the areas of data mining and machine learning. Srivatsan received his Ph.D. in Electrical Engineering from Indian Institute of Science, Bangalore.

P. S. Sastry received B.Sc.(Hons.) in Physics from IIT, Kharagpur and B.E. in Electrical Communications Engineering and Ph.D. from Department of Electrical Engineering, from IISc, Bangalore. Since 1986, he has been a faculty member in the department of Electrical Engineering, Indian Institute of Science, Bangalore, where currently he is a professor. He has held visiting positions at University of Massachusetts, Amherst, University of Michigan, Ann Arbor and General Motors Research Labs, Warren, USA. He is an Associate Editor of IEEE Transactions on Systems, Man and Cybernetics, Part-B and IEEE Transactions on Automation Science and Engineering. He is a receipient of Sir C. V. Raman Young Scientist award, from Govt of Karnataka, Hari Om Ashram Dr. Vikram Sarabhai research award from PRL, Ahmedabad and Most Valued Collegue award from General Motors Corporation, USA. He is a Fellow of Indian National Academy of Engineering. His research interests include Machine Learning, Pattern Recognition, Data Mining and Computational Neuroscience.