Discovering Frequent Generalized Episodes When Events Persist for Different Durations

Srivatsan Laxman, P.S. Sastry, *Senior Member*, *IEEE*, and K.P. Unnikrishnan

Abstract—This paper is concerned with the framework of frequent episode discovery in event sequences. A new temporal pattern, called the *generalized episode*, is defined, which extends this framework by incorporating event duration constraints explicitly into the pattern's definition. This new formalism facilitates extension of the technique of episodes discovery to applications where data appears as a sequence of events that persist for different durations (rather than being instantaneous). We present efficient algorithms for episode discovery in this new framework. Through extensive simulations, we show the expressive power of the new formalism. We also show how the duration constraint possibilities can be used as a design choice to properly focus the episode discovery process. Finally, we briefly discuss some interesting results obtained on data from manufacturing plants of General Motors.

Index Terms—Data mining, sequential data, frequent episodes, efficient algorithms, event durations.

1 Introduction

Pattern discovery in event sequences is an important temporal data mining task that has received much attention in recent times [1], [2], [3], [4]. The framework of frequent episodes discovery in event sequences, introduced by Mannila, et al. [2], is a popular pattern discovery method that has been found useful in many applications [5], [6], [7], [8]. The data in this framework is a sequence of events (or symbols), where each event is associated with a corresponding time of occurrence, with an implicit assumption that events are essentially *instantaneous*. In some applications, this assumption is viable, for example, alarm sequences in telecommunication networks, transaction sequences in grocery stores, and so forth. However, in many situations, different events tend to persist for different periods of time. In such cases, the data is a sequence of events with both *start* and *end* times. For example, the time-evolving status of an assembly line could be logged as a discrete sequence of states, along with the start and end times for each state. Similarly, Web navigation data could carry time stamps of when a user opened and closed each Web page during a session of Internet browsing. In all such cases, the durations of events carry important information from a data mining perspective.

This paper describes a pattern discovery framework that utilizes event duration information in the temporal data mining process. We extend the frequent episode discovery framework in [2] by defining what we call as *generalized episodes*. The generalized episode incorporates event duration information explicitly into the episode structure. This improves the descriptive power of episodes and helps unearth event duration-dependent temporal correlations in the data.

Stated informally, an episode in the framework in [2] is a partially ordered collection of symbols or event types. An episode is said to occur in a slice of data if the event types associated with the episode appear in the data with the same ordering as prescribed by the episode. The data mining task here is to discover all episodes whose frequency (which is some suitable measure of how often an episode occurs in the data stream) is above some user-specified threshold. Even in the extended framework presented in this paper, our generalized episode consists of a partially ordered collection of event types. In addition, now, each node of the episode also contains a set of time intervals. A generalized episode is said to occur in a data sequence if all the event types of the episode appear as events in the data with the same ordering as specified by the episode, and in addition, the durations of these events belong to the time intervals associated with the corresponding nodes in the episode. Thus, now, the discovery of frequent (generalized) episodes includes discovering the partial order on the event types, as well as discovering relevant time durations.

In general, given a sequence of events, frequent episode discovery helps in unearthing causative chains of events. For example, consider the problem of analyzing an event sequence in the form of fault report logs from an assembly line in a manufacturing plant (we discuss such an application of our algorithms in Section 5.6). Here, the event types would be various fault codes, and a serial episode (which is a set of totally ordered event types) can capture a specific causative chain. Discovery of frequent episodes can then help in the diagnosis of root causes for persistent fault situations. In such an application, the time durations for which different faults persist carry important information in the context of unearthing useful correlations among different faults. This is true of many other application scenarios as well. In all such cases, we need techniques that can naturally

- *S. Laxman is with Microsoft Research Labs, "Scientia," 196/36, 2nd Main, Sadashivnagar, Bangalore-560080 India. E-mail: slaxman@microsoft.com.*
- *P.S. Sastry is with the Department of Electrical Engineering, Indian Institute of Science, Bangalore-560012, India. E-mail: sastry@ee.iisc.ernet.in.*
- *K.P. Unnikrishnan is with Manufacturing Systems Research, General Motors R&D Center, 30500 Mound Road, Warren, MI 48090-9055. E-mail: k.unnikrishnan@gm.com.*

Manuscript received 10 Sept. 2005; revised 22 Apr. 2006; accepted 30 Mar. 2007; published online 1 May 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0358-0905.
Digital Object Identifier no. 10.1109/TKDE.2007.1055.

handle time durations for events. Current methods for frequent episode discovery (for example, [2] and [4]) treat the events in the data stream as instantaneous and, hence, if we want to use such techniques, then we have to ignore time durations. It may appear at first that we can still use such techniques by filtering out events of different durations of interest and then running frequent episode discovery on each such (filtered) substream. However, such an approach is not really satisfactory. First, we have to make ad hoc decisions on which time durations to consider in each substream, whereas what is more attractive is to automatically discover the relevant durations from some resolution information on time durations provided by the user. Second, and more importantly, this approach of ignoring time durations (after possibly filtering out events with durations thought to be relevant by the user) can miss out on correlations that are duration dependent. For example, a short duration event of type A followed by a long duration event of type B may lead to an event of type C, whereas a long duration event of type A followed by a short duration event of type B may lead to events of another type D. To handle all such cases automatically and in a natural manner, we need methods that retain events of all durations in the data stream and make the discovery process sensitive to event durations. This is the main motivation for the extended framework presented in this paper.

In this paper, we present the formalism needed for defining generalized episodes and some efficient algorithms for discovering all generalized episodes whose frequency is above some given threshold. The frequency of an episode was defined in [2] as the number of fixed-width sliding windows on the data in which the episode occurs. Episode frequency can be defined in other ways as well [9]. All such frequency measures can be adopted for generalized episodes also.

The rest of the paper is organized as follows: In Section 2, we define generalized episodes and develop a formalism to address the problem of discovering frequent generalized episodes. Sections 3 and 4 present the various algorithms. In Section 5, we discuss some results obtained with our algorithms. Section 6 contains discussions and conclusions.

2 Generalized Episodes in Event Sequences

This section presents our new formalism for incorporating time durations for events and defines *generalized episodes*.

2.1 Event Sequence

Each event in the data sequence consists of an event type, a start time, and an end time. The input *event sequence* is denoted as $s = \langle (E_1, t_1, \tau_1), (E_2, t_2, \tau_2), \ldots \rangle$, where each E_i takes values from a finite set $\mathcal{E}$ of the event types, t_i and τ_i denote, respectively, the start and end times of the ith event, and we have $\tau_i \geq t_i$. The events in s are sorted according to event start times, that is, $t_i > t_{i-1}$, $i = 2, 3, \ldots$.[1] The *duration* or *dwelling time* of the ith event, denoted Δ_i, is given by

$$\Delta_i = \tau_i - t_i, \ i = 1, 2, \ldots. \tag{1}$$

1. In principle, events in s may be ordered according to start times or end times. The definitions in this section assume that s is sorted based on the start times. These can be adapted with obvious modifications when s is sorted according to the event end times.

The following is an example event sequence with 10 events:

$$\begin{aligned} s \ = \langle & (A, 1, 4), (C, 5, 8), (A, 10, 14), (B, 16, 22), \\ & (A, 23, 32), (B, 33, 41), (C, 40, 44), \\ & (D, 48, 57), (A, 59, 68), (D, 65, 69) \rangle. \end{aligned} \tag{2}$$

The main difference here compared to the formalism in [2] is that events are not assumed to be instantaneous. The event $(A, 1, 4)$, for instance, has a dwelling time of 3 units. If the event sequence has $t_i = \tau_i \ \forall i$, then we get back the earlier formalism of a sequence of instantaneous events. By incorporating the event start and end times in the event sequences, we expect to be able to define and discover episodes with greater expressive capability in them. We note that our definition, in general, allows for successive events to have overlapping durations. That is, it is possible that $t_{i-1} < t_i \leq \tau_{i-1}$ for some i. An important special case is one where $t_{i+1} > \tau_i \ \forall i$. This would mean that exactly one event type exists at any given time in the event sequence.

2.2 Generalized Episode

As in [2], an episode would have a collection of event types with a partial order on it. In addition, each node in the episode is now associated with one or more time intervals, which specify the permitted dwelling times for events while considering the episode's occurrence. For this, we need to have a prespecified collection of all possible time intervals.

Let $\mathcal{I} = \{I_1, \ldots, I_{|\mathcal{I}|}\}$ denote a finite collection of *disjoint* time intervals. Then, given an event (E_i, t_i, τ_i) in the event sequence, there is a unique interval $I_j \in \mathcal{I}$ that can be associated with it such that the duration $\Delta_i \in I_j$. We are now ready to define *generalized episodes*.

A generalized episode is defined as a quadruple $\alpha = (V_\alpha, <_\alpha, g_\alpha, d_\alpha)$. V_α is a finite collection of nodes, and $<_\alpha$ specifies a partial order over it. g_α and d_α are maps, where $g_\alpha : V_\alpha \rightarrow \mathcal{E}$ specifies the map for associating an event type with each node, and $d_\alpha : V_\alpha \rightarrow 2^{\mathcal{I}}$ maps each node to the possible event durations (for that node) that episode α allows. The number of nodes in V_α, denoted $|\alpha|$, is referred to as the *size* of α. In the earlier framework of instantaneous events, an episode was defined by the triple $\alpha = (V_\alpha, <_\alpha, g_\alpha)$. The second map d_α is the new feature of the episode now, which is needed to allow for time durations of events.

We note here that the term *generalized episode* has been used earlier in the literature [10]. This is in the context of a framework whereby an event is described by several attributes (rather than by just a single "event type"). The framework presented in [10] did not at all consider the possibilities of event durations, and the special structures that come up when temporal constraints based on event durations need to be incorporated, as we shall do here (see Section 2.6). In addition, we present efficient algorithms to mine episodes with interesting time duration constraints. However, since in principle, one can think of event durations as another attribute, we decided to use the same term, namely, generalized episodes, for our new temporal pattern as well.

The set $\mathcal{I}$ of all allowable dwelling time intervals is a design choice for the user. In some cases, the data owner has prior knowledge of the kind of time intervals that might

yield interesting patterns. Otherwise, simple histograms over the data sequence can be used to decide on the time intervals that may be included in $\mathcal{I}$. Either way, the specification of $\mathcal{I}$ essentially allows the user to direct the search toward frequent episodes of the kind that are likely to be of interest.

In an episode $\alpha = (V_\alpha, <_\alpha, g_\alpha, d_\alpha)$, if the order $<_\alpha$ is *total*, then α is termed a generalized *serial* episode. If $<_\alpha$ is *empty*, then α is called a generalized *parallel* episode. These are similar to the serial and parallel episodes in the case of instantaneous events [2]. A generalized serial episode is essentially an ordered sequence of nodes, where each node is associated with an event type and a collection of time intervals (which describe the event duration possibilities for that node). When seeking temporal correlations and dependencies in the data, serial episodes are often more interesting than parallel episodes. Although the formalism allows for generalized episodes with all kinds of partial orders, the algorithms described in this paper are mainly concerned with the generalized serial episodes discovery. However, in Section 3.4, we show how our algorithms can be modified for the case of parallel episodes also.

2.3 Episode Occurrence

Consider an episode $\alpha = (V_\alpha, <_\alpha, g_\alpha, d_\alpha)$. The episode α is said to *occur* in the event sequence s with n events if $\exists$ a one-one $h_\alpha : V_\alpha \rightarrow \{1, \ldots, n\}$ such that the following hold $\forall v, w \in V_\alpha$:

$$E_{h_\alpha(v)} = g_\alpha(v), \tag{3}$$

$$\exists b \in d_\alpha(v) \text{ such that } \Delta_{h_\alpha(v)} \in b, \text{ and} \tag{4}$$

$$v <_\alpha w \text{ in } V_\alpha \text{ implies } t_{h_\alpha(v)} \leq t_{h_\alpha(w)} \text{ in } s. \tag{5}$$

(Recall that $E_i \in \mathcal{E}$ denotes the event type of the ith event in the sequence s whose dwelling time is Δ_i.) The occurrence of episode α in the event sequence s is written as $\alpha \triangleright s$.

If we take $t_i = \tau_i \; \forall i$ and drop the condition (4) above, then the above definition reduces to that of the episode occurrence in the framework in [2]. For a generalized episode to occur, in addition to the usual requirement of events satisfying the ordering as demanded by the episode, the event durations have to be consistent with those prescribed by the episode.

Example 1. Consider $\mathcal{I} = \{[0{-}3], [4{-}7], [8{-}11], [12{-}15]\}$ as the collection of time intervals for the event dwelling times. The following are some generalized serial episode examples:

1. $\alpha_1 = (B(0{-}3, 4{-}7, 12{-}15))$,
2. $\alpha_2 = (B(0{-}3, 8{-}11) \rightarrow C(\mathcal{I} \setminus 0{-}3))$,
3. $\alpha_3 = (A(\mathcal{I}) \rightarrow D(\mathcal{I}))$,
4. $\alpha_4 = (B(0{-}3, 8{-}11) \rightarrow C(12{-}15))$, and
5. $\alpha_5 = (B(\mathcal{I} \setminus 12{-}15) \rightarrow A(\mathcal{I}) \rightarrow D(0{-}3, 4{-}7))$.

In our notation, each node is represented by its associated event type, followed by a set of time intervals enclosed in parentheses. For example, consider the episode α_2: Here, the event type C with the dwelling time in any interval of $\mathcal{I}$ except [0–3], follows the event type B with the dwelling time in intervals [0–3] or [8–11]. Observe that α_2 and α_4 represent different episodes, purely on the basis of having different timing information. It can immediately be seen that among these example episodes, all except α_4 occur in the example event sequence (2).

2.4 Subepisode

Let $\alpha = (V_\alpha, <_\alpha, g_\alpha, d_\alpha)$. An episode $\beta = (V_\beta, <_\beta, g_\beta, d_\beta)$ is said to be a *subepisode* of α, written $\beta \preceq \alpha$, if $\exists$ a one-one $f_{\beta\alpha} : V_\beta \rightarrow V_\alpha$ such that $\forall v, w \in V_\beta$, we have

$$g_\alpha(f_{\beta\alpha}(v)) = g_\beta(v), \tag{6}$$

$$d_\alpha(f_{\beta\alpha}(v)) \subseteq d_\beta(v), \text{ and} \tag{7}$$

$$v <_\beta w \text{ in } V_\beta \text{ implies } f_{\beta\alpha}(v) <_\alpha f_{\beta\alpha}(w) \text{ in } V_\alpha. \tag{8}$$

Except for the extra condition given by (7), this is the same as the subepisode definition in [2]. Here, it does *not* suffice if just the relative ordering of the event types in β matches that of α (which was the case in episodes over instantaneous events). In addition, the dwelling time constraints on the event types of β must be consistent with those that α itself would allow.

Notice that among the episodes defined in *Example 1*, whereas $\alpha_1 \not\preceq \alpha_2$, we have $\alpha_3 \preceq \alpha_5$ and $\alpha_2 \preceq \alpha_4$. *Strict* subepisodes, that is, $\beta \preceq \alpha$ and $\beta \neq \alpha$, are denoted by $\beta \prec \alpha$. When $\beta \preceq \alpha$, we also say that α is a *superepisode* of β (written $\alpha \succeq \beta$).

Our definition of episode occurrence is such that if $\beta \preceq \alpha$, then $\alpha \triangleright s$ implies $\beta \triangleright s$. This is because if $\alpha \triangleright s$, then there is a "suitable" map $h_\alpha : V_\alpha \rightarrow \{1, \ldots, n\}$, and further, since $\beta \preceq \alpha$, there is also a "suitable" map $f_{\beta\alpha} : V_\beta \rightarrow V_\alpha$. Now, we can simply define a new map $h_\beta : V_\beta \rightarrow \{1, 2, \ldots, n\}$ as $h_\beta(v) = h_\alpha(f_{\beta\alpha}(v)) \; \forall v \in V_\beta$, and this is a "suitable" map, which demonstrates that $\beta \triangleright s$.

2.5 Episode Frequency

As stated earlier, the pattern discovery problem is that of finding all episodes whose frequency exceeds a given threshold. Episode frequency is some measure of how often an episode occurs in the data sequence, and it can be defined in many different ways. An important consideration while defining a frequency measure for episodes is the need for efficient algorithms for counting (or obtaining the frequency of a set of) candidate episodes in a given event stream.

In the original formulation in [2] (for episodes over streams of instantaneous events), the frequency of an episode is defined as the number of fixed-width sliding windows over the data in which the episode occurs at least once. An automata-based algorithm is used to simultaneously count the frequencies of a set of episodes. Another frequency defined in [2] is the number of minimal occurrences of an episode, where a minimal occurrence is a window such that the episode occurs at least once in the window, and no proper subwindow of it contains an occurrence of the episode. The algorithm for counting the minimal occurrences, whereas being very efficient time-wise when compared to the windows-based count, is very inefficient in terms of space complexity. The windows-

based count is not easily relatable to the intuitive notion of frequency, namely, the number of occurrences of an episode. The windows-based frequency is also dependent on a user-defined window-width parameter though there are some ways to mitigate this [5]. Both windows-based frequency and minimal-occurrences-based frequency are readily applicable to generalized episodes as well. Two new frequency measures for an episode are proposed in [9], based on the so-called nonoverlapped occurrences and noninterleaved occurrences. These count some suitably restricted subsets of the set of all occurrences of the episode. Although nonoverlapped-occurrences-based frequency is applicable to all episodes, noninterleaved-occurrences-based frequency can be used only for serial episodes. For both frequencies, efficient algorithms are available to obtain the frequencies of a set of candidate episodes [9] in the case of data streams with instantaneous events. We now define these frequencies.

Definition 1. *Consider an N-node episode $\alpha = (V, <, g, d)$, where $V = \{v_1, \ldots, v_N\}$. Two occurrences, h_1 and h_2, of α are said to be* nonoverlapped *if either 1) $h_2(v_i) > h_1(v_j)\ \forall v_i, v_j \in V$ or 2) $h_1(v_i) > h_2(v_j)\ \forall v_i, v_j \in V$. The corresponding* frequency *for episode α is defined as the maximum number of nonoverlapped occurrences of α in the given event sequence.*

Two occurrences are nonoverlapped if no event associated with one occurrence occurs in between the events associated with the other occurrence. In case of serial episodes, this means that the event corresponding to the first node of one occurrence appears after all events corresponding to the other occurrence.

Definition 2. *Consider an N-node* serial *episode $\alpha = (V, <, g, d)$, where $V = \{v_1, \ldots, v_N\}$ and $v_1 < \cdots < v_N$. Two occurrences, h_1 and h_2, of α are said to be* noninterleaved *if either $h_2(v_j) > h_1(v_{j+1})\ \forall v_j \in V \setminus \{v_N\}$ or $h_1(v_j) > h_2(v_{j+1})\ \forall v_j \in V \setminus \{v_N\}$. The corresponding* frequency *for episode α is defined as the number of noninterleaved occurrences of α in the given event sequence.*

It is easy to see that two occurrences that are nonoverlapped are also noninterleaved, but the converse does not always hold. This means that for any given (serial) episode, the frequency count based on noninterleaved occurrences is bounded below by the frequency count based on nonoverlapped occurrences. The count based on nonoverlapped occurrences is easy to visualize. In addition, for the case of episodes over instantaneous event sequences, it facilitates connecting frequent episode discovery to generative model learning in terms of some specialized hidden Markov models (HMMs) [4]. The noninterleaved-occurrences-based count, on the other hand, gives us a way to count at least some overlapping occurrences of the episode, which may be relevant in some applications.

2.6 Similar and Principal Episodes

Incorporation of durations for events (through the d-map) in the episode definition implies that there can be *different* episodes, which contain the same sequence of event types. (That is, we can have episodes with identical g-maps but different d-maps.) This gives rise to the idea of similar and principal episodes, which are explained in this section. It may be noted that in case of episodes over instantaneous events [2], episodes with identical g-maps are identical and, hence, the notions of similar and principal episodes are needed only in the generalized episodes framework. The definition of these depends on the frequencies of episodes. In this section, we let $\lambda_s(\alpha)$ denote the frequency of episode α in data stream s. The λ_s function can be any of the frequencies discussed earlier.

2.6.1 Similar Episodes

Consider the following examples of 2-node episodes under the same $\mathcal{I}$ as in *Example 1*:

1. $\beta_0 = (C(4{-}7) \rightarrow A(4{-}7))$,
2. $\beta_1 = (C(4{-}7) \rightarrow A(4{-}7, 8{-}11))$,
3. $\beta_2 = (C(4{-}7) \rightarrow A(\mathcal{I} \setminus 12{-}15))$, and
4. $\beta_3 = (C(4{-}7) \rightarrow A(\mathcal{I}))$.

Notice that all of the episodes have identical event type assignments, that is, they have the same g-maps but differ in their d-maps. Let us consider an event sequence s and suppose that β_0 occurs in s (that is, $\beta_0 \triangleright s$). Since $\beta_3 \prec \beta_2 \prec \beta_1 \prec \beta_0$, it can be inferred that $\beta_i \triangleright s$, $0 \leq i \leq 3$. The β_is differ only in the durations that they allow for the second node (with event type A). As more and more intervals are allowed for that node in the episode, we naturally expect the episode frequency to increase in s. This is true for all the frequencies that we mentioned earlier, and we have $\lambda_s(\beta_3) \geq \lambda_s(\beta_2) \geq \lambda_s(\beta_1) \geq \lambda_s(\beta_0)$. Suppose that, in this example, the episode frequencies satisfy

$$\lambda_s(\beta_3) = \lambda_s(\beta_2) > \lambda_s(\beta_1) > \lambda_s(\beta_0). \tag{9}$$

Then, with respect to their frequencies in s, the episodes β_2 and β_3 are indistinguishable. Episodes like β_2 and β_3 are referred to as being similar in s. Formally, two episodes α and β are said to be *similar* in event sequence s (written $\alpha \overset{s}{\sim} \beta$) if *all* of the following hold:

$$\alpha \preceq \beta \text{ or } \beta \preceq \alpha, \quad |\alpha| = |\beta|, \quad \text{and} \quad \lambda_s(\alpha) = \lambda_s(\beta). \tag{10}$$

It is important to note that similar episodes are always defined with respect to a given event sequence and a *given frequency definition*. Two episodes are similar if they are identical, except for the fact that one allows some more time intervals (for the event dwelling times) than the other, and if their frequencies are equal in the given event sequence.

2.6.2 Principal Episodes

Consider the example episodes discussed above. Using the relationship given by (9), we can say that the additional time intervals for the second node in β_3 (compared to those permitted by β_2) are clearly not interesting. In contrast, the additional time intervals on the second node in β_1 (compared to those permitted by β_0) are interesting. We therefore wish to keep episodes like β_3 out of the eventual rule discovery process. This leads to the idea of principal episodes. Informally, a principal episode is one in which each time interval in each event duration set (of the episode) has earned its place by virtue of a strictly positive contribution to the episode's frequency.

An episode α^* is said to be *principal* in an event sequence s if the following holds:

$$\alpha^* \triangleright s \quad \text{and} \quad \nexists \alpha \succ \alpha^* \text{ such that } \alpha \overset{s}{\sim} \alpha^*. \tag{11}$$

That is, α^* is *principal* if (in the event sequence s) it is not similar to any of its strict superepisodes. In the above example, β_2 is principal, whereas β_3 is not.

The idea of principal episodes resembles that of *closed* itemsets in frequent itemsets mining [11]. An itemset I^* is regarded as closed if there is no other itemset I such that $I \supset I^*$ and $support(I) = support(I^*)$. Thus, if a closed itemset is found to be frequent, then it automatically implies that all of its subsets are also frequent (and, so, there is no need to separately list them as frequent itemsets). In much the same way, a principal episode is the only episode of interest among all episodes that are similar to it. All other episodes that are similar to it are its subepisodes with identical event types and frequency count and, hence, carry no new information.

Subepisodes of principal episodes are, in general, not principal. However, all subepisodes generated by dropping the nodes (rather than by appending extra time intervals to the nodes) of a principal episode are principal. Let α^*_{N-1} be an $(N-1)$-node subepisode of an N-node principal episode α^*_N obtained by dropping one of its nodes (and keeping everything else identical). Since α^*_N is principal, dropping a time interval from any node in α^*_N will reduce its frequency by a strictly positive quantity. Removing the same time interval from the corresponding node in α^*_{N-1} will naturally cause a similar fall in frequency. This shows that there can be no "redundant" time intervals in the nodes of α^*_{N-1} (and, therefore, it must also be principal in s). This is an important property, which is exploited during the candidate generation step of our algorithms.

2.7 Event-Specific $\mathcal{I}$ Sets

Recall from Section 2.2 that $\mathcal{I}$ is a collection of *distinct* time intervals. If $\mathcal{I}$ contains two overlapping time intervals, say, [1–3] and [1–12], then (1-node) episodes like $A(1{-}3, 1{-}12)$ and $A(1{-}12)$ are indistinguishable based on their occurrences whatever the input event stream. This is not desirable and, hence, the need for the restriction of *disjoint* time intervals on $\mathcal{I}$. The main disadvantage with this restriction is that well-defined episodes like $A(1{-}3) \rightarrow B(1{-}12)$, even when frequent, can never be discovered.

This disadvantage can be removed by allowing event-specific $\mathcal{I}$ sets in the formalism. For instance, in the above example, we may define the set of time intervals allowed with event type B as $\mathcal{I}_B = \{[1{-}12]\}$, whereas for all other event types, we may use the set $\mathcal{I} = \{[1{-}3], [7{-}9], [10{-}12]\}$. We note that the formalism developed so far is easily extended to the case of event-specific $\mathcal{I}$ sets. Such a choice will allow the discovery of episodes like $A(1{-}3) \rightarrow B(1{-}12)$. There is, however, a new restriction in the formalism now, namely, that the time intervals allowed for a given event type must be *disjoint*. This is a milder constraint. In Section 5.5, we show some results that highlight the utility of event-specific $\mathcal{I}$ sets.

2.8 Single Interval d-Maps

An interesting special case of the generalized episode defined in Section 2.2 is a restriction that allows only single time intervals to be associated with nodes in the episode. This can be done by restricting the range of d-maps in the generalized episodes to $\mathcal{I}$ rather than $2^{\mathcal{I}}$. Stated formally, for episode $\alpha = (V_\alpha, <_\alpha, g_\alpha, d_\alpha)$, we now have $d_\alpha : V_\alpha \longrightarrow \mathcal{I}$ and V_α, $<_\alpha$ and g_α, as defined earlier. It is easily seen that this new definition does not alter anything else in the formalism, such as episode occurrence, subepisodes, and so forth. We are essentially reducing the set of all possible episodes. Consequently, the number of episodes discovered as frequent may be significantly fewer. This is very useful in some situations where tracking very large number episodes is inefficient. In the case of episodes over instantaneous events, frequent episode discovery under the nonoverlapped-occurrences-based frequency can be related to learning generative models for the data source in terms of some specialized HMMs, and this allows one to assess the statistical significance of the episodes discovered [4]. Such results can be extended to generalized episodes also under some assumptions if we allow only single interval d-maps (see [12] for details).

2.9 Discussion

In this section, we have extended the formalism in [2] to the case of events with nonzero durations. The main features of the extension are as follows: Now, each event has two time stamps associated with it: a start time and an end time. The generalized episode carries two maps: one to associate each node of the episode with an event type and the other to associate each node with possible dwelling times. This extra duration constraint gives rise to the notion of principal episodes. The notions of episode occurrence, subepisodes, and so forth are all modified to take care of the event duration constraints now possible in the episode structure. All notions of the frequency of an episode are naturally extended to the case of generalized episodes.

3 Algorithms under Nonoverlapped Frequency

This section presents algorithms for pattern discovery in the generalized frequent episode framework. Given an event sequence s, a collection of time intervals $\mathcal{I}$, and a frequency threshold, the objective is to obtain the complete set of frequent principal episodes in the data.

The three main constituents of this procedure are candidate generation, frequency counting, and principal episode check. The objective of candidate generation is to present as few episodes as possible of a given size to the frequency counting step and yet not miss any frequent episodes of that size. The frequency counting step takes as input a set of episodes of a particular size, computes frequencies for all episodes in the set in one pass through the data, and outputs all episodes with frequency above a threshold. After this, all frequent episodes are passed through a principal episode check, thereby obtaining all frequent principal episodes of the given size. These steps are repeated for episodes of all desired sizes.

We discuss these steps in detail in the subsections to follow. The main focus of this paper is the generalized serial episodes discovery and, so, we consider the algorithm for serial episodes first. Later, we discuss how this algorithm can be adapted for the case of parallel episodes as well.

3.1 Candidate Generation

In Section 2.6.2, we showed that a necessary condition for an N-node episode to be frequent and principal is that each of its $(N-1)$-node subepisodes obtained by dropping one of its nodes (and not altering the time intervals associated with any other node) are frequent and principal. Given $(N-1)$-node frequent principal episodes, candidate generation obtains all N-node episodes that satisfy this necessary condition.

In order to do this efficiently, we follow the same general strategy as the candidate generation for serial episodes proposed in [2]. The set $\mathcal{F}^*_{N-1}$ of $(N-1)$-node frequent principal episodes is organized in *blocks*, where within each block, all episodes have identical first $(N-2)$ nodes. Consider two $(N-1)$-node frequent principal episodes, say, α_1 and α_2, from the same block of $\mathcal{F}^*_{N-1}$. A potential N-node candidate β is constructed as follows: The event types and time intervals corresponding to the $(N-1)$ nodes of α_1 are copied into the first $(N-1)$ nodes of β. The event type and time intervals of the last node of α_2 are copied into node N of β. Next, we need to check whether all the $(N-1)$ node subepisodes of this β (obtained by dropping a node) are frequent and principal. If they are, then β is declared a candidate N-node episode. This process is repeated for every possible pair of episodes α_1 and α_2 in a block (including the case when $\alpha_1 = \alpha_2$) and for each block in $\mathcal{F}^*_{N-1}$. Also, since we are looking for candidate *serial* episodes, α_1 and α_2 can be combined in two ways to generate a potential candidate β: one is with the last node of α_2 as the last node of β, and the second is with the last node of α_1 as the last node of β. Like the candidate generation of the algorithm in [2], this algorithm too has $\mathcal{O}(N^2|\mathcal{F}_N|^2 \log|\mathcal{F}_N|)$ time complexity.

3.2 Frequency Counting

We now need an algorithm to compute the frequencies of the candidates obtained. In Section 2.5, two occurrences-based frequency counts were defined, namely, the nonoverlapped occurrences count and the noninterleaved occurrences count. As mentioned in that section, it is also possible to extend the windows-based counting scheme in [2] for the case of generalized episodes. In terms of the quality of episodes discovered as frequent, all the three frequency counts are similar. In this paper, we use the nonoverlapped-occurrences-based count as the main frequency definition, and this section describes in detail the algorithm for counting nonoverlapped occurrences of generalized episodes. In Section 4, we briefly indicate how the noninterleaved and windows-based counting schemes can also be used.

Mannila et al. [2] introduced a finite-state automaton-based algorithm for obtaining windows-based frequencies for a set of candidate episodes (over instantaneous event streams). A similar algorithm is employed for counting nonoverlapped occurrences of episodes over instantaneous events [4]. In these algorithms, the occurrence of an episode is recognized by employing automata for each episode. For example, an automaton for episode $(A \rightarrow B \rightarrow C)$ would transit to its first state on seeing an event of type A, then waits for an event of type B to transit to the next state, and so on. When it reaches its final state, an occurrence is regarded as complete.

The same general strategy is used here for counting nonoverlapped occurrences of generalized episodes. However, there is one major difference. All earlier algorithms use N automata per episode to count the frequencies of a set of N-node episodes. Here, we present an algorithm that needs only one automaton per episode.

The inputs to the algorithm are the data stream s, the set of candidates $\mathcal{C}$, the frequency threshold $\lambda_{\min}$, and the set $\mathcal{I} = \{I_1, \ldots, I_{|\mathcal{I}|}\}$ of time intervals used in the pattern discovery process. Each interval I_m, $m = 1, \ldots, |\mathcal{I}|$, is like $[L_m - R_m]$, where L_m and R_m denote the left and right limits of the interval, respectively. We use the following notation: For an episode α, the event type associated with its jth node is denoted by $\alpha.g[j]$, and the set of time intervals associated with its jth node is denoted by $\alpha.d[j]$. *Algorithm 1* obtains the nonoverlapped-occurrences-based frequencies for a set of candidate serial episodes. The algorithm is given below as pseudocode, and we refer to the line numbers while explaining the algorithm.

Algorithm 1. Nonoverlapped count for generalized serial episodes

Input: Set $\mathcal{C}$ of candidate N-node generalized serial episodes, event stream $s = \langle (E_1, t_1, \tau_1), \ldots, (E_n, t_n, \tau_n)) \rangle$, frequency threshold $\lambda_{\min} \in [0, 1]$, and set $\mathcal{I} = \{I_1, \ldots, I_{|\mathcal{I}|}\}$ of allowed time intervals (where each interval $I_m = [L_m - R_m]$)

Output: Set $\mathcal{F}$ of frequent generalized serial episodes in $\mathcal{C}$

```
1: for all event types A and time intervals I_m do
2:     Initialize waits(A, I_m) = φ
3: for all α ∈ C do
4:     for all I_m ∈ α.d[1] do
5:         Add (α, 1) to waits(α.g[1], I_m)
6:         Initialize α.freq = 0
7: Initialize bag = φ
8: for i = 1 to n do
9:     /* n is length of data stream */
10:    Find I_K ∈ I such that L_K ≤ (τ_i − t_i) ≤ R_K
11:    for all (α, j) ∈ waits(E_i, I_K) do
12:        for all I_m ∈ α.d[j] do
13:            Remove (α, j) from waits(E_i, I_m)
14:        Set j' = j + 1
15:        if j' = (N + 1) then
16:            Set j' = 1
17:        for all I_m ∈ α.d[j'] do
18:            if α.g[j'] = E_i and I_m = I_K then
19:                Add (α, j') to bag
20:            else
21:                Add (α, j') to waits(α.g[j'], I_m)
22:        if j = N then
23:            Update α.freq = α.freq + 1
24:    Empty bag into waits(E_i, I_K)
25: Output F = {α ∈ C such that α.freq ≥ nλ_min}
```

The main data structure is the $waits(\cdot,\cdot)$ list. It links together all automata that accept a particular "event type, time interval" pair. The list $waits(A, I_m)$ stores pairs like (α, j), indicating that an automaton for episode α is waiting for the event type A to occur with a duration in the I_m time interval in order to transit to state j.[2]

Since the generalized episode definition allows more than one time interval to be associated with a node, an (α, j) entry can simultaneously appear in more than one $waits$ list. For example, let the time intervals allowed by the jth node of episode α be $\alpha.d[j] = \{I_k, I_l\}$. Then, (α, j) would appear in both $waits(\alpha.g[j], I_k)$ and $waits(\alpha.g[j], I_l)$. It is mainly in handling of this aspect of $waits$ that the algorithm differs from the nonoverlapped-occurrence-based counting algorithm for episodes over instantaneous events [4], [9].

Algorithm 1 makes one pass over the data to count the frequencies of all candidate episodes. At the start, there would be automata waiting for "event type, time interval" pairs corresponding to the first nodes of all candidate episodes in $\mathcal{C}$. This is how the $waits$ list is initialized (lines 1–6, *Algorithm 1*).

Proceeding left to right along the event stream, the automata of all episodes are simultaneously tracked by appropriately updating $waits$. For example, let (E_i, t_i, τ_i) be the next event in the data stream and let the event duration $(\tau_i - t_i)$ belong to time interval I_K. For each entry (α, j) in $waits(E_i, I_K)$, we effect a state transition as follows: The (α, j) entry is removed from all $waits(E_i, I_m)$, $I_m \in \alpha.d[j]$ (lines 11–13, *Algorithm 1*). $(\alpha, j+1)$ is added to all $waits(\alpha.g[j+1], I_m)$, $I_m \in \alpha.d[j+1]$ (except when j corresponds to the last node of α, in which case a fresh automaton for α is initialized by adding $(\alpha, 1)$ to all $waits(\alpha.g[1], I_m)$, $I_m \in \alpha.d[1]$; lines 14–21, *Algorithm 1*). Since we loop over elements of $waits(E_i, I_K)$, it is inappropriate to add to this list from within the loop (when $\alpha.g[j+1] = E_i$, and $I_m = I_K$). In such cases, $(\alpha, j+1)$ is added to a temporary list, called *bag*, which is later emptied into $waits(E_i, I_K)$ on coming out of the loop (lines 18–19 and 24, *Algorithm 1*). Finally, when an automaton reaches its final state, the corresponding episode frequency is incremented by 1 (lines 22–23, *Algorithm 1*).

Space and time complexity. The set $\mathcal{I}$ of time interval possibilities is an input to the algorithm. Only one automaton per episode is needed to track the nonoverlapped occurrences of a set of episodes. However, there may be as many as $|\mathcal{I}|$ time intervals associated with each node of the episode and, so, at a given instant, an automaton may be waiting in as many as $|\mathcal{I}|$ number of $waits(\cdot,\cdot)$ lists. Since there are $|\mathcal{C}|$ candidates, the space complexity of *Algorithm 1* is $\mathcal{O}(|\mathcal{C}||\mathcal{I}|)$.

Initializations (lines 1–7, *Algorithm 1*) take $\mathcal{O}(|\mathcal{C}||\mathcal{I}|)$ time. For an input sequence with n events, the outer loop (starting at line 10, *Algorithm 1*) is entered n times, and in the worst case, we may need to update each of the $|\mathcal{C}|$ automata every time this loop is entered. For each of these automata, state-transition updates (lines 12–21, *Algorithm 1*) take $\mathcal{O}(|\mathcal{I}|)$ time. Thus, the total worst-case time complexity is $\mathcal{O}(n|\mathcal{C}||\mathcal{I}|)$.

Since here, we have to take care of event duration possibilities, both the space and time complexities have an extra factor of $|\mathcal{I}|$ when compared to the complexities of algorithms for frequent episode discovery over instantaneous events. However, all of the available algorithms for frequent episode discovery (over instantaneous events) have space and time complexities that scale linearly with N, which is the size of episodes, whereas for our algorithms, the space and time complexities are independent of episode size.

3.3 Principal Episodes Check

Once the set $\mathcal{F}$ of frequent episodes is found, we need to check which of the episodes in $\mathcal{F}$ are principal. An episode α in $\mathcal{F}$ is declared as principal if there are no superepisodes of α in $\mathcal{F}$ with the same frequency as α. Any superepisode of α in $\mathcal{F}$ is of the same size as α (and, hence) with identical event type associations for its nodes and can differ from α only in the time intervals that it allows in one or more nodes. Since $\mathcal{F}$ is already organized in lexicographic order, all required superepisodes of α appear in a contiguous block of $\mathcal{F}$. In the worst case, all episodes in $\mathcal{F}$ may belong to the same block, which means that the search for the superepisodes of every α in $\mathcal{F}$ may need up to $\mathcal{O}(|\mathcal{F}|^2)$ time. However, this is often not the case. Blocks in $\mathcal{F}$ do not tend to be very large, and the actual time required for the principal episode check is much less.

3.4 Parallel Episodes Discovery

The algorithm presented above is concerned with the generalized serial episode discovery. We now briefly discuss the case of generalized parallel episodes discovery.

Candidate generation can be done along the same lines, as described in Section 3.1. We consider every possible pair of $(N-1)$-node frequent principal generalized parallel episodes (discovered in the $(N-1)$th level of the algorithm) such that both episodes have the same first $(N-2)$ nodes. A potential N-node candidate is constructed using the first $(N-1)$ nodes of one episode and the $(N-1)$th node of the other. This new episode is declared a candidate if all its $(N-1)$-node subepisodes can be found in the set of frequent principal generalized parallel episodes of size $(N-1)$. The only difference with respect to the candidate generation procedure for generalized serial episodes is that now, two $(N-1)$-node frequent episodes, α_1 and α_2, can be combined in only one way to generate a potential candidate β, since there is no ordering information among the nodes for parallel episodes.

The algorithm for counting nonoverlapped occurrences of the generalized parallel episodes also uses a $waits$ list indexed by an "event type, time interval" pair. However, it works a little differently from the earlier $waits$ list used in *Algorithm 1*. The entries in $waits(A, I_m)$ would still be pairs of the form (α, j). However, now, if $(\alpha, j) \in waits(A, I_m)$, then it indicates that there is a partial occurrence of α, which still needs j events of type A with durations in the interval I_m for it to become a complete occurrence of α. Consider, for example, the 3-node generalized parallel episode $\alpha = (A(1{-}3, 4{-}6)\ B(1{-}3)\ A(1{-}3, 7{-}9))$. The $waits$

2. This data structure is similar to the main data structure in other frequent episode discovery algorithms and was originally proposed in [2]. The main difference here is that it is indexed by "event type"-"time interval" pairs rather than by event type alone.

list is initialized for this α by adding $(\alpha, 2)$ to $waits(A, 1-3)$ and adding $(\alpha, 1)$ to each of $waits(A, 4-6)$, $waits(A, 7-9)$, and $waits(B, 1-3)$. Note that since there can be more than one time interval associated with a node, there may be more than N entries in the $waits$ lists corresponding to a given N-node generalized parallel episode.

The overall procedure for counting the generalized parallel episodes is explained as follows: On seeing the event (E_i, t_i, τ_i) in the event sequence, the time interval I_K corresponding to the duration $(\tau_i - t_i)$ is identified, and the elements in $waits(E_i, I_K)$ are accessed. For each $(\alpha, j) \in waits(E_i, I_K)$, having now seen an event of type E_i with time duration in time interval I_K, (α, j) is replaced by $(\alpha, j-1)$ in $waits(E_i, I_K)$ if $j \geq 1$. A counter called $\alpha.counter$ is maintained for each episode α. This counter stores the number of nodes for which corresponding events exist in the occurrence of α that is currently being tracked. $\alpha.counter$ is incremented each time an (α, j) is accessed in one of the $waits$ lists, and j is decremented. When $\alpha.counter = N$ (where N is the size of α), it means that the N events necessary for completing an occurrence have appeared in the event sequence. Thus, the frequency is incremented by one, $\alpha.counter$ is reset to zero, and the $waits$ lists are updated to wait for a fresh occurrence of α. The detailed algorithm for parallel episode discovery is available in [12].

4 Algorithms under Other Frequency Counts

In Section 2.5, two occurrences-based frequency counts were defined, namely, the nonoverlapped occurrences count and the noninterleaved occurrences count. As mentioned in that section, it is also possible to extend the windows-based counting scheme in [2] for the case of generalized episodes. In this section, we discuss algorithms under the noninterleaved-occurrences-based count and the windows-based count. Each of these algorithms is essentially an extension of the corresponding algorithm [2], [4] for episode discovery over instantaneous event streams. The main changes are with respect to the additional time duration checks that need to be incorporated when recognizing occurrences of generalized episodes. Hence, we only provide a general description of the algorithms and do not list any pseudocodes. More detailed descriptions are available in [12].

4.1 Noninterleaved-Occurrences-Based Count

When using the noninterleaved-occurrences-based frequency definition, we must bear in mind that subepisodes, in general, may not be as frequent as episodes. However, the two subepisodes formed by dropping the first and last nodes of an episode are guaranteed to be at least as frequent as the episode. This property, along with the fact that such subepisodes obtained by dropping the nodes of principal episodes will remain principal as well, are together exploited for candidate generation. Frequent principal generalized episodes of size N are combined to obtain candidates of size $(N+1)$ in the following manner. Episodes α and β are combined only if by dropping the first node of one (say α) and the last node of the other (say β), we get identical $(N-1)$-node subepisodes. They are combined to form an $(N+1)$-node candidate γ by copying the whole of α into the first N nodes of γ and the last node of β into the $(N+1)$th node of γ. This is done for every ordered pair of episodes from the set of frequent principal generalized serial episodes of size N.

The algorithm for counting noninterleaved occurrences needs $(N-1)$ automata per N-node episode. In order to track episodes with time duration constraints, the $waits$ lists, like in Section 3, are indexed by both an event type and a time interval. Scanning the event stream in time order, if (E_i, t_i, τ_i) is the current event being processed, then we obtain the appropriate time interval I_K and look at the automata in $waits(E_i, I_K)$.

There are two important characteristics of the noninterleaved-occurrences-based counting algorithm. First, if an entry (α, j) is found in a $waits(E_i, I_K)$, then we must check to see that there are no instances of $(\alpha, j+1)$ already in the $waits$ lists before allowing a transition of an automaton for α into state j. The second important characteristic is that since we need to count *some* overlapping occurrences, a fresh automaton for an episode may be initialized before an earlier automaton reaches its final state. More specifically, a fresh automaton is initialized after an earlier one reaches state 2. Proceeding this way, a third automaton may be initialized after the second automaton transited to state 2 (which, in turn, is possible only after the first automaton transited to state 3), and so on. Thus, at any given point in time, there can be up to $(N-1)$ automata that are currently active for an N-node episode α. As was the case in *Algorithm 1*, removing an automaton entry (α, j) from the $waits$ lists involves removing it from $waits(\alpha.g[j], I_m)$ for all $I_m \in \alpha.d[j]$. Similarly, putting an automaton entry $(\alpha, j+1)$ into the $waits$ lists involves putting $(\alpha, j+1)$ into $waits(\alpha.g[j+1], I_m)$ for all $I_m \in \alpha.d[j+1]$. The space and time complexities of such an algorithm are $\mathcal{O}(N|\mathcal{C}||\mathcal{I}|)$ and $\mathcal{O}(nN|\mathcal{C}||\mathcal{I}|)$, respectively.

4.2 Windows-Based Count

The windows-based counting algorithm for obtaining the frequencies of generalized serial episodes proceeds along the same lines as the algorithm for counting serial episodes in [2]. The crucial difference lies in the $waits$ data structure, which now is indexed by two entries: event type and time interval. Like in [2], here too the user must specify a window width, and an episode is deemed to occur in a sliding window if the start times of events corresponding to the first and last nodes lie within it. The left end of the sliding windows in the data also need to be tracked, just like in the algorithm in [2]. This, of course, is absent in the occurrences-based counting algorithms. Since we need N automata per N-node episode, the space complexity is $\mathcal{O}(N|\mathcal{C}||\mathcal{I}|)$. The time complexity is $\mathcal{O}(\Delta T N|\mathcal{C}||\mathcal{I}|)$, where ΔT is the total number of time ticks in the given input sequence.

5 Results

This section presents results obtained using the generalized frequent episode discovery framework on both synthetic data and data from General Motors (GM) manufacturing plants. All the results quoted in this section were obtained

using the nonoverlapped-occurrences-based count (see Section 2.5 and 3.2) as the episode frequency.

5.1 Synthetic Data

Synthetic data was obtained by embedding some temporal patterns in noise. The idea is to generate event sequences with specific bias for some temporal patterns so that it is possible to test whether our algorithm picks these temporal patterns. By varying the parameters of the data generation process, several different time-stamped sequences of events were generated.

A temporal pattern to be embedded consists of a specific (short) sequence of event types, with each event type associated with some time intervals (for its dwelling time). A few such temporal patterns are input to the data generation process, and the idea is to generate synthetic event sequences by embedding several arbitrarily interleaved occurrences of these patterns in noise. The degree of noise is controlled by a parameter ρ. If at the current time instant (which is maintained by a counter), an event is to be generated, then the event will be what we would call an independent and identically distributed (*i.i.d.*) event with probability ρ; that is, the event type is a randomly chosen element of $\mathcal{E}$, and its dwelling time is chosen based on a uniform distribution over some prespecified range of time durations. The event generated at the current time will correspond to one of the (input) temporal patterns with probability $(1-\rho)$. In this case, one of these temporal patterns is chosen randomly (according to some mixing probability distribution). Having chosen a temporal pattern, another random choice is made as to whether a new occurrence must be started or a previous partial occurrence continued. Either way, we would have the event type to be used for the next event in the sequence. The event's start time is the current time. The dwelling time is randomly generated using a uniform distribution over the set of allowed time durations, as defined in the pattern. This dwelling time is added to the start time to obtain the event's end time. The current time is advanced by a small random integer, and the process is repeated for this new current time. It may be noted here that due to the nature of our data generation process, embedding a temporal pattern is equivalent to embedding many episodes such as those corresponding to cyclic permutations of the pattern and their subepisodes.

In our formalism, $\mathcal{I}$ denotes the set of all possible intervals for the dwelling times that the algorithm considers during the episode discovery process. It is noted that the set $\mathcal{I}$ used by the episode discovery algorithm need not coincide with the set of time intervals used for data generation. In practice, whereas some typical dwelling times of events in the data can be observed by inspection, one would not know the distributions of dwelling times. However, the user can choose the set $\mathcal{I}$, depending on the kind of episodes that one is interested in discovering. Thus, like the frequency threshold and the confidence threshold in rule discovery, the choice of set $\mathcal{I}$ is a design choice, which the user can exploit to orient the episode discovery process in some desired direction(s). Section 5.4 illustrates through some simulation experiments how this can be done.

5.2 GM Engine Assembly Plant Data

This data set consists of real-time status logs from one of GM's engine assembly plants. Each engine is assembled by passing it through several stations (or machines), where different operations are performed on it. Whenever a station or machine reports a fault, it is automatically logged in a fault sheet. Each entry here includes a *machine code*, a *fault code*, a *start time*, and an *end time* (that is, the time when the problem was solved). The goal of temporal data mining here is to unearth interesting correlations that can help diagnose the root causes of a stoppage or other problems in the assembly line. Section 5.6 describes how we used the generalized frequent episodes framework to analyze this data and shows some interesting results obtained. Due to length constraints (and also issues of confidentiality), we only provide some summary discussion of this application here. The frequent episodes discovery algorithms presented in this paper have been integrated into a temporal data mining software tool and is currently being used to provide online help for fault analysis in several engine assembly plants of GM.

5.3 Utility of the Generalized Episodes Framework

The objective of our first simulation experiment is to show what kind of extra expressive power generalized episodes give us for discovering patterns with temporal dependencies. To do this, we generated synthetic data sequences by embedding two temporal patterns:

1. $A(1{-}3) \rightarrow B(7{-}9) \rightarrow C(1{-}3)$ and
2. $A(7{-}9) \rightarrow B(1{-}3) \rightarrow C(7{-}9)$.

The data generated contained 50,000 events over a 200-sized alphabet. Both temporal patterns were embedded in equal proportion, and the *i.i.d.* event probability, ρ was set to 0.20. The time durations for the noise events were chosen uniformly from the interval [1–50].

Note that the two patterns (embedded in the data) comprise exactly the same event types and differ only with regard to the duration information associated with the different nodes. Clearly, these two patterns are indistinguishable in the original framework of frequent episodes (over instantaneous event streams). If we disregard event durations in the data and use the earlier framework (for example, the algorithms in [2], [4]), then we would only be able to discover that $(A \rightarrow B \rightarrow C)$ is a frequent episode. We would not be able to infer that there are certain specific biases in the time durations of events in the data.

However, using the generalized episode framework, it is possible to unearth these time-duration-dependent correlations. Consider the generalized serial episodes framework using the set $\mathcal{I} = \{1{-}3, 7{-}9\}$ of time duration possibilities. Let us restrict our attention to only generalized episodes with single interval d-maps (see Section 2.8). Since there are two time interval possibilities for each node, and since here, we allow only one time interval for each node of an episode, there are eight possible generalized 3-node serial episodes, with the three event types being A, B, and C in that order (our embedded patterns correspond to two of these). We ran our generalized serial episode discovery algorithm on this synthetic data set to see which of these eight generalized episodes figured in the list of frequent generalized

TABLE 1
Expressive Power of Generalized Episodes Framework: Ranks, Frequencies, and $(\frac{f}{f_{max}})$ of the Eight 3-Node Generalized Serial Episodes Using Single-Interval d-Maps and with $\mathcal{I} = \{1{-}3, 7{-}9\}$

Generalized episode	$\mathcal{I} = \{1\text{–}3, 7\text{–}9\}$		
	Rank	f	$\frac{f}{f_{max}}$
$A(7\text{–}9) \rightarrow B(1\text{–}3) \rightarrow C(7\text{–}9)$	1	6767	1.0
$A(1\text{–}3) \rightarrow B(7\text{–}9) \rightarrow C(1\text{–}3)$	4	6629	0.98
$A(1\text{–}3) \rightarrow B(7\text{–}9) \rightarrow C(7\text{–}9)$	32	3321	0.49
$A(7\text{–}9) \rightarrow B(1\text{–}3) \rightarrow C(1\text{–}3)$	32	3321	0.49
$A(1\text{–}3) \rightarrow B(1\text{–}3) \rightarrow C(1\text{–}3)$	51	3320	0.49
$A(7\text{–}9) \rightarrow B(7\text{–}9) \rightarrow C(1\text{–}3)$	51	3320	0.49
$A(7\text{–}9) \rightarrow B(7\text{–}9) \rightarrow C(7\text{–}9)$	> 100	-	-
$A(1\text{–}3) \rightarrow B(1\text{–}3) \rightarrow C(7\text{–}9)$	> 100	-	-

Synthetic data was generated by embedding two of these eight episodes. Data length: 50,000. $\rho = 0.20$. Number of event types: 200.

episodes. The results are tabulated in Table 1. For each episode, the table quotes the rank of the episode in the frequency-sorted list of frequent episodes, the frequency f of the episode, and the ratio $(\frac{f}{f_{max}})$, where f_{max} denotes the frequency of the most frequent episode. It can be seen that of the eight generalized episodes, only two appear in the top five, and these two are exactly the generalized episodes corresponding to the patterns embedded in the data. One of the embedded patterns is the most frequent episode. The second embedded pattern has rank 4 and $(\frac{f}{f_{max}})$ of 0.98 (the episodes ranked 2 and 3 are the two cyclic permutations of the most frequent episode). All of the other six episodes have $(\frac{f}{f_{max}})$ less than 0.5 (some of them do not even figure in the list of the top-100 frequent episodes). The eight generalized episodes listed in Table 1 are indistinguishable under the earlier framework. Thus, this example shows that the generalized episodes framework is capable of unearthing time-duration-dependent correlations in the data (when they exist), which are outside the expressive power of the earlier framework of frequent episode discovery over instantaneous event streams.

We note that the effectiveness of the generalized episodes framework is not sensitive to various parameters in the data generation process. For example, we can have bigger time intervals in the nodes of patterns that are embedded, we can vary the level of the *i.i.d.* noise ρ, we need not restrict ourselves to single interval d-maps, and so forth. Our next experiment illustrates this aspect. The synthetic data is generated by embedding two patterns in varying levels of noise (with $\rho = 0, 0.2, 0.4, 0.6, 0.8$):

1. $\alpha_1 = (A(1{-}10) \rightarrow B(101 - 150){-}C(1{-}10))$ and
2. $\alpha_2 = (A(101{-}150) \rightarrow B(1 - 20){-}D(101{-}150))$.

Once again, the synthetically generated sequences are 50,000 long over an alphabet of 200 event types. The time durations for the noise events are now drawn uniformly from the interval [1–250]. Let the set of time interval possibilities be $\mathcal{I} = \{1{-}20, 101{-}150\}$. Consider the full set of generalized serial episodes over this set of time intervals so that this time, each node may be associated with more than one time interval. The results of frequent generalized serial episode discovery are presented in Table 2 by showing the ranks and $(\frac{f}{f_{max}})$ values of α_1 and α_2 under different ρ. The episodes α_1 and α_2 continue to appear among the top few frequent generalized episodes, and their $(\frac{f}{f_{max}})$ values are always greater than or equal to 0.96. Notice, however, that whereas one of them is always the most frequent episode, the rank of the other is somewhat lower when compared to the experiment in Table 1. This is because since more than one time interval can be associated with a node, and since the event types A and B appear with two different time intervals in the two embedded patterns, episodes like $(A(1{-}10, 101{-}150) \rightarrow B(1{-}10, 101 - 150) \rightarrow C(1{-}10))$ have a little more frequency than $(A(1{-}10) \rightarrow B(101 - 150) \rightarrow C(1{-}10))$. However, observe that their differences in frequency are very small, clearly from the fact that the $(\frac{f}{f_{max}})$ values of α_1 and α_2 are always very close to unity.

TABLE 2
Ranks of α_1 and α_2 in Synthetic Data with Patterns Embedded in Varying Levels of Noise

IID noise probability ρ	$\mathcal{I} = \{1\text{–}20, 101\text{–}150\}$			
	α_1		α_2	
	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$
0.0	12	0.98	1	1.00
0.2	12	0.99	1	1.00
0.4	1	1.00	12	0.99
0.6	1	1.00	11	0.96
0.8	1	1.00	10	0.97

Set of time interval possibilities: $\mathcal{I} = \{1{-}20, 101{-}150\}$. Data length: 50,000. Number of event types: 200.

We once again note in this example the role played by the different time durations in the two patterns embedded in the data. Although an event of type A of a short time duration followed by a B of long duration leads to an event of type C, a longer duration A followed by a short duration B leads to some other event of type D. The original framework of frequent episodes over instantaneous event streams would pick both $(A \rightarrow B \rightarrow C)$ and $(A \rightarrow B \rightarrow D)$ as frequent episodes but cannot unearth the specific time-duration-dependent bias among the correlations involving the event types A, B, C, and D. This is the extra expressive power gained by the generalized episodes framework.

Next, we show that this enhanced expressive power of the generalized episodes framework comes with a reasonable increase in computational cost. Table 3 lists the total runtimes for the generalized frequent episode discovery for synthetic data of different lengths and different levels of noise. The data length was varied from 10,000 to 50,000 and the noise level ρ from 0.0 to 0.80. The set of time interval possibilities was taken as $\mathcal{I} = \{1{-}20, 101{-}150\}$. The frequency thresholds were chosen in such a way that the final number of frequent 4-node generalized serial episodes discovered was around 150. The actual runtimes[3] are all very reasonable. These times are only about two or three times more than the frequent episode discovery in the earlier framework of instantaneous events (see results section in [12, chapter 6]).

Another experiment that we conducted was to study the effect of the size of $\mathcal{I}$ on the runtimes of the generalized

3. All runtimes quoted in this section are on a Pentium 4 machine with a 2-GHz clock and 1-Gbyte main memory running Linux.

TABLE 3
Runtimes (in Seconds) of Frequent Generalized Serial Episode Discovery for Different Data Lengths and Different Noise Levels

IID noise probability ρ	Length of event sequence				
	10000	20000	30000	40000	50000
0.0	0.70	1.56	1.89	2.22	3.05
0.2	1.01	1.49	2.08	2.56	3.20
0.4	0.77	1.28	1.91	2.45	2.76
0.6	0.65	0.99	1.58	1.90	2.12
0.8	0.53	0.76	1.03	1.43	1.59

Set of time interval possibilities: $\mathcal{I} = \{1{-}20, 101{-}150\}$. Data length: 50,000. Number of event types: 200.

frequent episode discovery algorithms. Table 4 lists the runtimes needed for the frequent generalized serial episode discovery using these different $\mathcal{I}$ sets. All three sets are capable of describing time durations in the interval [1–200]. Note that the class of generalized episodes under consideration itself grows with the number of intervals in $\mathcal{I}$. Hence, rather than trying to fix the size of the final set of frequent generalized episodes discovered (which is what we did earlier in Table 3), we simply quote the runtimes obtained with different frequency thresholds. For smaller thresholds, a larger number of episodes are output, and hence, the corresponding runtimes are also more (for each case, the number of frequent 4-node generalized serial episodes obtained is mentioned within brackets in Table 4). For larger frequency thresholds, the number of frequent generalized episodes output is small, and this explains why the factors by which the runtimes increase with respect to size of $\mathcal{I}$ are more for lower frequency thresholds. Based on the runtimes in Tables 3 and 4, we feel that the computational cost is very reasonable for the extra expressive power that the generalized episodes framework gives us for unearthing time-duration-dependent correlations in the data.

5.4 Set $\mathcal{I}$ as a Design Choice

The main exploratory tool in the generalized episodes framework is the choice of set $\mathcal{I}$ of time intervals. Clearly, irrespective of what patterns are actually frequent in the data, the frequent episode discovery process will only output those episodes that are within the expressive power of the $\mathcal{I}$ set used. In this section, we illustrate this. Consider the synthetic data generated in the previous section by embedding two patterns in different levels of noise:

TABLE 4
Runtimes (in Seconds) of Frequent Generalized Serial Episode Discovery for $\mathcal{I}$ Sets of Different Sizes under Different Frequency Thresholds

Frequency threshold	$\mathcal{I}$-sets of different sizes		
	{1–200}	{1–100, 101–150}	{1–50, 51–100, 101–150}
0.01	1.98 (65)	11.03 (280)	13.21 (300)
0.05	1.78 (55)	10.38 (175)	13.05 (189)
0.10	1.10 (15)	2.20 (20)	3.30 (33)
0.15	0.62 (0)	1.35 (0)	2.50 (0)

Numbers in brackets indicate number of frequent 4-node generalized serial episodes discovered. Data length: 50,000. $\rho = 0.40$. Number of event types: 200.

TABLE 5
Set $\mathcal{I}$ as a Design Choice: Ranks and $(\frac{f}{f_{max}})$ Values for α_1 and α_2 (in Synthetic Data with Different Levels of Noise)

IID noise probability ρ	$\mathcal{I} = \{1{-}10, 101{-}150\}$			
	α_1		α_2	
	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$
0.0	1	1.00	51	0.51
0.2	1	1.00	32	0.50
0.4	1	1.00	35	0.50
0.6	1	1.00	100	0.48
0.8	1	1.00	99	0.49

Set of time interval possibilities: $\mathcal{I} = \{1{-}10, 101{-}150\}$. Data length: 50,000. Number of event types: 200.

1. $\alpha_1 = (A(1{-}10) \rightarrow B(101 - 150){-}C(1{-}10))$ and
2. $\alpha_2 = (A(101{-}150) \rightarrow B(1 - 20){-}D(101{-}150))$.

Table 2 shows that by using $\mathcal{I} = \{1{-}20, 101{-}150\}$, both these generalized episodes are discovered as frequent with high ranks and with $(\frac{f}{f_{max}})$ values very close to unity. Suppose that now, we use $\mathcal{I} = \{1{-}10, 101{-}150\}$. The results obtained are shown in Table 5. Notice that the duration of B in α_2 takes values in the interval [1–20], which is not fully describable by using the new $\mathcal{I}$. This pushes up the rank of α_1 and significantly pushes down the rank of α_2. This can be seen from the results shown in Table 5.

By contrast, choosing $\mathcal{I} = \{11{-}20, 101 - 150\}$ has the opposite effect on the ranks of α_1 and α_2. The corresponding ranks and $(\frac{f}{f_{max}})$ values are quoted in Table 6. Under this $\mathcal{I}$, since the duration of A in α_1 takes values in the interval [1–10] (which cannot be described at all by using this $\mathcal{I}$), α_1 does not even figure in the list of the top-100 frequent generalized episodes. On the other hand, the ranks of α_2 improve with respect to those in Table 5. Observe, however, that the ranks of α_2 are still a bit low (compared to the ranks of α_1 in Table 5). This is because the time interval [11–20] (in $\mathcal{I}$) describes only half the time durations for B in α_2 and, so, some noise events push up the frequency of episodes like $A(11{-}20, 101{-}150) \rightarrow B(101{-}150) \rightarrow D(101{-}150)$. Now, consider the case when we use the same $\mathcal{I}$, that is, $\mathcal{I} = \{11{-}20, 101{-}150\}$, but the episodes have only single interval d-maps (see Section 2.8). The results obtained for this choice are quoted in Table 7. Since now, no combination of time intervals are allowed in any node, many of the "noise" episodes disappear, and the ranks of α_2 improve

TABLE 6
Set $\mathcal{I}$ as a Design Choice: Ranks and $(\frac{f}{f_{max}})$ Values for α_1 and α_2 (in Synthetic Data with Different Levels of Noise)

IID noise probability ρ	$\mathcal{I} = \{11{-}20, 101{-}150\}$			
	α_1		α_2	
	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$
0.0	> 100	-	1	1.00
0.2	> 100	-	20	0.83
0.4	> 100	-	23	0.82
0.6	> 100	-	17	0.82
0.8	> 100	-	17	0.83

Set of time interval possibilities: $\mathcal{I} = \{11{-}20, 101{-}150\}$. Data length: 50,000. Number of event types: 200.

TABLE 7
Set $\mathcal{I}$ as a Design Choice: Ranks and $(\frac{f}{f_{max}})$ Values for α_1 and α_2 (in Synthetic Data with Different Levels of Noise) Considering Only Generalized Episodes with Single Interval d-Maps

IID noise probability ρ	$\mathcal{I} = \{11\text{–}20, 101\text{–}150\}$			
	α_1		α_2	
	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$
0.0	> 100	-	1	1.00
0.2	> 100	-	7	1.00
0.4	> 100	-	10	0.99
0.6	> 100	-	4	0.99
0.8	> 100	-	4	0.97

Set of time interval possibilities: $\mathcal{I} = \{11{-}20, 101{-}150\}$. *Data length: 50,000, Number of event types: 200.*

TABLE 9
Set $\mathcal{I}$ as a Design Choice: Ranks and $(\frac{f}{f_{max}})$ Values for α_1, α_2, and α_3 (in Synthetic Data with Different Levels of Noise)

IID noise proba-bility ρ	$\mathcal{I} = \{1\text{–}100, 101\text{–}200\}$					
	α_1		α_2		α_3	
	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$
0.0	7	0.74	5	0.76	6	0.75
0.2	4	0.81	5	0.76	6	0.73
0.4	3	0.85	5	0.84	6	0.70
0.6	2	0.89	4	0.79	5	0.68
0.8	2	0.96	3	0.86	8	0.70

Set of time interval possibilities: $\mathcal{I} = \{1{-}100, 101{-}200\}$. *Data length: 50,000. Number of event types: 200.*

significantly. Once again, since there are no time intervals to describe events of durations [1–10] (which are needed for events of types A and C in an occurrence of α_1), α_1 does not appear in the list of the top-100 frequent episodes.

Next, we consider the case of synthetic data with a third pattern α_3 embedded in it, in addition to the earlier patterns α_1 and α_2. That is, three patterns are embedded in equal proportion in varying levels of noise:

1. $\alpha_1 = (A(1{-}10) \rightarrow B(101 - 150){-}C(1{-}10))$,
2. $\alpha_2 = (A(101{-}150) \rightarrow B(1 - 20){-}D(101{-}150))$, and
3. $\alpha_3 = (C(51{-}100) \rightarrow D(101{-}200){-}E(101{-}200))$.

If we choose $\mathcal{I} = \{51{-}100, 101{-}200\}$, since there are no intervals that can describe small time durations, then both α_1 and α_2 will get pushed down the list of frequent generalized episodes. The results obtained for this choice are listed in Table 8. In fact, as can be seen in the table, α_1 and α_2 do not even appear among the top-100 frequent generalized episodes. However, if we take $\mathcal{I} = \{1{-}100, 101{-}200\}$, then the ranks of α_1 and α_2 improve. This can be seen from the results quoted in Table 9. All three generalized episodes now have ranks in the top 10. This is because all the time durations in the embedded patterns can be described using one of the time intervals in $\mathcal{I}$.

5.5 Event-Specific $\mathcal{I}$ Sets

Another useful way to focus the frequent generalized episode discovery is to use event-specific $\mathcal{I}$ sets (see Section 2.7). For example, in the synthetic data generated by embedding α_1, α_2, and α_3, suppose that we are not interested in episodes with large time durations events (which we define here as, say, events with durations greater than 100), except for events of type B. If we choose $\mathcal{I}_B = \{101{-}200\}$ as the set of time interval possibilities for event type B, and $\mathcal{I} = \{1{-}20, 51{-}100\}$ for all other event types, then the most frequent generalized episode that we discovered was $A(1{-}20) \rightarrow B(101{-}200) \rightarrow C(1{-}20)$ (which corresponds to α_1). The corresponding results are quoted in Table 10. It can be seen that α_2 and α_3 do not figure even in the top-100 frequent generalized episodes, whereas α_1 always has rank 1. Thus, by using some event-specific $\mathcal{I}$ sets, we can exercise some extra control in the frequent generalized episodes discovery process.

5.6 Analysis of GM Engine Assembly Data

In this section, we briefly describe an application of our frequent episode discovery algorithms for analyzing fault logs in an automotive engine assembly plant of GM. To our knowledge, this is the first instance of application of temporal data mining in the manufacturing domain.

The GM engine assembly plant data (see Section 5.2) is available in real time as a sequence of faults. Each record contains a machine code, a fault code, a start time, and an end time. There are two obvious ways for constituting an event sequence (as defined in Section 2) from this data: 1) by considering just the machine code of the record as the event type and 2) by considering the "machine code–fault code" combination in the record as the event type. In practice, both methods are useful. When using machine codes as the

TABLE 8
Set $\mathcal{I}$ as a Design Choice: Ranks and $(\frac{f}{f_{max}})$ Values for α_1, α_2, and α_3 (in Synthetic Data with Different Levels of Noise)

IID noise proba-bility ρ	$\mathcal{I} = \{51\text{–}100, 101\text{–}200\}$					
	α_1		α_2		α_3	
	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$
0.0	> 100	-	> 100	-	1	1.00
0.2	> 100	-	> 100	-	1	1.00
0.4	> 100	-	> 100	-	1	1.00
0.6	> 100	-	> 100	-	1	1.00
0.8	> 100	-	> 100	-	1	1.00

Set of time interval possibilities: $\mathcal{I} = \{51{-}100, 101{-}200\}$. *Data length: 50,000. Number of event types: 200.*

TABLE 10
Event-Specific $\mathcal{I}$ Sets: Ranks and $(\frac{f}{f_{max}})$ Values for α_1, α_2, and α_3 (in Synthetic Data with Different Levels of Noise)

IID noise proba-bility ρ	$\mathcal{I} = \{1\text{–}20, 51\text{–}100\}$, $\mathcal{I}_B = \{101\text{–}200\}$					
	α_1		α_2		α_3	
	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$	Rank	$\frac{f}{f_{max}}$
0.0	1	1.00	> 100	-	> 100	-
0.2	1	1.00	> 100	-	> 100	-
0.4	1	1.00	> 100	-	> 100	-
0.6	1	1.00	> 100	-	> 100	-
0.8	1	1.00	> 100	-	> 100	-

Set of time interval possibilities: $\mathcal{I}_B = \{101{-}200\}$ *for event type* B *and* $\mathcal{I} = \{1{-}20, 51{-}100\}$ *for all other event types. Data length: 50,000. Number of event types: 200.*

event types, the generalized episode discovery yields a first-level analysis, where each frequent episode discovered represents a set of stations (or machines) in which failures tend to occur in a correlated fashion. On the other hand, when "machine code-fault code" combinations constitute the event types, the output episodes indicate fault correlations at a greater level of granularity. However, since the number of event types in this latter case becomes very large, the output tends to be severely fractured and difficult to interpret. Hence, our primary approach has been to use only the machine faults as the event types and to do the higher resolution "machine code-fault code" analysis only on certain limited segments of the data.

Here, frequent serial episodes represent frequently co-occurring (and, hence, hopefully, correlated) sequence of faults and thus aid in the diagnosis of root causes for persistent problems. Like in [2], the frequent generalized episodes discovered can be used to construct rules like "β implies α," where β is a subepisode of α. The frequency threshold used for episode discovery is basically a minimum support requirement for these rules. The *rule confidence*, as always, is the ratio of the frequency of β to that of α. In particular, for each frequent episode α, we are interested in rules of the form "$\alpha^{(i)}$ implies α," where $\alpha^{(i)}$ denotes the subepisode of α obtained by dropping its ith node. We define the *score* of a frequent N-node episode α as the maximum confidence among rules of the form "$\alpha^{(i)}$ implies α" for all $i = 1, \ldots, N$. An episode's score, being simply the confidence of the best rule that it appears in, is thus a measure of its inferential power. When analyzing the GM data by using the generalized episode framework, we used both a frequency threshold and a score threshold to identify the output episodes that are finally presented to the user. This analysis, along with a method to prune the set of frequent episodes by using some heuristic knowledge provided by the plant maintenance, resulted in a useful tool, whereby a small enough set of episodes are presented to the user, and these were useful in fault analysis. As mentioned earlier, such a system is currently being used in one of the plants. We present below an example to illustrate the effectiveness of our method.

5.6.1 Alert Generation Using Frequent Episodes

One useful application of the generalized episode framework in GM's engine assembly plant is an alert generation system that indicates on a daily basis which episodes seem to be dominating the data in the recent past. This alert generation system is based on a sliding window analysis of the data. Every day, the generalized episode discovery algorithms are run for faults logged in some fixed-length history (for example, data corresponding to the immediate past one week). Based on the frequencies and scores of the episodes discovered, we use the following heuristic to generate alerts: "Whenever an episode exhibits a nondecreasing frequency trend for some period (of, say, four days), with its frequency and score above some user-defined thresholds, generate an alert on that episode." Thus, each day, the system provides the plant maintenance group with a list of alerts for the day. This list has been found very useful for online fault diagnosis and root cause analysis.

The following is an example of one such alert generated while analyzing some historic data for the period of January to April 2004. This particular alert was considered very interesting because it corresponded to a serious problem that the plant had during this time (for this entire period, our method generated only about 15 alerts). As it happened, from about the second week of March 2004, the plant was experiencing throughput problems due to repeated failure of a particular station (referred to here as A). The root cause for this was identified by the plant maintenance group only by 4 April 2004, and this was in some other station, which we call here as B. In our analysis of the data, one of the frequent episodes during March 2004 was the episode $(B \rightarrow A)$. By about the middle of March, its frequency and score have exceeded the thresholds. Also, due to its continued and increasing frequency in the successive data slices, by 20 March 2004, our system has generated an alert on this episode. This analysis is depicted in Fig. 1. The figure shows the actual dates on the x-axis and the cumulative frequency of the episode on all data slices up to that date on the y-axis. Clearly, in the figure, by using our system, the plant maintenance group could have been alerted to this fault correlation about two weeks earlier than the time by which they had actually diagnosed the root cause. This 2-week period, in the opinion of the plant maintenance group, could have contributed significantly to the throughput of the line. We note here that at the time that we carried out this analysis of historic data, we were not aware of this problem. Only when we presented the results of our analysis to the plant maintenance group did they realize that one of the alerts that we generated corresponded to this problem. The discovery of this fault correlation was an important factor that influenced the final decision by the plant to deploy our method for routine use. After the system was deployed, another interesting correlation was unearthed by the system. The plant was having problems due to a robot repeatedly going into a fault mode, from which recovery is not possible without manual intervention. Since this robot happened to be a newly installed one, there was very little experience available regarding its failure modes. Our frequent episode analysis of the plant logs was able to help the plant maintenance group diagnose the root cause for this problem.

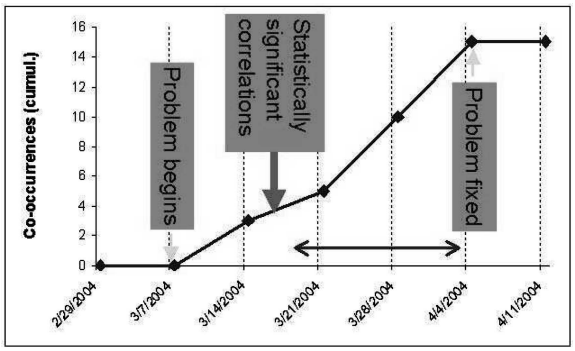

Fig. 1. Example of an interesting fault correlation discovered in historic data analysis of machine fault logs of the engine assembly plant.

6 Conclusions

In this paper, the frequent episodes framework in [2] was extended to incorporate explicit time duration constraints into the description of episodes. A new temporal pattern

called the generalized episode was defined. We developed the associated formalism and presented efficient algorithms for discovery of frequent generalized episodes in event sequences. We showed the utility and effectiveness of our algorithms through empirical studies on synthetic data. We also briefly described an application in the manufacturing domain.

For episode discovery over instantaneous event streams, the nonoverlapped-occurrences-based frequency count has an important theoretical property [4]. It facilitates a formal connection between episode discovery and learning of models for the data source from a class of models consisting of some specialized HMMs. This connection is very useful for assessing the significance of patterns discovered in the data, automatically fixing a frequency threshold, and so forth [4]. By restricting the dwelling times to take one of finitely many integer values only and considering single interval d-maps, as well as considering a model class consisting of HMM mixtures, it seems possible to derive similar theoretical results for generalized episodes also (see [12] for some details). We will address these theoretical aspects of the generalized episodes framework in the future.

ACKNOWLEDGMENTS

S. Laxman was with the Indian Institute of Science, Bangalore, while working on this paper. This research was partially funded by GM R&D Center, Warren, through the Society for Innovation and Development (SID), Indian Institute of Science (IISc), Bangalore. The authors would like to thank Basel Shadid for many helpful discussions.

REFERENCES

[1] R. Srikanth and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96),* Mar. 1996.

[2] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery,* vol. 1, no. 3, pp. 259-289, 1997.

[3] J. Han, G. Dong, and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database," *Proc. 15th Int'l Conf. Data Eng., (ICDE '99),* pp. 106-115, 1999.

[4] S. Laxman, P.S. Sastry, and K.P. Unnikrishnan, "Discovering Frequent Episodes and Learning Hidden Markov Models: A Formal Connection," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 11, pp. 1505-1517, Nov. 2005.

[5] G. Casas-Garriga, "Discovering Unbounded Episodes in Sequential Data," *Proc. Seventh European Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD '03),* pp. 83-94, 2003.

[6] M.J. Atallah, R. Gwadera, and W. Szpankowski, "Detection of Significant Sets of Episodes in Event Sequences," *Proc. Fourth IEEE Int'l Conf. Data Mining (ICDM '04),* pp. 3-10, Nov. 2004.

[7] M. Hirao, S. Inenaga, A. Shinohara, M. Takeda, and S. Arikawa, "A Practical Algorithm to Find the Best Episode Patterns," *Proc. Fourth Int'l Conf. Discovery Science (DS '01),* pp. 435-441, Nov. 2001.

[8] Z. Tronicek, "Episode Matching," *Proc. 12th Ann. Symp. Combinatorial Pattern Matching (CPM '01),* vol. 2089, pp. 143-146, July 2001.

[9] S. Laxman, P.S. Sastry, and K.P. Unnikrishnan, "Fast Algorithms for Frequent Episode Discovery in Event Sequences," *Proc. Third Workshop Mining Temporal and Sequential Data,* Aug. 2004.

[10] H. Mannila and H. Toivonen, "Discovering Generalized Episodes Using Minimal Occurrences," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining (KDD '96),* pp. 146-151, Aug. 1996.

[11] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering Frequent Closed Itemsets for Association Rules," *Proc. Seventh Int'l Conf. Database Theory (ICDT '99),* pp. 398-416, Jan. 1999.

[12] S. Laxman, "Discovering Frequent Episodes: Fast Algorithms, Connections with HMMs and Generalizations," PhD dissertation, Dept. of Electrical Eng., Indian Inst. of Science, Mar. 2006.

Srivatsan Laxman received the PhD degree in electrical engineering from the Indian Institute of Science, Bangalore, in 2006. Since April 2006, he has been a postdoctoral researcher at Microsoft Research Laboratories, Bangalore. He is currently working on learning paradigms with an adversarial or cryptographic twist. His research interests include machine learning, pattern recognition, data mining, and signal processing.

P.S. Sastry (S'82-M'85-SM'97) received the PhD degree in electrical engineering from the Indian Institute of Science, Bangalore, in 1985. Since 1986, he has been with the faculty of the Department of Electrical Engineering, Indian Institute of Science, where he is currently a professor. He has held visiting positions at the University of Massachusetts, Amherst, University of Michigan, Ann Arbor, and General Motors Research Labs, Warren, Michigan. His research interests include learning systems, pattern recognition, data mining, and image processing. He is a senior member of the IEEE.

K.P. Unnikrishnan received the PhD degree in physics (biophysics) from Syracuse University, Syracuse, New York, in 1987. He is currently a staff research scientist at the General Motors (GM) R&D Center, Warren, Michigan. Before joining GM, he was a postdoctoral member of the technical staff at the AT&T Bell Laboratories, Murray Hill, New Jersey. He has also been an adjunct assistant professor at the University of Michigan, Ann Arbor, a visiting associate at the California Institute of Technology (Caltech), Pasadena, and a visiting scientist at the Indian Institute of Science, Bangalore. His research interests include neural computation in sensory systems, correlation-based algorithms for learning and adaptation, dynamical neural networks, and temporal data mining.