Generating Diverse Realistic Data Sets for Episode Mining

Albrecht Zimmermann
albrecht.zimmermann@cs.kuleuven.be
KU Leuven

***Abstract*—Frequent episode mining has been proposed as a data mining task with the goal of recovering sequential patterns from temporal data sequences. While several episode mining approaches have been proposed in the last fifteen years, most of the developed techniques have not been evaluated on a common benchmark data set, limiting the insights gained from experimental evaluations. In particular, it is unclear how well episodes are actually being recovered, leaving an episode mining *user* without guidelines in the knowledge discovery process. One reason for this can be found in non-disclosure agreements that prevent real life data sets on which approaches have been evaluated from entering the public domain. But even easily accessible real life data sets would not allow to ascertain miners' abilities to identify underlying patterns. A solution to this problem can be seen in generating artificial data, which has the added advantage that patterns can be known, allowing to evaluate the accuracy of mined patterns. Based on insights and experiences stemming from consultations with industrial partners and work with real life data, we propose a data generator for the generation of diverse data sets that reflect realistic data characteristics. We discuss in detail which characteristics real life data can be expected to have and how our generator models them. Finally, we show that we can recreate artificial data that has been used in the literature, contrast it with real life data showing very different characteristics, and show how our generator can be used to create data with realistic characteristics.**

I. Introduction

In quite a few real life applications, data streams in continuously and is stored as a sequence of time-stamped events. Examples of such data include telecommunication status messages, web logs, sensor readings of earth quakes, sensor readings and status codes logged on assembly lines or industrial machinery, network traffic, user interface traces, and train schedules. If some of these events are considered exceptional in some way, e.g. an alarm in an industrial plant or a network traffic event that is considered an unauthorized intrusion, an obvious data mining setting consists of trying to identify patterns that appear in the time *before* these special events. Such patterns can then be used to predict the occurrence of related events in advance or to identify for instance the root causes of an alarm that have to be corrected to alleviate the situation.

Frequent episode mining has been proposed to address this issue and in the last fifteen years, numerous papers have proposed techniques for finding frequent episodes [1], [2], [3], [4], [5], [6], [7], [8].

When faced with a temporal data set in the context of an industry cooperation, we therefore turned to episode mining, and experienced a typical problem in pattern mining: a large output set, without a clear idea which of these patterns were relevant. We then turned to the literature, expecting to find guidelines but were disappointed: most of these techniques have not been evaluated on a common benchmark set, let alone a collection of benchmarks. The main reason for this is that event data that is collected in industrial settings is often covered by non-disclosure agreements (NDAs) that prohibit those data from being passed on or put into the public domain. This has prevented the emergence of collections of appropriate data sets that could fulfill the roles the UCI repository has for machine learning [9], the FIMI repository for itemset mining [10], or the UCR collection of data for time series classification and clustering [11].

Additionally, the enumeration of application settings above shows how varied data contexts are. Since most NDAs prohibit even the publication of data characteristics that could be stored together with episode mining results in reference databases [12], it is therefore unclear whether results from one application setting can be expected to transfer to the other settings. Finally, even *if* real life data with similar characteristics as the data set in question were available, the lack of knowledge about the underlying patterns in such data would make it difficult to find out which of the returned patterns are relevant.

As a solution to this problem, we propose *artificially* generating data with controlled characteristics and known hidden phenomena, as used in the single graph (network) mining [13] and SAT solving communities [14]. Such a data generator could then be used to guide knowledge discovery in the following way: generate data with similar characteristics as a real life data set under consideration, mine patterns on it, apply the insights about pattern recovery to the result from the real life data set. Such a generator should satisfy three requirements:

1) It should make it possible to verify the results of mining operations. (**Verifiability**)
2) It should be flexible, making it possible to generate diverse data sets for benchmarking. (**Diversity**)
3) It should mirror real life data characteristics. (**Realism**)

In this paper we attempt to provide such a solution. The starting point of this work is that, despite the lack of data, we can develop a data generator that reflects industrial *common sense* and practical experience and that is sufficiently general to create wide ranges of datasets matching this experience.

In the following section we recount the basics of episode mining. In Section III, we discuss related work, both in terms of episode mining and data generation. In Section IV, we discuss Laxman *et al.*'s data generator [5], critique some of the assumptions made, and propose and argue for alternatives and extensions. We also give a description of source episode and data generation. In Section V, we show characteristics of artificial data used in earlier work, contrast them with the real life data at our disposal and show how to generate data with similar characteristics as the real life data. Finally, in Section VI, we summarize our work and conclude.

II. Episode Mining

We mainly follow the notation of [1]:

Given a class $\mathcal{E}$ of event types, an event is a pair $(E, t), E \in \mathcal{E}, t \in \mathbb{N}^+$. An event sequence $\mathcal{S}$ is a triple (t_s, t_e, S) with t_s the starting time, t_e the end time, and S an ordered sequence of events:

$$\langle (E_1, t_1), \ldots, (E_m, t_m) \rangle$$

with $E_i \in \mathcal{E}$, $\forall i : t_s \leq t_i \leq t_e$ and $\forall t_i, t_j, i < j : t_i \leq t_j$, e.g.:

$(1, 141, \langle (E, 1), (A, 12), (B, 15), (C, 25), (D, 26), (A, 36), (B, 38), (C, 55), (E, 66), (D, 75), (A, 94), (E, 109), (B, 124), (C, 131), (D, 141) \rangle)$

An episode $\epsilon = (V_\epsilon, \trianglelefteq_\epsilon, g_\epsilon)$ is a set of nodes V_ϵ, a partial order $\trianglelefteq_\epsilon$ on V_ϵ and a mapping $g_\epsilon : V_\epsilon \mapsto \mathcal{E}$ associating each node with an event type. If the order is a total order, ϵ is called *serial*, if there is no ordering at all, *parallel*, if both orders are allowed in episodes, they are referred to as *general*. An episode ϵ is said to *occur* in an event sequence $\mathcal{S}$ at interval $[l, u]$ if the events to which V_ϵ are mapped occur in that interval in the same order as they occur in the episode.

Example 1: An episode occurring in the event sequence above would be $(\{v_1, v_2\}, \{v_1 \lhd v_2\}, \{v_1 \mapsto A, v_2 \mapsto B\})$. *It occurs, e.g., in* $[94, 131]$. *If there is no repetition of events in an episode, we fold the mapping of nodes to event types into the set of vertices itself.*

A *window* on $\mathcal{S}$ is an event sequence $\mathcal{W} = \langle t_{wb}, t_{we}, W \rangle$ with $t_s \leq t_{wb} \leq t_{we} \leq t_e$ and

$$W = \{(E_i, t_i) \in S \mid t_{wb} \leq t_i \leq t_{we}\}.$$

A window is said to be of size $w = t_{we} - t_{wb} + 1$. Given $\mathcal{S}$ and w we can define the set of all windows of size w on $\mathcal{S}$: $a_w(\mathcal{S})$

Example 2: If we assume a maximal window size 15*, the first five windows were in order* $[1, 15]$, $[2, 16]$, $[3, 17]$, *[4,18],* $[5, 19]$ *comprising the episodes:* $\langle E, A, B \rangle$, $\langle A, B \rangle$, $\langle A, B \rangle$, $\langle A, B \rangle$, $\langle A, B \rangle$.

Given a fixed window size w, the *frequency* of ϵ is the number of fixed-size windows on $\mathcal{S}$ in which it occurs:

$$\text{freq}(\epsilon, \mathcal{S}, w) = |\{\mathcal{W} \in a_w(\mathcal{S}) \mid \epsilon \text{ occurs in } \mathcal{W}\}|$$

So-called *windows-based* episode mining (WinEpi) [1] approaches require the user to specify a window size and a frequency threshold.

By the same authors, an alternative frequency definition has been proposed, that of minimal occurrences. A *minimal* occurrence of ϵ is an interval $[l, u]$ at which ϵ occurs, and for which no proper sub-interval $[l', u'] \subset [l, u]$ exists such that ϵ occurs at $[l', u']$. The frequency definition in this case changes to the number of minimal occurrences of ϵ (MinEpi [2]).

Example 3: A minimal occurrence of $(\{A, B\}, \{A \lhd B\})$, *for instance, is* $[12, 15]$, *and of* $(\{B, C, D\}, \{B \lhd C, B \lhd D, C \lhd D\})$ $[15, 26]$. *The frequency definition based on the number of minimal occurrences of* ϵ*, e.g.,* $freq((\{A, B\}, \{A \lhd B\})) = |\{[12, 15], [36, 38], [94, 124]\}| = 3$*, gives an arguably more intuitive count.*

As an improvement of the minimal occurrence semantic, non-overlapping occurrence counts have been proposed [5], [6], while [3], [4] aimed to move away from the fixed nature of maximal windows and the problems it entails by proposing frequency formulations based on *inter-event* time-gap constraints.

III. Related Work

A recent overview of temporal pattern mining techniques can be found in [15]. Early work in the field [1], [2], [16], [3], [4] used real life data for which the ground truth was not known, augmented with non-temporal sequential data, i.e. text or protein sequences. Non-temporal data has the characteristic that only the *order* in which elements occur in the episode is relevant, not the time delay between events. Most of the real life data is not publicly available, and with the exception of [16], algorithms have not been compared to each other on those data.

More recent works [5], [6], [7], [8] have used artificial data to experimentally validate their proposed approaches in addition to real life data. Laxman et al. [5] proposed a generative model based on HMMs, which assumes uniform noise distribution. We will discuss their generator and the underlying assumptions in more detail in the next section. Tatti *et al.* [6], [8] used artificial data corresponding to extreme cases to demonstrate the superiority of their technique in those contexts.

Artificial data generation has been explored in more depth in related fields. In the context of classification experiments in stream data with concept drift, [17] generated data by generating three-dimensional data points whose classification is chosen based on the sum of the first two dimensions compared against a threshold value. Class noise can be added. The STAGGER concept generator was introduced in [18]. It generates instances described by three nominal attributes, having two class labels. Class noise can be added. Bifet *et al.* [19] used the tree generator introduced in [20] by generating one source tree for each class and adding label drift during data generation. Classification in data streams is clearly different from episode mining – most importantly data points can be expected to satisfy i.i.d. assumptions to a certain degree. Nevertheless, as we will lay out in the next section, both the concepts of class noise, and of drift correspond to aspects of realistic data generation.

Mueen *et al.* [21] used a random walk generator to produce large sets of numerical time series data for the purpose of testing time series motif discovery algorithms. From the field of process control comes the Tennessee Eastman generator first described in [22], a complex generative model relating numerical values.

IV. Data Characteristics and Generation

The considerations outlined in the introduction lead us to believe that there is need for a data generator that is capable of mimicking real life data to aid with episode mining use, and facilitate improvements in episode mining research. Unfortunately, as we explained, the lack of real life data or even *characterizations* of real life data sets makes it difficult to come up with an unambiguous prescription of how artificial data should look.

However, consultations with our industrial partners in the context of a project allowed us to collect "common sense" assumptions and practical experience with data and the systems generating them. Specific properties of event sequences in an industrial context are that

- P1 time stamp information is important – events that occur with large delay are unlikely to be related
- P2 events can be missing – sensors may fail, or network connections be interrupted, for instance
- P3 events can repeat within a sequence – several threshold violations could be needed before an alarm is triggered
- P4 events can occur in more than one sequence – too high and too low pressure could be detected by the same sensor but lead to different alarm events
- P5 machinery can be in different states – and therefore generate different episodes at different times in the life or production cycle

We use this information to guide our design decisions: in the next section we quickly summarize the data generator proposed in [5], contrasting its underlying assumptions with the real-world insights listed above (Section IV-B), before we discuss the plausibility of used distributions in Section IV-C.

A. The HMM-based generator

To the best of our knowledge, the HMM-based generative model proposed in [5] is the so far most comprehensive generator for artificial episode mining data. A very attractive feature of this generator is that it first generates *source* episodes which it then embeds in the data sequence. Any mining results could therefore be compared to those source episodes to gauge their accuracy, fulfilling the verifiability requirement. The parameters of the model encompass:

- The noise probability parameter ($p \in [0,1]$), i.e. the probability that events do not belong to an embedded episode. The authors report on experiments for $p \in \{0, 0.2, 0.3, 0.4, 0.5\}$.

Example 4: It is important to realize that the noise probability is applied per event*. To give an example, let an embedded episode take the form $A \to B \to C \to D$. A noise probability of 0.2 would result in 1 noise event per 5 events, i.e. 1 noise event and 1 episode, on average in the data, e.g.: $\mathbf{E}, A, B, C, D, A, B, C, \mathbf{E}, D, A, B, \mathbf{E}, C, D$ (time-stamps omitted). A noise probability $p = 0.4$ would result in 2 noise events per 5 events on average while conversely $p = 0.1$ results in only a single noise event per 10 events etc.*

Any events that are not the result of episodic behavior are expected to arise independently which makes event-wise noise probability a plausible choice for modeling.
Additional parameters of the HMM generator are:

- The number of source episodes of which instances will be embedded in the data (n).
- The length of source episodes (N).
- The size of $\mathcal{E}$ (M).
- The total length of the data sequence (T).

Artificial data generated by this model can therefore be characterized by a tuple $\langle p, n, N, M, T\rangle$ with domain $\langle [0,1], \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+\rangle$.

B. Extending the generator to fit real-world experiences

The HMM model makes a number of explicit and implicit assumptions about data characteristics that are invariable, violating the diversity requirement, in addition to the realism requirement, as we will argue. First and foremost, the authors consider time stamp information itself irrelevant:

> "the actual values of event times are not important. The event times are used only to order events and only this ordering is needed to count episode occurrences. Thus, when analyzing the frequent episode discovery process, it is enough to consider a model which generates an ordered sequence of event types."

As a result, time stamps, with which events are annotated, are incremented by a "small random integer" every time an event is created. No information is given about the interval from which this integer is sampled and the sampling process. On the one hand, this collides with practical property P1. On the other hand, a larger interval from which to sample delays would also lead to more variability, allowing to test a wider range of data characteristics. We therefore add a parameter controlling

- the explicit *maximum* delay between any two successive events ($g \in \mathbb{N}^+$). In the default setting time delays are sampled uniformly from the interval $[1, g]$.

Example 5: For a maximal delay of $g = 20$, the data sequence in the preceding example ($p = 0.2$) might for instance take the form: $(\mathbf{E}, 1)$, $(A, 12)$, $(B, 15)$, $(C, 25)$, $(D, 26)$, $(A, 36)$, $(B, 38)$, $(C, 55)$, $(\mathbf{E}, 66)$, $(D, 75)$, $(A, 94)$, $(\mathbf{E}, 109)$, $(B, 124)$, $(C, 131)$, $(D, 141)$.

Also, in viewing only the order of events as relevant, Laxman *et al.* view the delay between any two successive events in an embedded episode as unconstrained. In real world phenomena one would expect a temporal correlation of events generated by the same process, and we therefore add an additional parameter to the model determining

- whether or not to enforce that any two successive events of a *source episode* have at most a time delay of g when embedded in the data ($h \in \{true, false\}$).

Example 6: To illustrate the effect that unconstrained time delays have, consider a setting with $g = 20$, $p = 0.2$, and an embedded episode of length 4. Uniform sampling over $[1, 20]$ will lead to an average delay of 10 between successive events. For $p = 0.2$, this leads to an average episode duration of 40. For $p = 0.4$, however, the average duration will increase to 67. This means that the probability of noise *affects the* duration of the signal *in the unconstrained model.*

Laxman *et al.* also make two assumptions about the events involved in source episodes: a) event types *do not* repeat within a single episode and b) different source episodes *do not* share any event types, in contrast to P3 and P4. This is not necessarily the case in real life data:

Example 7: A sensor reading could, e.g., exceed a certain safety value but depending on the system, it might take several such events for an alarm *to be triggered, leading to an embedded episode with repeating event types. Similarly, a given warning might be caused by any of a* variety *of sensors exceeding a safety threshold, causing several episodes to share the last event type.*

These considerations lead us to extend the model additionally by parameters governing

- whether or not event types repeat in a single source episodes ($r \in \{true, false\}$).
- whether or not different source episodes share event types ($s \in \{true, false\}$).

The data in [5] is generated by interleaving the embedded episodes randomly. While this not an unrealistic assumption, interleaving episodes can be expected to make it much harder for mining techniques to recover underlying patterns since false occurrences of regularity can be detected. For the sake of flexibility, we therefore add a parameter controlling

- whether embeddings of the same source episode can interleave ($i \in \{true, false\}$).

The HMM model also embeds source episodes concurrently. In contrast to this, a system, e.g. a production machine, could be in different *stages*, e.g. peak performance, deterioration, and near breakdown, with different episodes generated in each stage (property P5). This is similar to the concept drift explored in data streams. We therefore add an additional parameter that affects

- whether source episodes are embedded successively or not, i.e. concurrently ($S \in \{true, false\}$).

Finally, as mentioned in point P2, our experiences with real world data lead us to add a failure probability parameter modeling:

- whether information that *should* be logged, such as a sensor reading, is in fact not logged ($o \in [0, 1]$). This parameter is arguably related to the concept of "class noise" in classification in data streams.

So far, we have only extended the generator with a number of additional parameters, bringing it much more in line with our (and others') experiences with real-world data. Yet, the data generator we propose will already allow to generate data sets of a much larger variety that can be described by the tuple $\langle p, n, N, M, T, g, o, h, r, s, i, S\rangle$ with domain $\langle [0, 1], \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+, [0, 1], \{t, f\}, \{t, f\}, \{t, f\}, \{t, f\}, \{t, f\}\rangle$.

C. Adding realistic distributions

Among the assumptions made by Laxman *et al.* and others are uniform distributions for noise events and distinct source episodes. Deciding on appropriate distributions is more difficult than introducing parameters allowing for real-world like behavior. On the one hand we would posit, however, that uniform distributions are not common in real-world phenomena, generally speaking, with, e.g., normal or Poisson distributions more common. On the other hand, empirical distributions observed in the data at our disposal appear to be normally or Poisson generated, as we will illustrate in Section V.

Arguably the first of the choices that can be questioned is the assumption that all source episodes arise with the same probability. In a real life system, some phenomena are more likely to occur than others, with resulting effects on the probability of recovering the patterns. We therefore make this distinction explicit by adding a parameter that controls

- whether or not source episodes have different weights ($W \in \{true, false\}$).

Also, while noise events can be expected to arise independently from each other, assuming a uniform distribution is rather restrictive since individual noise events can have different probabilities. The data at our disposal shows event types that are not normally distributed. Additionally, this difference will have direct effects on any episode mining operation since uniformly distributed noise should look very different than recurrent phenomena whereas, for instance, poisson-distributed noise can give the appearance of regularity. We therefore add a parameter governing

- whether noise is normally or Poisson-distributed ($P \in \{u, p\}$).

Finally, a similar argument can be made w.r.t. time delays between events: the time delays we observed show a distribution resembling a normal distribution and normal distributions can be expected to occur in realistic data:

Example 8: Continuous sensor readings might overwhelm the data storage of a production machine. If sensor readings are taken every minute and an event is generated if readings have changed by at least a certain amount, all events can be expected to occur multiples of sixty seconds apart.

We therefore add parameters affecting the delay distribution:

- for noise ($d \in \{u, n\}$), and
- embedded episodes ($D \in \{u, n\}$).

Since it could happen that, for instance, noise has normally-distributed delays while episode events appear with uniformly distributed time delays, we do not want to supersede g, and instead introduce a parameter G that determines the base mean of the normal distribution (variance is set to $G/10$). To allow for more complex delay distributions, we assume that

the actual distribution is a mixture of m normal distributions with means $G, \ldots, m \cdot G$, respectively. The parameters of our generator are shown in Table I, together with default values.

D. Data Generation

In this section, we outline how the different parameter settings affect data generation algorithmically. We begin by giving pseudocode of the algorithm generating source episodes (Algorithm 1). Since the algorithm for generating the data itself consists mainly of a number of nested *if-then* statements, we abstain from pseudocode and instead give a natural language description.

a) Generation of source episodes: As a first step in data generation, we generate the source episode(s). For each of the N event types per episode we sample uniformly from $\mathcal{E}$. If $r =$ *false*, we reject the event if it already occurs in the same source episode, if $s =$ *false*, we reject it if it occurs in a source episode that has been generated already. If $r =$ *true* and there has been no repetition on reaching the Nth element of an episode, we sample from the already involved event types of this episode. If $s =$ *true* and there has been no shared event type on reaching the Nth element of an episode, we sample from the event types of already generated source episodes. Finally, if $W =$ *true*, we sample a weight for the episode uniformly from $(0, 1]$, and normalize the weights in the end so that they sum to 1.

b) Data set generation: For the actual data set, we start with a time stamp $t = 1$ and $\mathcal{S} = \emptyset$. As long as $|\mathcal{S}| < T$, we generate a noise event with probability p, sampled according to P from $\mathcal{E}$, at time stamp t.

With probability $1 - p$, an episode event is generated. If $S =$ *true*, source episode ϵ_i becomes embedded as long as the event count $M \cdot \sum_{j=1}^{i-1} weight_j < M' \leq M \cdot \sum_{j=1}^{i} weight_j$, otherwise all source episodes are embedded concurrently along the entire length of the data sequence. In the latter case, if $i =$ *false*, one of the n source episodes is chosen randomly (with equal probality if $W =$ *false*, weighted otherwise). If a partially embedded instance of that episode exists, the next event type is read from it, otherwise a new embedding started.

If $i =$ *true*, let $\mathcal{P}_E$ be the set of partial embeddings. A random integer value from $[1, |\mathcal{P}_E|+1]$ is chosen, with $|\mathcal{P}_E|+1$ corresponding to starting a new embedding (subject to the constraints imposed by S).

Finally, t is increased with a random value $\delta \in [1, g]$, if the delay distribution for the type of event (noise, or otherwise) is uniform. Otherwise, the normal distribution to be used is chosen by sampling from a geometric distribution with mean $1/m$, and δ sampled from the normal distribution with the respective mean. If $h = true$ and there is any partial embedding whose last event occurred at time stamp $t_e : t + \delta - t_e > g$ ($t_e : t + \delta - t_e > m \cdot G$) for a uniform (normal) distribution, an event is generated from that partial embedding at $t_e + g$ ($t_e + (m \cdot G)$).[1]

[1]We have implemented the data generator in Java and it is available for download at people.cs.kuleuven.be/~albrecht.zimmermann/software.html.

Algorithm 1 Source episode generation

source = $\emptyset$
for $1 \leq i \leq n$ **do**
  $\epsilon_i = (V_{\epsilon_i} = \emptyset, \trianglelefteq_{\epsilon_i} = \emptyset, g_{\epsilon_i} = \emptyset)$
  repeated = false
  shared = false
  for $1 \leq j \leq N$ **do**
    while E not accepted **do**
      E sampled uniformly from $\mathcal{E}$
      if r = false $\wedge \exists v \in V_\epsilon : v \mapsto E \in g_\epsilon$
        reject E
      else if r = true $\wedge \exists v \in V_\epsilon : v \mapsto E \in g_\epsilon$
        accept E
        repeated = true
      else if r = true $\wedge$ repeated = false $\wedge$ j = N
        sample E uniformly from $g_{\epsilon_i}(V_{\epsilon_i})$
        accept E
      if s = false $\wedge \exists v \in V_{\epsilon_{i'}}, i' < i : v \mapsto E \in g_{\epsilon_{i'}}$
        reject E
      else if s = true $\wedge \exists v \in V_{\epsilon_{i'}}, i' < i : v \mapsto E \in g_{\epsilon_{i'}}$
        accept E
        shared = true
      else if s = true $\wedge$ shared=false $\wedge j = N$
        sample E uniformly from $\bigcup_{i'<i} g_{\epsilon_{i'}}(V_{\epsilon_{i'}})$
        accept E
    $V_{\epsilon_i} = V_{\epsilon_i} \cup v_j$
    if $j > 1$
      $\trianglelefteq_{\epsilon_i} = \trianglelefteq_{\epsilon_i} \cup \{v_{j-1} \trianglelefteq v_j\}$
    $g_{\epsilon_i} = g_{\epsilon_i} \cup \{v_j \mapsto E\}$
  source = source $\cup \epsilon_i$
  Sample $weight_i$
for $1 \leq i \leq n$ **do**
  $weight_i = \frac{weight_i}{\sum_{j=1}^{n} weight_j}$

V. Data examples

As the preceding section shows, the proposed generator satisfies the first two requirements: 1) the embedding of source episodes allows the verification of episode mining results, and 2) a number of numerical and nominal parameters give it the flexibility to generate diverse data sets. We have also argued that the third requirement – realism – is fulfilled. In this section, we add empirical support that our generator satisfies the realism requirement. We begin by generating data consistent with the assumptions made in [5], [8] and discuss their characteristics in the context of the respective papers' underlying motivation. We then contrast those data with the real life data sets at our disposal and outline how the generator can be used to generate data with similar characteristics, demonstrating its flexibility along the way.

A. Effect of noise probability and distribution on event type distributions

The first two figures show the event type distribution of data generated using settings used in [5], i.e. $n = 2$, $p = 0.3$, $i =$

Numerical parameters (default values)		Nominal parameters (default settings)	
p – probability of noise events	(0.2)	$P \in \{u, p\}$ – noise is uniformly or Poisson distributed	(u)
N – length of source episodes	(4)	$r \in$ {t,f} – event types repeat in source episodes	(f)
M – size of the alphabet of event types	(20)	$s \in$ {t,f} – event types are shared among source episodes	(f)
n – number of source episode	(1)	$i \in$ {t,f} – embeddings of source episodes can interleave	(f)
T – total number of events in the data	(5000)	$W \in$ {t,f} – source episodes have different probabilities	(f)
g – maximal delay between any two successive events	(20)	$h \in$ {t,f} – delays between successive events of a source episode $\leq g$ ($\leq m \cdot G$)	(f)
o – probability that a source episode's event is *not* embedded	(0.0)	$S \in$ {t,f} – source episodes are embedded successively	(f)
G – "base mean" of a normal distributions (variance $G/10$)	(300)	$d \in \{u, n\}$ – noise delays are uniformly or normally distributed	(u)
m – number of normals to mix, with means $G, 2G, \ldots, m \cdot M$	(1)	$D \in \{u, n\}$ – embedding delays are uniformly or normally distributed	(u)

TABLE I
THE PARAMETERS OF THE DATA GENERATOR WITH DEFAULT SETTINGS IN PARENTHESES

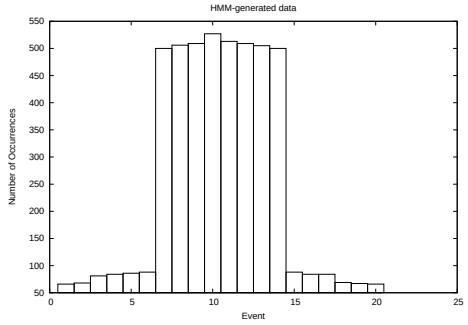

Fig. 1. Event type distribution of data generated by the HMM generator

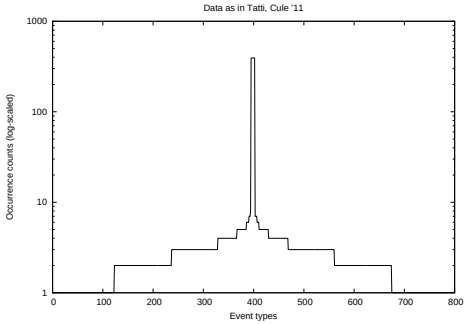

Fig. 2. Event type distribution of data generated using a very large alphabet

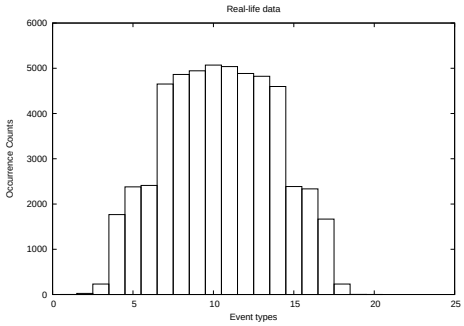

Fig. 3. Event type distribution in a example data set of our real life data

true, $h =$ *false*, $P = u$, and [8] ($N = 8$, $p = 0.38$, $P = u$). As we mentioned in passing in Section III, these are specific data sets generated to demonstrate superiority of the proposed methods in particular circumstances.

In the first case (Figure 1), the event type distribution on the one hand shows strongly expressed events involved in embedded episodes, making recovering something related to the underlying patterns relatively easy. On the other hand, there are only few noise events so that overlap and close proximity of embeddings can lead to the "discovery" of patterns that are different from the source episodes. This is therefore data set well-suited to demonstrating the superiority of non-overlapping occurrence counts over traditional methods. In addition, we assume that the General Motors data Laxman *et al.* worked with showed characteristics that could be generated using uniform distributions. Tatti *et al.* proposed mining closed episodes [8]. Drawing noise labels from a very large alphabet means clearly dominating embedded pattern (and combinations and permutations of it), as Figure 2 show, leading to an extremely large search space that would overwhelm less sophisticated techniques. These two data sets are rather specific and not necessarily representative of real life data. To illustrate this further, consider Figure 3, showing an example of the event type distribution of real life data we have worked with, looking rather different.

To motivate our design decisions, we first attempt to approximate the distribution using uniform noise and episode distributions (Figure 4). This data has been generated using uniform noise distribution, $n = 2$, $N = 4$, and $p = 0.7$, much higher than in the artificial data we just discussed. The distribution has more in common with the real life data than with the artificial data sets but does not fit yet. Alternatively, we can use source episodes having different weights, but continue generating noise that is uniformly distributed. The event type distribution of data generated in this way ($n = 3$, $p = 0.2$) can be seen in Figure 5. We can see that the abrupt changes in event type occurrence counts that can be observed in data that uses only one source episode, or several source episodes of equal probability, are becoming less pronounced. Finally, Figure 6 shows the event type distribution that occurs if we use poisson-distributed noise instead of uniformly distributed noise. This data looks like the real life data at our proposal, supporting our design decisions.

B. Parameter effects on time delay distribution

The second characteristic that we can use to describe a data sequence compactly is the distribution of the time delays between adjacent events in the data. Figure 7 shows the

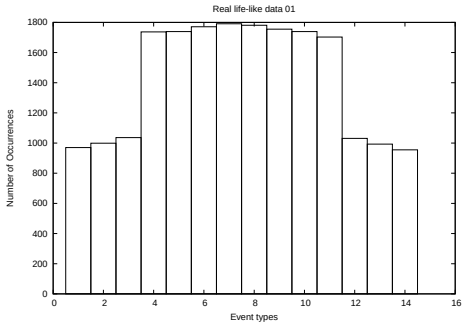

Fig. 4. Approximated real life like event type distribution using uniformly distributed noise, and source episodes having equal probability

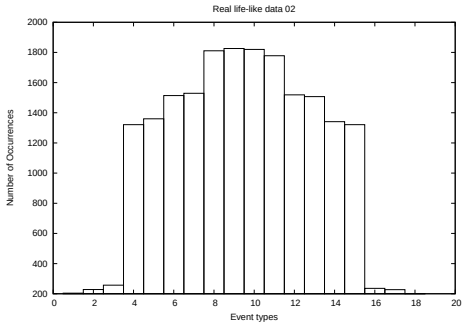

Fig. 5. Approximated real life like event type distribution using uniformly distributed noise, and source episodes having different probability

distribution of delays that results if we use the HMM generator with $g = 20$.

Time delays are roughly uniformly distributed as per the parameter used in generating them, and can therefore be considered uninformative, in keeping with Laxman *et al.*'s view of time information. If, however, the time delays of embedded episodes are independent of the amount and timestamps of noise that occurred between episode events, i.e. if h is set to *true*, the time delay distribution changes considerably (Figure 8), which can be expected to have an effect when windows- or maximal-gap constraints are used in mining.

Once again, this difference in delay distributions will not affect all episode mining approaches, in particularly if no or very lenient time constraints are used. To be able to evaluate the effect such differences will have, however, it is necessary

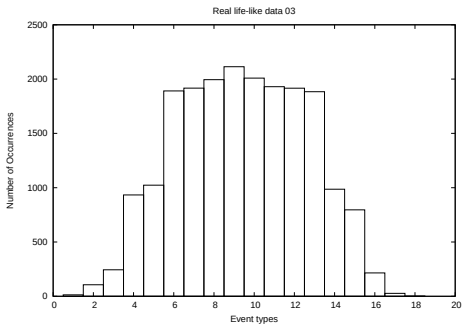

Fig. 6. Approximated real life like event type distribution using Poisson distributed noise, and source episodes having different probability

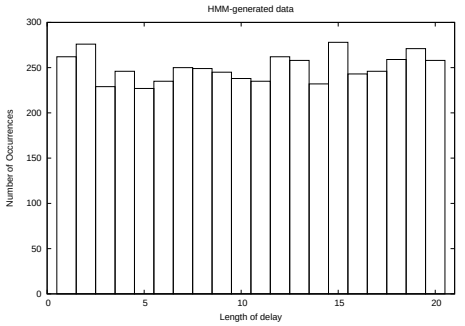

Fig. 7. HMM generator time delay distribution (unconstrained)

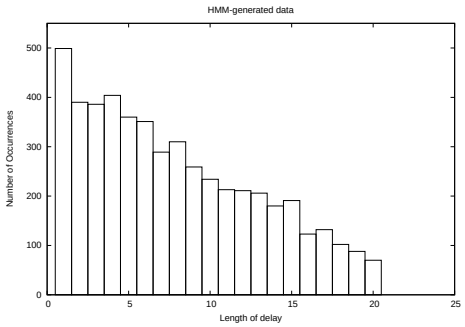

Fig. 8. HMM generator time delay distribution (constrained)

to be able to specify different temporal behavior in the data. This includes changing the distribution of time delay values. As Figure 9 shows, the real life data we have at our disposal shows a time delay distribution that is far different from a uniform distribution.[2]

By setting the delay distributions to normal distributions, $m = 15$, $G = 300$, we can achieve a similar looking distribution, as can be seen in Figure 10. This means that at this point, we can generate data with similar characteristics to the real life data at our disposal.

C. Discussion

As we have shown, our generator is capable of generating data with similar characteristics as our real life data and

[2]The figure shows only the first three peaks – the entire range of time stamps reaches into the hundreds of thousands, with increasingly lower peaks up to 4500

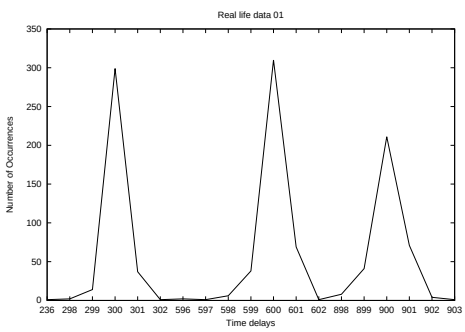

Fig. 9. First section of the time delay distribution in the first example data set of our real life data

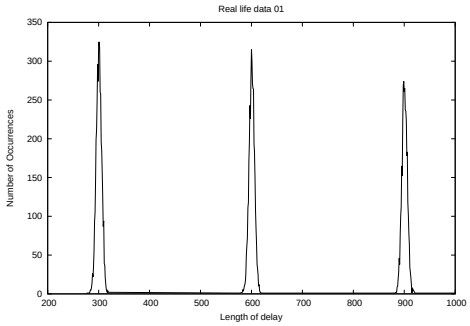

Fig. 10. First section of real life like time delay distribution

could therefore guide us in knowledge discovery.There are several parameters whose effect will not be obvious from event type and delay distributions but that can be expected to affect effectiveness and efficiency of mining operations. Whether several source episodes are embedded concurrently or successively, for instance, will not lead to changes in the event type distribution but a mining algorithm that is confronted by several overlapping regularities will have a harder time recovering them than if they can be identified one after the other.

We believe that the assumptions we made in defining parameters will allow others to simulate their real life data in a similar way, making episode mining use easier. In addition, generating data with a wide range of characteristics would allow to extensively benchmark episode mining approaches, which has not been done so far.

VI. Summary and Conclusion

In this work, we have proposed a data generator capable of generating real life like data sequences exhibiting a wide range of characteristics. An important feature of the generator is that the embedding of distinct source patterns makes it possible to compare the result of mining operations with the underlying phenomena. One way of using the generator for verifying and evaluating results from real life data would consist of generating data with similar characteristics as the data set in question, mining patterns, and gaining insights into pattern recovery. To support this, our generator is very flexible, generating data governed by a variety of parameters. In addition to adjusting the amount of noise, the number and length of source episodes, alphabet of possible event types, and length time delays, our generator has a number of additional parameters that govern how source episodes interact with each other, and how likely their embeddings are to be corrupted. Finally, the generator includes different distributions governing noise events and time delays.

We have also shown that we are capable of recreating data both with characteristics similar to challenging artificial data used in existing work on episode mining, and with characteristics in line with real life data sets we have worked with.

We have also performed an empirical evaluation on several episode mining approaches, using data generated by the generator proposed in this paper. Most interestingly, we find that even for relatively easy settings, source episodes rank at best among the top-10 results of a mining operation, and that for real life like settings recovery of source episodes is not very successful.

Acknowledgment

References

[1] H. Mannila and H. Toivonen, "Discovering frequent episodes in sequences," in *KDD'95*. AAAI Press, 1995, pp. 210–215.

[2] ——, "Discovering generalized episodes using minimal occurrences," in *KDD'96*. AAAI Press, 1996, pp. 146–151.

[3] G. Casas-Garriga, "Discovering unbounded episodes in sequential data," in *PKDD*, ser. LNCS, N. Lavrac, D. Gamberger, H. Blockeel, and L. Todorovski, Eds., vol. 2838. Springer, 2003, pp. 83–94.

[4] N. Méger and C. Rigotti, "Constraint-based mining of episode rules and optimal window sizes," in *PKDD*, ser. LNCS, J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, Eds., vol. 3202. Springer, 2004, pp. 313–324.

[5] S. Laxman, P. S. Sastry, and K. P. Unnikrishnan, "Discovering frequent episodes and learning hidden markov models: A formal connection," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1505–1517, 2005.

[6] N. Tatti, "Significance of episodes based on minimal windows," in *ICDM*, W. Wang, H. Kargupta, S. Ranka, P. S. Yu, and X. Wu, Eds. IEEE Computer Society, 2009, pp. 513–522.

[7] N. Tatti and B. Cule, "Mining closed strict episodes," in *ICDM*, G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, and X. Wu, Eds. IEEE Computer Society, 2010, pp. 501–510.

[8] ——, "Mining closed episodes with simultaneous events," in *KDD*, C. Apté, J. Ghosh, and P. Smyth, Eds. ACM, 2011, pp. 1172–1180.

[9] C. Blake and C. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: \url{http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html}

[10] B. Goethals and M. J. Zaki, "Advances in frequent itemset mining implementations: report on fimi'03," *SIGKDD Explorations*, vol. 6, no. 1, pp. 109–117, 2004.

[11] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, and C. A. Ratanamahatana, "The ucr time series classification/clustering homepage," 2011.

[12] H. Blockeel and J. Vanschoren, "Experiment databases: Towards an improved experimental methodology in machine learning," in *PKDD*, ser. LNCS, J. N. Kok, J. Koronacki, R. L. de Mántaras, S. Matwin, D. Mladenic, and A. Skowron, Eds., vol. 4702. Springer, 2007, pp. 6–17.

[13] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, generators, and algorithms," *ACM Comput. Surv.*, vol. 38, no. 1, 2006.

[14] D. M. Pennock and Q. F. Stout, "Exploiting a theory of phase transitions in three-satisfiability problems," in *AAAI/IAAI, Vol. 1*, 1996, pp. 253–258.

[15] S. Laxman and P. S. Sastry, "A survey of temporal data mining," *Sadhana-academy Proceedings in Engineering Sciences*, vol. 31, pp. 173–198, Apr. 2006.

[16] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Min. Knowl. Discov.*, vol. 1, no. 3, pp. 259–289, 1997.

[17] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *KDD*, 2001, pp. 377–382.

[18] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Machine Learning*, vol. 1, no. 3, pp. 317–354, 1986.

[19] A. Bifet and R. Gavaldà, "Adaptive xml tree classification on evolving data streams," in *ECML/PKDD (1)*, ser. LNCS, W. L. Buntine, M. Grobelnik, D. Mladenic, and J. Shawe-Taylor, Eds., vol. 5781. Springer, 2009, pp. 147–162.

[20] M. J. Zaki, "Efficiently mining frequent trees in a forest," in *KDD*. ACM, 2002, pp. 71–80.

[21] A. Mueen, E. J. Keogh, Q. Zhu, S. Cash, and M. B. Westover, "Exact discovery of time series motifs," in *SDM*. SIAM, 2009, pp. 473–484.

[22] E. F. V. J. J. Down, "A plant-wide industrial process control problem," *Computers & Chemical Engineering*, vol. 17, no. 3, pp. 245–255, March 1993.