# Understanding episode mining techniques: benchmarking on diverse, realistic, artificial data

Albrecht Zimmermann
DTAI Lab, Computer Science Department, KU Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
Email: albrecht.zimmermann@cs.kuleuven.be
Tel.-Nr: +3216327823

March 14, 2013

**Abstract**

Frequent episode mining has been proposed as a data mining task for recovering sequential patterns from temporal data sequences and several approaches have been introduced over the last fifteen years. These techniques have however never been compared against each other in a large scale comparison, mainly because the existing real life data is prevented from entering the public domain by non-disclosure agreements. We perform such a comparison for the first time. To get around the problem of proprietary data, we employ a data generator based on a number of real life observations and capable of generating data that mimics real life data at our disposal. Artificial data offers the additional advantage that the underlying patterns are known, which is typically not the case for real life data. Thus, we can evaluate for the first time the ability of mining approaches to recover patterns that are embedded in noise. Our experiments indicate that temporal constraints are more important in affecting the effectiveness of episode mining than occurrence semantics. They also indicate that recovering underlying patterns when several phenomena are present at the same time is rather difficult and that there is need to develop better significance measures and techniques for dealing with sets of episodes.

**pattern mining, episode mining, data generation, quality evaluation**

## 1 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 [18, 19, 9, 21, 17, 26, 27, 28].

A major problem is that 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 [7], the FIMI repository for itemset mining [13], or the UCR collection of data for time series classification and clustering [15].

Given this limited access to data, an extensive experimental evaluation on a large range of real life datasets is practically impossible, and most papers resort to comparisons on textual data or on one real life (often industrial) dataset. This in turn has its own pitfalls: On the one hand, textual data does not have a temporal aspect and it is unclear whether the generating process (grammar) has anything in common with generating processes in natural or industrial settings. On the other hand, 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 [8], 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 result of this, after 15 years of episode mining research, two important question remain unanswered:

Q1 Do episode mining approaches recover the patterns underlying the data at all?

Q2 What are the effects of data characteristics on the effectiveness of different episode mining approaches?

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 [10] and SAT solving communities [23]. Using such data would allow the kind of large scale comparisons that have been missing from the literature so far. Furthermore, they can be used to guide knowledge discovery by generating data with similar characteristics as a real life data set under consideration, mining patterns on it, and then applying the insights about pattern recovery to the result from the real life data set. To fulfill this purpose, the 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**)

Our first contribution is a data generator fulfilling these criteria, which we describe in the first part of the paper. The starting point of our 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.

The second contribution is that once we have our data generator, we use it to answer the two questions posed above, by evaluating a number of well-established episode mining techniques on data with a wide range of characteristics. Furthermore, we perform the kind of knowledge discovery guidance outlined above, by demonstrating how to manipulate the generator to generate data in line with real life characteristics, and then using this data to evaluate the capability of the episode mining techniques to recover the underlying patterns.

In the following section we recount the basics of episode mining. In Section 3, we discuss related work, both in terms of episode mining and data generation. In Section 4, we discuss Laxman *et al.*'s data generator [17], 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 5, 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 at our disposal. Using the generator to generate data sets with a variety of different characteristics, we compare several well-established episode mining techniques and identify challenging (and easy) data characteristics in Section 6. Following that, we show the performance of those techniques on the artificial data that mimics our own, in particular the ability to recover the underlying patterns (Section 6.5). Finally, in Section 7, we summarize our work and conclude.

## 2 Episode Mining

We mainly follow the notation of [18]:

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

An episode $\epsilon = (V_\epsilon, \trianglelefteq_\epsilon, g_\epsilon)$ is a set of nodes $V_\epsilon$, a partial order $\trianglelefteq_\epsilon$ on $V_\epsilon$ and a mapping $g_\epsilon : V_\epsilon \mapsto \mathcal{E}$ associating each node with an event type. If the order is a total order $\triangleleft$, $\epsilon$ 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 $\epsilon$ is said to *occur* in an event sequence $\mathcal{S}$ at interval $[l, u]$ if the events to which $V_\epsilon$ 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* $(\{v_1, v_2\}, \{v_1 \triangleleft v_2\}, \{v_1 \mapsto A, v_2 \mapsto B\})$. *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 $\epsilon$ is the number of fixed-size windows on $\mathcal{S}$ in which it occurs:

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

The total frequency of $A \triangleleft B$ for a maximal window size 15 would therefore be 25, with the first $A$ and $B$ contributing 12 counts, the second ones contributing 13. So-called *windows-based* episode mining (WinEpi) [18] 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 $\epsilon$ is an interval $[l, u]$ at which $\epsilon$ occurs, and for which no proper sub-interval $[l', u'] \subset [l, u]$ exists such that $\epsilon$ occurs at $[l', u']$. The frequency definition in this case changes to the number of minimal occurrences of $\epsilon$ (MinEpi [19]).

**Example 3** *A minimal occurrence of* $(\{A, B\}, \{A \triangleleft B\})$*, for instance, is* $[12, 15]$, *and of* $(\{B, C, D\}, \{B \triangleleft C, B \triangleleft D, C \trianglelefteq D\}$ $[15, 25]$. *The frequency definition based on the number of minimal occurrences of* $\epsilon$*, e.g., freq*$((\{A, B\}, \{A \triangleleft B\})) = |\{[12, 15], [36, 38], [94, 124]\}| = 3$, *gives an arguably more intuitive count.*

As an improvement of the minimal occurrence semantic, non-overlapping occurrence counts have been proposed [17, 26]. We reproduce here the definition given by Laxman *et al.*:

Two occurrences of an episode are said to be *non-overlapping* if no event associated with one appears in between the events associated with the other. The frequency of an episode is defined as the maximum number of non-overlapping occurrences of the episode in the event sequence.

**Example 4** *To illustrate this, let us change the example sequence from above to* $(1, 141, \langle(E, 1), (A, 12), (A, 13), (B, 15), (B, 20), (C, 25), (D, 25), (A, 36), (B, 38), (C, 55), (E, 66), (D, 75), (A, 94), (E, 109), (B, 124), (C, 131), (D, 141)\rangle)$ *The occurrence of* $\langle A \lhd B \lhd C\rangle$ *that starts at* 12 *and end at* 25 *prevents the occurrence of the same episode that starts at* 13 *from being counted in the non-overlapping semantic.*

Tatti *et al.*'s [27] definition for closed strict episodes is somewhat more involved. We only sketch it here but direct the reader to the original publication, should she be interested in details. They consider generalized episodes as directed acyclic graphs and define an episode $\epsilon$ as *strict* if for any $v, w \in V_\epsilon : g_\epsilon(v) = g_\epsilon(w)$ there exists a path from $v$ to $w$ (or from $w$ to $v$). They furthermore define a concept called *instance-closure*, and mine only instance-closed episodes, proving that those episodes are also frequency-closed.

Finally [9, 21] 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.

## 3 Related Work

A recent overview of temporal pattern mining techniques can be found in [16]. Early work in the field [18, 19, 20, 9, 21] 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 [20], algorithms have not been compared to each other on those data. The seminal paper [18] introduced WinEpi mining, performed efficiency experiments on one data set, and found episodes were evaluated by consulting domain experts. In a follow-up to their earlier work [19], the authors proposed the MinEpi formulation, evaluated efficiency on one data set and gave examples of discovered episodes. In a third work [20], the authors compared the two algorithms on five data sets (two temporal, three non-temporal). This series of papers therefore offered *some* evidence regarding the (relative) performance of the proposed techniques but that evidence was limited.

The technique proposed in [9] replaced the fixed-size windows within which an episode can occur by maximum constraints on the size of inter-event time-delays. The technique was evaluated on non-temporal data for which an existing pattern was known. The authors of [21] showed that the algorithm of [9] is incomplete. After correcting this, their technique was evaluated on a temporal data set w.r.t. efficiency. In both cases, only limited amounts of data were used.

More recent works [17, 26, 27, 28] have used artificial data to experimentally validate their proposed approaches in addition to real life data. Laxman et al. [17] 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. They introduced *non-overlapping occurrence counts*, and experimentally showed on artificially generated data that using this occurrence measure allows recovering the underlying patterns, and leads to more efficient episode mining than the WinEpi approach. The authors also argue that their technique will be more efficient than MinEpi mining. In addition, mining on a temporal real life data set is performed. Tatti *et al.* [26, 28] used artificial data corresponding to extreme cases to demonstrate the superiority of their technique in those contexts. Tatti [26] took a different approach to establishing significance by comparing the average lengths of discovered minimal windows to an independence model, using the same non-overlapping occurrence count. The technique was evaluated on four data sets, two non-temporal, two artificial. Recent work by the same author [27, 28] introduced the concept of *closed strict* episodes, in which any two nodes in an episode having same the label need to be connected by a path, mined using fixed windows (ClosEpi). The authors show that this includes all serial, parallel and most general episodes yet allows to enforce a closure property. The authors evaluated their technique on three text data sets in [27], a single artificial data set and two real life data sets whose characteristics are not specified in [28]. While those evaluations were more systematic than in earlier works, there were still only few data sets involved, showing a small range of characteristics.

The current situation of episode mining has parallels in the developments in itemset mining. Since unsupervised pattern mining does not offer itself to such a clear "right or wrong" evaluation measure as accuracy, evaluating whether any mining operation actually returns the underlying patterns is not an easy task. It has been established only tentatively that such underlying patterns would be recovered. The original Apriori paper [2] proposed a data generator meant to mirror real life distribution characteristics of supermarket transaction data to evaluate the efficiency and effectiveness of itemset mining techniques. The first comparison of itemset mining techniques on new data was performed by Zheng *et al.* [31] showing that the artificially generated data has rather different characteristics from real world data. In the evaluation of several algorithms proposed as improvements to the original Apriori algorithm, results on the artificial data set *could not* be transferred to real world data. Partially in reaction to this, the pool of benchmark data for itemset mining approaches was extended with the newly introduced real life data and several large UCI data sets, and two Frequent Itemset Mining Implementation competitions held [12, 4]. The focus lay on efficiency questions, with the nature of the data making evaluation of pattern recovery impossible. De Bie [5] used a maximum entropy model to evaluate whether unexpected patterns mined from a database are still exceptional after item permutation under certain constraints.

These works therefore combine to show how problematic it can be to depend on a small number of data sets showing a narrow range of characteristics when evaluating unsupervised pattern mining techniques. Furthermore, they illustrate the lack of knowledge regarding the effectiveness of pattern mining approaches and the difficulty of establishing this effectiveness using existing data sets.

Artificial data generation has been explored in more depth in other fields related to episode mining. In the context of classification experiments in stream data with concept drift, [25] 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 [24]. It generates instances described by three nominal attributes, having two class labels. Class noise can be added. Bifet *et al.* [6] used the tree generator introduced in [30] by generating one source tree for each class and adding label drift during data generation. Such data generators can therefore be used to evaluate the performance of different algorithms under varying conditions. 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.* [22] 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 [14], a complex generative model relating numerical values.

Outside of episode mining, there have been attempts in sequential mining to establish the significance of found patterns and identify challenging problems. Atallah *et al.* [3] used a reference source to calculate a significance threshold giving a guarantee on the error probability. The authors evaluated their analytic framework on a real life temporal data set and find agreement between the analytic results and empirical patterns mined. Chudova et al. [11] used artificially generated data to evaluate how, e.g., alphabet size, pattern length, pattern frequency, and auto-correlation influence the difficulty of recovering patterns. They formulated several equations relating the afore-mentioned characteristics of the data to the Bayes error rate. These works therefore established some of the limitations of existing approaches and gave some guidance regarding what to expect when using them.

The single-graph (or network) mining community, finally, has introduced data generators [10], the authors of which were in the advantageous position that information about the characteristics of real life networks existed that could then be exploited in data generation. The SAT solving community also uses artificially generated data [23] to ascertain the run time behavior and success probabilities of SAT solvers [29].

As this enumeration shows, artificial data sets can and have been used to evaluate mining and learning approaches, and establish their effectiveness and run time characteristics, and can therefore help closing the gap in knowledge about episode mining.

## 4 Data Characteristics and Generation

The considerations outlined in the introduction lead us to believe that a robust collection of artificial data showing varied, real-world inspired characteristics will 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 like.

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. The project had to do with mining episode patterns from industrial log files to help modeling the life cycle of machines. Especially early on, the mined patterns were perplexing – alarms occurring or repeating when they should not have, surprisingly large time delays in the data etc. – which lead to consultations with the engineers designing the machines. From those consultations arose the properties we report here and attempt to model later on.

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 [17], contrasting its underlying assumptions with the real-world properties listed above (Section 4.2), before we discuss the plausibility of used distributions in Section 4.3.

### 4.1 The HMM-based generator

To the best of our knowledge, the HMM-based generative model proposed in [17] 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 5** *It is important to realize that the noise probability is applied* per event*. To give an example, let an embedded episode take the form* $A \lhd B \lhd C \lhd D$*, and arbitrary noise events denoted by* $\mathbf{E}$*. 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.:* **E***, A, B, C, D, A, B, C,* **E***, D, A, B,* **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 $\langle [0, 1], \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+\rangle$.

## 4.2 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.” [17]

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

The mining techniques that we have described in Section 3 abstractly consider time delays only at one granularity. In fact, while actual data can include temporal information at different granularities, e.g. days, hours, minutes, it is necessary practice when using existing episode mining implementations to translate this information into time stamps represented by integers, e.g. by multiplying hours by 3600, adding minutes multiplied by 60 etc. We therefore use only one level of granularity for time stamps in our generator for the time being.

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 7** *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 8** *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 [17] is generated by interleaving the embedded episodes randomly. While this is a realistic 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 $\langle [0, 1], \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+, \mathbb{N}^+, [0, 1], \{t, f\}, \{t, f\}, \{t, f\}, \{t, f\}, \{t, f\}\rangle$.

## 4.3 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

| Numerical parameters (default values) | |
|---|---|
| $p$ – probability of noise events | (0.2) |
| $N$ – length of source episodes | (4) |
| $M$ – size of the alphabet of event types | (20) |
| $n$ – number of source episode | (1) |
| $T$ – total number of events in the data | (5000) |
| $g$ – maximal delay between any two successive events | (20) |
| $o$ – probability that a source episode's event is *not* embedded | (0.0) |
| $G$ – "base mean" of a normal distributions (variance $G/10$) | (300) |
| $m$ – number of normals to mix, with means $G, 2G, \ldots, m \cdot M$ | (1) |

Table 1: The numerical parameters of the data generator with default settings in parentheses

distributions is more difficult than introducing parameters allowing for real-world like behavior. 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. This position is supported by empirical distributions observed in the data at our disposal appear to be normally or Poisson generated, as we will illustrate in Section 5.

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 uniformly 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 uniformly or Poisson-distributed ($P \in \{u, p\}$).

As before, we let ourselves be guided by the characteristics of the data at our disposal. There are of course other distributions that could govern noise distribution, e.g. Pareto or other power law distributions. There is no reason not to include these distributions and we intend to continuously extend our data generator to make it more useful for benchmarking. Given the characteristics of the data at our disposal, as well as the already extensive experiments, we forwent the exploration of additional distributions for the time being.

| Nominal parameters (default settings) | |
|---|---|
| $P \in \{u, p\}$ – noise is uniformly or Poisson distributed | (u) |
| $r \in$ {t,f} – event types repeat in source episodes | (f) |
| $s \in$ {t,f} – event types are shared among source episodes | (f) |
| $i \in$ {t,f} – embeddings of source episodes can interleave | (f) |
| $W \in$ {t,f} – source episodes have different probabilities | (f) |
| $h \in$ {t,f} – delays between successive events of a source episode $\leq g$ ($\leq m \cdot G$) | (f) |
| $S \in$ {t,f} – source episodes are embedded successively | (f) |
| $d \in \{u, n\}$ – noise delays are uniformly or normally distributed | (u) |
| $D \in \{u, n\}$ – embedding delays are uniformly or normally distributed | (u) |

Table 2: The nominal parameters of the data generator with default settings in parentheses

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 9** *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 Tables 1 and 2, together with default values.

## 4.4 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. The alphabet for event types we use is actually a subset of $\mathbb{N}^+$, i.e. $M = |\mathcal{E}| \Rightarrow \mathcal{E} = \{1, \ldots, M\}$. This has two immediate advantage: on the one hand this means that we do not run into difficulties if $M > 94$, the maximum number of non-special ASCII symbols. On the other hand, this allows us to generate Poisson-distributed noise events directly, since our alphabet is already ordered, as we will explain below.

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

**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 $t$. The Poisson distribution has a single parameter, the mean, which is set to $\lceil M/2 \rceil$, and the result of sampling from it is used directly as noise event type. As an aside, this means that for a given $M$ the noise event types will have roughly the same distribution for different data sets but since the event types of source episodes have been selected independently, the interplay of noise and episodes will change.

With probability $1 - p$, an episode event is generated. If $S = \mathit{true}$, source episode $\epsilon_i$ becomes embedded as long as for the current event count $T'$ holds that $T \cdot \sum_{j=1}^{i-1} \mathit{weight}_j < T' \leq T \cdot \sum_{j=1}^{i} \mathit{weight}_j$, otherwise all source episodes are embedded concurrently along the entire length of the data sequence. In the latter case, if $i = \mathit{false}$, one of the $n$ source episodes is chosen randomly (with equal probability if $W = \mathit{false}$, 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 $i = \mathit{true}$, 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 $\delta$ sampled from the normal distribution with the respective mean. If $h = \mathit{true}$ 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

[1]This is another choice that could be made differently but that we made since we were given no indication to believe that certain event types were inherent more (or less) common.

**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 $\wedge \exists v \in V_\epsilon : v \mapsto E \in g_\epsilon$ then
                reject E
            else if r = true $\wedge \exists v \in V_\epsilon : v \mapsto E \in g_\epsilon$ then
                accept E
                repeated = true
            else if r = true $\wedge$ repeated = false $\wedge$ j = N then
                sample E uniformly from $g_{\epsilon_i}(V_{\epsilon_i})$
                accept E
            if s = false $\wedge \exists v \in V_{\epsilon_{i'}}, i' < i : v \mapsto E \in g_{\epsilon_{i'}}$ then
                reject E
            else if s = true $\wedge \exists v \in V_{\epsilon_{i'}}, i' < i : v \mapsto E \in g_{\epsilon_{i'}}$ then
                accept E
                shared = true
            else if s = true $\wedge$ shared=false $\wedge j = N$ then
                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 $j > 1$ then
            $\trianglelefteq_{\epsilon_i} = \trianglelefteq_{\epsilon_i} \cup \{v_{j-1} \trianglelefteq v_j\}$
        $g_{\epsilon_i} = g_{\epsilon_i} \cup \{v_j \mapsto E\}$
    source = source $\cup \epsilon_i$
    Sample $weight_i$
for $1 \leq i \leq n$ do
    $weight_i = \frac{weight_i}{\sum_{j=1}^{n} weight_j}$
```

$t_e + g\ (t_e + (m \cdot G))$.[2]

## 5 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 [17, 28] 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.

Characterizing unsupervised data sets is notoriously difficult and few characterizations and similarity measures have been developed, in particular if relabeling effects can occur: if one takes an event sequence and randomly permutes the labels of its event types, existing measures will wrongly consider the two data sets different. We therefore limit ourselves to distributional characteristics, i.e. the (shape of) the distribution of event types in the data and the distribution of time delays in the data, and consider two data sets similar when they approximately agree in those distributions. In particular, this is the way we assess the similarity of our generated data with the real life data at our disposal.

### 5.1 Effect of noise probability and distribution on event type distributions

The first two figures show the shape of the event type distribution of data generated using settings used in [17], i.e. $n = 2$, $p = 0.3$, $i = \mathit{true}$, $h = \mathit{false}$, $P = u$, and [28] ($N = 8$, $p = 0.38$, $P = u$).

In the first case (Figure 1), the event type distribution shows that on the one hand the events involved in embedded episodes are strongly expressed, which should make 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 an intended effect since this data set is therefore well-suited to demonstrating the superiority of non-overlapping occurrence counts over traditional methods.

A similar motivation underlies the generation of the artificial data in [28], shown in Figure 2. Drawing noise labels from a very large alphabet means that the embedded pattern (and combinations and permutations of it) clearly

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

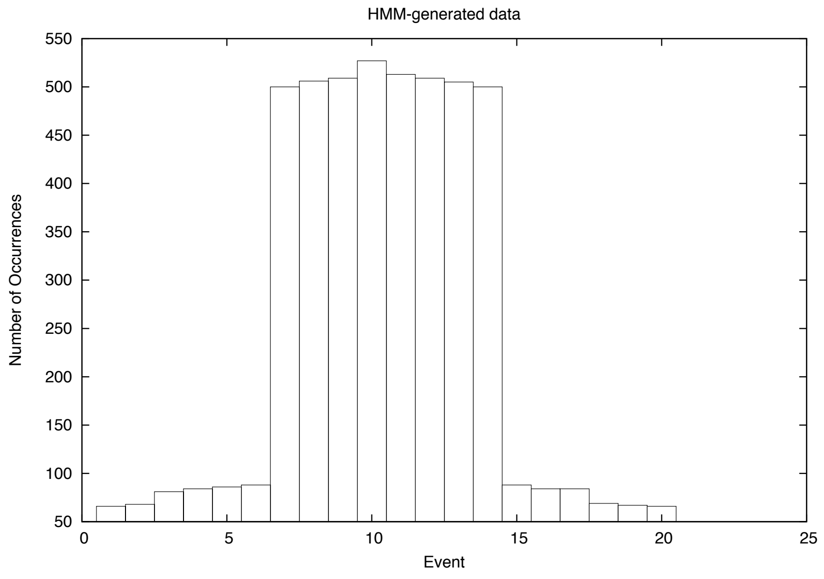

Figure 1: Event type distribution of data generated by the HMM generator

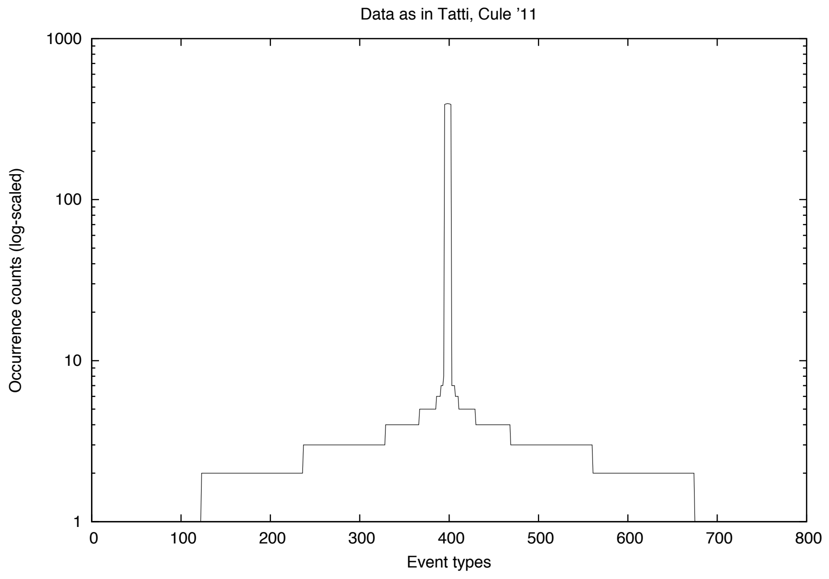

Figure 2: Event type distribution of data generated using a very large alphabet

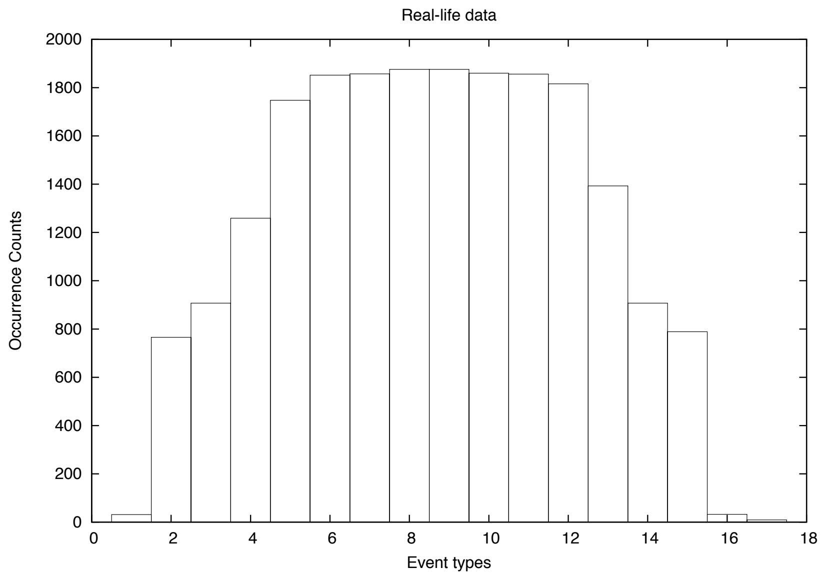

Figure 3: Event type distribution in an first example data set of our real life data

dominates, leading to an extremely large search space that would overwhelm less sophisticated techniques.

Using uniform distributions is therefore useful when then goal is the generation of data that leads to challenging enumeration and search problems. In addition, we assume that the General Motors data Laxman *et al.* worked with showed characteristics that could be generated using uniform distributions. However, these data sets are rather specific and not necessarily representative of real life data. To illustrate this further, consider Figures 3 and 4, showing examples of the event type distribution of real life data we have worked with, which are looking rather different.

To show how one can use the generator to approximate real life data, we will show here step-by-step how to arrive at data showing the same characteristics as the one at our disposal. In a first step, we attempt to approximate the distribution using uniform noise and episode distributions (Figure 5). This data has been generated using uniform noise distribution, two source episodes of length four, and a noise probability of 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 con-

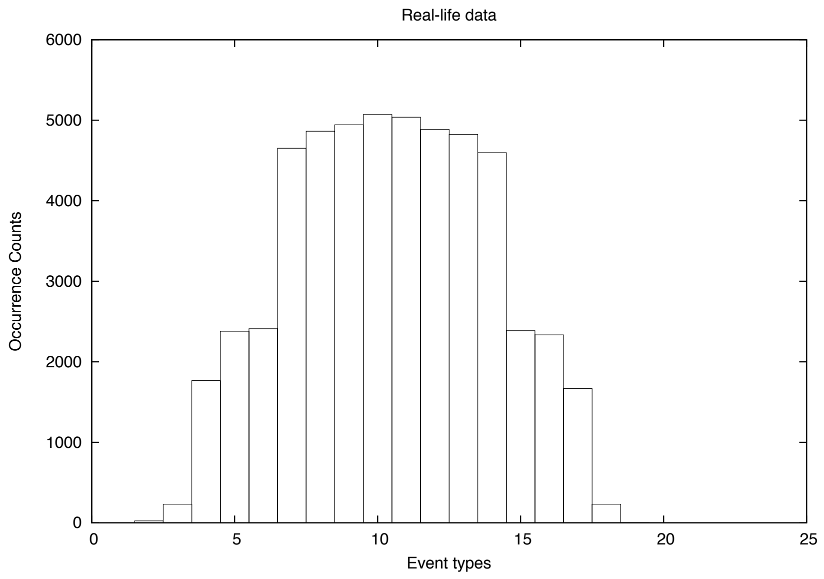

Figure 4: Event type distribution in a second example data set of our real life data

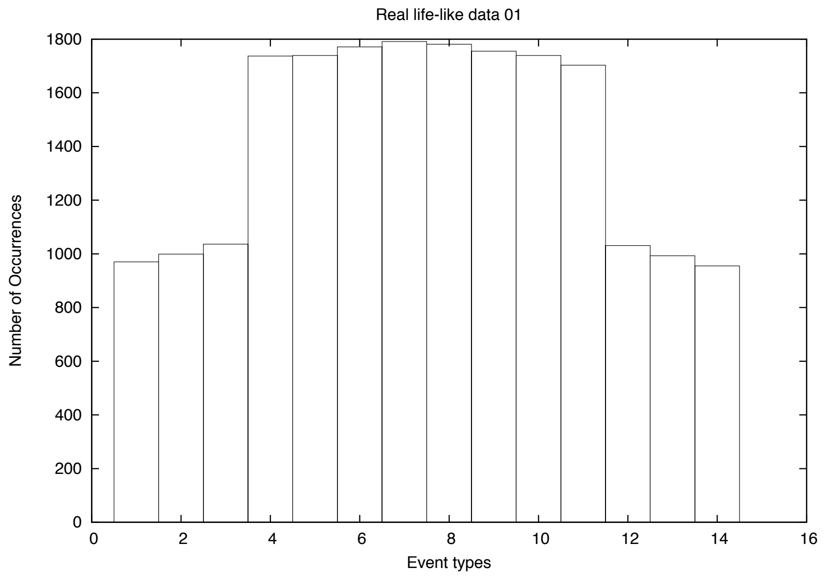

Figure 5: Approximated real life like event type distribution using uniformly distributed noise, and source episodes having equal probability

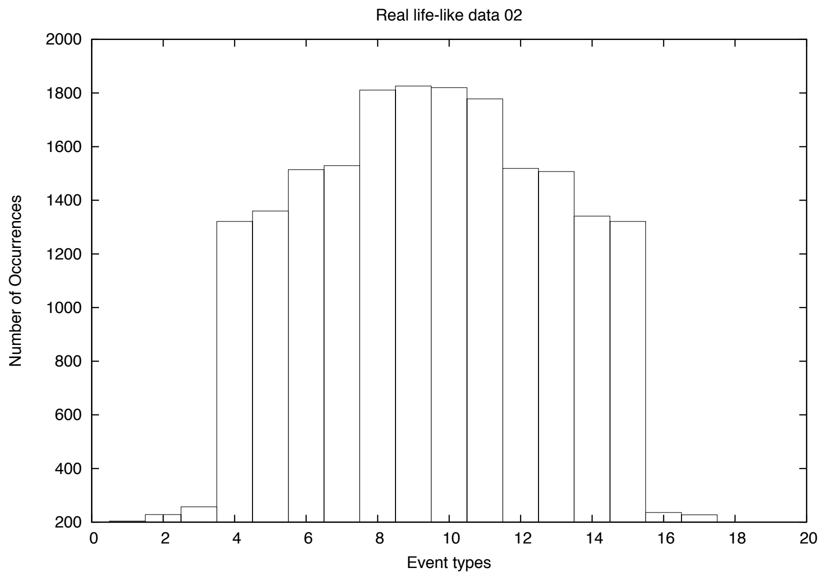

Figure 6: Approximated real life like event type distribution using uniformly distributed noise, and source episodes having different probability

tinue 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 6. 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 7 shows the event type distribution that occurs if we use poisson-distributed noise instead of uniformly distributed noise. The shape of the distribution of event types in this data resembles that of the real life data. looks like the real life data at our proposal, supporting our design decisions.

## 5.2 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 8 shows the 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

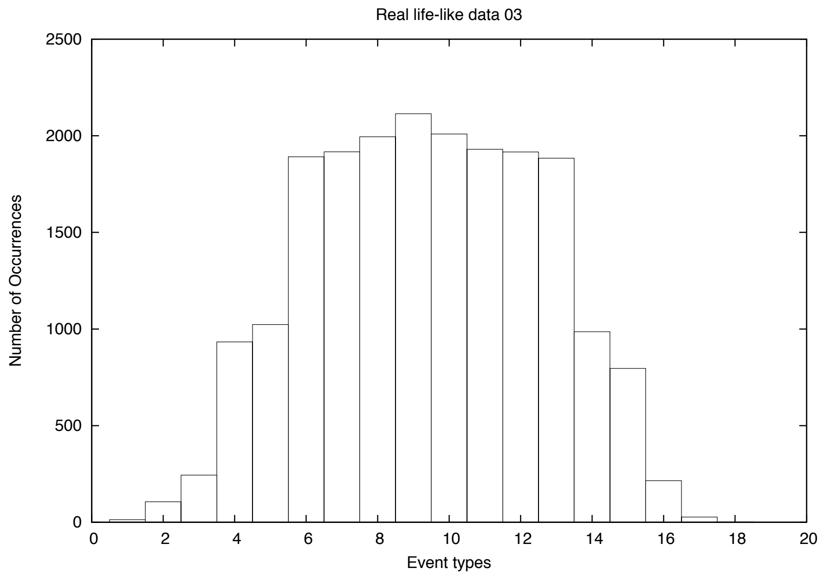

Figure 7: Approximated real life like event type distribution using Poisson distributed noise, and source episodes having different probability

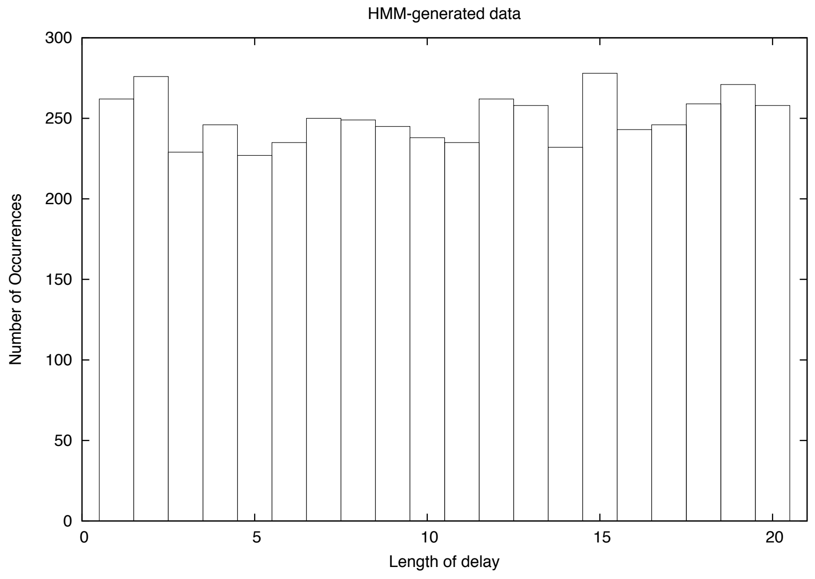

Figure 8: HMM generator time delay distribution (unconstrained)

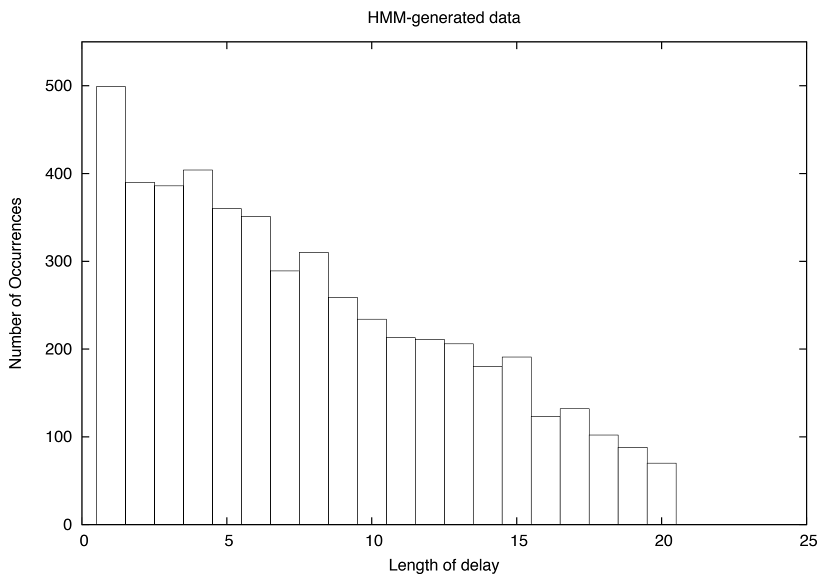

Figure 9: HMM generator time delay distribution (constrained)

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 9), 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 to be able to specify different temporal behavior in the data. This includes changing the distribution of time delay values. As Figure 10 shows, the real life data we have at our disposal shows a time delay distribution that is far different from a uniform distribution.[3]

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 11. This means that at this point, we can generate data with similar characteristics to the real life data at our disposal.

[3]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

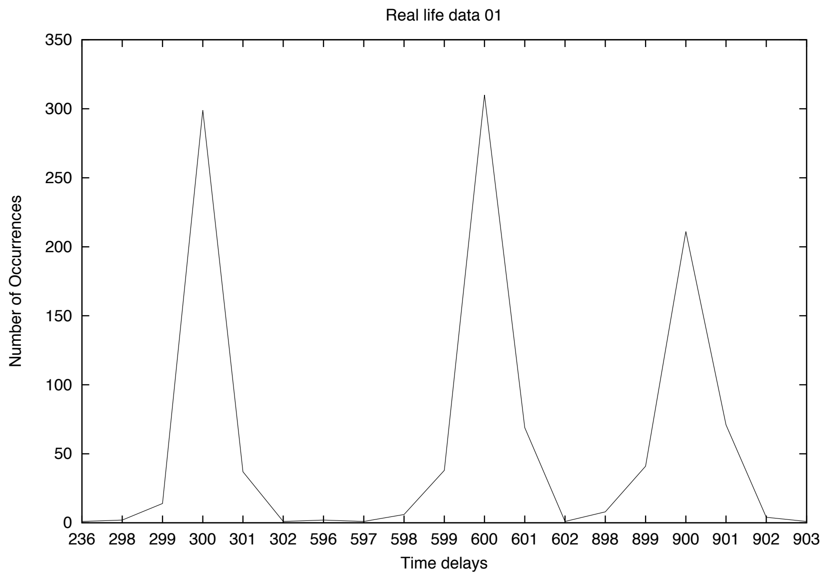

Figure 10: First section of the time delay distribution in the first example data set of our real life data

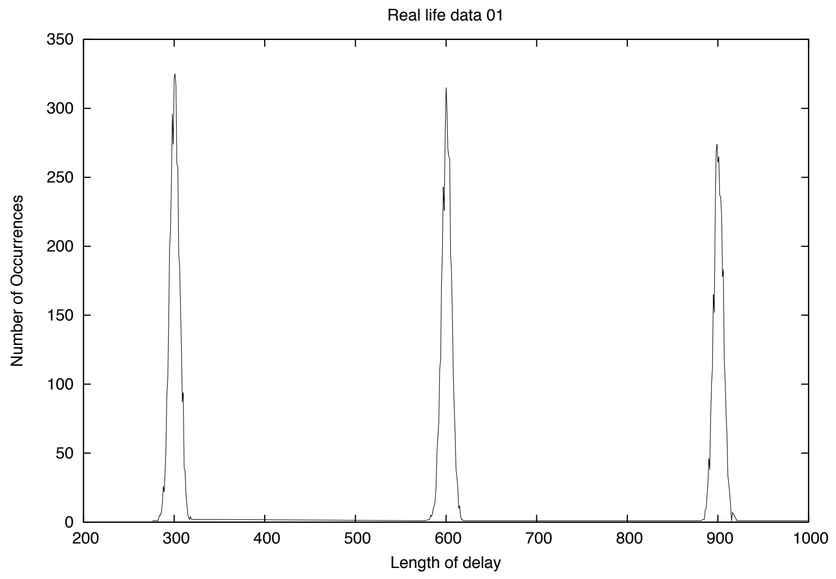

Figure 11: Approximated real life like time delay distribution, using normally distributed delays

## 5.3 Discussion

As we have shown so far, our generator is capable of recreating data with characteristics similar to the ones we have observed in real life data. We have to reiterate, however, that we cannot know whether the data at our disposal is representative. Instead of declaring a generative model with weighted source episodes, poisson-distributed noise, low noise probability, and normally distributed delays in the data the gold standard, our generator allows for a wide range of different parameter settings. Benchmarking episode mining approaches over a range of different settings will allow us to draw general conclusions about the effectiveness of mining techniques in recovering the embedded patterns, as well as about the mining behavior of different frequency count definitions.

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. This can be expected to be even more pronounced when interleaving concurrent episodes. In a similar vein, increasing failure probability will lower the occurrence count of event types from embeddings which in terms of event type distribution would look the same as a higher noise probability.

Instead of choosing a single generative model and optimizing episode mining techniques for it, we propose to use our generator to evaluate episode mining methods on many different types of data to gain a robust understanding of the strengths and limitations of different approaches.

## 6 Benchmark Experiments

The main contribution of our work lies in evaluating three state of the art episode mining approaches under a variety of conditions to answer

Q1 Do episode mining approaches recover the patterns underlying the data at all?

Q2 What are the effects of data characteristics on the effectiveness?

To the best of our knowledge, this is the first time that such a comparison has been performed on such a scale and using data whose underlying patterns were known, enabling the evaluation of the quality of episode mining results.

### 6.1 The Techniques

We compare three episode mining approaches against each other. The first episode miner has been generously supplied to us by Christophe Rigotti. Additional information can be found at `http://liris.cnrs.fr/~crigotti/dmt4sp.html`. The miner is a MinEpi miner for serial episodes. Result episodes can be filtered using a maximal window size or maximal gap constraint and we evaluate both temporal constraints, denoting the techniques by MinEpi (Window) and MinEpi (Gap), respectively. The second miner encompasses the ClosEpi techniques described in [27, 28] and has been supplied to us by Nikolaj Tatti. While it mines general episodes, we limit ourselves to the subset of serial episodes since the source episodes in our generator are effectively serial episodes. The temporal constraint is a maximum window size constraint. The third miner uses the non-overlapping occurrence semantic and a maximal gap constraint, and can be downloaded at `http://people.cs.vt.edu/patnaik/software`. We refer to it as TDMiner.

The first two miners present us with computational bottlenecks: the MinEpi miner for lenient constraint thresholds quickly generates such an excessive amount of episodes that writing the entire result to disk is not possible, while the ClosEpi technique is based on WinEpi counting, leading to the overcounting problem described above. We have to take care in setting constraint thresholds that allow us to perform a fair evaluation of both techniques in such a manner that we neither bias the process too much towards nor against finding the source episodes (for instance because the window is too narrow).

The first parameter that can distort the evaluation if chosen wrongly is the window width. If we knew the noise probability, the length of source episodes and the maximal and average time delay, we could define a window size that will facilitate discovery of embedded episodes without being too lenient: $w_{max} = \frac{1}{1-p} * N * g$

**Example 10** *For a 4-element embedded episode,* 0.2 *noise probability and a maximum time delay of* 20*, we can expect any* $\frac{1}{0.8} * 4 = 5$ *events to contain a full episode, which means that* $w_{max} = 100$ *should be enough to discover most occurrences.*

The problem is that while time delays can be gleaned from data, a practitioner is unlikely to know noise probability and length of source episodes. We can however make the assumption that $p \leq 0.95$. The condition given in [17] for the use of a frequency threshold as significance criterion is that $p \leq \frac{M}{M+1}$. Since we adopt the default value of $M = 20$ for most of our experiments, we felt this to be a useful assumption.

This significance threshold for an episode of length $N'$ is $\frac{T}{N' \cdot M}$. One method of addressing the combinatorial explosion of the MinEpi lies in restricting the maximal length of episodes. We therefore restrict it to 6, filter episodes using the support threshold, and fix the maximum window to $w_{max} = \frac{1}{0.05} \cdot 6 \cdot g_{avg}$ for uniform noise delay values. For normally distributed delay values, we use the most frequent delay value (equivalent to $G$): $w_{max} = \frac{1}{0.05} \cdot 6 \cdot G$

Setting the maximum gap constraint is much easier, since we can simply use the maximal delay value $g$ or the largest frequent delay value $M \cdot G$, for uniform and normal delay distribution, respectively. This is informed by our experiences with real life data where setting the gap constraint to the maximum delay in the data lead to millions of patterns. This gap constraint is used for MinEpi (Gap) and TDMiner.

As explained in Section 2, wide windows have a direct effect on the frequency of episodes in WinEpi like techniques. In terms of ClosEpi this meant that mining with the window used for the MinEpi technique either ran for days or ran out of memory. Therefore, we start with a narrower window for this technique, $w_{max} = \frac{1}{0.05} \cdot g_{avg}$ $(\frac{1}{0.05} \cdot G)$, and a minimum frequency of $t_T$, successively reducing the minimum frequency. If the miner finds the source episode(s) for a certain frequency at a certain window width, we choose this value as starting value for the next data set, if not we double the window width and reiterate the process. If the process runs out of memory for two data sets for a given setting or does not find the source episodes within 24 hours, we terminate this setting. Finally, since the miner mines closed episodes, we also accepted super-episodes of source episodes if those are found earlier than the actual source episodes.

For each parameter setting, we mine episodes on 100 generated data sets, using MinEpi (Window), MinEpi (Gap), and TDMiner. Due to the longer running times of ClosEpi, we evaluated this technique on only 20 data sets for each setting. The main question that we are attempting to evaluate is whether and how well embedded episodes can be recovered. To this end we rank found

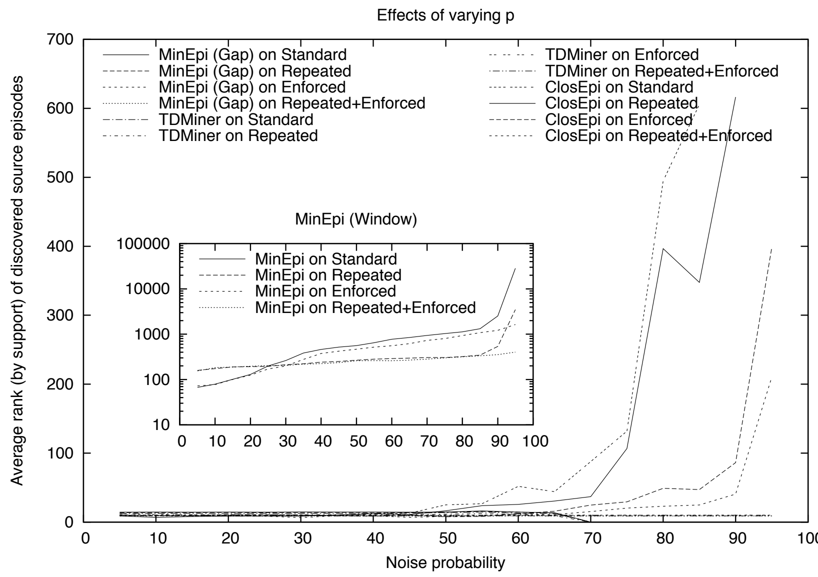

Figure 12: Average rank (by frequency) of source episodes among all discovered episodes for varying noise probability

episodes by decreasing frequency and report the average rank of the source episodes over all 100 data sets. This means that higher rank number is worse.

## 6.2 The Results for Uniformly Distributed Noise

In the first setting, we follow the work reported in [17] by assuming that both noise and delay values are sampled from a uniform distribution.

### 6.2.1 Effects of noise probability

Following the work of [17], we expect noise probability to be an important parameter. We therefore vary p over $[0.05, 0.95]$, increment 0.05, while keeping the other numerical parameters fixed at $(n = 1, N = 4, M = 20, T = 5000, g = 20, o = 0)$.

If event types repeat in source episodes (and therefore in embedded episodes), it should become *easier* to recover episodes since repeating multiples of noise event types have a low probability. Also, if time constraints are enforced on embedded episodes, this can be expected to counteract noise and make discovery easier. We therefore evaluate all combinations of values for $h$ and $r$, while

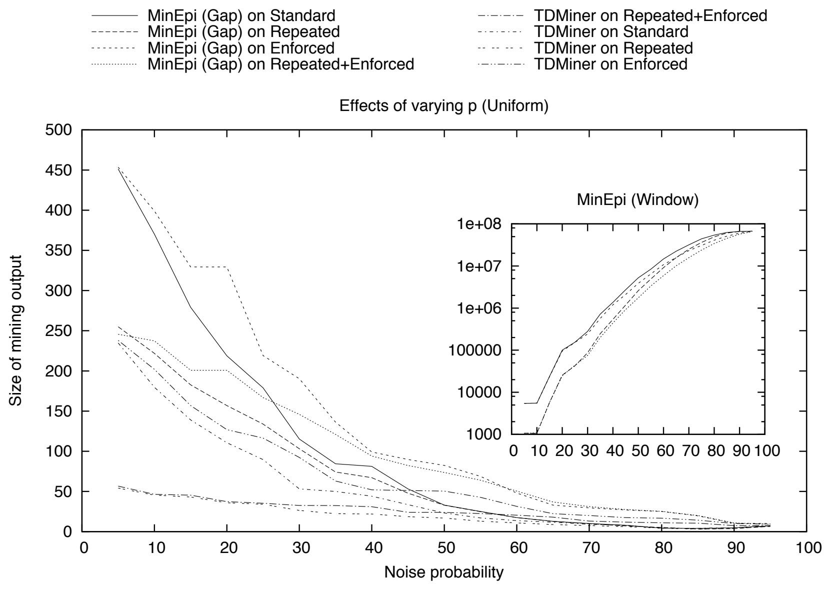

Figure 13: Average sizes of mining result for varying noise probability

keeping the other boolean parameters fixed at *false*, leading to the four curves shown in Figure 12.

The first thing to notice is the much worse performance of MinEpi (Window), as seen in the inset plot (whose y-axis is log-scaled). In the best case, with low noise probability, the ranking of the actual source episode is close to 100, making it unlikely that a user could identify it. It can also be seen that repetition of event types prevents the deterioration of the quality until very high noise probabilities.

More interesting is that the other three approaches perform very similar. Especially when maximal time delays for source episodes are enforced, the found episode is ranked within the top-10 until ClosEpi's performance starts deteriorating at $p = 0.65$. For non-enforced time delays, the ranking is still in the top-20 but the methods using a gap constraint do not recover the episode at all anymore at certain noise probabilities while ClosEpi ranks it very high.

In this context, it is interesting to consider the number of episodes mined in total by the respective approaches, shown in Figure 13. Using MinEpi (Window) leads to more than 1000 episodes for low noise probability and quickly moves into the region of millions. When it comes to MinEpi )Gap) and TDMiner, we see that TDMiner consistently mines fewer patterns than MinEpi (Gap) but as we have seen, the ranking of the found episode is very similar. This

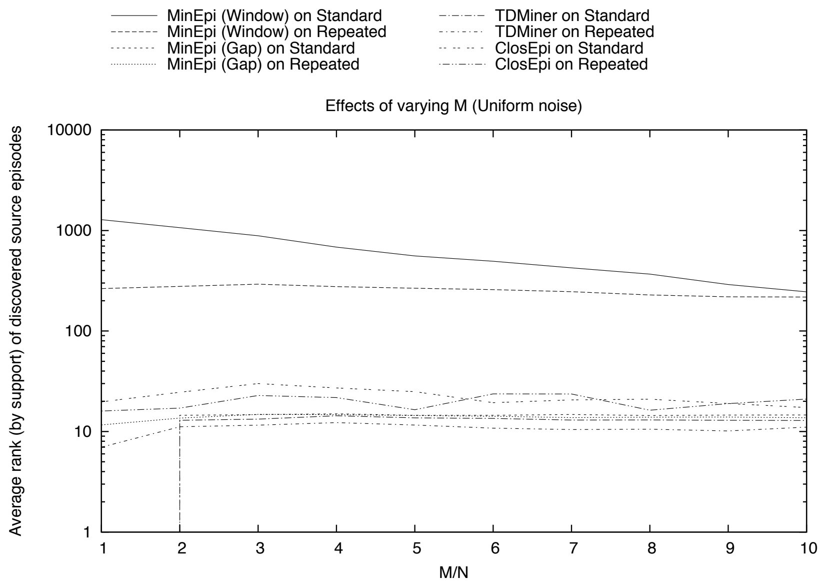

Figure 14: Average rank (by frequency) of source episodes among all discovered episodes for varying alphabet size

indicates that while using non-overlapping occurrence counts filters superfluous episodes, it does not improve identifying the actual pattern. As in the case of MinEpi (Window), repetition of event types reduces the output size. We did not plot ClosEpi's output sizes since it includes parallel episodes in the output and depending on data characteristics this can lead to distortions. Its output sizes are typically somewhat larger than those of the gap-using techniques but much smaller than those of MinEpi (Window).

For the rest of the experiments with uniform noise we use $p = 0.5$, a value where recovering embedded episodes is not too difficult yet. This is also the highest $p$-value reported on in Laxman *et al.*'s work.

### 6.2.2 Effects of $\frac{|\mathcal{E}|}{N}$

In the SAT solving setting, the phenomenon of "phase transition" has been described for the 3-*SAT* problem in propositional logic [23]. Depending on the proportion of number of clauses to number of variables per clause, benchmark problems pass from a region of *easy satisfiability* through a region of *hard decidability* into a region of *easy proof of unsatisfiability*.

In a similar vein, we vary $M$ in the interval $[4, 40]$, increment 4, for fixed

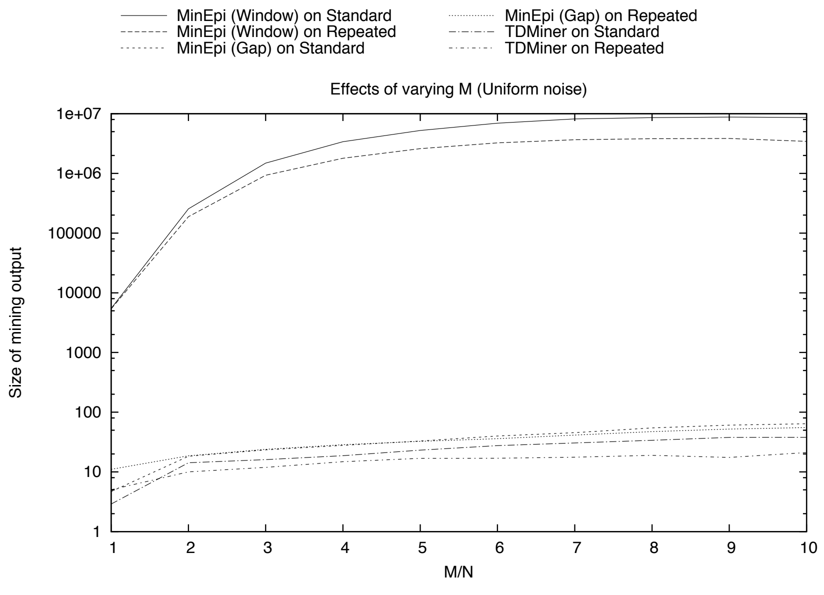

Figure 15: Average sizes of mining result for varying alphabet size

($p = 0.5, n = 1, N = 4, T = 5000, g = 20, o = 0$). Since we would expect the repetition of event types in a source episode to make a difference, we perform experiments for both values of $r$, while keeping the other boolean parameters fixed to *false*, leading to the curves shown in Figures 14.

As expected, increasing $M$ makes it easier for MinEpi (Window) to recover the underlying episodes since drawing noise uniformly from a larger alphabet decreases the appearance of regularity in the noise. The other three techniques are not really influenced by the change in alphabet size, with the exception that MinEpi (Gap) and TDMiner are stumped when $M = 4$ and there is no repetition in the source episodes. While this could be seen as evidence that mining closed strict episodes or using non-overlapping occurrences leads to better results, the use of the MinEpi semantic and temporal gap constraints seems to be as effective. In general, this plot can be considered a zoom into the quality results of the probability experiment in the preceding section, making the rankings more fine-grained, showing in more detail how the rankings compare to each other.

Figure 15 also shows that while using non-overlapping occurrence counts removes some spurious patterns, the advantage over the MinEpi (Gap) is minimal, especially compared to the large output sizes of using a maximal window size constraint.

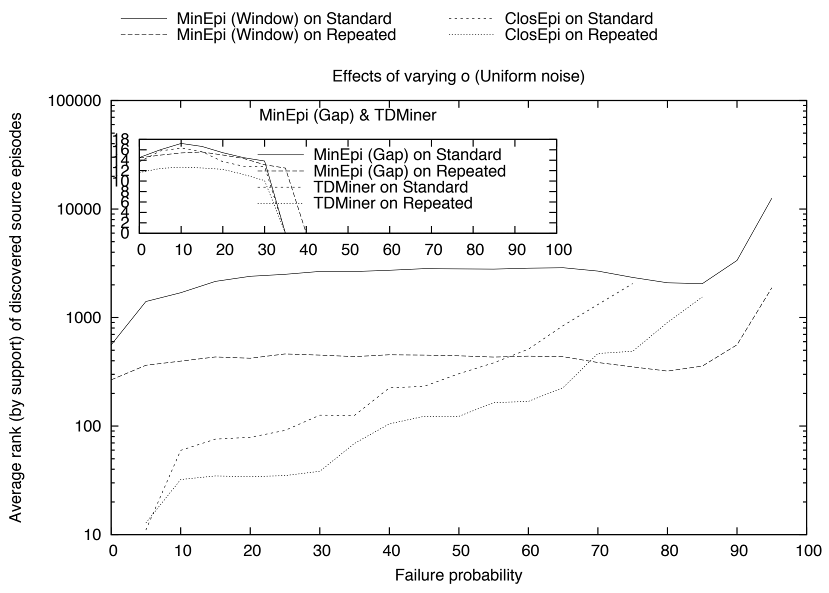

Figure 16: Average rank (by frequency) of source episodes among all discovered episodes for varying failure probability

### 6.2.3 Effects of failure probability

In real life situations, it is possible that events that should be logged, such as sensor readings or alarm codes, get lost, e.g. if the log file is not hosted directly on the machine itself. The higher the probability that such a failure of logging an event occurs, the more difficult it should be to recover the source episodes.

In the experiments reported here, we vary $o$ in the interval $[0, 0.95]$, increments 0.05, and keep the other parameters fixed at ($p = 0.5, n = 1, N = 4, M = 20, g = 20, T = 5000$). This is another setting in which we expect a repetition of event types in source episodes to make a difference, hence both values of $r$ are evaluated while keeping the other boolean parameters fixed to *false*, leading to the curves shown in Figure 16 and 17, respectively.

Similar to the results that we saw for varying noise probability, MinEpi (Window) is rather robust to the effects of failure probability, as opposed to ClosEpi, which quickly shows deteriorating ranks, or the other two techniques that simply fail to recover source episodes at all once failure probability is larger than 0.35. The reason for this can be seen in Figure 17, which shows that the total number of episodes mined falls quickly for both techniques, quicker for TDMiner than for MinEpi (Gap). We consider this experiment additional

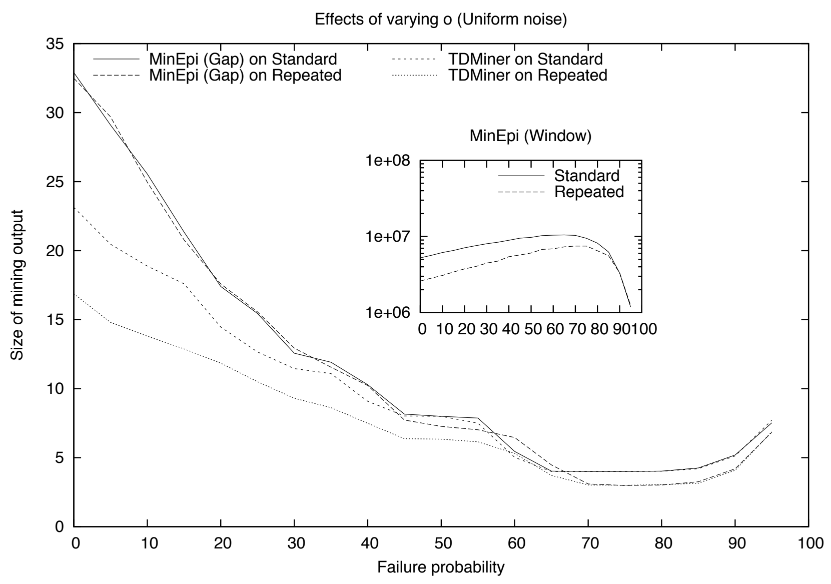

Figure 17: Average sizes of mining result for varying failure probability

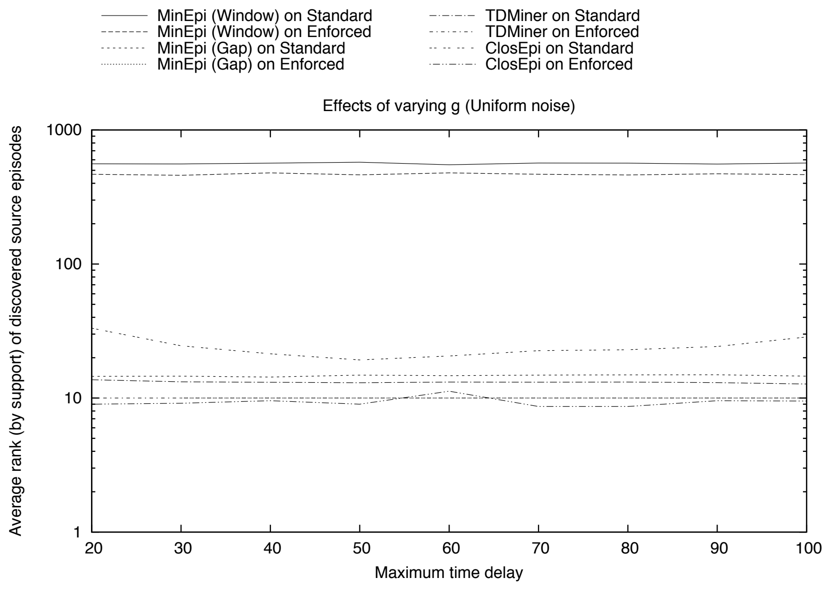

Figure 18: Average rank (by frequency) of source episodes among all discovered episodes for varying maximal time delay

support for the assumption that while using non-overlapping occurrence counts can remove some spurious patterns from the output, the parameter that makes or breaks the success of those techniques is the temporal gap constraint, which on the one hand helps filtering episodes but on the other can be vulnerable to noise effects if temporal delays are not enforced in the data by the generative process.

### 6.2.4 Effects of maximum time delay

To test this influence further, we next vary the maximal time delay in the data. Increasing $g$ leads to a larger interval $[1, g]$ and therefore to a larger variability of delays between consecutive events. In this experiment, we generate data varying $g$ in the range $[20, 100]$, increment 10, holding other parameters fixed at $(p = 0.5, n = 1, N = 4, M = 20, T = 5000, o = 0)$.

Since we expect enforcement of a maximal time delay between event types of an embedded episode to have an effect, we perform those experiments for both values of $h$ while keeping the other boolean parameters fixed to *false*, leading to the curves shown in Figure 18

Changing the maximal time delay has remarkable little effect on the rank-

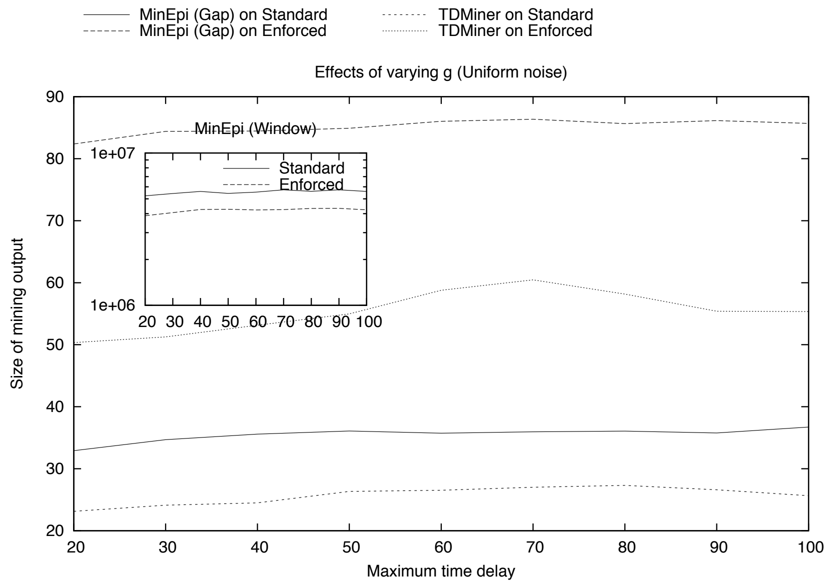

Figure 19: Average sizes of mining result for varying maximal time delay

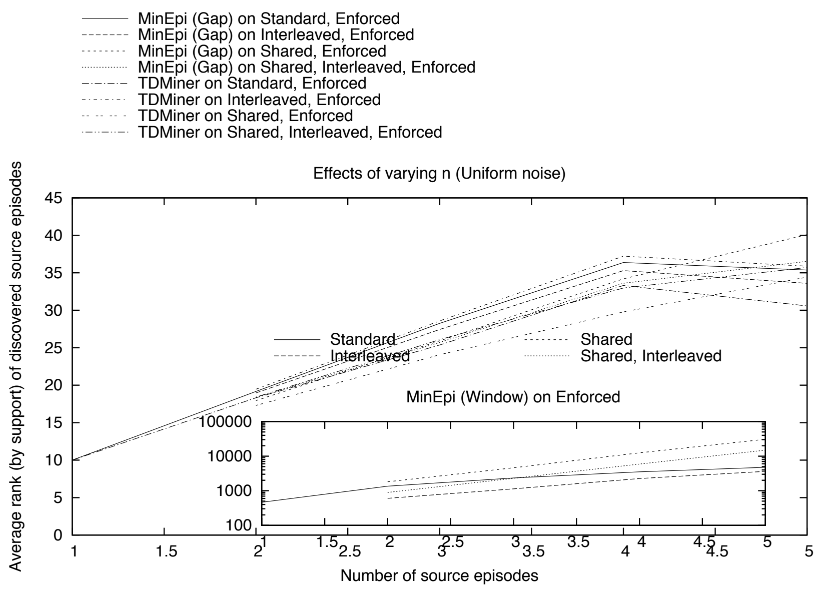

Figure 20: Average rank (by frequency) of source episodes among all discovered episodes for varying number of source episodes

ings for the different methods, probably due to the time-agnostic nature of the MinEpi, and ClosEpi semantics, and the effect of the gap constraints that effectively "stretch" with maximal time delays in the data. Figure 19 continues the trend of TDMiner mining fewer episodes than MinEpi (Gap) and we see that enforcing the maximal time delay leads to more mined patterns, even though the embedded source episode is actually better ranked than in the non-enforced case.

### 6.2.5 Effects of number of embedded episodes and episode interaction

While the experimental results so far indicate that MinEpi (Gap) is equal to TDMiner in terms of recovering the underlying patterns, all of the experiments involved only a single source episode and no interleaving, meaning that any advantage of non-overlapping occurrence counts cannot fully materialize. In this set of experiments, we therefore embed additional source episodes and evaluate the effect that different types of interaction among them have on the miners' results.

The clearest effect of embedding more episodes in the same amount of data

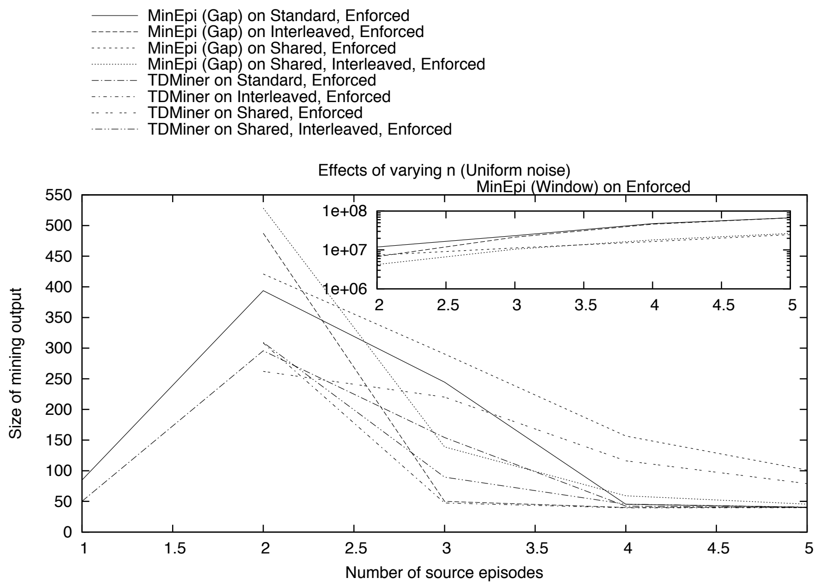

Figure 21: Average sizes of mining result for varying number of source episodes

will be that each episode occurs less often. To evaluate these effects, we vary $n$ in the range $[1, 5]$, while keeping the other parameters fixed at $(p = 0.5, N = 4, M = 20, g = 20, T = 5000, o = 0)$. All episodes have the same probability and are embedded concurrently. Even if episodes are non-interleaved, shared event types among episodes can make episode discovery harder, an effect that will be exacerbated if different instances of the same source episode can be interleaved in the data. Hence, we generate data for all combinations of values of $s$ and $i$, while keeping the other boolean values fixed, leading to the curves shown in Figure 20. In addition to averaging the rank of episodes over all 100 data sets, we averaged over the number of source episodes to give a concise picture. Since each episode has an equal probability of being chosen for embedding at each step, we felt that this was a justified choice.

Even for $n = 2$, the ClosEpi technique did not recover embedded episodes successfully within 24 hours. We therefore do not show this technique in the comparison. Furthermore, as was to be expected given the nature of the temporal gap constraint, MinEpi (Gap) and TDMiner were not capable of effectively recovering embedded episodes if the maximal time delays were not enforced. We therefore report all results on enforced time delays.

It comes as a surprise that for up to four source episodes the results of TDMiner on interleaved episodes are worst – setting aside the predictably high rankings achieved by MinEpi (Window). Generally speaking, both TDMiner and MinEpi (Gap) achieve very similar quality, and both techniques are not thrown off too much by shared event types and interleaving. But even on this setting which would be expected to favor non-overlapping occurrence counts, using MinEpi with a temporal gap constraint performs well. It has also to be said that an end user might have to find out which episodes out of thirty to forty are actually meaningful and which are artifacts of the mining process, a non-trivial task.

It is at this point that we have to raise the fact that our results differ markedly from the ones reported in [17]. When interleaving a 3- and a 4-event episode at $p = 0.5$ in data of size $T = 5000$ that work reports that the source episodes were found at positions 1 and 3 in the frequency-ranked list by their technique and at 1 and 6 by a windows-based miner, a result that we cannot replicate using the TDMiner or MinEpi (Window) technique even for a single source episode. We can only speculate at this point but a possible explanation could be that the authors inadvertently set a window width threshold that fit the actual noise probability, thus avoiding spurious episodes.

It is interesting to see in Figure 21 that after an initially relatively large output size, more source episodes quickly lead to manageable sizes, except when using MinEpi (Window). On the practitioner's side this could mean that when one sees few episodes being mined by TDMiner or MinEpi (Gap), several phenomena are actually buried in the data.

## 6.3 The Results for Poisson-distributed noise

Uniformly distributed noise assumes that the generating phenomena are truly random while just about any natural or cultural process can be expected to exhibit some kind of pattern. We therefore perform the experiments of the preceding setting again, using Poisson-distributed noise this time. We would expect Poisson-distributed noise to make it harder to recover underlying episodes since the higher probability for some noise events would give an appearance of regularity where none exists.

### 6.3.1 Effects of noise probability

The experiments varying the noise probability support this expectation, as Figure 22 shows: compared to Figure 12, the ranking deteriorates earlier. On the other hand, MinEpi (Gap) and TDMiner can recover episodes from data on which maximal delays are not delayed somewhat longer. It is also interesting to see that the number of episodes mined by MinEpi (Gap) and TDMiner decreases slower or even increases for higher noise probabilities when maximal delays are enforced, as Figure 23 shows – the higher noise probability also creates more opportunities for noise to appear like regular phenomena.

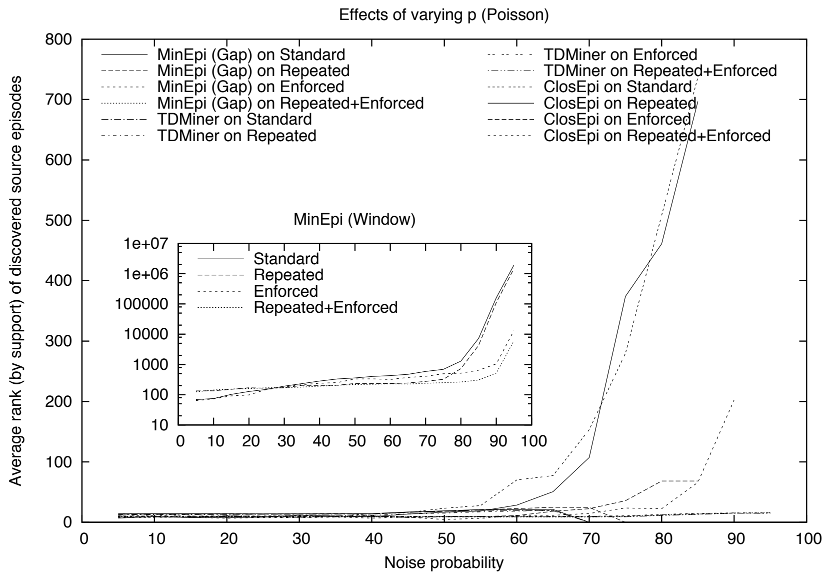

Figure 22: Average rank (by frequency) of source episodes among all discovered episodes for varying noise probability

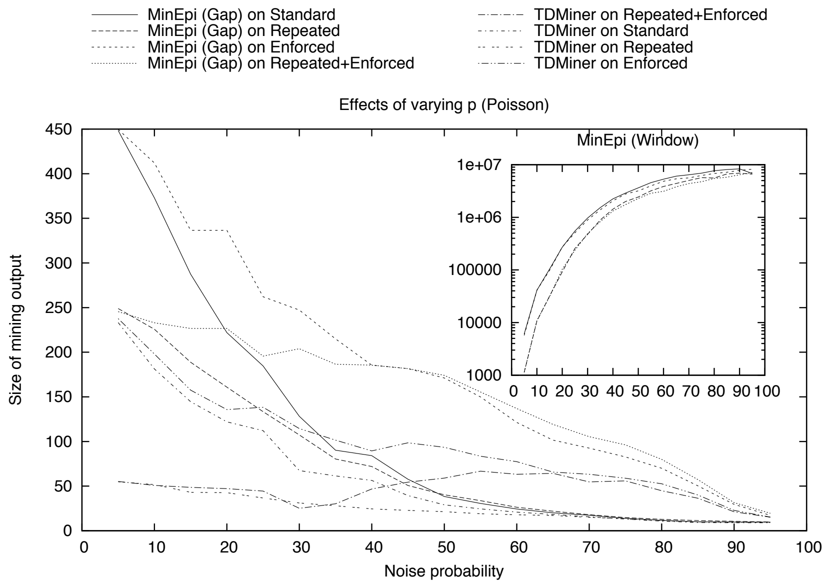

Figure 23: Average sizes of mining result for varying noise probability

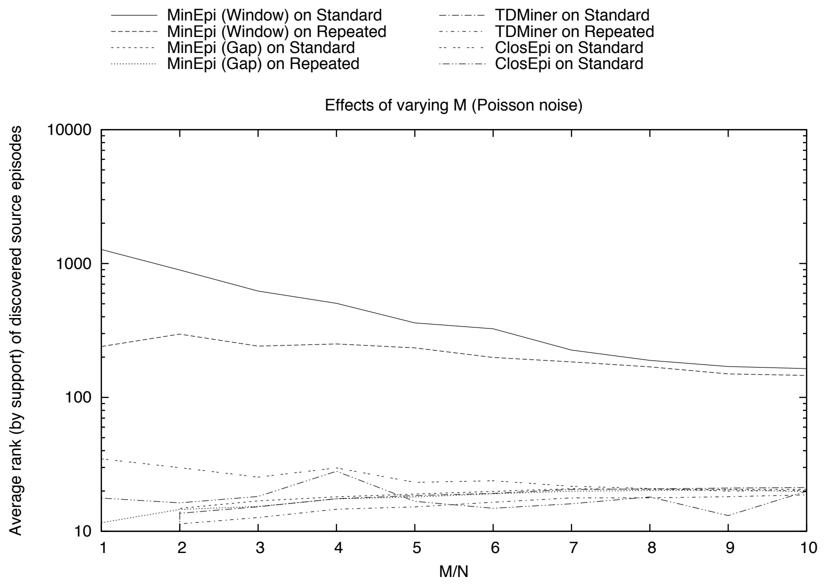

Figure 24: Average rank (by frequency) of source episodes among all discovered episodes for varying alphabet size

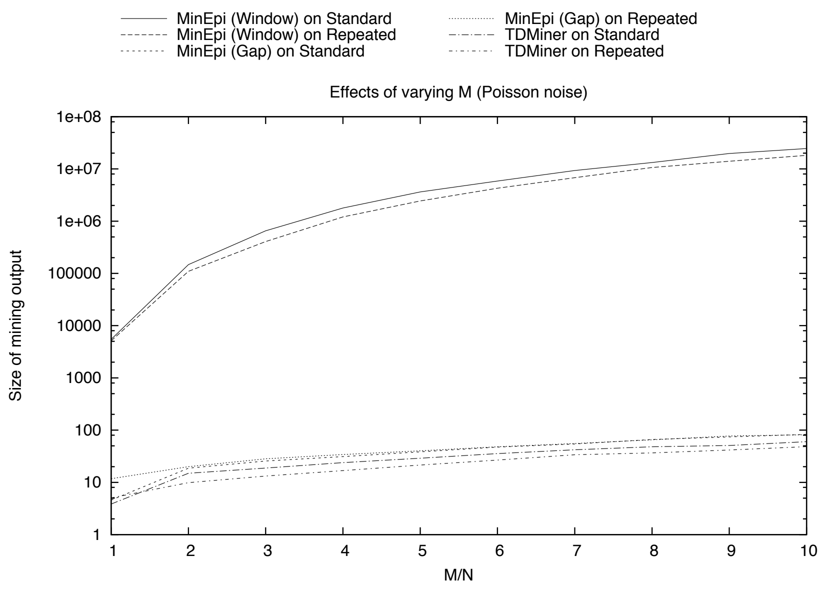

Figure 25: Average sizes of mining result for varying alphabet size

### 6.3.2 Varying M

This is also borne out by the results when the size of the alphabet is varied: while the average rankings for MinEpi (Gap) and TDMiner were slightly larger than 10 for uniformly distributed noise and did not vary much for different values of M, ranks rise to 20 for those two techniques when M is increased for Poisson-distributed noise (Figure 24). Connected to this is a slight increase in the number of episodes returned in total by the different techniques, seen in Figure 25.

### 6.3.3 Varying o

Changing the failure probability has more pronounced effects for Poisson-distributed noise as well, at least for MinEpi (both temporal constraints), and TDMiner, as seen in Figure 26. ClosEpi seems less affected by this but on the other hand is not able to recover any episodes at all anymore at a lower failure probability than for uniform noise. Poisson-distributed noise also leads to a generally larger output size for MinEpi (Gap) and TDMiner (Figure 27).

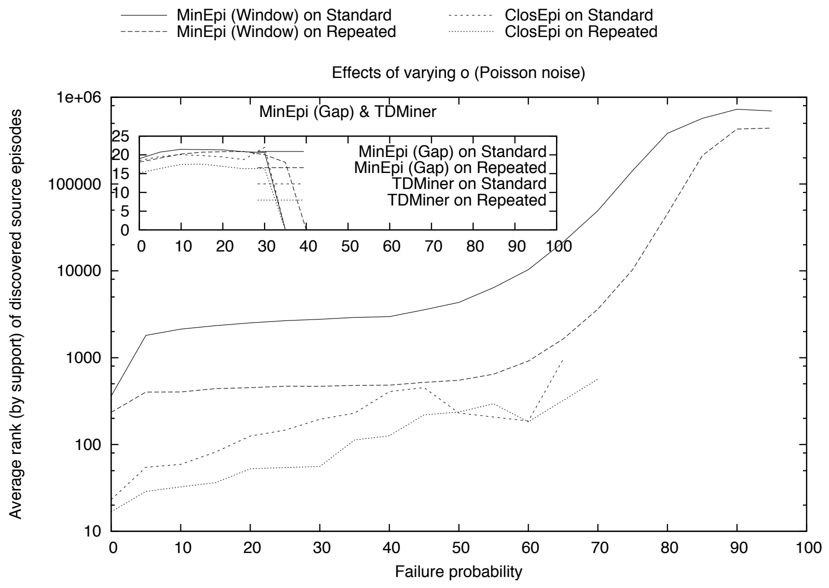

Figure 26: Average rank (by frequency) of source episodes among all discovered episodes for varying failure probability

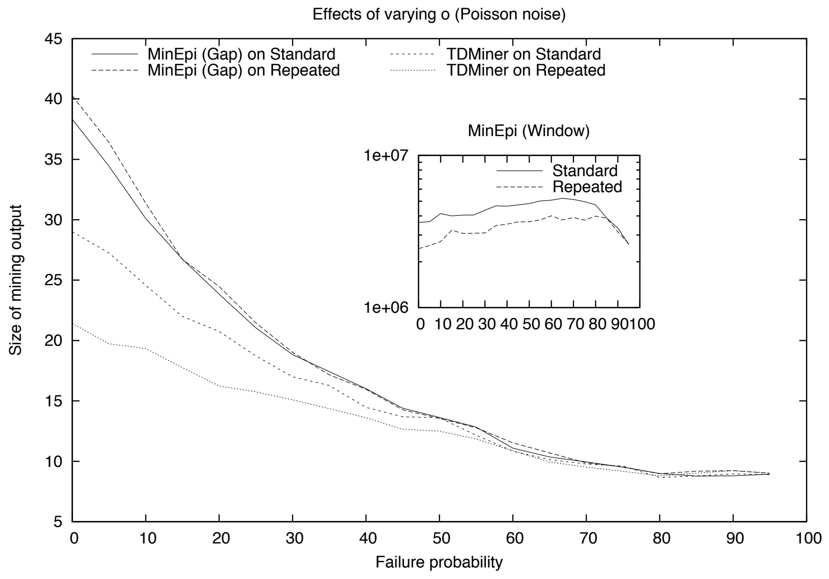

Figure 27: Average sizes of mining result for varying failure probability

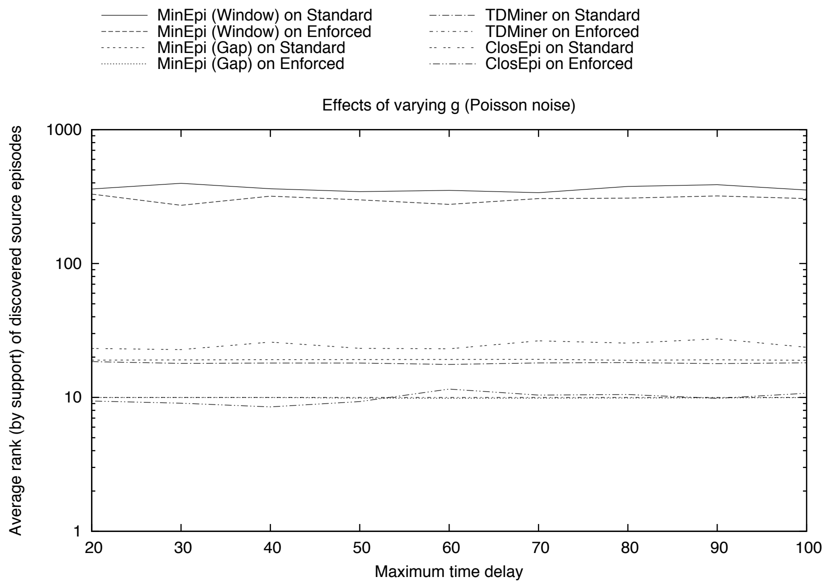

Figure 28: Average rank (by frequency) of source episodes among all discovered episodes for varying maximal time delay

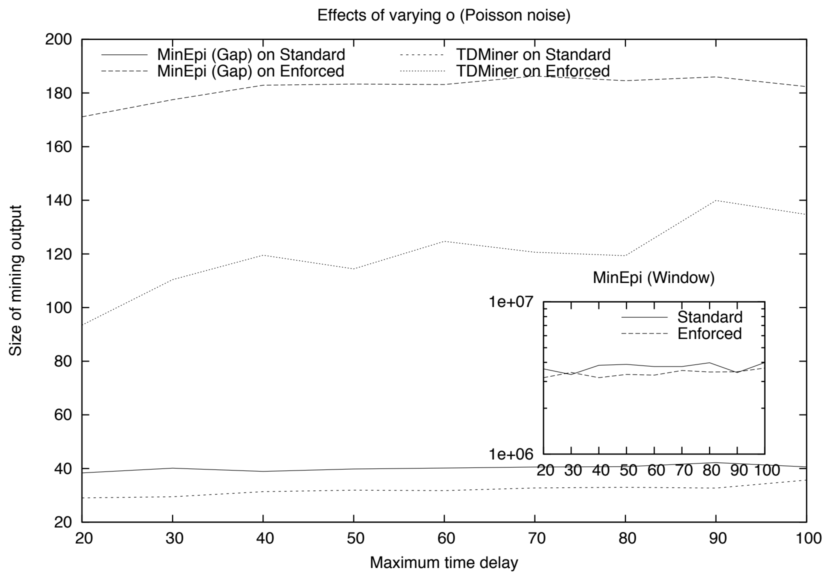

Figure 29: Average sizes of mining result for varying maximal time delay

### 6.3.4 Varying g

When the maximal time delay is varied, the most interesting result is actually found in Figure 29 since the number of episodes that are being mined by MinEpi (Gap) and TDMiner on data on which this delay is enforced effectively doubles. While the average rank also increases, this increase is only slight (Figure 28).

### 6.3.5 Effects of number of embedded episodes and episode interaction

These results let us expect that mining several source episodes will also be harder but as Figure 30 shows, this is not the case. Especially TDMiner attains somewhat better results, while for MinEpi (Gap) there is some change regarding what are the harder settings. This is also reflected in the output sizes (Figure 31) that do not change much between the two noise distributions.

## 6.4 Discussion

The data used for evaluation in the preceding sections is artificial in the sense that we generated it based on a priori assumptions, without data sources whose characteristics we tried to recreate. As we have shown, however, changing data

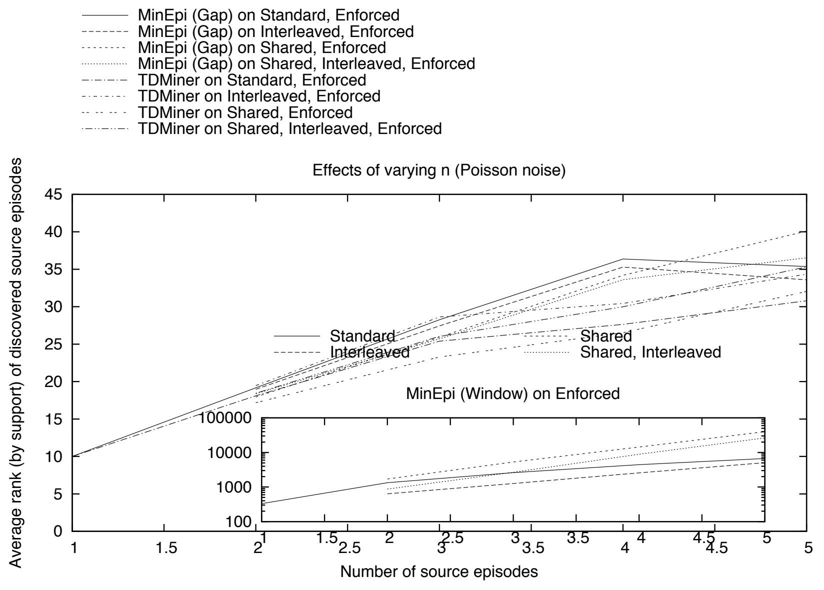

Figure 30: Average rank (by frequency) of source episodes among all discovered episodes for varying number of source episodes

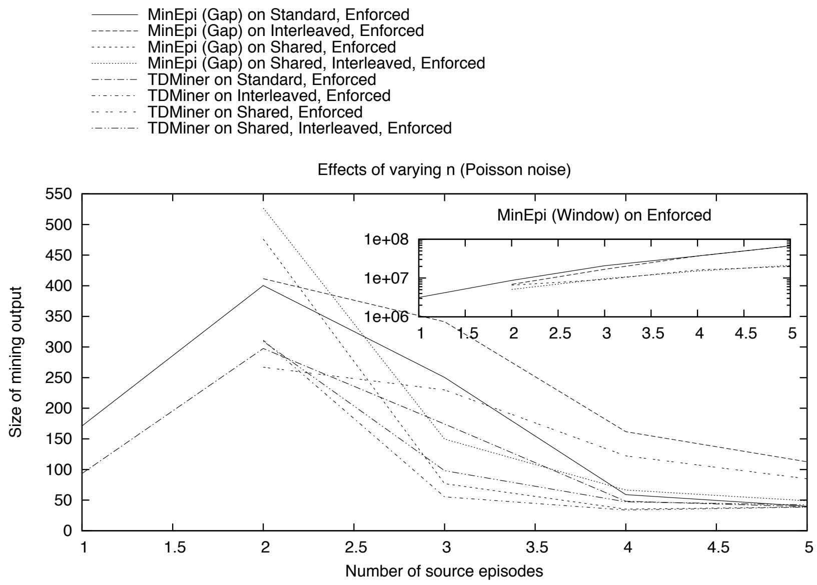

Figure 31: Average sizes of mining result for varying number of source episodes

characteristics can have a profound effect on the effectiveness of episode mining approaches. Depending on the value of different parameters that can describe data, a problem can become markedly harder or easier, and *a priori* assumptions can be proved wrong, as the results for failure probability, and time delay enforcement show. What we found generally surprising is the relatively weak overall performance of MinEpi (Window) mining using frequency as a significance criterion (as proposed in [17]). The best rank is typically still in the hundreds and would therefore likely be missed by a practitioner sifting through the output. Generally speaking, the performance of an episode mining technique seems to depend more on the temporal constraint used than on the occurrence semantic used:

- In an effort not to bias the mining process, we ran MinEpi (Window) with rather large window sizes and the results are typically unsatisfactory.
- For ClosEpi, on the other hand, we effectively report the results for the tightest window for which embedded episodes can be recovered, and the results are much better. It is rather difficult to set this maximum window size constraint a priori, however, and systematically exploring different settings systematically can lead to infeasible running times, as we have seen.
- MinEpi (Gap) and TDMiner use different occurrence semantics but both use temporal gap constraints and their results look consistently very similar (and better than the other two techniques').

Part of the motivation of this work lies in developing benchmark criteria that can be used to evaluate future proposals for episode mining techniques. The challenge to future research in the community would lie in the harder settings. In terms of translating our results into benchmarking problems, data on which episode mining techniques are evaluated, should thus show at least some of the following characteristics:

- A noise probability of at least 40%,
- A ratio $\frac{M}{N}$ that is not too large since otherwise noise events cannot give the appearance of regularity,
- A relatively high failure probability, especially for MinEpi-like techniques,
- Non-enforced time delays since enforcement makes the setting much easier for temporal gap constraints,
- Several episodes, ideally sharing event types and/or having interleaved embeddings.

As a side-note, repeating event types in source episodes make mining settings somewhat easier and should therefore be avoided in benchmark problems.

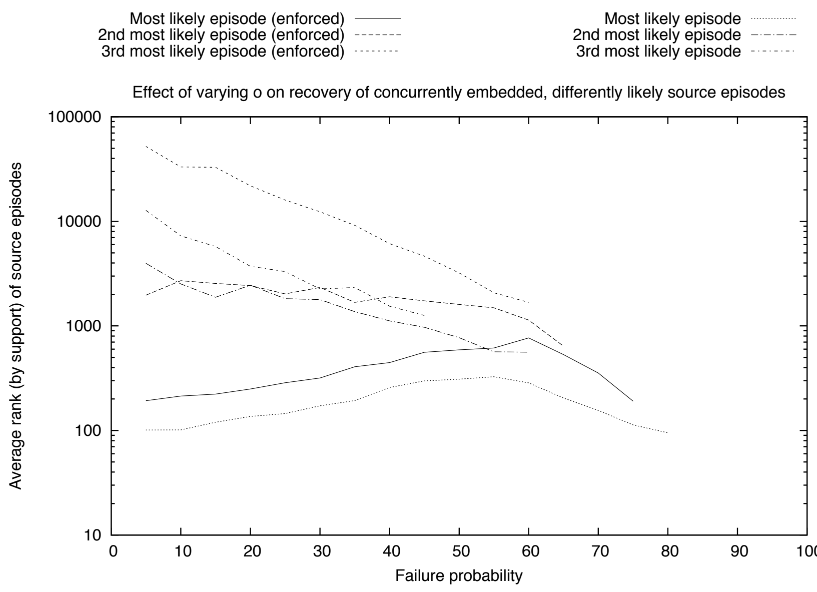

Figure 32: Average rank (by frequency) of source episodes of different probability on real life like data for different failure probability, mined using MinEpi (Window)

## 6.5 The Results for real life like settings

The second part of our motivation was to use artificial data as a stand-in for real life data to gain some guidance about the expected quality of mining results on such data. We showed in Section 5 how we can use our data generator to generate data mimicking the characteristics of the real life data at our disposal.

The parameters that we found to give rise to such characteristics were the following: using several source episodes ($n = 3$), low noise probability ($p \in \{0.1, \ldots, 0.3\}$), source episodes having different probabilities ($W =$ true), Poisson-distributed noise, and time delays samples from 15 normal distributions with enforced maximal time delays. These data might not be representative of all real life phenomena but we consider it interesting to consider the performance of the different techniques on them.

### 6.5.1 Effects of failure probability

A frequent topic in consultations with our industry partners was that the logging of events cannot be expected to be reliable, motivating us to introduce the

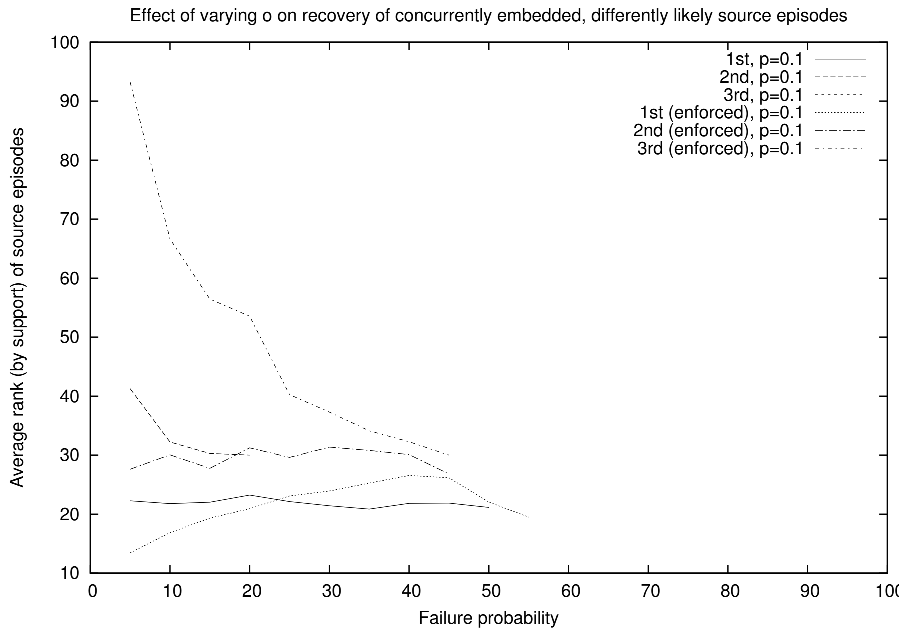

Figure 33: Average rank (by frequency) of source episodes of different probability on real life like data for different failure probability, mined using MinEpi (Gap)

failure probability parameter. In a first setting, we therefore evaluate the effects of different failure probabilities for noise probabilities $0.1, 0.2, 0.3$, respectively. Figures 32, 33, and 34 show the average rank of the each of the three source episodes for MinEpi (Window), MinEpi (Gap), and TDMiner for $p = 0.1$, respectively. The settings for $p = 0.2$ and $p = 0.3$ were very similar, with the only difference being that a higher noise probability prevented recovering episodes earlier. ClosEpi did not manage to recover any of the source episodes within 24 hours and we therefore do not show any results for this technique. Since these episodes have different probabilities, we cannot simply average over their ranks and we therefore show each episode's rank individually, denoting with "1st" the most likely episode, with "2nd" the second most likely etc. Given the impact that enforcing the maximal time delay has on MinEpi (Gap) and TDMiner, and the fact that we do not know whether such enforcement is realistic or not, we evaluated both settings.

As could by now unfortunately be expected, Figure 32 shows that using MinEpi with a maximum window size constraint would not be effective in recovering underlying episodes. The situation looks better for MinEpi (Gap) (Figure 33) and TDMiner (Figure 34), at least when time delays for source episodes are enforced. At least the most likely episode has a good chance of being recovered and ranked well. The graphs look counterintuitive since higher

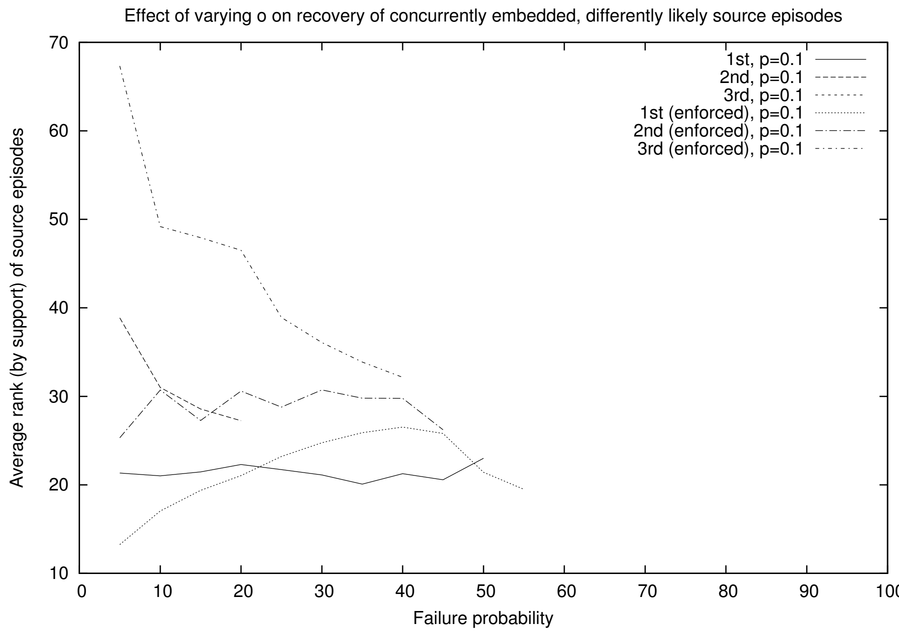

Figure 34: Average rank (by frequency) of source episodes of different probability on real life like data for different failure probability, mined using TDMiner

failure probabilities improve the ranking of recovered episodes. This is in a sense an artifact of the data generation: since the probabilities for source episodes are chosen at random, the least likely episode can be more likely in some generated data sets than in others. For higher failure probabilities, unlikely episodes are not recovered at all, and the averaged ranking therefore reflects only relatively likely, and therefor well-ranked, episodes. This also means that improvements in ranking, e.g. $o = 0.4$ for the most likely episode on time-delay enforced data for TDMiner, do not indicate that it has in fact become easier to recover this episode. Quite contrary, it indicates that recovering has become impossible on some data sets.

Figure 35 shows how the total output sizes compare and we see once again that the two temporal gap constrained techniques have very similar characteristics.

Generally speaking, these results indicate that as long as the failure probability is relatively low, it might be possible to recover the most likely and second most likely phenomena in the data, *if* there is actually a time delay enforcement mechanism at play in the data.

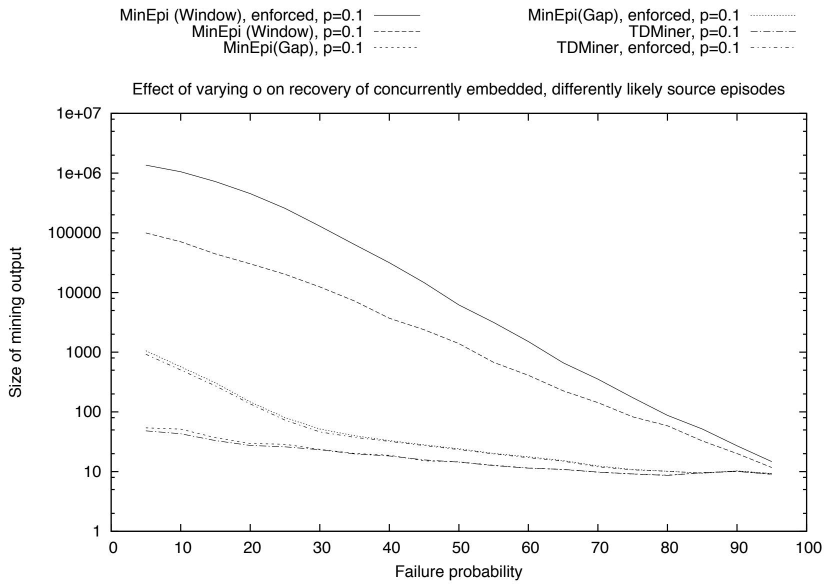

Figure 35: Average sizes of mining result of source episodes of different probability on real life like data for different failure probability

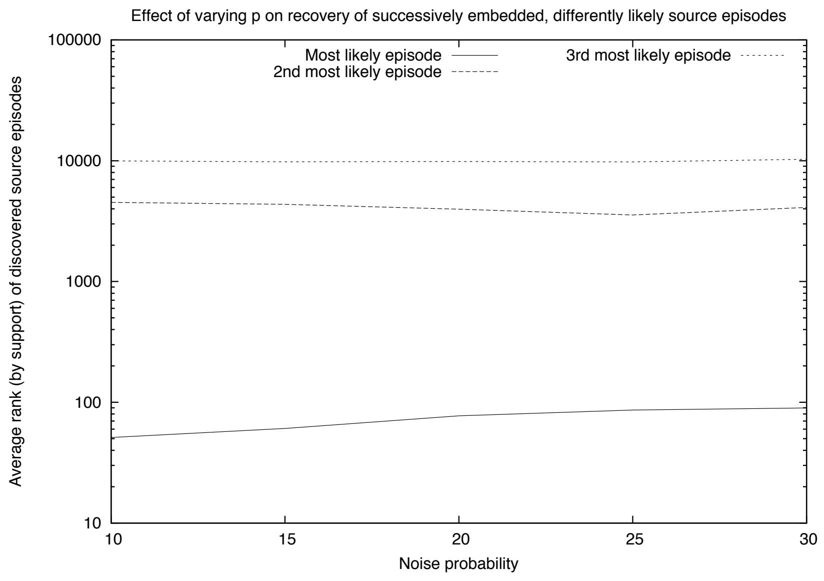

Figure 36: Average rank (by frequency) of successively embedded source episodes of different probability on real life like data for different noise probability, mined using MinEpi (Window)

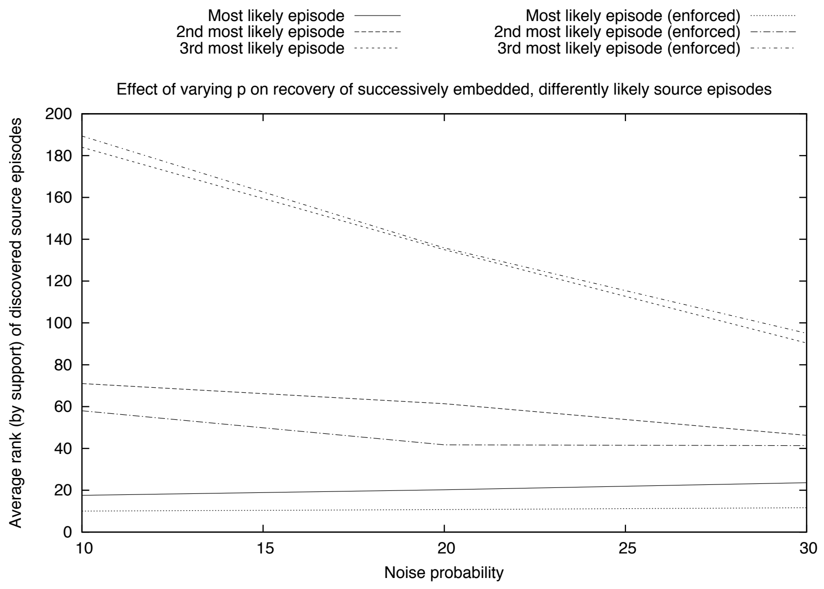

Figure 37: Average rank (by frequency) of successively embedded source episodes of different probability on real life like data for different noise probability, mined using MinEpi (Gap)

### 6.5.2 Effects of successively versus concurrently embedded source episodes

It is possible that the system generating the data, e.g. an industrial machine, goes through different stages during its life cycle, generating different patterns in each one. Since this was a possibility that was mentioned in the context of the project, we therefore also generate data in which data is not concurrently but successively embedded. We vary the noise probability over the full range of settings for which we observed data characteristics matching the real life data. The probability of an episode appearing in the data is still the same as in the preceding experiment but the fact that each episode occurs on its own should make recovery easier.

This assumption is supported by Figure 36, which shows that the most likely episode is ranked much better than for the concurrent setting when mined using MinEpi (Window). In fact, this is the first time in the experiments that this technique has ranked a recovered episode low enough that it might be useful. Unfortunately, however, the ranking is still relatively high. This also means that it would be difficult to leverage the gained knowledge, e.g. by identifying the

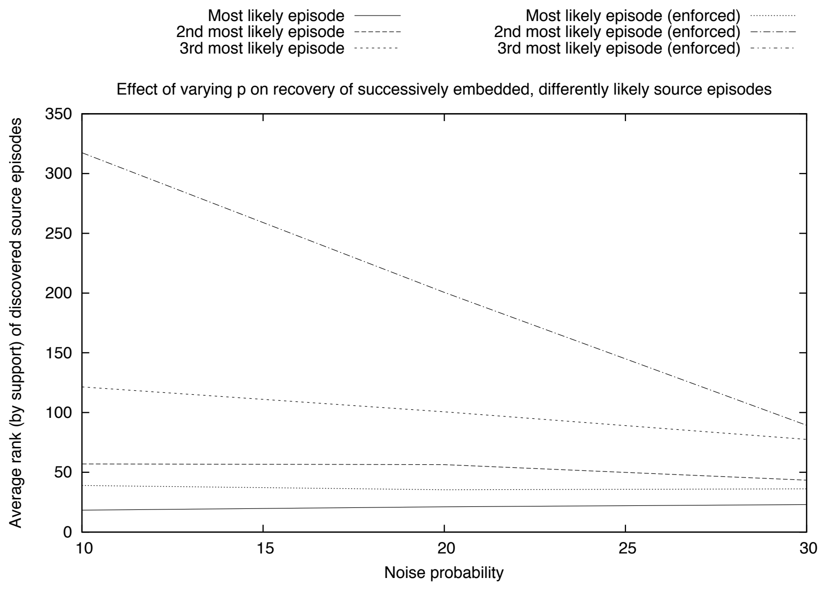

Figure 38: Average rank (by frequency) of successively embedded source episodes of different probability on real life like data for different noise probability, mined using TDMiner

"main" state via its representative episode, and detecting deviations from it. Figures 37 and 38 paint a more positive picture w.r.t. the most likely episode which is ranked low enough by both MinEpi (Gap) and TDMiner that it would be useful. Already the second most likely episode, however, could be problematic to identify. Figure 39, finally, shows that the total output size is larger than for concurrent episodes, making the task of identifying the meaningful episodes harder.

### 6.5.3 Discussion

The experiments on data showing characteristics similar to real life ones are in line with the multi-episode experiments we have performed before: while the gap-constrained techniques are capable of recovering some source episodes, in many cases they also return quite a few spurious patterns mixed in. Especially if some patterns are less likely (and therefore less strongly expressed), it is possible that they are not recovered at all or buried in so many other episodes that they will be hard to identify.

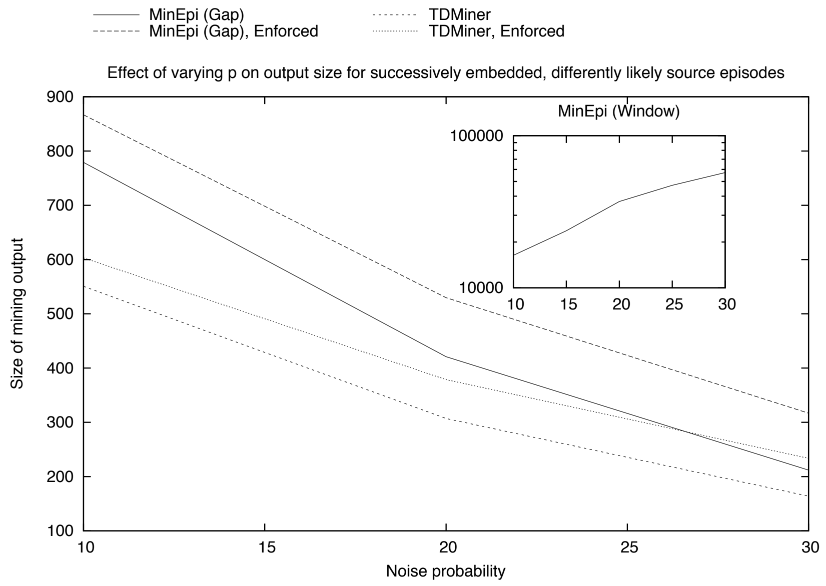

Figure 39: Average sizes of mining result of successively embedded source episodes of different probability on real life like data for different noise probability

# 7 Summary and Conclusions

Unsupervised pattern mining faces a challenge when evaluating methods proposed in the literature since for most real life data the underlying phenomena are unknown. The challenge is even greater for frequent episode mining where not even a common body of real life data sets exists on which all techniques can be evaluated. This means that it is actually unknown how exactly different methods compare to each other in practice and whether any of them are capable of finding underlying patterns at all.

To alleviate this problem, we have proposed to generate artificial data for evaluating episode mining approaches. Artificial data has an advantage over real life data sets in that the embedded patterns are known and can therefore be used to evaluate how successful episode mining approaches are in recovering them. Furthermore, if such a data generator is flexible enough to generate data with different characteristics, mining techniques can be benchmarked much more comprehensively than has been done before.

Our first contribution to addressing this problem consists of describing and implementing such a data generator. Starting from a number of real life observations, we have proposed a number of numerical and nominal parameters that can be manipulated to generated a wide range of different data sets. We proceeded to show that the data used in prior work has very specific characteristics that, while useful for illustrating those techniques' merits, has little in common with real life data at our disposal. We have then gone on to demonstrate how the different parameters of our generator can be adjusted to generate data that mimics characteristics of that real life data.

Our second contribution consists of using our data generator to evaluate the performance of four episode mining techniques. To the best of our knowledge, is this not only the first time that several such techniques have been compared on a large number of data sets, it is furthermore the first time that episode mining techniques are evaluated w.r.t. whether they are able to recover patterns embedded in the data. We varied a number of parameters to generate data with different noise probabilities, time delays, or number of embedded patterns, to name just a few. In addition, we performed experiments on real life like data we generated. We find that for several of the settings a single episode can be recovered relatively well, but also that the problem becomes much more challenging once several source episodes are involved. Interestingly, much of the success of different methods seems to hinge on the temporal constraints used. A maximal window size constraint is difficult to decide on and large window sizes can lead to weak results, while exploring different settings increases the computational cost of episode mining significantly. Maximal gap constraints, however, can be relatively easily derived from the data and when combined with the MinEpi semantic lead to similarly good results as gap constraints and the non-overlapping occurrence semantic.

If we use the results on the real life like data as guidance in exploring results on the actual real life data, we have to draw the conclusion that it is unlikely that all phenomena that occur in the data can be identified using the output of frequent episode mining. This statement comes with a caveat though: the setting is one in which several episodes are embedded in the data, and the output episodes were ranked using support in the data under the different semantics. If episode mining research gives rise to and adopts new measures for identifying the significance of mined episodes, and develops methods for dealing with sets of episodes, the good results achieved for single source episodes should become stronger and settings with several source episodes could be tackled successfully.

We do not consider our proposed generator or the benchmarking experiments we performed as the final step but instead as a beginning. Even though the parameters and their values seem justified to us given a priori considerations and the real life data at our disposal, it is likely that other researchers work with data sets that our generator cannot model yet, meaning that it would need to be extended. In particular, we will continue to add additional distributions for noise events, time delays, event types in source episodes, and source episode weights to the generator. In a similar vein, research in episode mining will continue and new techniques and measures will be proposed. We have shown that the lack of a collection of publicly available data does not have to stand in the way of comprehensively evaluating episode mining approaches. We expect that future work in the field will include more extensive evaluations and comparisons, especially when it comes to mining sets of episodes, so that potential users can find guidance in how to use those potential powerful techniques.

## Acknowledgments

I would like to thank Jan Lemeire and Stijn Meganck for providing the motivation for this work. I am also grateful to my colleagues Tias Guns and Siegfried Nijssens who read and commented on my work and those anonymous reviewers whose constructive feedback help me improve both research and text.

## References

[1] *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada.* ACM, 2002.

[2] Rakesh Agrawal and Ramakrishan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 487–499, Santiago de Chile, Chile, September 1994. Morgan Kaufmann.

[3] Mikhail J. Atallah, Robert Gwadera, and Wojciech Szpankowski. Detection of significant sets of episodes in event sequences. In *ICDM*, pages 3–10. IEEE Computer Society, 2004.

[4] Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors. *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, 2004.

[5] Tijl De Bie. Explicit probabilistic models for databases and networks. *CoRR*, abs/0906.5148, 2009.

[6] Albert Bifet and Ricard Gavaldà. Adaptive xml tree classification on evolving data streams. In Wray L. Buntine, Marko Grobelnik, Dunja Mladenic, and John Shawe-Taylor, editors, *ECML/PKDD (1)*, volume 5781 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2009.

[7] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[8] Hendrik Blockeel and Joaquin Vanschoren. Experiment databases: Towards an improved experimental methodology in machine learning. In Joost N. Kok, Jacek Koronacki, Ramon López de Mántaras, Stan Matwin, Dunja Mladenic, and Andrzej Skowron, editors, *PKDD*, volume 4702 of *Lecture Notes in Computer Science*, pages 6–17. Springer, 2007.

[9] Gemma Casas-Garriga. Discovering unbounded episodes in sequential data. In Nada Lavrac, Dragan Gamberger, Hendrik Blockeel, and Ljupco Todorovski, editors, *PKDD*, volume 2838 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2003.

[10] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1), 2006.

[11] Darya Chudova and Padhraic Smyth. Pattern discovery in sequences under a markov assumption. In *KDD* [1], pages 153–162.

[12] Bart Goethals and Mohammed Javeed Zaki, editors. *FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[13] Bart Goethals and Mohammed Javeed Zaki. Advances in frequent itemset mining implementations: report on fimi'03. *SIGKDD Explorations*, 6(1):109–117, 2004.

[14] E. F. Vogel J. J. Down. A plant-wide industrial process control problem. *Computers & Chemical Engineering*, 17(3):245–255, March 1993.

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

[16] Srivatsan Laxman and P. S. Sastry. A survey of temporal data mining. *Sadhana-academy Proceedings in Engineering Sciences*, 31:173–198, April 2006.

[17] Srivatsan Laxman, P. S. Sastry, and K. P. Unnikrishnan. Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE Trans. Knowl. Data Eng.*, 17(11):1505–1517, 2005.

[18] H. Mannila and H. Toivonen. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210–215. AAAI Press, 1995.

[19] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 146–151. AAAI Press, 1996.

[20] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, 1997.

[21] Nicolas Méger and Christophe Rigotti. Constraint-based mining of episode rules and optimal window sizes. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *PKDD*, volume 3202 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2004.

[22] Abdullah Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney Cash, and M. Brandon Westover. Exact discovery of time series motifs. In *SDM*, pages 473–484. SIAM, 2009.

[23] David M. Pennock and Quentin F. Stout. Exploiting a theory of phase transitions in three-satisfiability problems. In *AAAI/IAAI, Vol. 1*, pages 253–258, 1996.

[24] Jeffrey C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[25] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *KDD*, pages 377–382, 2001.

[26] Nikolaj Tatti. Significance of episodes based on minimal windows. In Wei Wang, Hillol Kargupta, Sanjay Ranka, Philip S. Yu, and Xindong Wu, editors, *ICDM*, pages 513–522. IEEE Computer Society, 2009.

[27] Nikolaj Tatti and Boris Cule. Mining closed strict episodes. In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM*, pages 501–510. IEEE Computer Society, 2010.

[28] Nikolaj Tatti and Boris Cule. Mining closed episodes with simultaneous events. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *KDD*, pages 1172–1180. ACM, 2011.

[29] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *J. Artif. Intell. Res. (JAIR)*, 32:565–606, 2008.

[30] Mohammed Javeed Zaki. Efficiently mining frequent trees in a forest. In *KDD* [1], pages 71–80.

[31] Zijian Zheng, Ron Kohavi, and Llew Mason. Real world performance of association rule algorithms. In *KDD*, pages 401–406, 2001.