Conception of Semantic Complex Event Pattern Mining methods on Event Streams

Focussing on predictive Episode Mining

Leon Bornemann

Department of Mathematics and Computer Science
Freie Universität Berlin

This thesis is submitted for the degree of
Master of Science

July 2016

Declaration

Does the FU have a declaration text in which I declare that I worked on this alone, up to scientific standard, did not copy anything without citing etc? If yes I will put this here.

Leon Bornemann
July 2016

Acknowledgements

And I would like to acknowledge (TODO) ...

Abstract

It is a little early for an abstract, I guess I could already write a preliminary Abstract...

Table of contents

List of figures

List of tables

Chapter 1

Introduction

This chapter serves as a rather broad introduction to the topic of this thesis and provides motivation for the work.

1.1 General Introduction and Motivation

Almost any application domain of information systems has some data that is being generated. Data is available in many different forms. One of these forms are data streams. Data streams are not limited to the recent rise in popularity of video and audio streams. On the contrary the application domains that produce data in the form of streams are very diverse. They include for example constantly running business applications that log business activities and events, sensor networks that report usage data or devices that take measurements of physical quantities (such as temperature, pressure, humidity, etc...) at certain points of time.

The fact that streams generate a constant stream of data and thus lead to a constantly growing database is a significant difference to classic applications of data mining in which there is a static (training) database. Despite that significant difference in the data representation, many fields of interest in the context of static databases remain the same for data streams. Common areas of interest are frequent patterns, predictive patterns, association rules, clustering and classification of the data entities. Approaches and algorithms that solve these problems for static databases, while by no means fully researched, are rather well known and evaluated. Applying these methods to data streams can present challenges and may demand many modifications due to the large and possibly infinite amounts of data produced by streams. Naturally, data mining methods for stream data must be especially fast, scalable and memory efficient.

Apart from the additional, algorithmic constraints on memory and computation time, data streams also present conceptual challenges. In contrast to static databases streams may evolve over time, which can make it very difficult for algorithms to assess which past data of the stream should be considered when analyzing the currently incoming data. Recognizing these so called concept drifts is one challenge among many when processing or mining data streams.

A suitable way to look at most data streaming scenarios is that of event streams. An event can be anything that happens in the real world, which can be represented as an element of the stream. These events are commonly referred to as basic or simple events. A frequent area of interest when processing event streams is to mine complex events that consist of multiple basic events with different relationships between each other. Discovering interesting complex event patterns can be tough, especially since there may be a lot of potential candidates. Often we are only interested in specific event combinations. One possible approach to improve the mining process is to use domain knowledge. If the domain knowledge about the underlying event stream contains semantic information about the different event types it is possible to use that knowledge directly in the mining process. Semantic knowledge can take many different forms, a common one being an RDF-graph that can be examined using queries.

Figure 1.1 presents the basic idea of semantic complex event mining algorithms. On the lowest level of abstraction we have the low-level event stream, which is the unrefined data coming directly from the sources (for example sensor data). The low-level stream needs to be transformed in some way to an annotated event stream, which then in turn gets mined to discover complex events. The mining algorithm's basic input is the stream of annotated events, but it can also use the previously mentioned semantic knowledge, an ontology, which contains additional information about the event types.

In terms of the very broad term of complex events this thesis will focus on the subtopic of episode pattern mining from stream data or very large log files or databases. Episodes are a specific kind of complex events and are formally defined in chapter 2. For now it is sufficient to know that episodes are essentially complex events in which single events or entire episodes can be combined using two operators: the conjunction and the sequence operator. Figure 1.2 visualizes a simple episode pattern and example occurrences in a data stream.

So why is the mining of episodes of interest? There are many real-life use cases in which episode discovery is relevant. The discovery of frequent episodes can for example be related to the discovery of underlying models for data generation, which can help to better understand the data generation process. Another application is to find predictive episodes, meaning episodes which can help to predict events occurring in the future. This has already been applied to predict outages in Finnish power grids (TODO: find a citable source for this). The discovery of predictive episodes is relevant for many domains, for example sensor networks (if a certain chain of events leads to failure, predictive episodes can be used to

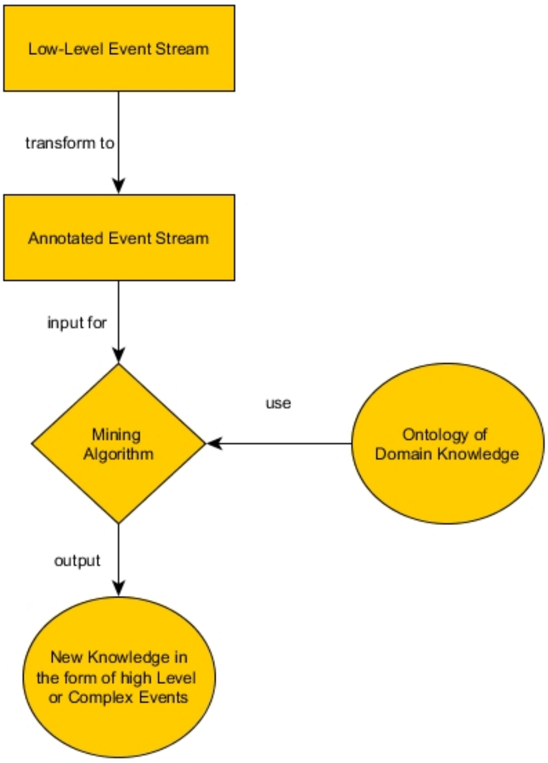


Fig. 1.1 General structure of a semantic mining process of complex events

preemptively expect failures and react accordingly). The domain that will be used in the evaluation of this thesis is stock market prediction. Both predicting the overall direction of the stock market and predicting whether individual stocks will rise or fall are a difficult problems that have the obvious application of generating investment strategies. Apart from this, predictive episodes also have use cases in the enforcement of regulations. Predictive episodes could for example be used to detect illegal price arrangements of certain companies, which have been known to happen between oil and gas stations of major companies (TODO: find and cite a source for this).

In contrast to other regression and forecasting methods, such as artificial neural networks, predictive episodes have the advantage that once they are discovered they can make ad-hoc predictions in a fast moving data-stream, whereas neural networks usually forecast the closing values of stock markets of the next day based on the values of previous days. This gives predictive episodes a niche: fast moving (real-time) data-streams that need quick predictions of future events.

The rest of the thesis is outlined as follows: Chapter 2 introduces the basic terminology, reviews the related work, and gives a detailed, formal introduction to episode mining. Chapter **??** presents the suggested algorithm to mine predictive episodes from event streams in detail

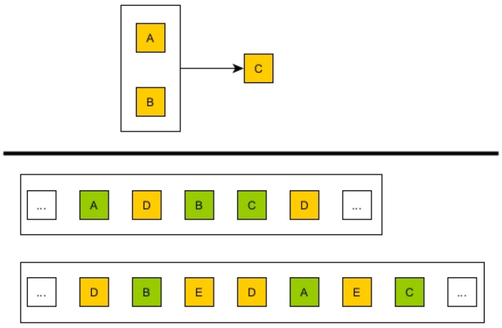


Fig. 1.2 The top half visualizes an example episode pattern, which consist of a conjunction (A and B must both occur, but the order does not matter) and a sequence (C must occur after A and B). The bottom half shows two windows of an example stream, in which occurrences of the episode are shown in green.

and analyzes its complexity. Subsequently chapter **??** presents an evaluation using both synthetically generated data, as well as real stock market data. Finally chapter **??** concludes the paper and mentions possible future work.

1.2 Problem Definition and Exact Research Question

Even though many terms have not yet been precisely defined (which will be done in 2) this section presents a loose, intuitive definition of the problem to be tackled by this thesis. Given a stream of basic events and a special event type P one is interested to predict occurrences of that special event type P in the stream, so that actions can be taken before P actually occurs. For this purpose we are interested to discover episodes in the stream of which occurrences are likely to be followed by P. A window of an example stream is visualized in figure 1.3.

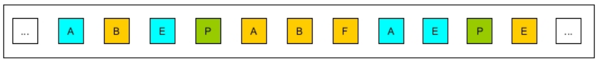


Fig. 1.3 The figure visualizes an event stream in which events of type P are to be predicted. The occurrences of the possible predictive episode A followed by E is colored in teal, whereas events of types P are colored in green and all other events are yellow

Note that it is very important to the mining process how much time is allowed to pass between the occurrence of the predicting episode and the event that is to predict. If this time span is too small a mining algorithm might not be any good predictors. If the timespan is too

large a mining algorithm might be overwhelmed by the number of candidates that need to be considered, since the more time may pass the larger are the sequences before each event P that must be considered.
To solve the problem outlined above a semantic mining algorithm will be suggested in this thesis. Since the underlying data structure to be analyzed are data streams, the algorithm must have the following properties:

- The algorithm must be able to adapt to a changing context (as the stream progresses, the underlying model may change completely)
- The recognition of predicting episodes must be quick, since streams, especially in the domain of stock markets, can have a high velocity. This makes it important to be able to quickly predict events. If the prediction takes too long, the event which we want to predict may have already occurred before the algorithm outputs its prediction, thus robbing the user of the opportunity to take action.
- The Algorithm must not require to store the entire stream of events seen so far, since that is not feasible for most streams.

The algorithm will be evaluated on both synthetically generated data and real-life datasets. The latter ones will be from the domain of stock market prediction, in which the developed algorithm will be empirically compared to other approaches in terms of accuracy measures and execution time.

In summary the research question in particular is:

How can event streams be mined for episodes that effectively predict certain event types and how can domain knowledge be used to improve the results or speed of the mining algorithm?

Chapter 2

Related Work

This chapter reviews the related work relevant for this thesis. Since there are many different related work areas that all have their relevance to this thesis, this chapter is divided into different sections dealing with each of them in turn. First section 2.1 introduces and defines basic the basic terminology that will be used in this thesis. Subsequently section **??** gives a broad overview over the rather general area of mining data streams with a focus on prediction. Finally section 2.3 introduces the concept of episodes and summarizes different episode mining approaches.

2.1 Basic Definitions and Terminology

This section introduces relevant definitions and terminology that was introduced in previous work and will be used in this thesis.

2.1.1 Event Processing Terminology

The basic event terminology in this subsection is paraphrased from the event processing glossary created by the Event Processing Technical Society [27]. Note that some of the definitions may be slightly altered or simplified. This is due to the fact that the event processing technical society uses these terms for a very general description of event processing and event processing architectures and thus some original definitions are more complex than what is needed in this thesis. The definitions given here aim to establish a clear terminology for this thesis.

Definition 1 ***Event*** *An event is either something that is happening in the real world or in the context of computer science an object that represents a real world event and records its* *properties. The latter can also be referred to as an event object or an event tuple. Note that the term is overloaded, but the context usually gives a clear indication of what is meant.*

The event processing society claims that the context usually solves the ambiguity in the above definition. Since this may not always be the case we will only use the term *event* in this thesis to refer to event objects unless it is otherwise specified.

Definition 2 ***Simple Event*** *A simple event is an event that is not viewed as summarizing, representing, or denoting a set of other events. Sometimes also referred to as a basic events.*

These two definitions can sometimes cause confusion. It is important to note that the term event is the most general term, since it can refer to any kind of event, be it simple, derived or complex (see definitions 3 and 4). A simple event however is the most basic form of an event and often the ingredient for the creation of more complex events: Given simple events it is possible to derive events from those or the absence of those. For example the absence measurement events of a sensor could be used to derive the event of that very same sensor becoming defect. These events are called derived events:

Definition 3 ***Derived Event*** *A derived event or synthesized event is an event that is generated according to some method or based on some reasoning process.*

It is also possible to combine multiple simple events to form what we refer to as complex events:

Definition 4 ***Complex Event*** *A complex event is a derived event that is created by combining other events. The events can be combined by using certain operators, for example disjunction, conjunction or sequence. An example would be* $(A \wedge B) \rightarrow C$ *(event A and B in any order followed by event C).*

This is a very broad definition of complex events. The choice of allowed operators strongly impacts the expressiveness of complex events. A specific kind of complex events are episodes (see section 2.3), which will be the main focus of this thesis. The next notion that needs to be considered is that each individual event normally belongs to a certain class of events, which we refer to as the event type:

Definition 5 ***Event type*** *The event type, sometimes also referred to as event class, event definition, or event schema is a label that identifies events as members of an event class.*

Another important term that was not explicitly defined in the event processing glossary, but is very relevant to the topic at hand is the notion of a type alphabet:

Definition 6 ***Type Alphabet*** *The type alphabet, often simply called the event alphabet, is the set of all possible event types that can occur in the observed system.*

Event alphabets are often implicitly defined when mining frequent itemsets, patterns or episodes. So far we have looked at events without considering the scenario we are most interested in which are event streams. To do so we need the notion of timestamps:

Definition 7 ***Timestamp*** *A time value of an event indicating its creation or arrival time.*

Given that we can define an event stream:

Definition 8 ***Event Stream*** *An event stream is an ordered sequence of events, usually ordered by the event timings.*

Note that this rather broad definition of an event stream does not assume anything about the kind of event that is contained in it. A stream can contain very basic forms of events (simple events) but can also be made up out of derived events or even complex events. Also other properties for example that the stream is constantly updating (new events coming in) are not considered yet.

2.2 Data Streams

As already mentioned in the introduction data streams present a challenge to data miners in many ways. This section aims to give both a general overview over the broad research area of mining data stream as well as specifically cover the related topics of forecasting and stock market prediction.

2.2.1 Data Stream Mining

Many research areas in classical data mining are also of interest when processing streams. As already mentioned in the introduction however, data streams impose severe restrictions on the algorithms, such them having to use only one pass over the data and that they should be incremental. Normally data mining algorithms for the classical scenario in which the underlying data is a static database or even a data-set that fits into main memory have these properties. Thus the algorithms need to be modified and often approximations have to be made. A comprehensive basic overview over the application of different data mining tasks and how they can be applied to streams was comprised by Gaber et. al. [15]. The authors structure the state of the art in data streaming according to a small taxonomy with three basic parts:

1. Theoretical Foundations
2. Mining Techniques and Applications
3. Systems

The taxonomy is visualized in figure

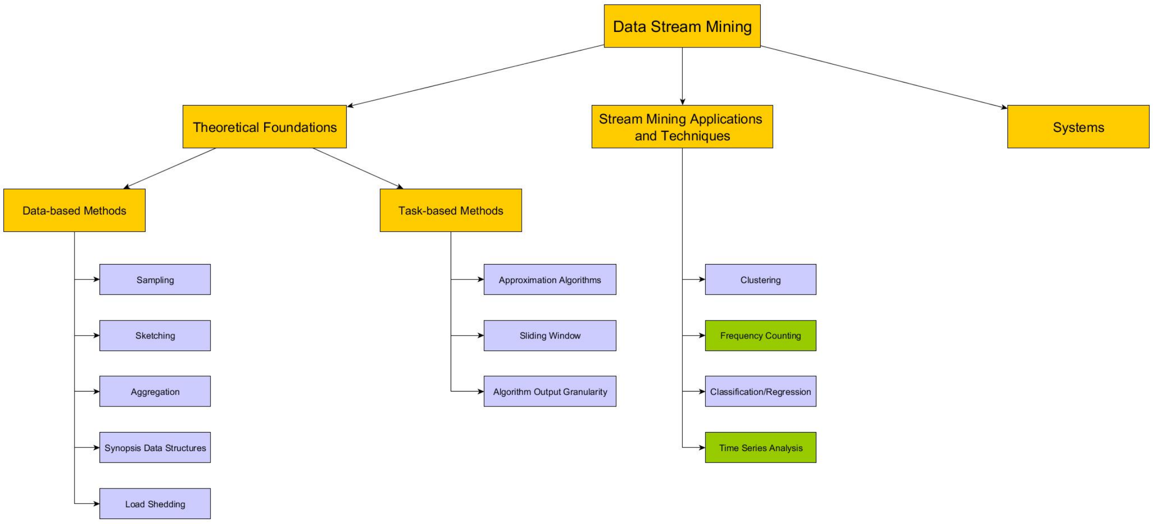


Fig. 2.1 The taxonomy of the research areas of data stream mining as presented by Gaber et. al. [15]. The subcategories that are marked green are the categories to which this thesis will make contributions.

In the theoretical foundations they present general approaches on how to deal with the issues of data streams. They distinguish between data-based techniques and task-based techniques. Data-based techniques aim to reduce the data being processed or apply transformations on the data stream, while task-based techniques are modifications of existing techniques to meet the requirements for time and space. The subcategories of data-based techniques are:

- **Sampling:** The idea of sampling is basically the same as in statistics. Instead of processing the whole stream, only a subset of the stream is looked at. Due to the unknown size of the stream, sampling methods are more complex than in static systems like databases (TODO: extra reference).
- **Load Shedding:** Load shedding is similar to sampling in that it simply drops certain incoming data without processing it, however load shedding is a dynamic approach that is only applied if needed, for example if there are volume spikes (large amounts

of data coming in suddenly). Thus in contrast to sampling, which is applied over the whole lifetime of the stream, load shedding is a more reactionary strategy [6].

- **Sketching:** When we speak of sketching we refer to the process of building a summary of the data stream that only uses very little memory compared to the whole stream. Examples of this are the so-called frequency moments [5]
- **Synopsis Data Structures:** Synopsis data structures summarize the stream in data structures that are then used to approximately answer queries. TODO: difference between synopsis and sketches.
- **Aggregation:** Aggregation is mainly useful when statistical measures are to be computed over a stream. TODO: reference?

The task-based techniques mentioned by the authors are:

- **Approximation Algorithms:** Approximation algorithms are common for hard problems (such as NP-complete problems) but also many classic data-mining problems can be solved on data streams using approximate variants of the original algorithms. A commonly cited example are approxiamtion algorithms for the mining frequent items or itemsets, such as the sticky sampling or the lossy counting algorithm [30].
- **Sliding Window:** The usage of sliding windows over the data streams is common when the user is only interested in the most recent events as opposed to the complete history. The main challenge here is to construct algorithms that work with the incremental updates that happen whenever new data arrives (the window slides forward). (TODO: cite something)
- **Algorithm Output Granularity:** The term algorithm output granularity refers to the strategy of dynamically reacting to the available memory and fluctuating data rates. The basic idea is to mine the incoming data stream as long as possible (normally until the device runs out of memory). If this happens the generated knowledge structures are merged and summarized in order to free up memory and continue the mining.

The actual mining techniques and applications are well known from classical data mining:

- **Clustering** TODO
- **Classification and Regression** Classification and regression in evolving data streams presents the additional challenges of concept drift, which means that the underlying class distribution may change, which will make the originally built model invalid over time. TODO: cite stuff here

- **Frequency Counting** TODO
- **Time Series Analysis** TODO

The last category in the taxonomy is less focused on research issues and conceptual or algorithmic problems but instead lists the existing data stream processing systems (to that date). Since that is less relevant to this thesis we do not mention or describe existing systems but instead refer the interested reader to the original paper by Gaber et. al. [15].

TODO: Note that the paper by Gaber et. al. was published in 2005, so quite a lot of work has been done since then, which means that these are obviously not present in their overview. An example for this is the popular hadoup system (TODO: verify that). However the established taxonomy is still very useful, since it brings structure to the large field of data stream mining.

As already mentioned many data mining algorithms for data streams make compromises of some sort or employ appoximations. Especially in the area of pattern mining there are quite a few approaches that use approximations or focus on the most recent data. Gianella et. al. have for example developed an algorithm to incrementally maintain approximately frequent itemsets for the most recent time windows using tilted time window frames [16]. Approaches to mine frequent items without tilting the time windows exist as well, such as the sticky sampling or the lossy counting algorithm [30]. These algorithms can be generalized to mine frequent itemsets, too. If this is done however pruning the candidates may become an issue if the main memory is small (the number of candidates to maintain may be too large). A paper also related to regression and time series forecasting was published by Papadimitriou et. al. [35]. The authors tackle the problem of cyclic time series mining, as well as time series prediction of numerical values in potentially infinite data streams.

The rise in popularity of mining data streams also pushes research areas like distributed data mining, in which parallelized versions of data mining algorithms are researched. Large data streams and distributed systems often go hand in hand, which creates the need for distributed algorithms. Such solutions have been researched for example for frequent pattern mining [26], association rule mining [3], clustering [18] and classification (in this case a distributed boosting algorithm) [24].

A slightly different direction to data stream processing is querying data streams. Some contributing data scientists have thus approached streams in a similar manner as relational databases [34]. The authors present their progress at building a general purpose Data Stream Management System (DSMS) prototype, as a pendant to the traditional Database Management Systems (DBMS). They suggest the usage of an extension of SQL as a stream query language to allow for continuous queries and discuss approximation techniques.

2.2.2 Classification and Regression

Classification and regression are the main applications of supervised learning. They are very similar and closely interlinked to each other. Classification aims to assign new, unseen objects to a class based on a model that was created from training data. Class labels are categorical. Regression is similar, but instead of assigning categorical class labels to new data points it is now the goal to generate a numerical (real) value that is close to the actual value. A comprehensive overview over most classification approaches was comprised by Kotsiantis et. al. [20]. The authors distinguish between different types of classifiers:

- logical classifiers such as Decision Trees and rule based algorithms
- preceptron based classifiers, such as the single layered perceptron or artificial neural networks
- statistical classifiers, such as naive bayes or bayesian networks
- instance based classifiers, such as k nearest neighbor
- maximum margin classifiers, such as support vector machines

It is notable that the authors do not mention ensemble learners [12], which combine several classifiers and classify new examples by letting each classifier vote. Arguably the most notable ensemble learning technique is the random forest [25]. It is important to keep in mind that most classification approaches can also be tweaked to do regression, a random forest for instance can either be used to estimate a (binary) class label (classification) or a numerical value (regression). Classification and regression have also been attempted by using pattern mining and association rule generation [28]. Instead of mining all association rules the authors focus on finding so called class association rules. An especially relevant application of rule based regression was the usage of classification rules using minimal rule generation for the prediction of equity returns [2]. The authors modify a classification rule generation technique called R-MINI to be able to do regression. They evaluate their approach on historical stock market data from the S&P 500 data-set (data is aggregated to monthly values).

2.2.3 Prediction and Financial Time Series Forecasting

Similar to regression and classification, but not quite the same is prediction. Prediction (sometimes also called forecasting) introduces a new dimension, which is time. The task in prediction is to build a model that if given data points on a timeline will predict which data point will show up next. Due to its dependency on time, prediction plays a large role when mining time series data. Arguably the most popular models for time series prediction are artificial neural networks which are used by multiple authors with differnt variations and application domains [11] [33] [14]. Of particular interest for this thesis are of course prediction approaches that were used in the same domain, namely stock market or financial time series prediction. Naturally this has been looked into. Gestel at. al. used support vector machines to predict the closing values of stock market indices [36].Another, more recent approach used an artificial neural network with an improved learning algorithm (by integrating improved bacterial chemotaxis optimization into the back propagation of the neural network) to successfully predict the Standard's & Poor's 500 index (changes were aggregated to daily values). Another different approach is to use grey system models to predict time series [19], in this specific case the authors predict the daily foreign currency exchange rate of euro to dollar. A comprehensive comparison of different models for predicting the S&P CNX NIFTY index was carried out by kumar et. al. [21]. The best performing model in their study is the support vector machine, closely followed by a random forest. This is interesting, since random forests have not received as much attention for time series prediction as for example artificial neural networks. Of course no general conclusions can be drawn from the performance of these models for one index. An very detailed overview over a lot of work in the area can be found in the literature study done by Atsalakis et al. [4] although the focus seems to be on papers that use neural networks or support vector machines. The attentive reader will have noticed that most of the previously named examples focus on predicting stock indices, a.k.a the general direction of the market. Predicting individual stock values has been looked at less, but there is of source some previous work to be considered. For example mahfoud et. al. compare genetic algorithms to neural networks for the prediction of individual stocks [29].

2.2.4 Complex Event Processing and Episode Mining

Before talking about the state of the art in complex event processing it is first important to clarify the term *"event"* which is a is very broad term that is used in many different areas of science. Even when restricted to computer science there may be different definitions with subtle differences. As already explained in the introduction this thesis will focus on events as defined by the glossary of the event processing society [27]. When talking about processing complex events there is an important difference between two cases:

- The patterns of interest are known before looking at the data. This is called complex event detection.

- There is no prior knowledge about which patterns might be interesting, they need to be discovered while looking at the data. This is called complex event discovery or complex event mining (which basically means that data mining methods need to be employed)

Complex event detection usually revolves around specification and query languages for complex events [13]. Quite a few different specification and query languages have been developed, such as SNOOP [10] or the SASE event processing language [37].
Discovering interesting complex events of arbitrary structure in data streams is a very challenging task, thus most work focuses on specific types of complex events. A popular example is mining frequent sequences of events (basically complex events that only use the sequence operator) [9] [17].
This thesis deals with a more expressive type of complex events: Episodes. Originally episodes were researched without any relation to event processing [31]. Episodes have roughly been categorized as serial, parrallel or composite and there are different mining methods proposed for each of these [31] [38]. The connection between episodes and hidden markov models was also explored in a PHD thesis by Laxman [22]. Evaluating episode mining algorithms on real-life datasets is often difficult due to a lack of knowledge about the ground truth. Thus, generation of realistic, synthetic datasets has been looked into as well [39].

2.2.5 Semantic Web

TODO

2.3 Episodes

This section gives a detailed introduction to the topic of episode mining and presents an overview over the most relevant related work.

2.3.1 Basic Definitions

As already mentioned, episodes are complex events whose basic building blocks are simple events. Note that in order to make use of episodes we require that all simple events have a type and we have a finite, previously known event alphabet, that contains these types, which we refer to as Σ. We follow up with a formal definition:

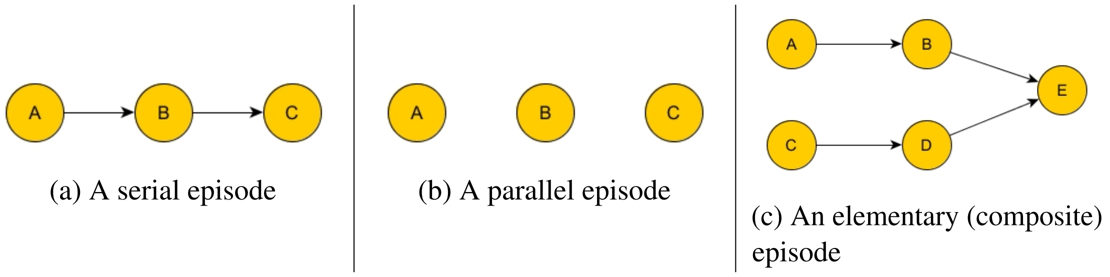
Definition 9 ***Episode*** *An episode (also sometimes called episode pattern or elementary episode)* α *of length m (also called m-episode) is defined as a triple:* $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ *where* $V_\alpha = \{v_1, ..., v_m\}$ *is a set of nodes,* $\leq_\alpha$ *is a partial order over* V_α *and* $g_\alpha : V_\alpha \rightarrow \Sigma$ *is a mapping that maps each node of* V_α *to an event type [31].*

Put more simply an episode is a multiset of event types, whose elements can be, but do not have to be ordered by a relation ($\leq_\alpha$). Another way of putting it is that an episode is essentially a partially ordered sequence of events. Before we look at examples there are a two special types of episodes that need to be mentioned since they have received the most attention in the available literature. These are called serial and parallel episodes:

Definition 10 ***Serial Episode*** *An episode* $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ *is called a serial episode if* $\leq_\alpha$ *is a total order [31].*

Definition 11 ***Parallel Episode*** *An episode* $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ *is called a parallel episode if* $\leq_\alpha = \emptyset$*, in other words if there is no ordering imposed on* V_α *at all [31].*

Essentially, serial episodes are sequences, while parallel episodes are multisets. Figure 2.2 visualizes example episodes as directed acyclic graphs (DACs), which is a useful way to visualize episodes.



(a) A serial episode

(b) A parallel episode

(c) An elementary (composite) episode

Fig. 2.2 Different example episodes visualized as directed acyclic graphs

In fact each episode can be formally transformed to a DAC using the following simple procedure: Given an Episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$, create the corresponding DAC $G = (V, E)$ by executing the following:

1. For each $v \in V_\alpha$ add v to V and label v with $g_\alpha(v)$
2. For each pair $v, w \in V_\alpha$ where $v \leq_\alpha w$ add edge (v, w) to E

The original paper by Manilla et.al. [31] also introduces the notion of composite episodes. We repeat the definition here:

Definition 12 ***Composite Episode*** *An episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ is called a composite episode if $g_\alpha : V_\alpha \rightarrow \Sigma \cup C^*$, where C^* is the set of all composite episodes.*

This recursive definition of composite episodes may be confusing at first but it has the advantage that any elementary episode can be represented as a composite episode which is exclusively a serial or parallel composition of serial, parallel or composite subepisodes (see definition 13).

Interestingly, there are other parts of the related work that use the term *composite episodes* but deviate from definition 12. For example Baathorn et. al. propose a method for finding composite episodes [7]. However they define composite episodes as a sequence of parallel episodes, which is more restrictive than the original definition. Also Baumgarten et. al. use this definition [8] when they present an approach to mine descriptive composite episodes. Note that not all elementary episodes can be represented as sequences of parallel episode. A simple example shown in figure 2.3 illustrates this. If the presented episode were to be represented as a sequence of parallel episodes obviously A and B would have to be in different parallel episodes in order to fulfill the requirement that B must be after C. After that the problem is that it is impossible to assign C to any of those sets. If it gets assigned to the same parallel episode set as A then this would prohibit A and B occurring first followed by C, which is allowed in the original definition. Likewise if C gets assigned to the same parallel episode as B then that eliminates the possibility of C occurring before A and B, which once again was allowed in the elementary episode. In this thesis we will stick to the original definition as presented in definition 12.

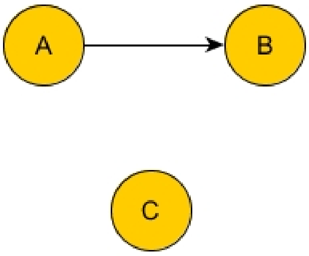


Fig. 2.3 An elementary episode that can not be represented as a sequence of parallel episodes

If we want to denote quick, simple episodes formally without drawing a DAC, we will use $\rightarrow$ as the sequence (ordering) operator. To show that there is no order specified between two nodes we use $||$ as the parallel operator. For example $(A \,||\, B) \rightarrow C$ denotes a composite episode of length 3, which specifies that it does not matter in which order A or B occur, but C must occur after both A and B. If we want to discuss more complex episodes we will

visualize them graphically like in the previous figures
The notion of sub- and superpatterns, which is very important for most pattern mining applications also applies to episodes, as shown in the next definition.

Definition 13 ***Subepisode*** *An episode $\beta = (V_\beta, \leq_\beta, g_\beta)$ is said to be a subepisode of episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ if all of the following conditions hold:*

1. $V_\beta \subseteq V_\alpha$
2. $\forall v \in V_\beta \ : \ g_\beta(v) = g_\alpha(v)$
3. $\forall v, w \in V_\beta, v \leq_\beta w \ : \ v \leq_\alpha w$

In this context we also refer to α as the superepisode of β [31] [23].

It is important to note that in the original definition of sub- and superepisodes by Mannila et. al. [31], the first property of the above definition was actually defined as $V_\beta \subset V_\alpha$, which implies that subepisodes would always have to consist of at least one less node than their superepisodes. This definition was changed by Laxman et. al. [23] to allow set equality as well. The implications of this change are that parallel episodes like $A||B$ are now subepisodes of their serial counterparts of the same length $A \rightarrow B$. In this thesis we stick to the definition that allows set equality between super- and subepisodes.
It is helpful to think of episodes as a template or pattern for concrete occurrences. In order to define what we mean by an episode occurance, we first need to formally introduce the notion of an event sequence.

Definition 14 ***Event sequence*** *An event sequence is defined as an ordered list of tuples $S = [(T_1, t_1), ..., (T_n, t_n)]$ where $T_i \in \Sigma$ is the event type of the i-th event and $t_i \in \mathbb{N}^+$ is the timestamp of the i-th event. The sequence is ordered according to the timestamps, which means that $\forall i, j \in 1, ..., n \; i < j \implies t_i \leq t_j$.*

Definition 15 ***Episode Occurrence*** *An event episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ is said to occur in a sequence S if events of the types that the nodes in V_α are mapped to by g_α, occur in S in the same order that they occur in the episode. More formally if we are given a sequence of events $S = [(T_1, t_1), ..., (T_n, t_n)]$ we can define an occurrence of α as an injective Map $h : V_\alpha \rightarrow \{1, ..., n\}$, where $g_\alpha(v) = T_{h(v)}$ and $\forall v, w \in V\alpha : v \leq_E w \implies t_{h(v)} \leq t_{h(w)}$ holds [31].*

2.3.2 Episode Detection and general Mining Algorithm

When discovering episodes in a sequence one is usually interested in those episodes that occur frequently, meaning more often than a user defined threshold. This is similar to all the different kinds of pattern mining algorithms, such as the apriori algorithm for finding frequent itemsets [1]. A general algorithm for mining the episodes occurring frequently in a sequence is given in algorithm 1. The algorithm is very alike the basic apriori algorithms, since it uses a level-wise, breadth first search by first identifying all frequent episodes of a certain length i and then uses these frequent episodes to generate candidates (possibly frequent episodes) of length $i+1$. In order for this to be correct, episode frequency must follow the apriori principle [1], which formally means that if an episode is frequent, all its subepisodes must be frequent as well. Intuitively one would assume that this is always true for episodes but strictly speaking this depends on the definition of episode frequency. However, any frequency definition of episodes that does not satisfy the apriori principle would be highly questionable to say the least, since this would eliminate the possibility of an efficient candidate generation that prunes those episodes which have infrequent subepisodes. To the best of our knowledge all frequency definitions proposed in the literature satisfy the apriori principle.

Algorithm 1 General mining algorithm for frequent episodes

```
function EpisodeMining
    $C_i \leftarrow$ Episodes of Size 1
    $freq \leftarrow \emptyset$
    $i \leftarrow 1$
    while $C_i \neq \emptyset$ do
        Count frequencies of each Episode $E \in C_i$
        $L_i \leftarrow \{E \mid E \in C_i \wedge C_i\ is\ frequent\}$
        $freq \leftarrow freq \cup L_i$
        $C_{i+1} \leftarrow$ Generate Episode Candidates of length $i+1$ from $L_i$
        $i \leftarrow i+1$
    return $freq$
```

In summary, the general mining algorithm for frequent episodes requires:

- A definition of episode frequency, that does not violate the apriori principle
- An algorithm for counting episode frequency (of concrete candidates) according to this definition

- An algorithm to generate candidate episodes

It may be a bit confusing that we need a definition of episode frequency for such a mining algorithm. Since we have already defined what an occurrence of an episode looks like it would seem that counting all occurrences of an episode would yield its frequency. While this is a possible definition of frequency, it is important to note that finding all occurrences of an episode within a sequence is neither practical nor useful. An example will demonstrate the problem with this. Consider the simple serial episode $A \rightarrow B$ and a sequence of length $2 \cdot n$ which repeats the subsequence (A,B) n times. One quickly realizes that the number of episode occurrences is very large due to the possibility of overlapping episode occurrences. In this particular case there are already $\frac{n \cdot (n+1)}{2}$ possible occurrences. This number swiftly increases with the size of the episode pattern, since it introduces more potential overlappings. Naturally the number of possible parallel and composite episode occurrences is even larger, since they are less restrictive in the order of the events. Additionally, such a frequency definition would violate the apriori principle, since subepisodes can have less occurrences than their superepisodes. Consider the example sequence $[A,B,C,A,B,C]$ (timestamp values are left out). In this sequence there are 3 distinct occurences for the episode $A \rightarrow B$ and four distinct occurrences for its superepisode $A \rightarrow B \rightarrow C$. These detrimental effects of this naive frequency definition are nicely summarized by Laxman et. al. in a paper presenting the non-overlapped frequency definition [23].
This leads to various frequency definitions of episodes in the literature, which will be dealt with in subsections 2.3.3 and 2.3.4. Each frequency definition comes with its own frequency counting algorithm. The procedure for generating candidates is independent of the frequency definition and is presented in 2.3.5.

2.3.3 Window based frequency

To the best of our knowledge the window based frequency was the first frequency definition for episodes to gain general popularity. It was conceived by Mannila et. al. [31], although the frequency counting algorithms were only mentioned in text form. The same authors specified the algorithms in a later paper [32], which acts as the primary source for the overview given in this subsection. In order to define the window based frequency we first need the notion of a time window:

Definition 16 ***Time Window*** *Given a sequence of events S we define the Time Window $W(S,q,r)$ with $q,r \in \mathbb{N}^+$ and $q < r$ as the ordered subsequence of S that includes all events* *of the annotated event stream S that have a timestamp t where $q \leq t \leq r$. We call $w = r - q + 1$ the size of Window W.*

Definition 17 ***Episode Frequency - Window based Definition*** *Given a sequence of events S, a fixed window size of w and an episode α, we define the window based frequency $w\_freq(\alpha)$ as the number of windows W with size w of S in which α occurs: $w\_freq(\alpha) = |\{W(S,q,r) \mid r - q + 1 = w \wedge \alpha \text{ occurs in } W\}|$.*

For example given a sequence $S = [(12,A),(14,B),(19,C),(22,A),(34,D)]$ the Episode $\alpha = B \to A$ occurs in window $W(S,14,22)$.
This definition can be confusing at first since it is intended that episode occurrences that are comprised of the exact same events count just as many times as there are windows in which the events appear. If we have a window size of $w = 11$ for the previously mentioned example, we can find the episode $B \to A$ in the consecutive windows $W(S,12,22)$, $W(S,13,23)$ and $W(S,14,24)$, which means we will get a frequency of 3 just for the two events $(14,B)$ and $(22,A)$. This effect obviously increases with the window size. Note that for each episode α we only count one occurrence per window W, no matter how many occurrences of α there are in W.
When determining the window based frequency the naive approach would be to check each window of the sequence separately. Since the windows are adjacent there is a better approach, which makes it possible to only iterate over the sequence once and determine the window based frequency for each candidate episode. Most papers focus purely on parallel and serial episodes and do not give an algorithm for composite episodes. The algorithms to determine the window based frequency of serial and parallel episodes are given in algorithm 2 and 3 respectively. The basic ideas are described below.

Frequency Counting of parallel Episodes

The algorithm for counting the frequency of parallel episodes uses the following data structures and variables:

- For each event type A store the number of occurrences of this event in the current window in $A.count$
- For each episode α store
  - $\alpha.freq$ - the number of windows in which alpha occurred so far
  - $\alpha.eventCount$ - the number of events of $alpha$ that are present in the current window

  - $\alpha.inwindow$ - this variable gets set to the current time whenever α becomes fully present ($alpha.eventCount = |\alpha|$).

- additionally maintain $contains$ which is a set of sets. Each set in $contains$ is identified by a tuple (T, i), where $T \in \Sigma$ is an event type and $i \in \mathbb{N}^+$. These sets are referred to as $contains(T, n)$. The set $contains(T, n)$ contains all candidate parallel episodes, which contain the event type T exactly n times. This is done to be able to efficiently access candidates, when certain events enter or leave the window.

The algorithm works as follows. First the above mentioned variables and index structures are initialized. Subsequently we perform the main loop which iterates through every point of time between the start and the end time of the sequence (TODO: they start earlier for some reason?). Essentially each iteration is sliding the window one step forwards. The only two things that need to be handled inside the loop are new events coming into the window and old events dropping out of the sliding window. If a new event comes in its count gets updated and each episode α that is affected by this will get its event count updated and, if it is completed, the current time will be saved in $\alpha.inwindow$.

If an event A drops out of the window, for all episodes α that occurred in the previous window(s) and now no longer have an occurrence in the current window (by losing A) the number of windows they were present in gets added to $\alpha.freq$.

Algorithm 2 Calculate Window based Frequency for parallel Episodes

Require: Let C be the set of candidate parallel episodes, and let $S = [(T_1,t_s),...,(T_n,t_e)]$ be a sequence of events, let *win* be the window size and finally let *minS* be the minimum support.

```
// Initialization
for each $\alpha \in C$ do
    for each $A \in \alpha$ do
        $A.count \leftarrow 0$
        for $i \in \{1,...,|\alpha|\}$ do
            $contains(A,i) \leftarrow \emptyset$
for each $\alpha \in C$ do
    for each $A \in \alpha$ do
        $a \leftarrow$ number of events of type $A$ in $\alpha$
        $contains(A,a) \leftarrow contains(A,a) \cup \{\alpha\}$
    $\alpha.eventCount \leftarrow 0$
    $\alpha.freq \leftarrow 0$
// Recognition
for $start \leftarrow t_s - win + 1$ to $t_e$ do
    //Bring new events to the window
    for each $(t,A) \in S$ where $t = start + win - 1$ do
        $A.count \leftarrow A.count + 1$
        for each $\alpha \in contains(A,A.count)$ do
            $\alpha.eventCount \leftarrow \alpha.eventCount + A.count$
            if $\alpha.eventCount = |\alpha|$ then
                $\alpha.inwindow \leftarrow start$
    // Drop old events out of the window
    for each $(t,A) \in S$ where $t = start - 1$ do
        for each $\alpha \in contains(A,A.count)$ do
            if $\alpha.eventCount = |\alpha|$ then
                $\alpha.freq \leftarrow \alpha.freq + \alpha.inwindow - start$
            $\alpha.eventCount \leftarrow \alpha.eventCount - A.count$
        $A.count \leftarrow A.count - 1$
return $\{\alpha \mid \alpha \in C \wedge \frac{\alpha.freq}{t_e - t_s + win - 1} \geq minS\}$
```

Frequency Counting of serial Episodes

The basic idea when determining the window based frequency of serial episodes is that a serial episodes can be recognized by using automaton that accepts the events of its corresponding serial episode in exactly the specified order and ignores all other input. For each serial episode α there can be several instances of the recognizing automaton (in different states) at the same time. This is necessary in order to replace old occurrences with newer ones whenever possible. The algorithm for counting the frequency of serial episodes uses the following data structures and variables:

- Each episode α is represented as an array in which the event types contained in α are stored in the correct order
- For each episode α the number of windows in which alpha occurred so far is stored in $\alpha.freq$
- An automaton is simply represented by a tuple (α, i), where α is the corresponding candidate serial episode and $i \in \{1,...,|\alpha|\}$ is the state (position in the episode) in which the automaton currently is.
- The automata are grouped by the event type that will allow them to perform the next transition. These lists are referred to as $waits(T)$, where $T \in \Sigma$ is an event type.
- For each automata belonging to episode α that is currently in state i, the time at which it this automaton was initialized is stored in $\alpha.initialized[i]$.
- Additionally any automata that were initialized at point of time t will be contained in a list referred to as $beginsat(t)$.
- transitions to be made are stored in a list named transitions and are stored in the form (α, i, t), where α is the corresponding episode, i is the index of the state from which the automaton will transition to the next one and t is the time in which this automaton was initialized.

The structure of the algorithm is the same as the one of algorithm 2. First the variables are initialized, then the sequence is looped over and the sliding window moves by one time unit in each iteration. A new instance of an automaton is initialized, whenever an event A is the first event of an episode and enters the window. Additionally all automata that wait for A will be moved to the next state, while memorizing their initial starting time. If an automaton of episode α moves to a state which is already occupied by another automaton, the old automaton is discarded (since the newer one has a later starting time and thus will be present in more windows). If an automaton reaches its final state the current time is memorized in $\alpha.inwindow$. If an automaton in its final state expires (its starting time drops out of the window) the number of windows it was present in is added to its corresponding sequence.

Algorithm 3 Calculate Window based Frequency for serial Episodes

Require: Let C be the set of candidate serial episodes, and let $S = [(T_1, t_s), ..., (T_n, t_e)]$ be a sequence of events, let *win* be the window size and finally let *minS* be the minimum support. TODO: part of the algorithm is cut off

```
// Initialization
for each $\alpha \in C$ do
    for $i \in \{1, ..., |\alpha|\}$ do
        $\alpha.initialized[i] \leftarrow 0$
        $waits(\alpha[i]) \leftarrow \emptyset$
for each $\alpha \in C$ do
    $waits(\alpha[1]) \leftarrow waits(\alpha[1]) \cup \{(\alpha, 1)\}$
    $\alpha.freq \leftarrow 0$
for $i \in \{t_s - win, ..., t_s - 1\}$ do
    $beginsat(t) \leftarrow \emptyset$
// Recognition
for $start \leftarrow t_s - win + 1$ to $t_e$ do
    //Bring new events to the window
    $beginsat(start + win - 1) \leftarrow \emptyset$
    $transitions \leftarrow \emptyset$
    for each $(t, A) \in S$ where $t = start + win - 1$ do
        for each $(\alpha, j) \in waits(A)$ do
            if $j = |\alpha| \wedge \alpha.initialized[j] = 0$ then
                $\alpha.inwindow \leftarrow start$
            if $j = 1$ then
                $transitions \leftarrow transitions \cup \{(\alpha, 1, start + win - 1\}$
            else
                $transitions \leftarrow transitions \cup \{(\alpha, j, initialized[j-1])\}$
                Remove $(\alpha, j-1)$ from $beginsat(\alpha.initialized[j-1])$
                $\alpha.initialized[j-1] \leftarrow 0$
                Remove $(\alpha, j)$ from $waits(A)$
    for each $(\alpha, j, t) \in transitions$ do
        $\alpha.initialized[j] \leftarrow t$
        $beginsat(t) \leftarrow beginsat(t) \cup \{(\alpha, j)\}$
        if $j \leq |\alpha|$ then
            $waits(\alpha[j+1]) \leftarrow waits(\alpha[j+1]) \cup \{(\alpha, j+1)\}$
    // Drop old events out of the window
    for each $(\alpha, l) \in beginsat(start - 1)$ do
        if $l = |\alpha|$ then
            $\alpha.freq \leftarrow \alpha.freq + start - \alpha.inwindow$
        else
            Remove $(\alpha, l+1)$ from $waits(\alpha[l+1])$
```

Naturally there are enhancements to these algorithms. For example both counting algorithms iterate over each point of time t between the start and end time of the sequence, regardless of the fact that there might not be anything to do at this point of time (no event drops out of the window and no new event comes in). Especially if the sequence of events is sparse, meaning that the time between events is usually large, this will become problematic. This issue can be fixed rather easily: instead of increasing *start* by one in each iteration, one can increase start by the amount of time that is needed until a change in the window occurs and update the data structures accordingly.

There is a notable absence of frequency counting algorithms for elementary (composite) episodes in literature. Mannila et. al. claim that each composite episode can be broken down into partial episodes, which are serial and/or parallel [32]. However they neither specify an algorithm for breaking down composite episodes into purely serial and parallel parts, nor do they specify a frequency counting algorithm for composite episodes. Subsequent research, such as alternate frequency definitions and counting algorithms has also mainly focused on parallel and serial episodes. If composite episodes have been studied they were usually studied in the above mentioned, more restrictive form of sequences of parallel episodes.

2.3.4 Other frequency definitions

In this subsection we briefly present alternative frequency definitions. However we do not present the respective counting algorithms, since they are not as relevant for this thesis. For the exact algorithms we refer the reader to the respective papers.

Most alternative definitions tried to move away from the ideas of fixed windows and tried to improve the performance of the counting algorithms.

Minimal Occurrence Based Frequency Definition

The first alternate definition does uses the concept of minimal occurrences:

Definition 18 ***Minimal Occurrence*** *An event episode α is said to occur minimally in a window $W(S,q,r)$ if α occurs in W and there is no subwindow of W in which α also occurs.*

Definition 19 ***Episode Frequency - Minimal Occurrence based Definition*** *Given a sequence of events S and an Episode α, we define the minimal occurrence based frequency $mo\_freq(\alpha)$ as the number of minimal occurrences of α in S. TODO: find and cite the original source*

The second alternative definition introduces the concept of non-overlapping occurrences:

Definition 20 ***Non-Overlapping Occurrences*** *Given a m-Episode* $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ *where* $V_\alpha = \{v_1, ..., v_m\}$*, two occurrences* h_1 *and* h_2 *of* α *are non-overlapped if either*

- $\forall v_j \in V_\alpha : h_2(v_1) > h_1(v_j)$ *or*
- $\forall v_j \in V_\alpha : h_1(v_1) > h_2(v_j)$

A set of occurrences is non-overlapping if every pair of occurrences in it is non-overlapped [23].

This leads to the Definition:

Definition 21 ***Episode Frequency - Non-Overlapping Occurrences based Definition*** *Given a seqeunce of events S and an Episode* α*, we define the non-overlapping occurrence based frequency* $noo\_freq(\alpha)$ *as cardinality of the largest set of non-overlapped occurrences of* α *in S [23].*

When looking at these definitions in comparison to the window based frequency definition it is not clear whether any of these is always superior to or more useful than the other since they have different properties. We mention them briefly:

- As already mentioned the window based frequency counts an episode occurrence that is comprised of the same events in multiple windows. This might especially distort the count if the window size is high and the events in the episode happen with minimal delay between them.
- The minimal occurrence based definition of frequency does not suffer from the problem of the previous point
- The window based definition has the advantage that it already incorporates a fixed size during which episodes may occur, meaning there can not be episodes that stretch over a time period larger than the fixed window size w. This might be beneficial for potential algorithms, since it reduces the search space for episodes. On top of that it is also closer to reality, since in many domains episodes happen within a small time window [**?**].
- The non-overlapping occurrence based frequency offers the fastest counting algorithm of all three definitions. However when incorporating expiry times for serial episodes it looses this advantage. Additionally previous literature has not yet identified an efficient algorithm to count non-overlapped occurrences of parallel episodes with expiry times.

2.3.5 Candidate Generation

Most previous work generates candidates for serial and parallel episodes separately, using a levelwise approach for both cases. The candidate generation procedure presented below was originally specified by Manila et. al. [32]. A more detailed explanation can be found in Laxman's PHD thesis [22].

We first consider the case for parallel episodes. Given F_k as the set of all frequent parallel episodes of length k we can generate the candidate parallel episodes C_{k+1} of length $k+1$ by doing the following:

1. Represent each candidate $\alpha \in F_k$ as a lexicographically sorted array of length k.
2. For each unordered pair (α, β), $\alpha, \beta \in F_k$ where α and β share the same first $k-1$ nodes, generate candidate γ by copying α and appending $\beta[k]$.

For example the two frequent parallel episodes $A||A||B$ and $A||A||C$ will generate the candidate $A||A||C||D$.

The same procedure can be applied to serial episodes, except that

- we do not order lexicographically, instead the serial episodes remain in the array in their natural order
- each pair (α, β) with the same properties as above now generates two candidates:
  - γ_1 by copying α and appending $\beta[k]$.
  - γ_2 by copying β and appending $\alpha[k]$.

Thus the two frequent serial episodes $A \rightarrow A \rightarrow B$ and $A \rightarrow A \rightarrow C$ will now generate the two candidates $A \rightarrow A \rightarrow B \rightarrow C$ and $A \rightarrow A \rightarrow C \rightarrow B$.

Since the mining of composite episodes has not received much attention, it is unsurprising that there little related work that mentions candidate generation strategies for these general types of episodes (TODO: refer to later chapter, since I will do that). For a more strict definition of composite episodes which only includes sequences of parallel episodes, Baumgarten et. al. use a tree growth strategy to generate candidates for composite episodes [8].

References

[1] Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.

[2] Apte, C. and Hong, S. J. (1994). Predicting equity returns from securities data with minimal rule generation. In *KDD Workshop*, pages 407–418.

[3] Ashrafi, M. Z., Taniar, D., and Smith, K. (2004). Odam: An optimized distributed association rule mining algorithm. *IEEE distributed systems online*, (3):1.

[4] Atsalakis, G. S. and Valavanis, K. P. (2009). Surveying stock market forecasting techniques–part ii: Soft computing methods. *Expert Systems with Applications*, 36(3):5932–5941.

[5] Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM.

[6] Babcock, B., Datar, M., Motwani, R., et al. (2003). Load shedding techniques for data stream systems. In *Proc. Workshop on Management and Processing of Data Streams*. Citeseer.

[7] Bathoorn, R. and Siebes, A. (2007). Finding composite episodes. In *International Workshop on Mining Complex Data*, pages 157–168. Springer.

[8] Baumgarten, M., Büchner, A. G., and Hughes, J. G. (2003). Tree growth based episode mining without candidate generation. In *IC-AI*, pages 108–114.

[9] Bettini, C., Wang, X. S., and Jajodia, S. (1998). Mining temporal relationships with multiple granularities in time sequences. *IEEE Data Eng. Bull.*, 21(1):32–38.

[10] Chakravarthy, S. and Mishra, D. (1994). Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26.

[11] Connor, J. T., Martin, R. D., and Atlas, L. E. (1994). Recurrent neural networks and robust time series prediction. *Neural Networks, IEEE Transactions on*, 5(2):240–254.

[12] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer.

[13] Eckert, M. and Bry, F. (2009). Complex event processing (cep). *Informatik-Spektrum*, 32(2):163–167.

[14] Frank, R. J., Davey, N., and Hunt, S. P. (2001). Time series prediction and neural networks. *Journal of intelligent and robotic systems*, 31(1-3):91–103.

[15] Gaber, M. M., Zaslavsky, A., and Krishnaswamy, S. (2005). Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26.

[16] Giannella, C., Han, J., Pei, J., Yan, X., and Yu, P. S. (2003). Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212:191–212.

[17] Hasan, A., Teymourian, K., and Paschke, A. (2015). Probabilistic event pattern discovery. In *Rule Technologies: Foundations, Tools, and Applications*, pages 241–257. Springer.

[18] Januzaj, E., Kriegel, H.-P., and Pfeifle, M. (2004). Dbdc: Density based distributed clustering. In *Advances in Database Technology-EDBT 2004*, pages 88–105. Springer.

[19] Kayacan, E., Ulutas, B., and Kaynak, O. (2010). Grey system theory-based models in time series prediction. *Expert systems with applications*, 37(2):1784–1789.

[20] Kotsiantis, S. B., Zaharakis, I., and Pintelas, P. (2007). Supervised machine learning: A review of classification techniques.

[21] Kumar, M. and Thenmozhi, M. (2006). Forecasting stock index movement: A comparison of support vector machines and random forest. In *Indian Institute of Capital Markets 9th Capital Markets Conference Paper*.

[22] Laxman, S. (2006). *Discovering frequent episodes: fast algorithms, connections with HMMs and generalizations*. PhD thesis, Indian Institute of Science Bangalore.

[23] Laxman, S., Sastry, P., and Unnikrishnan, K. (2007). A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 410–419. ACM.

[24] Lazarevic, A. and Obradovic, Z. (2001). The distributed boosting algorithm. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 311–316. ACM.

[25] Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

[26] Lin, K. W., Chung, S.-H., and Lin, C.-C. (2015). A fast and distributed algorithm for mining frequent patterns in congested networks. *Computing*, pages 1–22.

[27] Luckham, D. and Schulte, R. (2011). Epts event processing glossary v2. 0. *Event Processing Technical Society*.

[28] Ma, B. L. W. H. Y. (1998). Integrating classification and association rule mining. In *Proceedings of the fourth international conference on knowledge discovery and data mining*.

[29] Mahfoud, S. and Mani, G. (1996). Financial forecasting using genetic algorithms. *Applied Artificial Intelligence*, 10(6):543–566.

[30] Manku, G. S. and Motwani, R. (2002). Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment.

[31] Mannila, H., Toivonen, H., and Verkamo, A. I. (1995). Discovering frequent episodes in sequences extended abstract. In *Proceedings the first Conference on Knowledge Discovery and Data Mining*, pages 210–215.

[32] Mannila, H., Toivonen, H., and Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289.

[33] Martinetz, T. M., Berkovich, S. G., and Schulten, K. J. (1993). Neural-gas' network for vector quantization and its application to time-series prediction. *Neural Networks, IEEE Transactions on*, 4(4):558–569.

[34] Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., and Varma, R. (2003). Query processing, resource management, and approximation in a data stream management system. CIDR.

[35] Papadimitriou, S., Brockwell, A., and Faloutsos, C. (2003). Adaptive, hands-off stream mining. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 560–571. VLDB Endowment.

[36] Van Gestel, T., Suykens, J. A., Baestaens, D.-E., Lambrechts, A., Lanckriet, G., Vandaele, B., De Moor, B., and Vandewalle, J. (2001). Financial time series prediction using least squares support vector machines within the evidence framework. *Neural Networks, IEEE Transactions on*, 12(4):809–821.

[37] Wu, E., Diao, Y., and Rizvi, S. (2006). High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM.

[38] Zhou, W., Liu, H., and Cheng, H. (2010). Mining closed episodes from event sequences efficiently. In *Advances in Knowledge Discovery and Data Mining*, pages 310–318. Springer.

[39] Zimmermann, A. (2012). Generating diverse realistic data sets for episode mining. In *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*, pages 611–618. IEEE.