Searching for Patterns in Long Sequences

Proefschrift

voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica
aan de Universiteit Antwerpen
te verdedigen door

Boris CULE

Promotor: prof. dr. Bart Goethals

Antwerpen, 2012

Searching for Patterns in Long Sequences
Nederlandse titel: *Zoeken naar patronen in lange sequenties van data*

Cover photo by Suzana Čule

If we knew what it was we were doing, it would not be called research, would it?

— Albert Einstein

Acknowledgements

Before we begin with the scientific content of this thesis, I would like to give a quick mention to the people without whom none of this work would ever have materialised.

First of all, I would like to thank my supervisor Bart Goethals, for inviting me to work as a researcher under his guidance, and for his unconditional support throughout my years as a Ph.D. student. I would also like to thank Jan Paredaens, for offering me the position of a teaching assistant within the ADREM research group, and my predecessor Nele Dexters, whose efforts in preparing me for the job ensured that I hit the ground running from day one.

Early on in my career as a researcher, I had the great fortune of working together with Céline Robardet, who taught me a great deal about scientific methods and writing academic papers. Later on, I had the pleasure of working together with Nikolaj Tatti, in an ongoing collaboration that resulted in hours of occasionally useful discussions, not to mention a number of publications.

A pleasant atmosphere at work is a necessity if work is to yield the desired results, and I have been lucky that my ADREM colleagues over the years (and I have seen quite a few of them come and go) continuously made sure that going to work was something to look forward to. Roel, Jan H., Adriana, Calin, Wim, Michael, Alvaro, Koen S., Sandy, Jilles, Jeroen, Nghia, Emin, Cheng, Tayena, Floris, Koen V., Kris, Tim, Stefan, Antonio, Martin, Thomas and Reuben — I thank you all.

Equally important, in my experience, was domestic bliss. My partner Vesna not only helped motivate me to bring this stage of my career to a successful end, but also made numerous personal sacrifices to allow me to dedicate the necessary time to my research. From working late at night or in the weekend, to spending a week on an Australian beach, it was always met with utmost sympathy and understanding. My daughters, Frida and Ramona, brought me further happiness, which could only reflect positively on my work.

Finally, to complete the circle, I must return to the very beginning, and thank my parents for opening the doors to the world to me all those years ago. It was probably the most difficult decision they ever had to make, but it was one I will always remain grateful for.

Thanks!

Boris Cule
Antwerpen, 2012

Contents

Publications

- **Boris Cule**, Bart Goethals, and Céline Robardet. A new constraint for mining sets in sequences. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2009)*, pages 317–328, 2009.
- **Boris Cule** and Bart Goethals. Mining Association Rules in Long Sequences. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2010)*, pages 300–309, 2010.
- Nikolaj Tatti and **Boris Cule**. Mining Closed Strict Episodes. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM-2010)*, pages 501–510, 2010.
- Nikolaj Tatti and **Boris Cule**. Mining Closed Episodes with Simultaneous Events. In *Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD-2011)*, pages 1172–1180, 2011.
- **Boris Cule**, Bart Goethals, Sven Tassenoy, and Sabine Verboven. Mining Train Delays. *Proceedings of the 10th International Symposium on Intelligent Data Analysis (IDA 2011)*, pages 113–124, 2011.
- **Boris Cule**, Nikolaj Tatti, and Bart Goethals. MARBLES: Mining Association Rules Buried in Long Event Sequences. *Proceedings of the SIAM International Conference on Data Mining (SDM 2012)*, pages 248–259, 2012.
- Nikolaj Tatti and **Boris Cule**. Mining Closed Strict Episodes. In *Data Mining and Knowledge Discovery*, 25(1): 34–66, 2012.

CHAPTER 1

Introduction

The world is overflowing with data. The rise of the internet has gone hand in hand with the technological advances which allow both the storage and the transfer of massive amounts of data. As a result, many users are now able to collect large quantities of data, which they struggle to process and analyse meaningfully. Data mining aims to solve this problem. Typically, a data mining algorithm takes as input a huge dataset, and aims to produce some useful information for the user. This information may be a summary of the dataset, a prediction of future events based on historical data, a grouping of the data into easily manageable clusters, or any other type of analysis the user wishes to obtain.

An important field in data mining is pattern mining. In simple terms, a pattern is typically considered to be a set of events that reoccur in the data. In this thesis, we mainly aim to solve the problem of identifying reoccurring patterns in long sequences of data. Before we begin with our own proposed methods, however, we would like to introduce some basic pattern types that the pattern mining field has been busy with in the last few decades.

1.1 Itemsets

As mentioned above, a pattern typically consists of a set of items (or an *itemset*) that reoccurs in the data. In the itemset mining problem, the dataset consists of a number of transactions, each containing a number of items. A classical transaction database can be illustrated by a supermarket shopping basket example, given in Table 1.1. In this setting, the goal is to discover which products (items) are often

bought together. In our example, we notice that *beer* and *crisps* have been bought together three times, so we can conclude that {beer, crisps} is a frequent itemset.

Transaction ID	Items
t_1	{beer, crisps}
t_2	{beer, coke, crisps}
t_3	{bread, butter}
t_4	{butter, cheese}
t_5	{beer, crisps, diapers}

Table 1.1: A market basket dataset.

More formally, the *support* of an itemset is usually defined as the number of transactions in the dataset that the itemset occurs in, and, given a transaction database, and a user-chosen support threshold σ, we aim to discover all itemsets that occur in the dataset at least σ times. Alternatively, the *frequency* of an itemset can be defined as the proportion of transactions in the dataset that the itemset occurs in, and, given a user-chosen frequency threshold ϕ, we aim to discover all itemsets with a frequency higher than or equal to ϕ.

1.2 Association Rules

Looking back at the example given in Table 1.1, it is very easy to see that frequent itemsets do not tell the whole story. Patterns of interest to the user can be much more subtle than that. For example, in a supermarket setting, it is not only interesting to know which items are often bought together, but also which items actually determine this correlation. Looking at the data in Table 1.1, we can notice that each customer who bought bread also bought butter, but not vice versa. This means that we can, with a 100% confidence (in our very limited example), predict that a customer that buys bread will also buy butter, but the chance that a customer who buys butter will also buy bread is just 50%.

Formally, an *association rule* of the form $X \Rightarrow Y$ tells us that an occurrence of itemset X implies the occurrence of itemset Y in the same transaction. The *confidence* of such an association rule is expressed as $c(X \Rightarrow Y)$, and is traditionally defined as the proportion of transactions that contain X that also contain Y. Again, given a transaction database, and a user-chosen confidence threshold ϵ, we aim to discover all association rules whose confidence is at least as high as ϵ. Just as with itemsets, a number of alternative ways to measure the value of an association rule have been proposed, which we will not go into at this stage. For reasons of efficiency, in most proposed approaches, the search for interesting association rules has been limited to rules consisting of already identified frequent itemsets.

1.3 Sequences

Naturally, itemsets and association rules are far from the only patterns we can look for in a dataset. If the data is sequential by nature, itemsets do not possess the ability

to express the order in which events occur, which is a major limitation. Assume that our database consists of sequences of events, rather than transactions. An example of such a dataset is given in Table 1.2.

Sequence ID	Sequence
s_1	$ABCBAD$
s_2	$BBAB$
s_3	$ACCAD$
s_4	$CDDA$
s_5	$BAABAD$

Table 1.2: A dataset consisting of a number of event sequences.

If we applied traditional itemset mining to this example, given a support threshold of 3, we would find that $\{A, C, D\}$ is a frequent itemset. However, all sequential information would be lost. If we look closely, we can actually find no sequential pattern with a support of 3 or more, consisting of items A, C and D. Sequential patterns ACD and CAD occur in two sequences, while CDA occurs in one. In fact, the only sequential pattern of size 3 with a sufficiently high support is BBA, which occurs in sequences s_1, s_2 and s_5. The search for frequent sequences, therefore, aims to find all sequences that occur in the dataset often enough.

1.4 Episodes

In a number of applications, though, the data does not always come in the form of many sequences. Sometimes the dataset consists of one long sequence. In this context, when we talk of reoccurring patterns, we mostly talk of frequent *episodes*. An episode, in its most general form, can be described by a directed acyclic graph (DAG). An example of an episode is given in Figure 1.1.

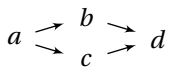


Figure 1.1: An episode.

This episode represents the following pattern: event a occurs first, it is followed by events b and c, that are then followed by event d. An episode, therefore, imposes a partial order on the events. It is immediately obvious that the expressive power of episodes is much greater than that of itemsets and sequences. More importantly, episode mining can discover frequent itemsets, as well as frequent sequences. An itemset is simply a directed acyclic graph with no edges, and the order in which the events making up the episode occur is therefore irrelevant. This kind of an episode is also referred to as a *parallel* episode. A sequence, on the other hand, can be expressed as a directed acyclic graph imposing a total order on the events, and can also be described as a *serial* episode.

Example 1. *Assume we are given an input sequence $abcdacbd$, and the user-chosen support threshold is equal to 2. With the itemset mining approach, the largest pattern we can find is $\{a,b,c,d\}$, while sequence mining will find sequences abd and acd. Episode mining, however, will find the episode shown in Figure 1.1. Itemset mining will identify the events that often reoccur together, but all information about the (partial) order in which they reoccur will be lost. Sequence mining will fail to discover a pattern consisting of all four reoccurring events, because they do not always occur in exactly the same order. Only episode mining offers us a true insight into the reoccurring pattern.*

1.5 Thesis Outline

The outline of the remainder of this thesis is as follows.

- **Chapter 2** investigates how to detect association rules between episodes, rather than itemsets. We introduce the concept of strict episodes, whereby we ease the computational complexity of the problem without sacrificing much in terms of the patterns we find. We build association rules based on closed strict episodes, avoiding the unnecessary output of all frequent episodes. Three different confidence measures are presented, and their respective merits discussed. We further reduce the size of the output by restricting it to closed association rules, and we go a step further by adopting the concept of confidence boost, to reduce redundancy to a minimum.

- **Chapter 3** extends the definition of an episode to allow a node in the DAG to carry multiple events. This allows us to describe patterns in which multiple events occur simultaneously. In this setting, we no longer mine only strict episodes introduced in **Chapter 2**. This, naturally, increases the complexity of the problem, and poses a number of new challenges. Here, too, we limit the output to closed episodes, and tackle a number of problems that arise when trying to define a subset relationship among the newly defined episodes.

- **Chapter 4** describes a novel technique for evaluating patterns in long sequences, taking into account not only how often the pattern occurs, but also how cohesive the occurrence of the pattern is. We evaluate patterns using an interestingness measure that combines the frequency and the cohesion of the pattern. We extend this approach to search for interesting association rules, too, and we present some preliminary analysis of how this approach could be used in a setting where the dataset consists of many sequences, rather than of just one long sequence.

- **Chapter 5** summarises the main contributions of the thesis and offers some insight into possible future work.

CHAPTER 2

MARBLES: Mining Association Rules Buried in Long Event Sequences

The discovery of sequential patterns is a well-studied field in data mining. Episodes are sequential patterns that describe events that often occur in the vicinity of each other. Episodes can impose restrictions on the order of the events, which makes them a versatile technique for describing complex patterns in the sequence. Most of the research on episodes deals with special cases such as serial and parallel episodes, while discovering general episodes is surprisingly understudied. This is particularly true when it comes to discovering association rules between them.

In this chapter[1] *we propose an algorithm that mines association rules between two general episodes. On top of the traditional definitions of frequency and confidence, we introduce two novel confidence measures for the rules. The major challenge in mining these association rules is pattern explosion. To limit the output, we aim to eliminate all redundant rules. We define the class of closed association rules, and show that this class contains all non-redundant output. To make the algorithm efficient, we use further pruning steps along the way. First of all, we generate only free and closed frequent episodes from which we create candidate rules, we speed up the evaluation of the rules, and then prune the remaining non-closed rules from the output. Finally, we provide the user with the additional option of using a confidence boost threshold to remove the less informative rules from the output.*

[1] This chapter is based on work published in SDM 2012 as "MARBLES: Mining Association Rules Buried in Long Event Sequences" by Boris Cule, Nikolaj Tatti and Bart Goethals [14].

2.1 Introduction

Discovering frequent patterns in an event sequence is an important field in data mining. Episodes, first defined by Mannila et al. [23], represent a rich class of sequential patterns, enabling us to discover events occurring in the vicinity of each other while at the same time capturing complex interactions between the events.

More specifically, a frequent episode is traditionally considered to be a set of events that reoccurs in the sequence. Gaps are allowed between the events and the order in which the events are allowed to occur is specified by the episode. The frequency of an episode is usually expressed as the number of windows of specified length in which the episode occurs, but can also be defined incorporating other concepts, such as minimal windows that contain the episode. However, it is important that the frequency is monotonically decreasing so we can use the well-known level-wise approach to mine all frequent episodes.

The order restrictions of an episode are described by a directed acyclic graph (DAG): the set of events in a sequence covers the episode if and only if each event occurs only after all its parent events (with respect to the DAG) have occurred (see the formal definition in Section 2.3). Usually, only two extreme cases are considered. A parallel episode poses no restrictions on the order of the events, and a window covers the episode if the events occur in the window, in any order. In such a case, the DAG associated with the episode contains no edges. The other extreme case is a serial episode. Such an episode requires that the events occur in one, and only one, specific order in the sequence. Clearly, serial episodes are more restrictive than parallel episodes. If a serial episode is frequent, then its parallel version is also frequent.

An association rule between two episodes expresses the fact that the occurrence of one episode implies, with a high enough probability, that another episode can be found nearby. Typically, association rules are defined such that an occurrence of a smaller episode implies the occurrence of a greater episode.

The main contribution of this chapter is an algorithm that mines association rules consisting of two episodes represented by DAGs. On top of that, we introduce two novel ways to define the confidence of an association rule, based on more intuitive concepts than the traditional method using sliding windows of fixed length. To reduce the size of the output, we adopt the traditional concept of closed patterns to obtain only non-redundant association rules. We present a collection of algorithms, called MARBLES, to mine all such rules, handling all three possible approaches.

So far, very little research has gone into the search for episodes based on DAGs and even less has gone into the discovery of association rules between them. In practice, such episodes have been overshadowed by parallel and serial episodes. The main reason for this is the pattern explosion. Consider the case of itemsets, where a frequent itemset of size k has 2^k frequent subsets. With an episode of size k, described by a DAG, the number of possible subepisodes is much larger, as we also have to look at all possible subsets of its edges, the number of which grows quadratically with the number of nodes. On top of that, each association rule consists of two episodes, so the number of combinations is huge, as illustrated by the following example.

Example 2. *Consider a sequence within which subsequence $abcd$ occurs frequently, using a fixed window of size 4. Assume that, outside these occurrences, events a, b, c and d never occur. It is easy to see that episodes $G = \{a,d\}$ and $H = a \to b \to c \to d$ will have the same frequency. The same, of course, is also true for any episode X, such that $G \subseteq X \subseteq H$. Therefore, the confidence of all association rules $X \Rightarrow Y$, where $G \subseteq X \subset Y \subseteq H$, will be equal to 1. In this case, a single simple reoccurring pattern in the sequence results in 158 different association rules[2]. However, just one of those rules is actually not redundant, namely $G \Rightarrow H$, as all others can be derived from that one.*

However, the advantage of episodes based on DAGs is that they allow us to capture dependencies between the events while not being too restrictive. The following example illustrates that parallel and serial episodes may be insufficient as a means of discovering all interesting association rules in a dataset.

Example 3. *As an example we will use text data, namely inaugural speeches by presidents of the United States (see Section 2.7 for more details). Protocol requires the presidents to address the chief justice and the vice presidents in their speeches. Hence, we have discovered association rule*

$$\{chief, justic, vice, president\} \Rightarrow \{chief \to justic, vice \to president\}.$$

This rule tells us that when these four words appear near each other, then 'chief' precedes 'justice' and 'vice' precedes 'president'. Since the actual address order varies from speech to speech, the pattern does not impose any additional restrictions. The discovered rule informs us of the frequent usage of phrases 'chief justice' and 'vice president' in each others vicinity, something we could never discover using only parallel or serial episodes.

A popular method of reducing the size of the output in any pattern mining problem is to discover only closed patterns. A pattern is closed if there exists no superpattern with the same frequency. Some work has gone into the discovery of closed frequent episodes, but we go a step further, by introducing the concept of closed association rules. We output only the rules where the left-hand side is minimal, and the right-hand side maximal. Defining this class of association rules is not trivial, and we use a variety of computational tricks to speed up the execution of our algorithm. This allows us to output association rules consisting of two episodes represented by DAGs, and yet keep the size of the output well under control.

Once the redundant rules have been removed in this way, some of the remaining rules can still be less informative than others. We adopt the concept of confidence boost [4], a measure of how informative a rule is, to our setting, and allow the user to remove the less informative rules from the output by means of a confidence boost threshold. We experimentally confirm that the output can be significantly reduced in this way.

Apart from the traditional definition of the confidence of an association rule, based on the frequencies of the two episodes using a sliding window of fixed size, we introduce two additional ways to define the confidence, using either minimal

[2] we counted this number manually

windows or weighted minimal windows. We show that all three methods have their merits, and can be valuable and intuitive, depending on the nature of the input sequence and the wishes of the end user.

The rest of the chapter is organised as follows: In Section 2.2, we discuss the most relevant related work, before presenting the main notations and concepts in Section 2.3. Section 2.4 introduces the notion of association rules using three different methods, while in Section 2.5 we discuss how we can limit the size of the output by eliminating redundant association rules. The algorithms that allow us to achieve this goal are presented in detail in Section 2.6. In Section 2.7 we show the results of our experiments, before presenting our conclusions in Section 2.8. Our implementation of the algorithm is available online[3].

2.2 Related Work

The first attempt at discovering frequent subsequences, or serial episodes, was made by Wang et al. [32]. The dataset consisted of a number of sequences, and a pattern was considered interesting if it was long enough and could be found in a sufficient number of sequences. A complete solution to a more general problem was later provided by Agrawal and Srikant [3] using an APRIORI-style algorithm [2].

Looking for frequent general episodes in a single event sequence was first proposed by Mannila et al. [23]. The WINEPI algorithm finds all episodes that occur in a sufficient number of windows of fixed length, and generates association rules $X \Rightarrow Y$, where $X \subset Y$ and both X and Y are frequent episodes. Specific algorithms were given for the case of parallel and serial episodes, but no algorithm for detecting general episodes was provided. Tatti and Cule [28] extend the definition of an episode to be able to depict simultaneous events. They provide an algorithm for generating all frequent episodes, but not association rules.

Mannila et al. also propose MINEPI [23], an alternative interestingness measure for an episode, where the frequency is defined as the number of minimal windows that contain the episode. In this context, the authors also define association rules. Unfortunately, this frequency measure is not monotonically decreasing. However, the issue can be fixed by defining frequency as the maximal number of non-overlapping minimal windows [27, 22]. Zhou et al. [35] proposed mining closed serial episodes based on the MINEPI method. However, the paper did not address the non-monotonicity issue of MINEPI. None of these follow-up papers handled the problem of association rules.

Méger and Rigotti [24] propose a method for mining association rules of the form $X \Rightarrow Y$, such that X and Y are both serial episodes, and X is a prefix of Y. Cule et al. [13] introduce an alternative interestingness measure for episodes, combining frequency with the cohesion of an episode. They further extend this work to mine association rules [12], but the method works only for parallel episodes.

A lot of research in the field of pattern discovery has gone into eliminating redundancy. An important way to tackle this problem is by outputting only closed patterns. Within sequence mining, some research has gone into outputting only closed subsequences, where a sequence is considered closed if it is not properly contained in any other sequence which has the same frequency. Yan et al. [34],

[3] `http://adrem.ua.ac.be/implementations`

Tzvetkov et al. [30], and Wang and Han [31] proposed methods for mining such closed patterns, while Garriga [9] further reduced the output by post-processing it and representing the patterns using partial orders. Harms et al. [21], meanwhile, experiment with closed serial episodes. In another attempt to trim the output, Garofalakis et al. [16] proposed a family of algorithms called SPIRIT which allow the user to define regular expressions that specify the language that the discovered patterns must belong to.

Pei et al. [26], and Tatti and Cule [29] considered restricted versions of the general problem setup of finding frequent episodes. The former approach assumes a dataset of sequences where the same label can occur only once. Hence, an episode can contain only unique labels. The latter pointed out the problem of defining a proper subset relationship between general episodes and tackled it by considering only strict episodes, where two nodes having the same label had to be connected by a path. In this chapter, we adopt the latter approach, extend it by allowing events in the sequence to take place at the same time, and build on it further in order to mine association rules.

Further interestingness measures for episodes, either statistically motivated or aimed at removing bias towards smaller episodes, were made by Garriga [8], Gwadera et al. [19, 18], Calders et al. [7], and Tatti [27]. All these methods, however, were limited to finding interesting episodes, and stopped short of discovering association rules between them.

Tackling redundancy within association rules, and not only within the patterns they consist of, has been done within the field of frequent itemset mining, but not within episode mining. Bastide et al. [5] define rules as non-redundant if they consist of a minimal antecedent and a maximal consequent, while Balcazar [4] introduces the concept of confidence boost, a measure of how informative a rule is. We incorporate both these concepts in our approach, while dealing with the complexity of terms such as minimal and maximal in the context of episodes.

2.3 Preliminaries

In this section, we introduce the basic concepts that we will use throughout the chapter. First we will describe our dataset.

Definition 1. *We define a* sequence event $e = (id(e), lab(e), ts(e))$ *as a tuple consisting of three entries, a unique id number* $id(e)$*, a label* $lab(e)$ *coming from an alphabet* Σ*, and a time stamp integer* $ts(e)$*. We will assume that if* $id(e) > id(f)$*, then* $ts(e) \geq ts(f)$*. A* sequence *is a collection of sequence events ordered by their ids.*

Note that we are allowing multiple events to have the same time stamp. However, for the sake of simplicity, we will use the notation $s_1 \cdots s_N$ to mean a sequence $((1, s_1, 1), \ldots, (N, s_N, N))$. In the rest of the chapter, we will assume all sequences to be of this form. The remaining definitions can easily be adapted for the case when multiple events occur simultaneously.

Definition 2. *Given a sequence* s *and two integers* i *and* j *we define a* subsequence $s[i, j] = s_i, \ldots, s_j$ *containing all events occurring between* i *and* j*. We define the* length

of subsequence $s[i,j]$ *as*

$$len(s[i,j]) = ts(s_j) - ts(s_i) + 1.$$

Our next step is to define the patterns we are interested in.

Definition 3. *An* episode *G is represented by a directed acyclic graph with labelled nodes, that is, $G = (V, E, lab)$, where $V = (v_1, \ldots, v_K)$ is the set of nodes, E is the set of directed edges, and lab is the function $lab : V \to \Sigma$, mapping each node v_i to its label.*

When there is no danger of confusion, we will use the same letter to denote an episode and its graph.

Definition 4. *A node n in an episode graph is a* descendant *of a node m if there is a path from m to n. In that case, node m is an* ancestor *of node n.*

We are now ready to give a precise definition of an occurrence of a pattern in a sequence.

Definition 5. *Given a sequence s and an episode G we say that s* covers *G, or G* occurs *in s, if there is an* injective *map f mapping each node v_i to a valid index such that the node v_i in G and the corresponding sequence element $s_{f(v_i)}$ have the same label, $s_{f(v_i)} = lab(v_i)$, and that if there is an edge (v_i, v_j) in G, then we must have $f(v_i) < f(v_j)$. In other words, the parents of v_j must occur in s before v_j. If the mapping f is surjective, that is, all events in s are used, we will say that s is an* instance *of G.*

An example of a sequence covering an episode is given in Figure 2.1.

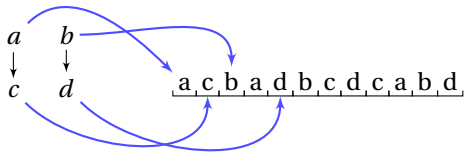


Figure 2.1: An example of an episode covered by a sequence.

In order to be able to discover association rules, we must first be able to compare episodes using some sort of a subset relationship.

Definition 6. *Given two episodes G and H, we say that G is a* subepisode *of H, denoted $G \subseteq H$, if the DAG describing episode G is a subgraph of the DAG describing episode H.*

Definition 7. *Given two episodes G and H, such that $G \subset H$, we can express an* association rule *$G \Rightarrow H$. We call G the* head *of the rule, and H the* tail *of the rule.*

2.4 Association Rules

In this section we present three possible methods to measure the frequency of an episode and the confidence of an association rule. The first one uses a sliding window of fixed size, the second is based on disjoint minimal windows, while the third one introduces the concept of weighted minimal windows.

Using Fixed Windows

We start off by defining the frequency of an episode in the traditional manner, based on windows of fixed length. This definition corresponds to the definition used in WINEPI [23]. The frequency is monotonically decreasing which allows us to do effective pruning while discovering frequent episodes.

Definition 8. *Given a window size ρ and a sequence s, we define the* fixed-window frequency *of an episode G in s, denoted $fr_f(G;s)$, to be the number of windows of size ρ in s covering the episode,*

$$fr_f(G;s) = \left|\left\{s[i, i+\rho-1] \mid s[i, i+\rho-1] \text{ covers } G\right\}\right|.$$

*We will use $fr_f(G)$ whenever s is clear from the context. An episode is σ-*frequent *(or simply* frequent*) if its frequency is higher than or equal to some given threshold σ.*

In this context, association rules can be defined in the traditional manner.

Definition 9. *Given a window size ρ and episodes X and Y, such that $X \subset Y$, we define the* fixed-window confidence *of the association rule $X \Rightarrow Y$, denoted $c_f(X \Rightarrow Y)$, to be the ratio of their respective frequencies,*

$$c_f(X \Rightarrow Y) = \frac{fr_f(Y)}{fr_f(X)}.$$

Informally, we can interpret this definition as follows: $c_f(X \Rightarrow Y)$ is the percentage of windows that contain X that also contain Y. In other words, if we encounter a window that contains X, $c_f(X \Rightarrow Y)$ represents the probability that the window also contains Y.

Using Minimal Windows

Using a sliding window of fixed length has some drawbacks, particularly in the context of association rules, as can be seen in the following examples.

Example 4. *Consider a sequence within which subsequence $abcd$ occurs frequently, using a fixed window of size 4. Assume that, outside these occurrences, events a, b, c and d never occur. The fact that $fr_f(\{a,d\}) = fr_f(a \to b \to c \to d)$ implies that $c_f(\{a,d\} \Rightarrow a \to b \to c \to d) = 1$. However, each time $abcd$ occurs in the input sequence, episode $\{b,c\}$ can be found in three windows of size 4. Therefore, $c_f(\{b,c\} \Rightarrow a \to b \to c \to d) = \frac{1}{3}$. Intuitively, though, looking at the pattern, we note that every occurrence of $\{b,c\}$, just like every occurrence of $\{a,d\}$, implies an occurrence of $a \to b \to c \to d$. The confidences of the two rules do not reflect that.*

Example 5. *Consider sequence s_1 within which subsequence $abxycd$ occurs frequently, and sequence s_2 within which subsequence $axbcyd$ occurs frequently, where x and y are noise events and are not part of the patterns. Assume we are using a fixed window of size 6, and that, outside these occurrences, events a, b, c and d never occur. As in the previous example, $c_f(\{a,d\} \Rightarrow a \rightarrow b \rightarrow c \rightarrow d) = 1$. However, each time $abxycd$ occurs in s_1, episode $\{b,c\}$ can be found in three windows of size 6, while each time $axbcyd$ occurs in s_2, $\{b,c\}$ can be found in five windows of size 6. Therefore, $c_f(\{b,c\} \Rightarrow a \rightarrow b \rightarrow c \rightarrow d)$ is equal to $\frac{1}{3}$ in s_1 and $\frac{1}{5}$ in s_2. Once again, the confidence values are not intuitive.*

The problem with basing the definition of confidence of an association rule on the fixed-window frequencies of the two episodes is that this approach focuses on the occurrences of the episodes within the windows, rather than on their occurrences within the sequence. In the above examples, while there are some windows that contain $\{b,c\}$ and not $a \rightarrow b \rightarrow c \rightarrow d$, it still holds that *every* occurrence of $\{b,c\}$ *in the sequence* can be found within an occurrence of $a \rightarrow b \rightarrow c \rightarrow d$. Our first step towards addressing this issue is to consider a different definition of the frequency of an episode in a sequence.

Definition 10. *Given a sequence s and an episode G, a window $s[a,b]$ is called a* minimal window *of G in s, if $len(s[a,b]) \leq \rho$, $s[a,b]$ covers G, and if no proper subwindow of $s[a,b]$ covers G. We define $b(s[a,b]) = ts(s_a)$ as the beginning of the window, and $e(s[a,b]) = ts(s_b)$ as its end. We denote the set of all minimal windows of G in s with $mw(G;s)$, or simply $mw(G)$, when s is known from the context. Given a set of minimal windows W, we define a function $dis(W)$ to be equal to 1 if all windows in W are pairwise disjoint, and 0 otherwise.*

An attempt was made to define the frequency of an episode as the number of its minimal windows [23]. However, this measure proved to be non-monotonic. In order to satisfy the downward-closed property, we need to consider only the non-overlapping windows [27, 22].

Definition 11. *The* disjoint-window frequency *of an episode G in a sequence s, denoted $fr_m(G;s)$, is defined as the maximal number of non-overlapping minimal windows within s that contain episode G. Formally,*

$$fr_m(G;s) = \max\{|W| \mid W \subseteq mw(G), dis(W) = 1\}.$$

Once again, we will use $fr_m(G)$ whenever s is clear from the context.

A naïve way to define the confidence of an association rule $X \Rightarrow Y$ in the disjoint-window context would be to, once again, compute the ratio between the frequencies of Y and X. The following example shows that this might not be very intuitive either.

Example 6. *Consider sequence s_1 within which subsequence $abcxyd$ occurs 100 times, and sequence s_2 within which subsequence $abcbcd$ occurs 100 times, where x and y are noise events and are not part of the patterns. Assume that, outside these occurrences, events a, b, c and d never occur. Denote $G = \{b,c\}$ and $H = a \rightarrow b \rightarrow c \rightarrow d$. In s_1, $fr_m(G) = 100$ and $fr_m(H) = 100$. Therefore, $\frac{fr(G)}{fr(H)} = 1$. However, in s_2, $fr_m(G) = 200$ and $fr_m(H) = 100$, and $\frac{fr(G)}{fr(H)} = 0.5$. Clearly, in both sequences, each occurrence of G implies an occurrence of H, and the results for s_2 are not satisfactory.*

While defining the frequency of an episode using minimal windows solved some of the problems inherent in the fixed-window method, we clearly still need to find a better way to define the confidence of an association rule. Intuitively, we wish the confidence of rule $X \Rightarrow Y$ to express the probability of encountering a minimal window of Y having encountered a minimal window of X. More formally, we wish to know what percentage of minimal windows of X are contained within minimal windows of Y. However, in order to do this, we are forced to drop the constraint that the minimal windows in question must be disjoint. The reasoning behind this decision is shown in the following example.

Example 7. *Consider sequence $s = abcadbcbcd$. Denote $G = b \rightarrow c$ and $H = a \rightarrow b \rightarrow c \rightarrow d$. The disjoint-window frequency of H is 1, but the sequence contains two overlapping minimal windows of H, $s[1,5]$ and $s[4,10]$. There are three minimal windows of G ($s[2,3]$, $s[6,7]$ and $s[8,9]$), and each of them is contained within a minimal window of H. However, if we were to only use non-overlapping windows of H, we would be faced with two problems. First of all, the confidence of rule $G \Rightarrow H$ would depend on our choice of disjoint minimal windows — if we chose the first minimal window of H, $s[1,5]$, we would find two occurrences of G outside it and the confidence of the rule would be $\frac{1}{3}$, whereas if we chose the second minimal window of H, $s[4,10]$, we would find just one occurrence of G outside it, and the confidence would be $\frac{2}{3}$. More importantly, whichever choice we made, we would not be able to get the correct result, showing that* every *occurrence of G is contained within an occurrence of H.*

Now that we have seen that we cannot define the confidence of an association rule using either the disjoint-window frequencies, or the containment of the disjoint occurrences, of the two episodes, we are ready to present a definition that corresponds exactly to our intuition.

Definition 12. *Given episodes X and Y, such that $X \subset Y$, and a minimal window $s[a,b]$ of episode X. Assume there exists a minimal window $s[c,d]$ of Y such that $c \leq a$ and $b \leq d$, then we define the* minimal-extensibility *of occurrence $s[a,b]$ of X into an occurrence of Y as*

$$ext_m(s[a,b], X, Y) = 1.$$

If there exists no such minimal window of Y, we define $ext_m(s[a,b], X, Y) = 0$.

Definition 13. *Given episodes X and Y, such that $X \subset Y$, we define the* minimal-window confidence *of the association rule $X \Rightarrow Y$, denoted $c_m(X \Rightarrow Y)$, to be the proportion of minimal windows of X that are contained within a minimal window of Y,*

$$c_m(X \Rightarrow Y) = \frac{1}{|mw(X)|} \sum_{w \in mw(X)} ext_m(w, X, Y).$$

Weighted Minimal Windows

The problem with using minimal windows is that they do not take the cohesion of a pattern into account. A minimal window of size 2 will make the same contribution towards the frequency of an episode as a minimal window of size 10. Similarly, using only minimal windows, the confidence of the association rule $X \Rightarrow Y$ would depend

solely on the inclusion of minimal windows of X inside minimal windows of Y, regardless of how much the window would need to be expanded in order to find the whole of Y. The following example illustrates these problems further.

Example 8. *Consider sequences $s_1 = axbcyd$ and $s_2 = abxycd$. Episode $b \rightarrow c$ occurs once in each sequence, so its disjoint-window frequency would be equal to 1 in each sequence. However, the implied pattern* an occurrence of a b is followed by an occurrence of a c *is clearly stronger in s_1, where b is immediately followed by a c. In both sequences, the minimal window of $b \rightarrow c$ is contained within a minimal window of $a \rightarrow b \rightarrow c \rightarrow d$, and the confidence of the rule $b \rightarrow c \Rightarrow a \rightarrow b \rightarrow c \rightarrow d$ using minimal windows would therefore be equal to 1 for both sequences. However, the meaning of the rule,* an occurrence of episode $b \rightarrow c$ is likely to be contained within a nearby occurrence of $a \rightarrow b \rightarrow c \rightarrow d$, *can be seen more clearly in s_2. In s_1, we must look much further before we encounter the whole of the larger episode.*

To address these problems, we must first modify the definition of the frequency of an episode in a sequence.

Definition 14. *The* total weight *of a set of windows W in a sequence s, denoted $tw(W)$, is defined as*

$$tw(W) = \sum_{w \in W} \frac{1}{len(w)}.$$

The weighted-window frequency *of an episode G in a sequence s, denoted $fr_w(G;s)$, is defined as*

$$fr_w(G;s) = \max\{tw(W) \mid W \subseteq mw(G), dis(W) = 1\}.$$

Here, too, we will use $fr_w(G)$ if s is clear from the context.

Informally, the shorter the window, the greater its contribution towards the overall frequency of the episode. Note that, once again, we need to consider only non-overlapping windows in order to satisfy the downward-closed property. However, this time, as the following example illustrates, we need to be much more careful as to which windows we choose, hence the need to maximise the sum of the windows' inverse lengths, rather than simply their number.

Example 9. *Consider sequence $s = abxywzcbaxywzc$ and parallel episode $X = \{a,b,c\}$. Sequence s contains three minimal windows of this episode, namely $s[1,7]$, $s[7,9]$ and $s[8,14]$. Note that the second minimal window overlaps with both the first and the third, so the disjoint-window frequency of X would be equal to 2. The two windows that would count towards this frequency would be $s[1,7]$ and $s[8,14]$. However, if we want to maximise the sum of the inverse lengths of the windows, we cannot simply take the maximal number of non-overlapping windows and then compute this sum. In our example, we have two possible sets of non-overlapping windows, $s[1,7]$ and $s[8,14]$, and $s[7,9]$ alone. In the first case, we have two windows of size 7, and the resulting sum would be $\frac{2}{7}$. In the second case, we have one window of size 3, and the sum would, therefore, be equal to $\frac{1}{3}$. We see that by choosing a smaller number of non-overlapping windows, we actually get a higher weighted-window frequency for the episode. We conclude that $fr_w(X) = \frac{1}{3}$.*

As was already shown in Example 8, we also need to take the various lengths of the minimal windows into account when computing the confidence of association rules. For reasons similar to those discussed with respect to the minimal-window method, we once again cannot take only the disjoint minimal windows into account. Intuitively, we want the confidence of rule $X \Rightarrow Y$ to correspond to the likelihood of encountering the whole of Y once we have encountered X. The nearer the whole of Y is to X on average, the higher the confidence should be.

Definition 15. *Given episodes X and Y, such that $X \subset Y$, and a minimal window $s[a,b]$ of episode X. Assume there exists a minimal window $s[c,d]$ of Y such that $c \leq a$ and $b \leq d$, and that this is the smallest window that contains both Y and $s[a,b]$, then we define the* weighted-extensibility *of occurrence $s[a,b]$ of X into an occurrence of Y as*

$$ext_w(s[a,b],X,Y) = \frac{len(s[a,b])}{len(s[c,d])}.$$

If there exists no such minimal window of Y, we define $ext(s[a,b],X,Y) = 0$.

Definition 16. *Given episodes X and Y, such that $X \subset Y$, we define the* weighted-window confidence *of the association rule $X \Rightarrow Y$, denoted $c_w(X \Rightarrow Y)$, to be the average weighted-extensibility of an occurrence of X into an occurrence of Y,*

$$c_w(X \Rightarrow Y) = \frac{1}{|mw(X)|} \sum_{w \in mw(X)} ext_w(w,X,Y).$$

The following example illustrates that, when extending an occurrence of X in order to find an occurrence of Y, it is important that we find the smallest such occurrence.

Example 10. *Consider sequence $s = axybca$ and episodes $X = b \rightarrow c$ and $Y = \{a, b \rightarrow c\}$. Sequence s contains one minimal window of X, namely $s[4,5]$. This occurrence of X can be extended in search of an occurrence of Y. There are two candidate minimal windows of Y that can be considered, $s[1,5]$ and $s[4,6]$. If we choose the former, the extensibility of $s[4,5]$ into Y would equal $\frac{2}{5}$, and if we use the latter, this value would rise up to $\frac{2}{3}$. Since we are interested in how far we need to look in order to extend an occurrence of X into an occurrence of Y, we clearly need to look for the smallest minimal window of Y that satisfies the conditions.*

We now show what effect the new definitions would have on the patterns discussed in Example 8.

Example 11. *Consider sequences $s_1 = axbcyd$ and $s_2 = abxycd$, and episodes $X = b \rightarrow c$ and $Y = a \rightarrow b \rightarrow c \rightarrow d$. First of all, we see that $fr_w(X; s_1) = \frac{1}{2}$, while $fr_w(X; s_2) = \frac{1}{4}$. However, we also see that $c_w(X \Rightarrow Y)$ equals $\frac{1}{3}$ in s_1 and $\frac{2}{3}$ in s_2. This shows that both problems mentioned in Example 8 have been successfully addressed.*

Table 2.1 shows an overview of the results of applying the three methods presented in Section 2.4 on the patterns and sequences discussed in Examples 8 and 11.

Pattern	s_1	s_2
$fr_f(X)$	5	3
$fr_f(Y)$	1	1
$c_f(X \Rightarrow Y)$	0.2	0.33
$fr_m(X)$	1	1
$fr_m(Y)$	1	1
$c_m(X \Rightarrow Y)$	1	1
$fr_w(X)$	0.5	0.25
$fr_w(Y)$	0.17	0.17
$c_w(X \Rightarrow Y)$	0.33	0.67

Table 2.1: An overview of all presented methods applied to the sequences given in Example 8. The window size used for the fixed-window method was 6.

2.5 Eliminating Redundancy

In this section, we will denote the frequency of an episode G with $fr(G)$, and the confidence of a rule $X \Rightarrow Y$ with $c(X \Rightarrow Y)$, regardless of which method we are using, as what follows applies to all three methods.

Closed Association Rules

It is a known fact in episode mining that two different DAGs may actually represent the same episode, as illustrated by the following example.

Example 12. *Consider DAGs G_1 and G_2 shown in Figure 2.2. It is clear that these two DAGs actually represent the same episode, namely* event a is followed by event b, which is then followed by event c. *More interestingly, DAGs H_1 and H_2 given in Figure 2.2 also represent the same episode, if we assume that the same event cannot take place twice at the same time (which, in reality, is often the case). H_1 represents a parallel episode consisting of two occurrences of event a. Clearly, any time this episode occurs, an occurrence of a must be followed by another occurrence of a. Therefore, any such occurrence is also an occurrence of the serial episode $a \rightarrow a$, depicted by H_2.*

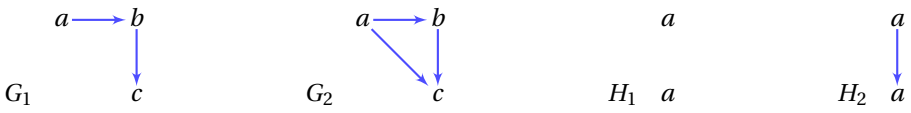


Figure 2.2: Toy episodes demonstrating closure.

To tackle this sort of redundancy, we must impose some restrictions on the types of DAGs we will discover.

Definition 17. *An episode G is called* transitively closed *if for two nodes n and m, such that there exists a path from n to m, there also exists an edge from n to m.*

Definition 18. *An episode G is called* strict *if for any two nodes v and w in G sharing the same label, there exists a path either from v to w or from w to v.*

As has been shown by Tatti and Cule [29], the issue of the above mentioned redundancy is resolved within the class of transitively closed strict episodes. Therefore, we will mine only association rules consisting of strict, transitively closed episodes. In the remainder of this chapter, we consider episodes to be strict and transitively closed, unless stated otherwise.

A traditional step towards further reducing the output is to generate only closed episodes.

Definition 19. *An episode G is* closed *if there exists no episode H, such that $G \subset H$ and $fr(G) = fr(H)$.*

However, we want to eliminate redundancy within the association rules we output, and not only among the frequent episodes. In fact, what we want to achieve is to output only the rules that give us most information. More precisely, we want the left-hand side of the rule to be minimal (the most general episode) and the right-hand side maximal (the most specific episode), among all those for which the rule holds. To achieve this, we first need to define what exactly we mean by minimal in this context.

Definition 20. *An episode G is* free *if there exists no episode H, such that $H \subset G$ and $fr(G) = fr(H)$.*

If we wish to fully eliminate redundancy in the output, we must consider only those rules that consist of a free episode on the left-hand side, and a closed episode on the right-hand side.

Definition 21. *Given two association rules $R_1 = X_1 \Rightarrow Y_1$ and $R_2 = X_2 \Rightarrow Y_2$, we say that R_1 is a* subset *of R_2 if $X_2 \subseteq X_1$ and $Y_1 \subseteq Y_2$. In this case, we denote $R_1 \subseteq R_2$. If $X_2 \subset X_1$ or $Y_1 \subset Y_2$, we denote $R_1 \subset R_2$.*

Unlike the frequency of an episode, the confidence of an association rule is not necessarily a monotonic measure, given this subset relationship between rules. The following example illustrates that, given rules R_1 and R_2, such that $R_1 \subset R_2$, the confidence of R_1 could be smaller than, greater than, or equal to the confidence of R_2.

Example 13. *Consider sequence $s = aacbabda$, using the minimal window method. Consider episodes $G_1 = a$, $G_2 = a \to b$, $G_3 = a \to c \to b$ and $G_4 = a \to c \to b \to d$, and rules $R_1 = G_1 \Rightarrow G_4$, $R_2 = G_2 \Rightarrow G_4$ and $R_3 = G_2 \Rightarrow G_3$. Clearly, it holds that $G_1 \subset G_2 \subset G_3 \subset G_4$, and therefore $R_3 \subset R_2 \subset R_1$. To compute the minimal-window confidence of the three rules, we first need to identify all minimal occurrences of the four episodes in s. There are four minimal occurrences of G_1, $s[1,1]$, $s[2,2]$, $s[5,5]$ and $s[8,8]$, two minimal occurrences of G_2, $s[2,4]$ and $s[5,6]$, one minimal occurrence of G_3, $s[2,4]$, and one minimal occurrence of G_4, $s[2,7]$. For R_1, we see that two of the four minimal occurrences of G_1 can be found within a minimal occurrence of G_4, and*

therefore $c_m(R_1) = 0.5$. *For* R_2, *we find that both minimal occurrences of* G_2 *can be found within a minimal occurrence of* G_4, *so* $c_m(R_2) = 1$. *Finally, for* R_3, *we see that just one minimal occurrence of* G_2 *can be found within a minimal occurrence of* G_3, *and* $c_m(R_3) = 0.5$. *To sum up,* $R_3 \subset R_2 \subset R_1$, *but* $c_m(R_1) = c_m(R_3) < c_m(R_2)$.

Example 13 illustrates that we cannot apply the usual definition of closure to association rules. It is possible for two rules, R_1 and R_2, such that $R_1 \subset R_2$, to have the same confidence purely *by coincidence*. However, in such a case, we cannot derive the confidence of R_1 using the confidence of R_2, and we cannot leave R_1 out of the output. We must therefore be a little bit more careful when defining non-redundant rules.

Definition 22. *Given episodes* X *and* Y, *such that* $X \subset Y$, *the association rule* $R = X \Rightarrow Y$ *is* not closed *if there exists a rule* R_1, *such that* $R \subset R_1$ *and* $c(R) = c(R_1)$, *and there exists no rule* R_2, *such that* $R \subset R_2 \subset R_1$ *and* $c(R) \neq c(R_2)$. *An association rule that does not satisfy these conditions is* closed.

Confidence Boost

In reality, mining only closed rules might not be enough to sufficiently reduce the output, as some redundant rules will still be discovered, as illustrated by the following example.

Example 14. *Assume we are given an input sequence* s, *in which subsequence* $abcde$ *occurs* 999 *times, and subsequence* $abxde$ *just once. Using the fixed window method with the window size of* 5, *the frequency of all subepisodes of* $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ *that include* a *and* e, *but do not include* c *will be* 1 000, *while the frequency of all subepisodes that include* a, c *and* e *will be* 999. *As a result, the confidence of rule* $\{a, e\} \Rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ *will be equal to* 0.999, *while the confidence of rule* $\{a, e\} \Rightarrow a \rightarrow b \rightarrow d \rightarrow e$ *would be equal to* 1. *Both rules would be closed, and both would appear in the output. However, it could be argued that the second rule is not very informative, given the first rule, as it, in fact, covers just one extra occurrence in the dataset.*

To solve this type of redundancy, we turn to the concept of *confidence boost*, as proposed by Balcazar [4].

Definition 23. *Given episodes* X *and* Y, *such that* $X \subset Y$, *the* confidence boost *of association rule* $X \Rightarrow Y$ *is defined as*

$$cb(X \Rightarrow Y) = \frac{c(X \Rightarrow Y)}{max_{X' \subseteq X, Y \subseteq Y'} c(X' \Rightarrow Y')},$$

under the conditions that $X \Rightarrow Y \neq X' \Rightarrow Y'$ *and that* Y' *is frequent. If the set of rules in the denominator is empty,* $cb(X \Rightarrow Y) = \infty$.

Given a certain confidence boost threshold β, the naïve approach would be to remove all rules from the output that have a confidence boost lower than β. However, such an approach involves a number of risks, as illustrated by the following example.

Example 15. *Assume, once again, that we are given an input sequence s, in which subsequence $abcde$ occurs 999 times, and subsequence $abxde$ just once. As seen in Example 14, $c(\{a,e\} \Rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e) = 0.999$ and $c(\{a,e\} \Rightarrow a \rightarrow b \rightarrow d \rightarrow e) = 1$. However, $cb(\{a,e\} \Rightarrow a \rightarrow b \rightarrow d \rightarrow e) = 1/0.999 \approx 1.001$. Given a confidence boost threshold of, say, 1.1, rule $\{a,e\} \Rightarrow a \rightarrow b \rightarrow d \rightarrow e$ would be removed from the output, as it is not very informative. However, if we chose a confidence threshold of 1, we would find neither of the two rules in the output, and important insight into the properties of the dataset would be lost.*

The informativeness of a rule clearly depends on the other rules in the output, and removing one uninformative rule from the output can make another previously uninformative rule become informative. Therefore, rather than looking at the absolute confidence boost of an individual rule, we need to evaluate the confidence boost of a rule relative to the other rules in the output.

Definition 24. *Given episodes X and Y, such that $X \subset Y$, the confidence boost of association rule $X \Rightarrow Y$* with respect to a set of rules $\mathscr{R}$ *is defined as*

$$cb_{\mathscr{R}}(X \Rightarrow Y) = \frac{c(X \Rightarrow Y)}{max_{X' \subseteq X, Y \subseteq Y'} c(X' \Rightarrow Y')},$$

under the conditions that $X \Rightarrow Y \neq X' \Rightarrow Y'$ and $X' \Rightarrow Y' \in \mathscr{R}$. If the set of rules in the denominator is empty, $cb_{\mathscr{R}}(X \Rightarrow Y) = \infty$.

The key, therefore, is to reduce the output to the bare minimum, but no more than that.

Definition 25. *Given a sequence s, a user-chosen window size ρ, a frequency threshold σ, a confidence threshold ϕ, and a confidence boost threshold β, a set of closed association rules $\mathscr{R}$ is* self-sufficient *if the following conditions hold:*

1. *for each $R = X \Rightarrow Y \in \mathscr{R}$, $fr(Y) \geq \sigma$,*
2. *for each $R \in \mathscr{R}$, $c(R) \geq \phi$,*
3. *for each $R \in \mathscr{R}$, $cb_{\mathscr{R}}(R) \geq \beta$,*
4. *for each $R' = X' \Rightarrow Y' \notin \mathscr{R}$, either $fr(Y') < \sigma$, $c(R') < \phi$, or $cb_{\mathscr{R}}(R') < \beta$*

Definition 25 states that an acceptable output set of association rules must be minimal. Either adding or removing a single rule would break at least one of the conditions for the set to be self-sufficient. The following proposition shows that a self-sufficient set of association rules is also unique.

Proposition 1. *Given a sequence s, a user-chosen window size ρ, a frequency threshold σ, a confidence threshold ϕ, and a confidence boost threshold β, there exists one and only one self-sufficient set of closed association rules.*

Proof. We will prove a more general statement: Given a set of rules $\mathscr{B}$, there is a unique set of rules $\mathscr{R} \subseteq \mathscr{B}$ such that $cb_{\mathscr{R}}(R) \geq \beta$ for any $R \in \mathscr{R}$ and $cb_{\mathscr{R}}(R) < \beta$ for any $R \in \mathscr{B} \setminus \mathscr{R}$. We call $\mathscr{R}$ to be a self-sufficient set with respect to $\mathscr{B}$.

If we prove this, then the proposition immediately follows if we set $\mathscr{B}$ to be the set of all closed rules that satisfy the thresholds.

We will prove this using induction over the size of $\mathscr{B}$. Obviously, the result holds for $|\mathscr{B}| = 1$.

Assume that the result holds for any base set of size smaller than k and assume that $|\mathscr{B}| = k$. Since the subset relation of rules is a partial order, there must be a rule R in $\mathscr{B}$ for which there is no rule $R' \in \mathscr{B}$ such that $R \subset R'$. This immediately implies that $cb_{\mathscr{U}}(R) = \infty$ for any subset $\mathscr{U} \subseteq \mathscr{B}$. This means that R must be included in a self-sufficient set. Let

$$\mathscr{V} = \left\{V \in \mathscr{B} \mid V \subset R,\ c(V)/c(R) < \beta\right\}.$$

Since R must be in a self-sufficient set, none of the rules in $\mathscr{V}$ can be in a self-sufficient set. Let $\mathscr{W} = \mathscr{R} \setminus (\mathscr{V} \cup \{R\})$. By the induction assumption, let $\mathscr{U}$ be the unique self-sufficient set with respect to $\mathscr{W}$ and let $\mathscr{S} = \mathscr{U} \cup \{R\}$. Set $\mathscr{S}$ is a self-sufficient set with respect to $\mathscr{B}$. To see the uniqueness, let $\mathscr{S}'$ be a self-sufficient set w.r.t. $\mathscr{B}$. We already saw that $R \in \mathscr{S}'$ and that $\mathscr{V} \cap \mathscr{S}' = \emptyset$. Let $\mathscr{U}' = \mathscr{S}' \setminus \{R\}$. Then $\mathscr{U}'$ is a self-sufficient set with respect to $\mathscr{W}$ and by the induction assumption $\mathscr{U} = \mathscr{U}'$ which implies that $\mathscr{S} = \mathscr{S}'$. □

Formally, given a sequence s, a user-chosen window size ρ, a frequency threshold σ, a confidence threshold ϕ, and a confidence boost threshold β we wish to find the self-sufficient subset $\mathscr{R}$ of closed association rules of the form $X \Rightarrow Y$, where $X \subset Y$, $fr(Y) \geq \sigma$, $c(X \Rightarrow Y) \geq \phi$, and $cb_{\mathscr{R}}(R) \geq \beta$. However, finding the self-sufficient set of association rules is not necessarily trivial, as illustrated by the following example.

Example 16. *Assume that our output $\mathscr{R}$ consists of four confident association rules, R_1, R_2, R_3 and R_4, such that $R_1 \subset R_2 \subset R_3 \subset R_4$. If $c(R_1) = 0.99$, $c(R_2) = 0.9$, $c(R_3) = 0.8$ and $c(R_4) = 0.72$, then $cb_{\mathscr{R}}(R_1) = 0.99/0.9 = 1.1$, $cb_{\mathscr{R}}(R_2) = 0.9/0.8 = 1.125$, $cb_{\mathscr{R}}(R_3) = 0.8/0.72 \approx 1.11$ and $cb_{\mathscr{R}}(R_4) = \infty$, as no superrule of R_4 has been found. Using a confidence boost threshold of 1.2, only R_4 would be informative enough. However, if we removed R_3 from the output to obtain a set of rules $\mathscr{R}_1 = \{R_1, R_2, R_4\}$, the relative confidence boost of R_2 would now be above the threshold — $cb_{\mathscr{R}_1}(R_2) = 0.9/0.72 = 1.25$. By removing R_3 from the output, R_2 has become informative enough and should be kept. By then removing R_1 from $\mathscr{R}_1$, we would obtain a self-sufficient set of rules, $\{R_2, R_4\}$. Alternatively, we may have chosen to first remove R_2 from $\mathscr{R}$, and thus obtain $\mathscr{R}_2 = \{R_1, R_3, R_4\}$. Now, R_1 would be informative enough, as $cb_{\mathscr{R}_2}(R_1) = 0.99/0.8 \approx 1.24$. By then removing R_3 from $\mathscr{R}_2$, we would obtain set $\mathscr{R}_3 = \{R_1, R_4\}$. However, note that this set is not self-sufficient, as condition 4. from definition 25 is not satisfied. There exists a rule $R_2 \notin \mathscr{R}_3$, such that $cb_{\mathscr{R}_3}(R_2) = 0.9/0.72 = 1.25 \geq \beta$.*

The above example shows that the order in which the rules are removed from the output is very important, a fact that must be taken into account when developing an algorithm for discovering the self-sufficient set of rules. Our algorithm, described in detail in Section 2.6, first generates all closed association rules, and then in a systematic and deterministic manner removes rules until the set becomes self-sufficient.

2.6 MARBLES

In this section, we present our algorithms. We start off by describing the first step, mining frequent episodes, before moving on to the algorithms we developed for mining closed association rules and finding their self-sufficient subset.

Mining Episodes

In order to generate association rules, we first need to generate frequent episodes. As our goal is to mine only closed association rules we do not need to consider all frequent episodes. We will resort to i-closed episodes [29].

Definition 26. *An i-closure is a function $icl(G;s) = H$ mapping an episode G to an episode H such that $G \subseteq H$. H is constructed from G in two phases. Firstly, if an event with a label l occurs in every minimal window of G and there is no node in G labelled with l, then we add a new node into G with label l. This is repeated until no new additions are possible. In the second phase, we add new edges. If in every instance of G a node v occurs before a node w, then we add an edge from v to w.*

Definition 27. *Given a sequence s, we say that an episode G is i-closed if $G = icl(G;s)$. From the fundamental properties of the closure, it follows that G is the maximal episode for which the closure is equal to G. Similarly, we say that G is i-free (or an i-generator) if there is no H such that $H \subset G$ and $icl(G;s) = icl(H;s)$. In other words, G is a minimal episode that can produce $icl(G;s)$.*

Note that there can be several minimal episodes that have the same i-closure but only one maximal episode.

The reason we are using i-closed episodes is that we can use them to generate closed association rules.

Proposition 2. *Let $X \Rightarrow Y$ be a closed association rule. Then X is i-free and Y is i-closed.*

Proof. All three confidence measures depend only on the minimal windows of X and Y. As demonstrated in the proof of Theorem 6 in the paper where i-closure was originally proposed [29], if two episodes, say G and H, have the same i-closure, i.e., $icl(G;s) = icl(H;s)$, then they have exactly the same minimal windows. This implies that $c(X \Rightarrow Y) = c(X \Rightarrow icl(Y;s))$ proving that Y must be i-closed. Similarly if we have $X' \subseteq X$ such that $icl(X';s) = icl(X;s)$, then $c(X \Rightarrow Y) = c(X' \Rightarrow Y)$ which immediately implies that $X = X'$. Consequently, X must be i-free. □

To mine episodes we employ the MINEEPISODES algorithm given in the original paper [29]. The algorithm follows a standard BFS-approach for closed patterns by first discovering all frequent i-free episodes and computing their closure. Hence, the output of this algorithm contains all frequent i-free episodes and their closures. Typically, free patterns are suppressed from the final output but in this case we need them to generate the left-hand sides of the association rules.

MINEEPISODES can either use fixed windows or minimal windows as a frequency constraint. The algorithm computes the frequency by first discovering all minimal

windows and then computing the frequency using the discovered windows. We extend the algorithm to handle weighted minimal windows by introducing a dynamic program (given in Algorithm 1). The algorithm takes a list of minimal windows for an episode G and computes $fr_w(G)$.

Algorithm 1: WEIGHTEDFREQUENCY. Computes weighted-window frequency $fr_m(X)$.

input : list of minimal windows $V = \{v_1, \ldots, v_N\}$ of episode X
output : $fr_m(X)$

```
1 j ← 1;
2 foreach v_i ∈ V do
3     while j ≤ N and e(v_i) ≥ b(v_j) do
4         j ← j + 1;
5     d_i ← j;
6 c_{N+1} ← 0;
7 for i = N...1 do
8     c_i ← max(1/len(v_i) + c_{d_i}, c_{i+1});
9 return c_1;
```

Consider that we have an ordered list of minimal windows $V = \{v_1, \ldots, v_N\}$. We need to select a subset of V, containing disjoint windows that maximise the total weight. In order to do that, let c_i be the maximal weight of a subset of disjoint windows of $\{v_i, \ldots, v_N\}$. Then it is easy to see that

$$c_i = \max(1/len(v_i) + c_d, c_{i+1}),$$

where d is the index of the next minimal window v_d that is disjoint with v_i. The left side in the max corresponds to using v_i in the subset and the right side corresponds to omitting v_i from the so far best disjoint collection.

The algorithm first finds d_i, the index of the next disjoint window for each v_i and then constructs the weights c_i. Both steps require $O(N)$ time and memory.

Mining Association Rules

Now that we have mined episodes, the next step is to build association rules. In order to do that, we introduce MARBLES given in Algorithm 2. The algorithm takes as input episodes mined in the first step and builds rules from these episodes. We assume that episodes are provided in specific groups. The input consists of a list of pairs, where the first element is an i-closed episode, say X, and the second element is a list, say $\mathcal{G}$, of all i-free episodes that have X as their closure. The reason for this grouping is that the confidence is equal for all rules of form $G \Rightarrow Y$, where $G \in \mathcal{G}$. In fact, $c(G \Rightarrow Y) = c(X \Rightarrow Y)$.

Of the three definitions of confidence given in Section 2.4, the fixed-window confidence is the only one that is monotonic. Confidence based on (weighted) minimal windows is not monotonic, hence we have to resort to an exhaustive enumeration.

The first loop in the algorithm discovers all confident association rules using i-free and i-closed episodes. This set contains all closed association rules, and

Algorithm 2: MARBLES. Mines closed association rules.

input : confidence threshold ϕ, list $\mathscr{X}$ of pairs $(X, \mathscr{G})$, where X is an i-closed episode and $\mathscr{G}$ are the i-free episodes having X as their i-closure
output : list of closed association rules

```
1  R ← ∅;
2  foreach (X1, G1), (X2, G2) ∈ X s.t. X1 ⊆ X2 do
3      foreach G ∈ G1 do
4          add R = G ⇒ X2 to R;
5          if c(R) < φ then mark R;
6  foreach R1 ∈ R do
7      S ← ∅;
8      foreach R2 ∈ R, R1 ⊊ R2 do
9          remove any rule S from S s.t. R2 ⊆ S;
10         if there is no S ∈ S s.t. S ⊆ R2 then
11             add R2 to S;
12     if there is S ∈ S s.t. c(R1) = c(S) then
13         mark R1;
14 return unmarked rules from R;
```

possibly some redundant rules. To remove the redundant rules, for each rule R_1, we first construct a set of its minimal superrules $\mathscr{S}$. If any of the rules in $\mathscr{S}$ has the same confidence as R_1, we mark R_1 as non-closed, and consequently purge it from the output.

In order to implement this algorithm efficiently, we need an efficient technique for enumerating all superepisodes of a given episode (used in the first loop) and for enumerating all superrules of a given association rule (used in the second loop). Assume that we have a list of episodes $\mathscr{E}$. We begin by grouping $\mathscr{E}$ into groups $\mathscr{E}_L$, where L is a multiset of labels and $\mathscr{E}_L$ contains episodes containing exactly the labels L. We then create index lists for these groups of form $I_{l,o}$, such that $I_{l,o}$ contains $\mathscr{E}_L$, where l occurs in L o times. Within each group $\mathscr{E}_L$, for each edge e, we create an index list of episodes I_e to contain all episodes containing edge e.

When searching for superepisodes of a given episode, say G, we first find episode groups $\mathscr{E}_L$ whose labels are a superset of or equal to the labels of G. After this is done, we then find a mapping between the labels of G and L (if there are several, we test them all). Once a mapping is found we map the edges of G and find all episodes that contain the edges of G.

To query association rules, we build a similar indexing structure, only now we group rules by their head episode. We index these groups using the head episode and within each group we index each rule with its tail.

Our final missing step in the algorithm is to actually compute the confidence. Computing confidence using fixed-windows is trivial. Hence, we focus on computing confidence based on minimal and weighted windows. The algorithms are given in Algorithm 3 and 4, respectively.

The algorithm MINWINCONFIDENCE takes as input two ordered lists of minimal

Algorithm 3: MINWINCONFIDENCE. Computes minimal-window confidence $c_m(X \Rightarrow Y)$.

input : list of minimal windows $V = \{v_1, \ldots, v_N\}$ of episode X, list of minimal windows $W = \{w_1, \ldots, w_M\}$ of episode Y
output : $c_m(X \Rightarrow Y)$

```
1  u ← 0; i ← 1;
2  foreach v ∈ V do
3      while i ≤ M and e(w_i) ≤ e(v) do
4          i ← i + 1;
5      if b(w_i) ≤ b(v) then u ← u + 1;
6  return u/N;
```

windows, V for the head episode and W for the tail episode. The algorithm then enumerates windows in V and tries to find a covering window from W. Since W and V are ordered, we do not need to search the covering window from beginning but instead from the last inspected window. This brings the run-time to $O(|V| + |W|)$.

Algorithm 4: WEIGHTEDCONFIDENCE. Computes weighted-window confidence $c_w(X \Rightarrow Y)$.

input : list of minimal windows $V = \{v_1, \ldots, v_N\}$ of episode X, list of minimal windows $W = \{w_1, \ldots, w_M\}$ of episode Y
output : $c_w(X \Rightarrow Y)$

```
1  c ← 0; i ← 1;
2  foreach v ∈ V do
3      while i ≤ M and e(w_i) ≤ e(v) do
4          i ← i + 1;
5      j ← i;
6      l ← ∞;
7      while j ≤ M and b(w_j) ≤ b(v) do
8          if len(w) ≤ l then
9              l ← len(w);
10             i ← j;
11         j ← j + 1;
12     c ← c + len(w) / l;
13 return c/N;
```

The algorithm WEIGHTCONFIDENCE is similar to MINWINCONFIDENCE. It also takes two ordered lists of minimal windows as input, V for the head episode and W for the tail episode. The only difference is that this time we do not need to simply find any covering window, but the smallest such window. This costs us an extra for-loop which brings the run-time to $O(|V| + |W| + |V|C)$, where C is the average number of windows in W covering a window $v \in V$. In practice, this number is small, making the algorithm efficient.

Finding the Self-Sufficient Subset

Our final step is to find the self-sufficient subset of rules from a given set of closed association rules. The proof of Proposition 1 gives us a simple approach for this: find a maximal rule R and add it to the output, remove R and the all the rules $R' \subset R$ with $c(R')/c(R) < \beta$, and repeat until there are no rules left. The pseudo-code for the algorithm is given in Algorithm 5.

Algorithm 5: BOOST. Computes self-sufficient subset.

input : set of rules $\mathscr{B}$, boost threshold β
output : self-sufficient subset $\mathscr{R}$ of $\mathscr{B}$

$\mathscr{R} \leftarrow \emptyset$;
while $\mathscr{B} \neq \emptyset$ **do**
  $R \leftarrow$ a maximal rule in $\mathscr{B}$;
  $\mathscr{R} \leftarrow \mathscr{R} \cup \{R\}$;
  $\mathscr{V} \leftarrow \{V \in \mathscr{B} \mid V \subset R,\ c(V)/c(B) < \beta\}$;
  $\mathscr{B} \leftarrow \mathscr{B} \setminus (\mathscr{V} \cup \{R\})$;
return $\mathscr{R}$;

2.7 Experiments

We tested our algorithm on five real-world datasets, three of which were text datasets — *address*, consisting of the inaugural addresses by the presidents of the United States[4], merged to form a single long sequence, *moby*, the novel Moby Dick by Herman Melville[5], and *abstract*, consisting of the first 739 NSF award abstracts from 1990[6], also merged into one long sequence. We processed the sequences using the Porter Stemmer[7] and removed the stop words. Our fourth dataset contained a sequence of alarms triggered in a factory, stretching over 18 months. An entry in the dataset consists of a time stamp and an event type. Finally, the fifth dataset contained information about trains delayed at departure from a Belgian railway station. The dataset consists of actual departure times of delayed trains, coupled with train numbers, stretching over a period of one month. The characteristics of the five datasets are summarised in Table 2.2. Note that the three text datasets are dense, as there are no time stamps involved. Implicitly, of course, the events (words) are assumed to have "taken place"on consecutive time stamps. The remaining two datasets, *alarms* and *trains*, are sparse. In both datasets, times are measured in seconds, and most time stamps are not associated with any event. Furthermore, these two datasets also contain events that occur at the same time, which is never the case in the text datasets.

In all our experiments, we kept the window size and the frequency threshold fixed, while varying the confidence and the confidence boost threshold. The main

[4] taken from `http://www.bartleby.com/124/pres68`
[5] taken from `http://www.gutenberg.org/etext/15`
[6] taken from `http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html`
[7] `http://tartarus.org/~martin/PorterStemmer/`

Sequence	Size	$\|\Sigma\|$	type
address	62 066	5 295	dense
moby	105 719	10 277	dense
abstract	67 828	6 718	dense
alarms	514 502	9 595	sparse
trains	10 115	1 280	sparse

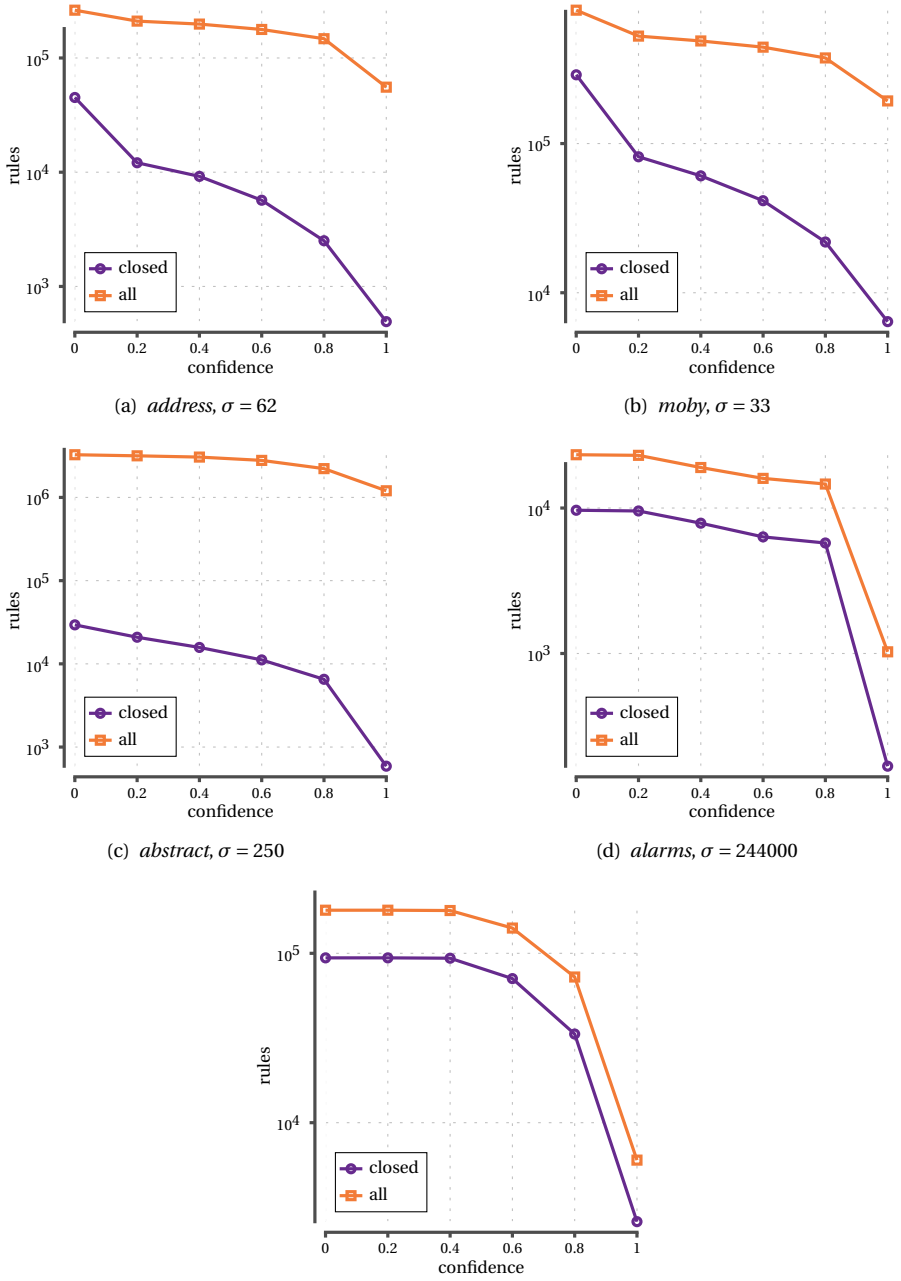
Table 2.2: Characteristics of the five datasets. The first column contains the size of the sequence, the second column the number of unique symbols in the sequence, and the third the type of data.

goal of our experiments was to demonstrate how we tackle the problem of pattern explosion. For each set of experiments, we therefore pushed the frequency threshold as low as possible, until the algorithm for generating all confident association rules began to take too long or ran out of memory. This demonstrated the need to generate i-closed and i-free episodes as an intermediary step, as our algorithm for mining only the closed rules ran much faster, and could handle much lower frequency thresholds. For the text datasets, we used the window size of 15, while for *alarms* and *trains* we used 10 and 30 minutes, respectively, after consulting domain experts in each field.

Figures 2.3, 2.4 and 2.5 show how the total number of confident association rules compared with the number of closed rules we discovered in all five datasets, using the fixed-window, minimal-window, and weighted-window method, respectively. The results show that the reduction was significant at all thresholds. Among the text datasets, the best reduction was obtained in the *abstract* dataset, which is not surprising, keeping in mind that this is the most structured dataset of the three. This was particularly visible in the minimal window method, where the reduction was smallest for the *address* and *moby* dataset, and highest for the other three datasets.

Figures 2.6, 2.7 and 2.8 show how the output could be further reduced by using a confidence boost threshold. We give results for all five datasets, using the fixed-window, minimal-window, and weighted-window method, respectively. Again, it can clearly be seen that the output can be significantly reduced in more structured datasets, while the results were not impressive for the more diverse *address* and *moby* datasets.

Finally, we give an overview of some of the more interesting association rules discovered in the three text datasets in Figures 2.9, 2.10 and 2.11. Note that our experiments confirmed that the three methods can rank rules differently. For example, Figures 2.9 (a) and (c) give the top rules of size 2 in the *address* dataset for $\phi = 1$ using the fixed window and the weighted window method, respectively, while Figures 2.10 (a) and (b) give the top rules of size 2 in the *moby* dataset for $\phi = 1$ using the fixed window and the minimal window method, respectively. Figure 2.11 (a), on the other hand, shows a rule of size 2 that was ranked top for $\phi = 1$ for all three methods in the *abstract* dataset. Figure 2.9 (g) shows a rule with $c_f \approx 0.88$ discovered in the *address* dataset. However, due to the rule shown in Figure 2.9 (h) having $c_f \approx 0.87$, the confidence boost of this rule is actually smaller than 1.01. Using a con-



(a) *address*, $\sigma = 62$

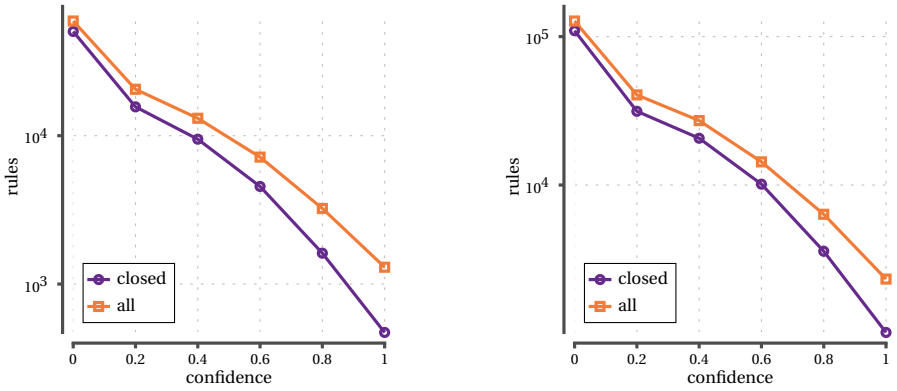
(b) *moby*, $\sigma = 33$

(c) *abstract*, $\sigma = 250$

(d) *alarms*, $\sigma = 244000$

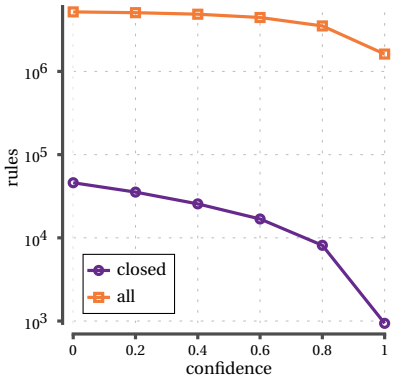
(e) *trains*, $\sigma = 17000$

Figure 2.3: The comparison of the number of closed association rules and the total number of rules discovered in the five datasets, using the fixed-windows method.

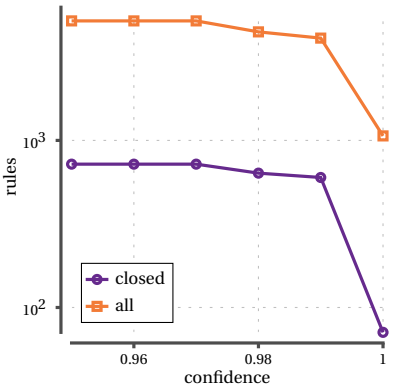


(a) *address*, $\sigma = 8$

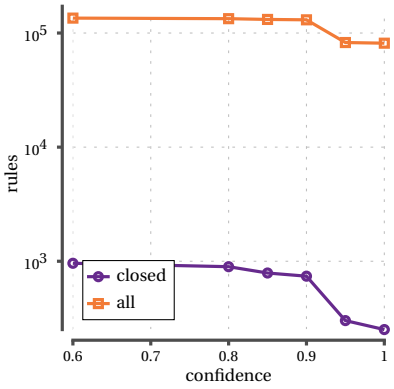
(b) *moby*, $\sigma = 7$



(c) *abstract*, $\sigma = 30$

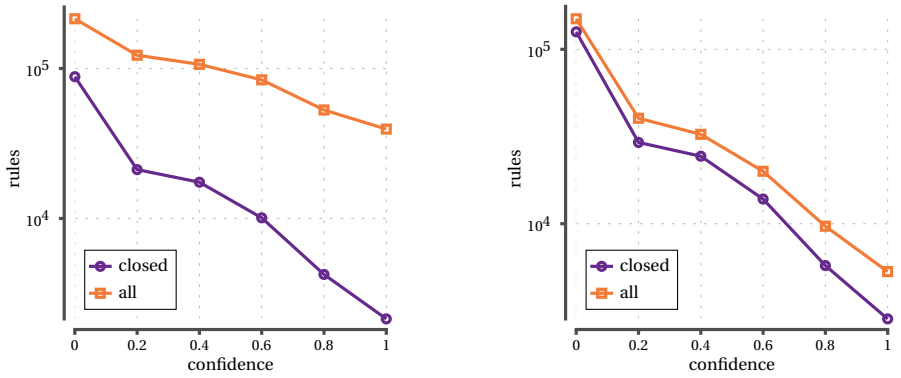


(d) *alarms*, $\sigma = 6035$



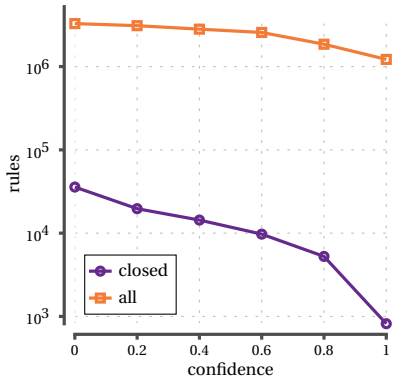
(e) *trains*, $\sigma = 18$

Figure 2.4: The comparison of the number of closed association rules and the total number of rules discovered in the five datasets, using the minimal-windows method.

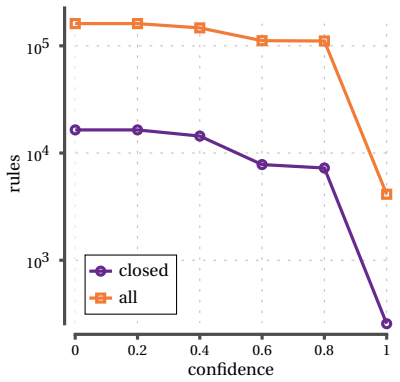


(a) *address*, $\sigma = 1$

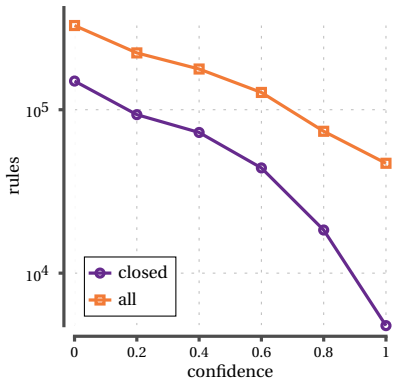
(b) *moby*, $\sigma = 1$



(c) *abstract*, $\sigma = 4$

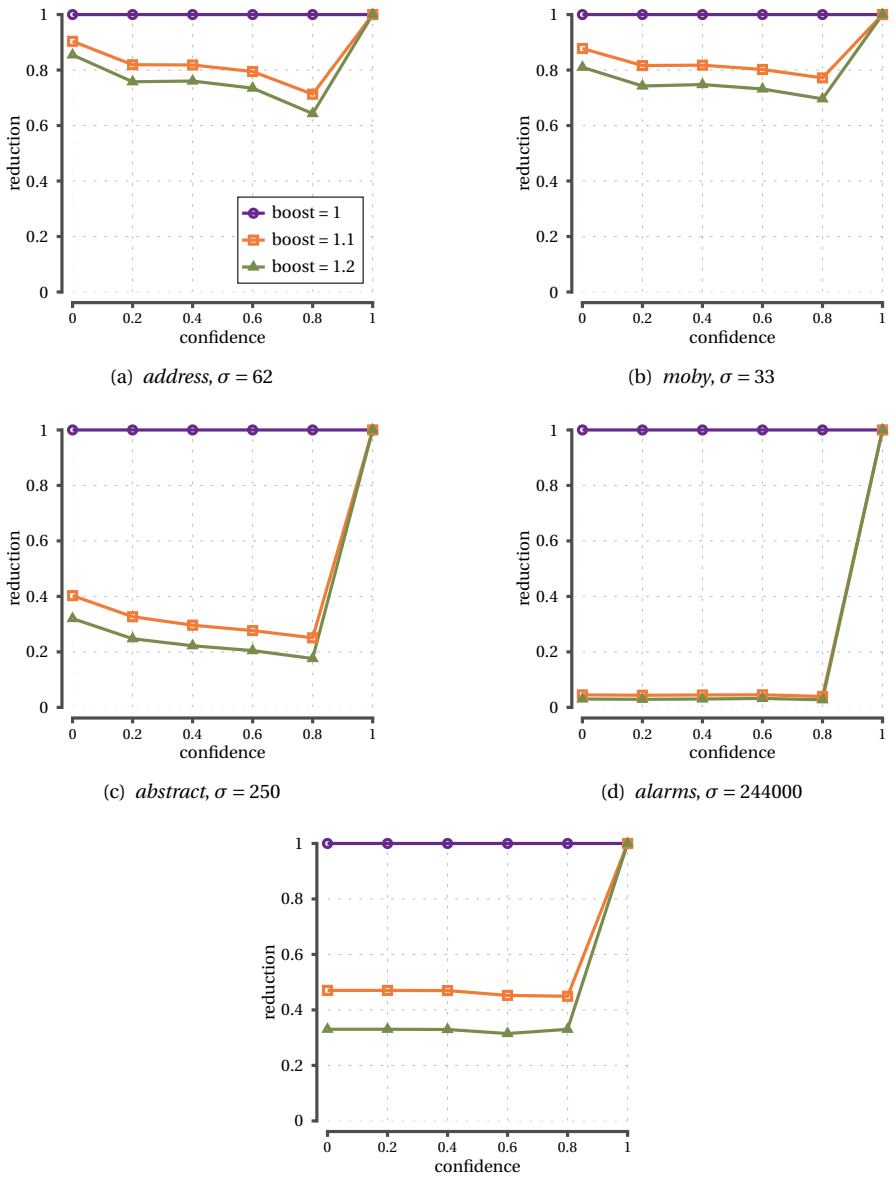


(d) *alarms*, $\sigma = 1300$



(e) *trains*, $\sigma = 0.03$

Figure 2.5: The comparison of the number of closed association rules and the total number of rules discovered in the five datasets, using the weighted-windows method.



(a) *address*, $\sigma = 62$

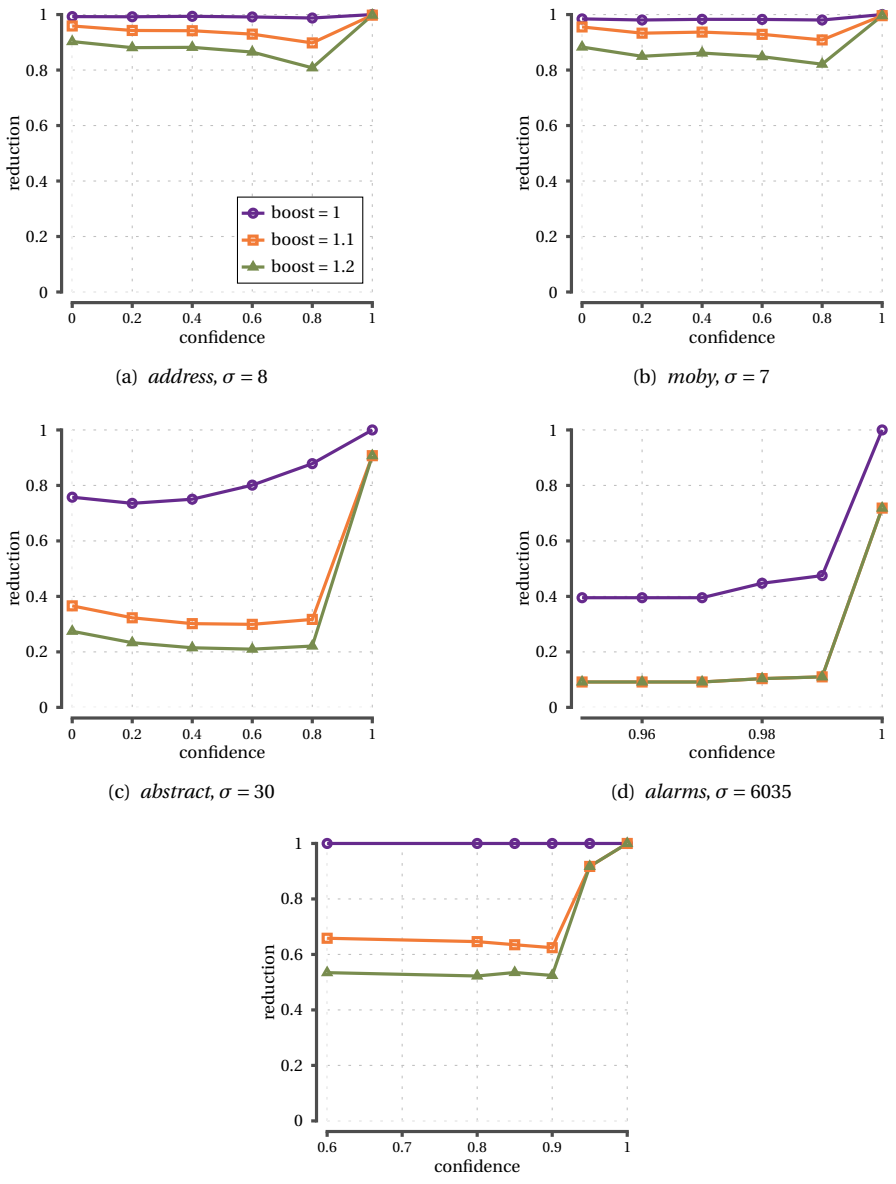
(b) *moby*, $\sigma = 33$

(c) *abstract*, $\sigma = 250$

(d) *alarms*, $\sigma = 244000$

(e) *trains*, $\sigma = 17000$

Figure 2.6: The comparison of the number of closed association rules discovered in the five datasets with varying confidence boost thresholds, using the fixed-windows method. For easier comparison, the number of rules is expressed as a proportion of the total number of closed rules discovered at each confidence threshold. The legend included in Figure (a) applies to all five plots.



(a) *address*, $\sigma = 8$

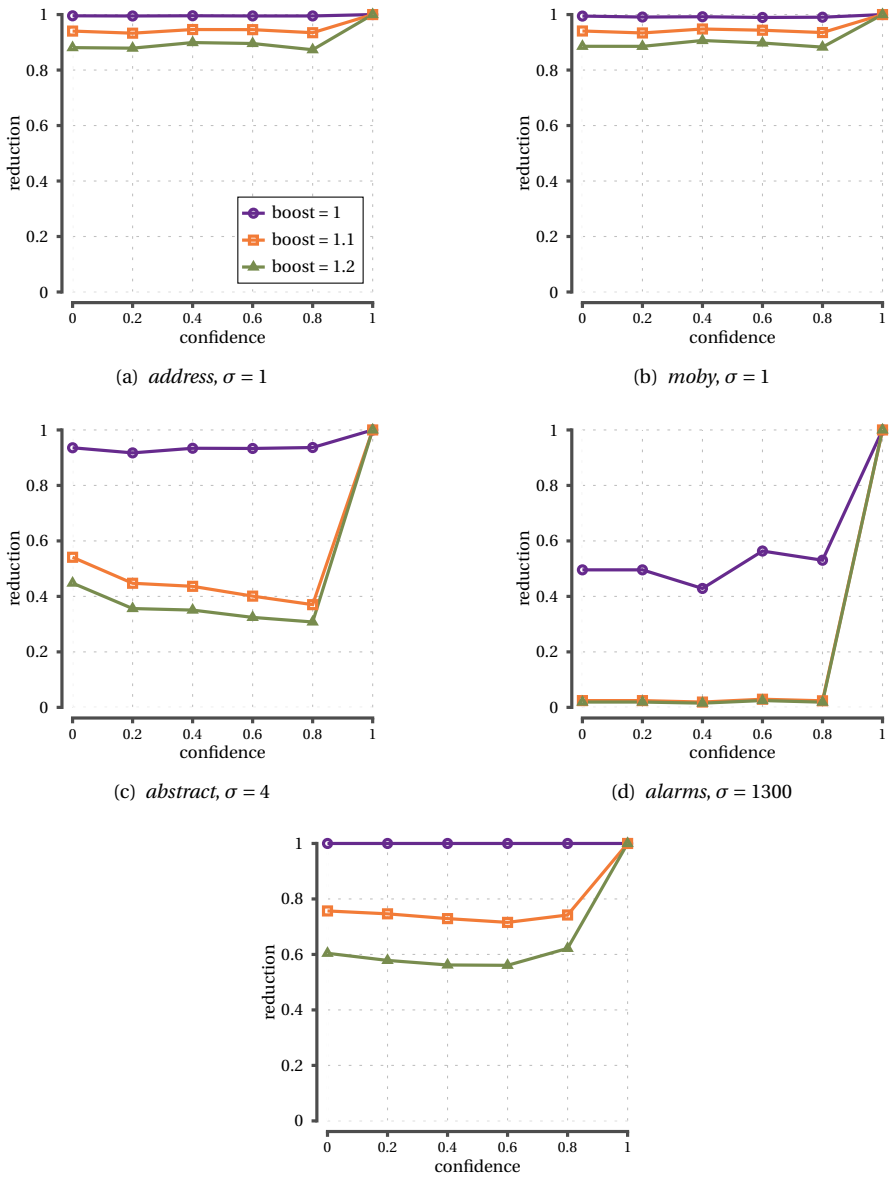
(b) *moby*, $\sigma = 7$

(c) *abstract*, $\sigma = 30$

(d) *alarms*, $\sigma = 6035$

(e) *trains*, $\sigma = 18$

Figure 2.7: The comparison of the number of closed association rules discovered in the five datasets with varying confidence boost thresholds, using the minimal-windows method. For easier comparison, the number of rules is expressed as a proportion of the total number of closed rules discovered at each confidence threshold. The legend included in Figure (a) applies to all five plots.



(a) *address*, $\sigma = 1$

(b) *moby*, $\sigma = 1$

(c) *abstract*, $\sigma = 4$

(d) *alarms*, $\sigma = 1300$

(e) *trains*, $\sigma = 0.03$

Figure 2.8: The comparison of the number of closed association rules discovered in the five datasets with varying confidence boost thresholds, using the weighted-windows method. For easier comparison, the number of rules is expressed as a proportion of the total number of closed rules discovered at each confidence threshold. The legend included in Figure (a) applies to all five plots.

fidence boost threshold of 1.1, rule shown in Figure 2.9 (g) would be removed from the output. A similar example from the *moby* dataset is shown in Figures 2.10 (c) and (d). Finally, a very large rule is shown in Figure 2.11 (e). Here, the occurrence of a general episode consisting of four words implies, with a confidence equal to 1, the occurrence of a serial episode consisting of six words. Unfortunately, non-disclosure agreements prevent us from presenting similar examples from the *alarms* and *trains* datasets.

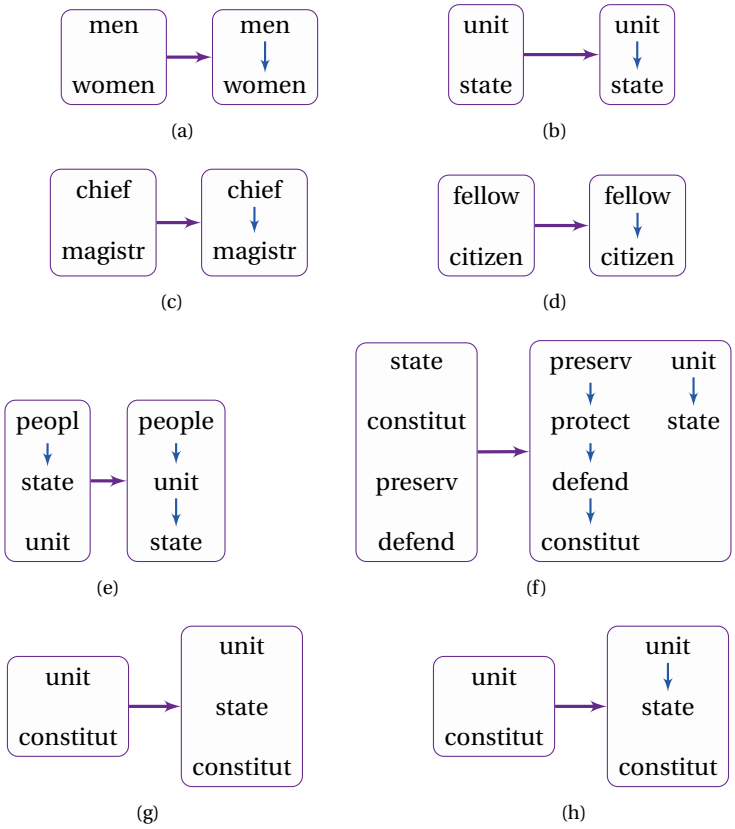


Figure 2.9: Examples of association rules discovered in the *address* dataset. (a) The most frequent rule with $c_f = 1$. (b) The most frequent rule with $c_f > 0.8$ and with $c_w > 0.8$. (c) The most frequent rule of size 2 with $c_w = 1$. (d) The most frequent rule with $c_m > 0.8$. (e) The most frequent rule with $c_w = 1$. (f) The most frequent rule of size 6 with $c_w = 1$. (g) A rule that can be removed using the confidence boost threshold, $c_f \approx 0.88$. (h) A more informative rule than the one in (g), $c_f \approx 0.87$.

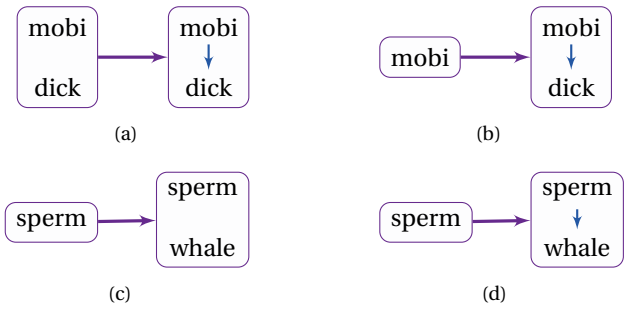


Figure 2.10: Examples of association rules discovered in the *moby* dataset. (a) The most frequent rule with $c_f = 1$. (b) The most frequent rule with $c_m = 1$. (c) A rule that can be removed using the confidence boost threshold, $c_m \approx 0.87$. (d) A more informative rule than the one in (c), $c_m \approx 0.84$.

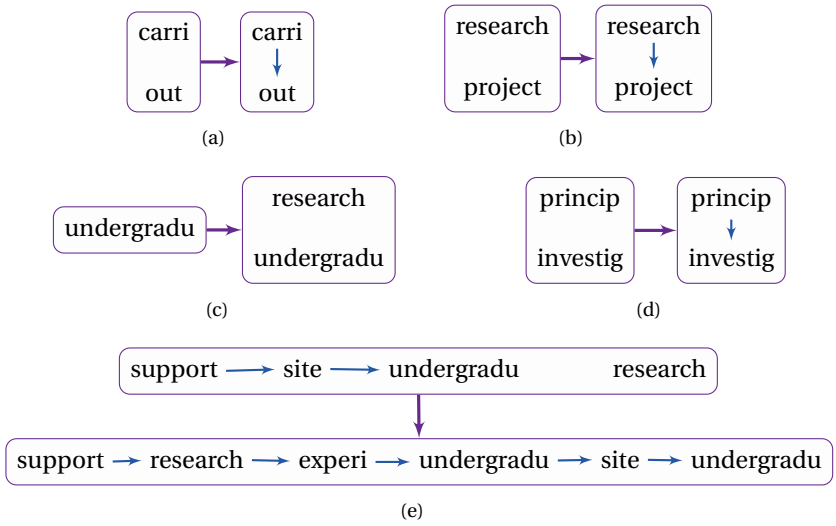


Figure 2.11: Examples of association rules discovered in the *abstract* dataset. (a) The most frequent rule of size 2 with $c_f = 1$, $c_m = 1$ and $c_w = 1$. (b) The most frequent rule with $c_f > 0.8$. (c) The most frequent rule with $c_m > 0.8$. (d) The most frequent rule with $c_w > 0.8$. (e) The most frequent rule of size 6 with $c_m = 1$.

2.8 Conclusions

In this chapter, we present a complete association rule miner for strict episodes, a large subclass of general episodes, represented by directed acyclic graphs (DAGs). We approach the problem from three angles, defining the frequency using windows of fixed size, minimal windows, and weighted minimal windows. While the first method is computationally the least demanding, we show that the other two can

be more intuitive and meaningful. We define and mine association rules within all three settings.

Furthermore, we tackle the problem of pattern explosion. The pattern explosion is exponential already when it comes to itemsets, it is much more of a problem within the field of episodes, and it culminates when we attempt to generate association rules between general episodes. We define closed association rules, thus eliminating all redundant rules from the output. Our algorithms take advantage of various properties of closed rules, such as the fact that the left-hand side must consist of a free episode, and the right-hand side of a closed episode, which allows us to discover closed association rules efficiently. Our experiments demonstrate that the reduction in the size of the output is considerable. Further reduction can be achieved by setting a confidence boost threshold, thus eliminating the least informative rules.

In future work, we intend to investigate if it is possible to extend this work to the complete class of general episodes, dropping the constraint that they must be strict. An interesting property of the confidence of an association rule is that it is not always a monotonic measure. Further research could also be dedicated to examining if it would still be possible to somehow prune some of the rules, based on some other criteria.

CHAPTER 3

Mining Closed Episodes with Simultaneous Events

General episodes depicted by directed acyclic graphs are a very rich pattern type, capable of expressing various interdependencies between the events making up the pattern. Typically, each node in such a graph carries a label corresponding to an event, and if two nodes are connected by a directed edge, the pattern tells us that the first event must happen before the second event. If two nodes are not connected by an edge, the two events may happen in any possible order, or even at the same time. However, there is one possible interdependency that cannot be depicted in this manner.

In this chapter[1] *we extend the definition of an episode in order to be able to represent cases where events often occur simultaneously. We present an efficient and novel miner for discovering frequent and closed general episodes. Such a task presents unique challenges. Firstly, we cannot define closure based on frequency. We solve this by computing a more conservative closure that we use to reduce the search space and discover the closed episodes as a postprocessing step. Secondly, episodes are traditionally presented as directed acyclic graphs. We argue that this representation has drawbacks leading to redundancy in the output. We solve these drawbacks by defining a subset relationship in such a way that allows us to remove the redundant episodes. We demonstrate the efficiency of our algorithm and the need for using closed episodes empirically on synthetic and real-world datasets.*

[1] This chapter is based on work published in KDD 2011 as "Mining Closed Episodes with Simultaneous Events" by Nikolaj Tatti and Boris Cule [28].

3.1 Introduction

Discovering interesting patterns in data sequences is a popular aspect of data mining. An episode is a sequential pattern representing a set of events that reoccur in a sequence [23]. In its most general form, an episode also imposes a partial order on the events. This allows great flexibility in describing complex interactions between the events in the sequence.

Existing research in episode mining is dominated by two special cases: parallel episodes, patterns where the order of the events does not matter, and serial episodes, requiring that the events must occur in one given order. Proposals have been made (see Section 3.2) for discovering episodes with partial orders, but these approaches impose various limitations on the events. In fact, to our knowledge, there is no published work giving an explicit description of a miner for general episodes.

We believe that there are two main reasons why general episodes have attracted less interest: Firstly, implementing a miner is surprisingly difficult: testing whether an episode occurs in the sequence is an **NP**-complete problem. Secondly, the fact that episodes are such a rich pattern type leads to a severe pattern explosion.

Another limitation of episodes is that they do not properly address simultaneous events. However, sequences containing such events are frequently encountered, in cases such as, for example, sequential data generated by multiple sensors and then collected into one stream. In such a setting, if two events, say a and b, often occur simultaneously, existing approaches will depict this pattern as a parallel episode $\{a, b\}$, which will only tell the user that these two events often occur near each other, in no particular order. This is a major limitation, since the actual pattern contains much more information.

In this chapter we propose a novel and practical algorithm for mining frequent closed episodes that properly handles simultaneous events. Such a task poses several challenges.

Firstly, we can impose four different relationships between two events a and b: (1) the order of a and b does not matter, (2) events a and b should occur at the same time, (3) b should occur after a, and (4) b should occur after or at the same time as a. We extend the definition of an episode to handle all these cases. In the remainder of this chapter, we consider events simultaneous only if they occur exactly at the same time. However, we can easily adjust our framework to consider events simultaneous if they occur within a chosen time interval.

Secondly, a standard approach for representing a partial order of the events is by using a directed acyclic graph (DAG). The mining algorithm would then discover episodes by adding nodes and edges. However, we point out that such a representation has drawbacks. One episode may be represented by several graphs and the subset relationship based on the graphs is not optimal. This ultimately leads to outputting redundant patterns. We will address this problem.

Thirdly, we attack the problem of pattern explosion by using closed patterns. There are two particular challenges with closed episodes. Firstly, we point out that we cannot define a unique closure for an episode, that is, an episode may have several maximal episodes with the same frequency. Secondly, the definition of a closure requires a subset relationship, and computing the subset relationship between episodes is **NP**-hard.

We mine patterns using a depth-first search. An episode is represented by a DAG and we explore the patterns by adding nodes and edges. To reduce the search space we use the *instance-closure* of episodes. While it is not guaranteed that an instance-closed episode is actually closed, using such episodes will greatly trim the pattern space. Finally, the actual filtering for closed episodes is done in a post-processing step. We introduce techniques for computing the subset relationship, distinguishing the cases where we can do a simple test from the cases where we have to resort to recursive enumeration. This filtering will remove all redundancies resulting from using DAGs for representing episodes.

The rest of the chapter is organised as follows: In Section 3.2, we discuss the most relevant related work. In Section 3.3, we present the main notations and concepts. Section 3.4 introduces the notion of closure in the context of episodes. Our algorithm is presented in detail in Sections 3.5, 3.6 and 3.7. In Section 3.8 we present the results of our experiments, before presenting our conclusions in Section 3.9. The code of the algorithm is available online[2].

3.2 Related Work

A review of the main work on finding episodes in long sequences, as well as an overview of attempts to reduce the output by, among other techniques, resorting to closed episodes, has been provided in Section 2.2 in the previous chapter. Here, we briefly revisit the most important work, and go deeper in detail when it comes to work specifically relevant to this chapter.

The concept of episodes in a single event sequence was introduced by Mannila et al. [23]. The proposed WINEPI method finds all episodes that occur in a sufficient number of windows of fixed length. While the concept of episodes represented by DAGs has been touched upon, specific algorithms were given only for parallel and serial episodes.

Mannila et al. also propose MINEPI [23], an alternative interestingness measure for an episode, where the support is defined as the number of minimal windows. Unfortunately, this measure is not monotonically decreasing. Fortunately, this issue can be fixed by defining support as the maximal number of non-overlapping minimal windows [27, 22].

In this chapter we use frequency based on a sliding window as it was defined for WINEPI.However, we can easily adopt our approach for other monotonically decreasing measures, as well as to a setup where the data consists of many (short) sequences instead of a single long one.

Pei et al. [26], and Tatti and Cule [29] considered restricted versions of our problem setup. The former approach assumes a dataset of sequences where the same label can occur only once. Therefore, an episode can contain only unique labels. The latter pointed out the problem of defining a proper subset relationship between general episodes and tackled it by considering only episodes where two nodes having the same label had to be connected. In our work, we impose no restrictions on the labels of events making up the episodes.

Using episodes to discover simultaneous events has, to our knowledge, not been done yet. However, this work is somewhat related to efforts made in discovering

[2] `http://adrem.ua.ac.be/implementations`

sequential patterns in multiple streams [25, 11, 20]. Here, it is possible to discover a pattern wherein two events occur simultaneously, as long as they occur in separate streams.

3.3 Episodes with Simultaneous Events

We begin this section by introducing the basic concepts that we will use throughout the chapter. First we will describe our dataset.

Definition 28. *A* sequence event $e = (id(e), lab(e), ts(e))$ *is a tuple consisting of three entries, a unique id number* $id(e)$*, a label* $lab(e)$*, and a time stamp integer* $ts(e)$*. We will assume that if* $id(e) > id(f)$*, then* $ts(e) \geq ts(f)$*. A* sequence *is a collection of sequence events ordered by their ids.*

Note that we are allowing multiple events to have the same time stamp even when their labels are equivalent. For the sake of simplicity, we will use the notation $s_1 \cdots s_N$ to mean a sequence $((1, s_1, 1), \ldots, (N, s_N, N))$. Similarly, we will also write $s_1 \cdots (s_i s_{i+1}) \cdots s_N$ to mean the sequence

$$((1, s_1, 1), \ldots, (i, s_i, i), (i+1, s_{i+1}, i), \ldots, (N, s_N, N-1)).$$

This means that s_i and s_{i+1} have equal time stamps.

Our next step is to define patterns we are interested in.

Definition 29. *An* episode event *e is a tuple consisting of two entries, a unique id number* $id(e)$ *and a label* $lab(e)$*. An* episode graph *G is a directed acyclic graph (DAG). The graph may have two types of edges:* weak edges $WE(G)$ *and* proper edges $PE(G)$*.*

An episode *consists of a collection of episode events, an episode graph, and a surjective mapping from episode events to the nodes of the graph which we will denote by* $nd(e)$*. A proper edge from node* v *to node* w *in the episode graph implies that the events of* v *must occur before the events of* w*, while a weak edge from* v *to* w *implies that the events of* w *may occur either at the same time as those of* v *or later.*

We will assume that the nodes of G are indexed and we will use the notation $nd(G, i)$ *to refer to the i th node in G.*

When there is no danger of confusion, we will use the same letter to denote an episode and its graph. Note that we are allowing multiple episode events to share the same node even if these events have the same labels.

Definition 30. *Given an episode G and a node n, we define* $lab(n) = \{lab(e) \mid nd(e) = n\}$ *to be the multiset of labels associated with the node. Given two multisets of labels X and Y we write* $X \leq_{lex} Y$ *if X is lexicographically smaller than or equal to Y. We also define* $lab(G)$ *to be the multiset of all labels in G.*

Definition 31. *A node n in an episode graph is a* descendant *of a node m if there is a path from m to n. If there is a path containing a proper edge we will call n a* proper descendant *of m. We similarly define a* (proper) ancestor*. A node n is a* source *if it has no ancestors. A node n is a* proper source *if it has no proper ancestors. We denote all sources of an episode G by* $src(G)$*.*

We are now ready to give a precise definition of an occurrence of a pattern in a sequence.

Definition 32. *Given a sequence s and an episode G, we say that s covers G if there exists an* injective *mapping m from the episode events to the sequence events such that*

1. *labels are respected, $lab(m(e)) = lab(e)$,*
2. *events sharing a same node map to events with the same time stamp, in other words, $nd(e) = nd(f)$ implies $ts(m(e)) = ts(m(f))$,*
3. *weak edges are respected, if $nd(e)$ is a descendant of $nd(f)$, then $ts(m(e)) \geq ts(m(f))$,*
4. *proper edges are respected, if $nd(e)$ is a proper descendant of $nd(f)$, then $ts(m(e)) > ts(m(f))$.*

Note that this definition allows us to abuse notation and map graph nodes directly to time stamps, that is, given a graph node n we define $ts(m(n)) = ts(m(e))$, where $nd(e) = n$.

The following example illustrates the above concepts.

Example 17. *Consider the three episodes given in Figure 3.1. A sequence $ab(cd)$ covers G_1 and G_3 but not G_2 (proper edge (c,d) is violated). A sequence $(ab)cd$ covers G_1 and G_2 but not G_3.*

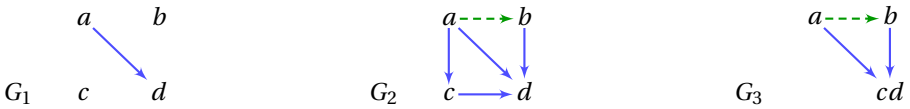


Figure 3.1: Toy episodes. Proper edges are drawn solid. Weak edges are drawn dashed.

Finally, we are ready to define support of an episode based on fixed windows. This definition corresponds to the definition used in WINEPI [23]. The support is monotonically decreasing which allows us to do effective pruning while discovering frequent episodes.

Definition 33. *Given a sequence s and two integers i and j we define a* subsequence

$$s[i,j] = \{e \in s \mid i \leq ts(e) \leq j\}$$

containing all events occurring between i and j.

Definition 34. *Given a window size ρ and an episode s, we define the* support *of an episode G in s, denoted $fr(G;s)$, to be the number of windows of size ρ in s covering the episode,*

$$fr(G;s) = \left|\{s[i,i+\rho-1] \mid s[i,i+\rho-1] \text{ covers } G\}\right|.$$

*We will use $fr(G)$ whenever s is clear from the context. An episode is σ-*frequent *(or simply* frequent*) if its support is higher or equal than some given threshold σ.*

Example 18. *Consider sequence $abcdacbd$ and set the window size $\rho = 4$. There are 2 windows covering episode G_1 (given in Figure 3.1), namely $s[1,4]$ and $s[5,8]$. Therefore, $fr(G_1) = 2$.*

Theorem 3. *Testing whether a sequence s covers an episode G is an **NP**-complete problem, even if s does not contain simultaneous events.*

Proof. If s covers G, then the map f mapping the nodes of G to indices of s provides the certificate needed for the verification. Hence, testing the coverage is in **NP**.

In order to prove the completeness we reduce 3SAT to the coverage. In order to do so, given a formula F, we will build an episode G and a sequence such that sequence s covers G if and only if F is satisfiable.

Assume that we are given a formula F with M variables and L clauses. We define the alphabet for the labels to be $\Sigma = \{\alpha_1, \ldots, \alpha_M\} \cup \{\beta_1, \ldots, \beta_L\}$, where α_i is identified with the ith variable and β_j is identified with jth clause.

We will now construct G. The nodes in the episode consist of three groups. The first group, $P = \{p_1, \ldots, p_M\}$, contains M nodes. A node p_i is labelled as $lab(p_i) = \alpha_i$. These nodes represent the positive instantiation of the variables. The second group of nodes, $N = \{n_1, \ldots, n_M\}$, also contains M nodes, and the labels are again $lab(n_i) = \alpha_i$. These nodes represent the negative instantiation of the variables. Our final group is $C = \{c_1, \ldots, c_{3L}\}$, contains $3L$ nodes, 3 nodes for each clause. The labels for these nodes are $lab(c_{3j-2}) = lab(c_{3j-1}) = lab(c_{3j}) = \beta_j$. The edges of the episode G are as follows: Let β_j be the jth clause in F and let α_i be the kth variable occurring in that clause. Note that $k = 1,2,3$. We connect p_i to c_{3j+k-3} if the variable is positive in the clause. Otherwise, we connect n_i to c_{3j+k-3}.

The sequence s consists of 5 consecutive subsequences $s = s_1 s_2 s_3 s_4 s_5$. We define $s_1 = s_3 = \alpha_1 \cdots \alpha_M$ and $s_2 = s_4 = s_5 = \beta_1 \cdots \beta_L$, that is, s is equal to

$$\alpha_1 \cdots \alpha_M \beta_1 \cdots \beta_L \alpha_1 \cdots \alpha_M \beta_1 \cdots \beta_L \beta_1 \cdots \beta_L.$$

Our final step is to prove that s covers G whenever F is satisfiable. First assume that F is satisfiable and let t_i be the truth assignment for the variable α_i. We need to define a mapping f. If t_i is true, then we map p_i into the first group s_1 and n_i into the third group s_3. If t_i is false, then we map n_i into s_1 and p_i into s_3. Since each clause is now satisfied, among the nodes c_{3j-2}, c_{3j-1}, and c_{3j} there is at least one node, say c_k, such that the parent of that node (p_i or n_i) is mapped to the first group s_1. We can map c_k into the second group s_2. The remaining two nodes are mapped into the fourth and the fifth group, s_4 and s_5. Clearly this mapping is valid since all the nodes are mapped and the edges are honoured.

To prove the other direction, let f be a valid mapping of G into s. Since the sequence has the same amount of symbols as there are nodes in the graph the mapping f is surjective. Define a truth mapping by setting the ith variable to true if p_i occurs in s_1, and false otherwise. Each symbol in the second group s_2 is covered. Select one symbol, say β_j from that group. The corresponding node, say c_k, has a parent (either p_i or n_i) that must be mapped into the first group. This implies that the truth value for the ith variable satisfies the jth clause. Since all clauses are satisfied, F is satisfied. This completes the proof. □

3.4 Subepisode Relationship

In practice, episodes are represented by DAGs and are mined by adding nodes and edges. However, such a representation has drawbacks [29]. The following example illustrates this.

Example 19. *Consider episodes H_1 and H_2 given in Figure 3.2. Even though these episodes have different graphs, they are essentially the same — both episodes are covered by exactly the same sequences, namely all sequences containing $abab$, $a(ab)b$, $(aa)(bb)$, $aa(bb)$, $(aa)bb$, or $aabb$.*

Figure 3.2: Two similar episodes.

The above example shows that essentially the same episode may be represented by several graphs. Moreover, using the graph subset relationship to determine subset relationships between episodes will ultimately lead to less efficient algorithms and redundancy in the final output. To counter these problems we introduce a subset relationship based on coverage.

Definition 35. *Given two episodes G and H, we say that G is a* subepisode *of H, denoted $G \preceq H$, if any sequence covering H also covers G. If $G \preceq H$ and $H \preceq G$, we say that G and H are* similar *in which case we will write $G \sim H$.*

This definition gives us the optimal definition for a subset relationship in the following sense: if $G \not\preceq H$, then there exists a sequence s such that $fr(G;s) < fr(H;s)$.

Example 20. *Consider the episodes given in Figures 3.1 and 3.2. It follows from the definition that $H_1 \sim H_2$, $G_1 \preceq G_2$, and $G_1 \preceq G_3$. Episodes G_2 and G_3 are not comparable.*

Theorem 4. *Testing $G \preceq H$ is an **NP**-hard problem.*

Proof. The hardness follows immediately from Theorem 3 as we can represent sequence s as a serial episode H. Then s covers G if and only if $G \preceq H$. □

As mentioned in the introduction, pattern explosion is *the* problem with discovering general episodes. We tackle this by mining only closed episodes.

Definition 36. *An episode G is closed if there is no $H \succ G$ with $fr(G) = fr(H)$.*

We should point out that, unlike with itemsets, an episode may have several maximal superepisodes having the same frequency. We illustrate this problem with the following example.

Example 21. *Consider episodes G_1, G_3, and G_4 given in Figure 3.3, sequence $abcbdacbcd$ and window size $\rho = 5$. The support of episodes G_1, G_3 and G_4 is 2. Moreover, there is no superepisode of G_3 or G_4 that has the same support. Thus, G_3 and G_4 are both maximal superepisodes having the same support as G_1.*

This implies that we cannot define a closure operator based on frequency. However, we will see in the next section that we can define a closure based on instances. This closure, while not removing all redundant episodes, will prune the search space dramatically. The final pruning will then be done in a post-processing step.

Figure 3.3: Toy episodes demonstrating closure.

Our final step is to define transitively closed episodes that we will use along with the instance-closure (defined in the next section) in order to reduce the pattern space.

Definition 37. *Let G be an episode. Its* transitive closure, *$tcl(G)$, is obtained by adding edges from a node to each of its descendants making the edge proper if the descendant is proper, and weak otherwise. If $G = tcl(G)$ we say that G is transitively closed.*

It is trivial to see that given an episode G, we have $G \sim tcl(G)$. Thus we can safely ignore all episodes that are not transitively closed. In the remainder of this chapter, all episodes are assumed to be transitively closed, unless stated otherwise.

3.5 Handling Episode Instances

The reason why depth-first search is efficient for itemsets is that at each step we only need to handle the current projected dataset. In our setup we have only one sequence so we need to transport the sequence into a more efficient structure.

Definition 38. *Given an input sequence s and an episode G, an* instance *i is a valid mapping from G to s such that for each $e \in range(i)$ there is no $f \in s - range(i)$ such that $lab(e) = lab(f)$, $ts(e) = ts(f)$ and $id(f) < id(e)$. We define $first(i) = \min ts(i(n))$ to be the smallest time stamp and $last(i) = \max ts(i(n))$ to be the largest time stamp in i. We require that $last(i) - first(i) \leq \rho - 1$, where ρ is the size of the sliding window. An* instance set *of an episode G, defined as $inst(G)$ is a set of all instances ordered by $first(i)$.*

Example 22. *Consider sequence $abcbdacbcd$ and G_1 in Figure 3.3. Then*

$$inst(G_1) = ((1,2,3,5),(1,4,3,5),(6,7,8,10),(6,9,8,10)).^3$$

The condition in the definition allows us to ignore some redundant mappings whenever we have two sequence events, say e and f, with $lab(e) = lab(f)$ and $ts(e) = ts(f)$. If an instance i uses only e, then we can obtain i' from i by replacing e with f. However, i and i' are essentially the same for our purposes, so we can ignore either i or i'. We require the instance set to be ordered so that we can compute the support efficiently. This order is not necessarily unique.

[3] For simplicity, we write mappings as tuples.

Using instances gives us several advantages. Adding new events and edges to episodes becomes easy. For example, adding a proper edge (n, m) is equivalent to keeping instances with $ts(i(n)) < ts(i(m))$. We will also compute support and closure efficiently. We should point out that $inst(G)$ may contain an exponential number of instances, otherwise Theorem 3 would imply that **P** = **NP**. However, this is not a problem in practice.

The depth-first search described in Section 3.6 adds events to the episodes. Our next goal is to define algorithms for computing the resulting instance set whenever we add an episode event, say f, to an episode G. Given an instance i of G and a sequence event e we will write $i+e$ to mean an expanded instance by setting $i(f) = e$.

Let G be an episode and let $I = inst(G)$ be the instance set. Assume a node $n \in V(G)$ and a label l. Let H be the episode obtained from G by adding an episode event with label l to node n. We can compute $inst(H)$ from $inst(G)$ using the AUGMENTEQUAL algorithm given in Algorithm 6.

Algorithm 6: AUGMENTEQUAL(I, n, l), augments I

input : $I = inst(G)$, a node n, a label l
output : $inst\big(G$ with n augmented with $l\big)$

1 **return** $\left\{ i+e \mid \begin{array}{l} i \in I, e \in s, lab(e) = l, \\ ts(n) = ts(e), i+e \text{ is an instance} \end{array} \right\}$;

The second augmentation algorithm deals with the case where we are adding a new node with a single event labelled l to a parallel episode. Algorithm AUGMENT, given in Algorithm 7, computes the new instance set. The algorithm can be further optimised by doing augmentation with a type of merge sort so that post-sorting is not needed.

Algorithm 7: AUGMENT(I, l), augments instances

input : $I = inst(G)$, label l of the new event
output : $inst(G$ with a new node labelled with $l)$

1 $E \leftarrow \{e \in s \mid lab(e) = l\}$;
2 $J \leftarrow \{i+e \mid i \in I, e \in E, i+e \text{ is an instance}\}$;
3 Sort J by $first(i)$;
4 **return** J;

Our next step is to compute the support of an episode G from $inst(G)$. We do this with the SUPPORT algorithm, given in Algorithm 8. The algorithm is based on the observation that there are $\rho - (last(i) - first(i))$ windows that contain the instance i. However, some windows may contain more than one instance and we need to compensate for this.

Theorem 5. SUPPORT$(inst(G))$ *computes* $fr(G)$.

Proof. Since I is ordered by $first(i)$, the first for loop of the algorithm removes any instance for which there is an instance j such that $first(i) \le first\big(j\big) \le last\big(j\big) \le last(i)$. In other words, any window that contains i will also contain j. We will show that the

Algorithm 8: SUPPORT(I), computes support

```
   input  : I = inst(G)
   output : fr(G)
1  J ← ∅; l ← ∞;
2  foreach i ∈ I in reverse order do
3      if last(i) < l then
4          Add i to J;
5          l ← last(i);
6  l ← −∞; f ← 0;
7  foreach i ∈ J do
8      d ← ρ − (last(i) − first(i));
9      a ← 1 + last(i) − ρ;
10     d ← d − max(0, 1 + l − a);
11     f ← f + d;
12     l ← first(i);
13 return f;
```

next for loop counts the number of windows containing at least one instance from W, this will imply the theorem.

To that end, let $i_n \in J$ be the nth instance in J and define S_n to be the set of windows of size ρ containing i_n. It follows that $|S_n| = \rho - (last(i_n) - first(i_n))$ and that the first window of S_n starts at $a_n = 1 + last(i_n) - \rho$. Let $C_n = \bigcup_{m=1}^{n} S_m$. Note that because of the pruning we have $a_{n-1} < a_n$, this implies that $C_{n-1} \cap S_n = S_{n-1} \cap S_n$. We know that $|S_{n-1} \cap S_n| = \max\big(0, 1 + first(i_{n-1}) - a_n\big)$. This implies that on Line 10 we have $d = |S_n - S_{n-1}|$ and since $|C_n| = |C_{n-1}| + d$, this proves the theorem. □

Finally, we define a closure episode of an instance set.

Definition 39. *Let $I = inst(G)$ be a set of instances. We define an* instance-closure*, $H = icl(I)$ to be the episode having the same nodes and events as G. We define the edges*

$$PE(H) = \{(a,b) \mid ts(i(a)) < ts(i(b)) \ \ \forall\, i \in I\} \text{ and}$$
$$WE(H) = \{(a,b) \mid ts(i(a)) \leq ts(i(b)) \ \ \forall\, i \in I\} - PE(H)\,.$$

If $G = icl(I)$ we say that G is i-closed.

Example 23. *Consider sequence $abcbdacbcd$ and episodes given in Figure 3.3. Since events b and c always occur between a and d in $inst(G_1)$, the instance closure is $icl(inst(G_1)) = G_2$. Note that G_2 is not closed because G_3 and G_4 are both superepisodes of G_2 with the same support.*

Despite the fact that some i-closed episodes are not actually closed, instance-closure reduces the search space dramatically because we do not have to iterate the edges implied by the closure. Note that the closure may produce cycles with weak edges. However, we will later show that we can ignore such episodes.

3.6 Discovering Episodes

We are now ready to describe the mining algorithm. Our approach is a straightforward depth-first search. The algorithm works on three different levels.

The first level, consisting of MINE (Algorithm 9), and MINEPARALLEL (Algorithm 10), adds episode events. MINE is only used for creating singleton episodes while MINEPARALLEL provides the actual search. The algorithm adds events so that the labels of the nodes are decreasing, $lab(nd(G,i+1)) \leq_{lex} lab(nd(G,i))$. The search space is traversed by either adding an event to the last node or by creating a node with a single event.

Algorithm 9: MINE, discovers frequent closed episodes

```
1  for x ∈ Σ do
2      I ← inst(singleton episode with the label x);
3      G ← TESTEPISODE(I, ∅, ∅);
4      if G ≠ null then MINEPARALLEL(I, G);
```

Algorithm 10: MINEPARALLEL(I,G), recursive routine adding episode events

```
   input : episode G, I = inst(G)
1  MINEWEAK(I, G, ∅);
2  M ← |V(G)|;
3  n ← nd(G, M);
4  for x ∈ Σ, x ≥ max lab(n) do
5      if M = 1 or lab(n) ∪ {x} ≤_lex lab(nd(G, M − 1)) then
6          J ← AUGMENTEQUAL(I, n, x);
7          H ← TESTEPISODE(J, ∅, ∅);
8          if H ≠ null then
9              MINEPARALLEL(J, H);
10 for x ∈ Σ, x ≤ min lab(n) do
11     J ← AUGMENT(I, x);
12     H ← TESTEPISODE(J, ∅, ∅);
13     if H ≠ null then
14         MINEPARALLEL(J, H);
```

The second level, MINEWEAK, given in Algorithm 11, adds weak edges to the episode, while the third level, MINEPROPER, given in Algorithm 12, turns weak edges into proper edges. Both algorithms add only those edges which keep the episode transitively closed. The algorithms keep a list of forbidden weak edges W and forbidden proper edges P. These lists guarantee that each episode is visited only once.

The following example illustrates the process of generating candidate episodes.

Example 24. *Assume we are given a sequence $(aa)ba$, window size $\rho = 2$, and support threshold $\sigma = 2$. Episodes $G_1, \ldots, G_5$ shown in Figure 3.4 are created by the first level of*

the algorithm. The edge in G_5 is augmented by the closure. G_6 and G_7 in Figure 3.4 are discovered by MINEWEAK. *However, the weak edges are converted into proper edges by the instance-closure.*

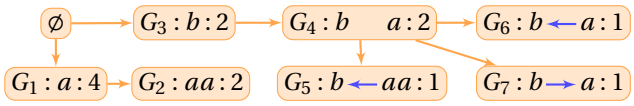


Figure 3.4: The candidate generation process discussed in Example 24. Each state shows the corresponding episode and its support.

Algorithm 11: MINEWEAK(I, G, W), recursive routine adding weak edges

```
   input  : ep. G, I = inst(G), forbidden weak edges W
1  A ← {(a,b) | a,b ∈ V(G), (a,b) ∉ E(G)};
2  MINEPROPER(I, G, A, ∅);
3  for (a,b) ∉ E(G) ∪ W do
4      if G + (a,b) is transitively closed then
5          J ← {i ∈ I | ts(i(a)) ≤ ts(i(b))};
6          H ← TESTEPISODE(J, W, ∅);
7          if H ≠ null then
8              MINEWEAK(J, H, W);
9          Add (a,b) to W;
```

Algorithm 12: MINEPROPER(I, G, W, P), recursive routine adding proper edges

```
   input  : episode G, I = inst(G), forbidden weak edges W, forbidden proper
            edges P
1  for (a,b) ∈ WE(G) − P do
2      if G + proper edge (a,b) is transitively closed then
3          J ← {i ∈ I | ts(i(a)) < ts(i(b))};
4          H ← TESTEPISODE(J, W, P);
5          if H ≠ null then
6              MINEPROPER(J, H, W, P);
7          Add (a,b) to P;
```

At each step, we call TESTEPISODE, given in Algorithm 13. This routine, given a set of instances I, will compute the instance-closure $icl(I)$ and test it. If the episode passes all the tests, the algorithm will return the episode and the search is continued, otherwise the branch is terminated. There are four different tests. The first test checks whether the episode is frequent. If we pass this test, we compute the instance-closure $H = icl(I)$. The second test checks whether H contains cycles. Let G be an episode such that $I = inst(G)$. In order for H to have cycles we must

have two nodes, say n and m, such that $ts(i(n)) = ts(i(m))$ for all $i \in I$. Let G' be an episode obtained from G by merging nodes in n and m together. We have $G \preceq G'$ and $fr(G) = fr(G')$. This holds for any subsequent episode discovered in the branch allowing us to ignore the whole branch.

If the closure introduces into H any edge that has been explored in the previous branches, then that implies that H has already been discovered. Hence, we can reject H if any such edge is introduced. The final condition is that during MINEPROPER no weak edges should be added into H by the closure. If a weak edge is added, we can reject H because it can be reached via an alternative route, by letting MINEWEAK add the additional edges and then calling MINEPROPER.

The algorithm keeps a list $\mathscr{C}$ of all discovered episodes that are closed. If all four tests are passed, the algorithm tests whether there are subepisodes of G in $\mathscr{C}$ having the same frequency, and deletes them. On the other hand, if there is a superepisode of G in $\mathscr{C}$, then G is not added into $\mathscr{C}$.

Algorithm 13: TESTEPISODE(I, W, P), tests the episode $icl(I)$ and updates $\mathscr{C}$, the list of discovered episodes

input : $I = inst(G)$, forbidden weak edges W, forbidden proper edges P
output : $icl(I)$, if $icl(I)$ passes the tests, **null** otherwise

```
1  f ← SUPPORT(I);
2  if f < σ then return null;
3  G ← icl(I);
4  if there are cycles in G then
5      return null;
6  if WE(G) ∩ W ≠ ∅ or PE(G) ∩ P ≠ ∅ then
7      return null;
8  fr(G) ← f;
9  foreach H ∈ C, lab(H) ∩ lab(G) ≠ ∅ do
10     if fr(G) = fr(H) and G ⪯ H then
11         return G;
12     if fr(G) = fr(H) and H ⪯ G then
13         Delete H from C;
14 Add G to C;
15 return G;
```

3.7 Testing Subepisodes

In this section we will describe the technique for computing $G \preceq H$. In general, this problem is difficult to solve as pointed out by Theorem 4. Fortunately, in practice, a major part of the comparisons are easy to do. The following theorem says that if the labels of G are unique in H, then we can easily compare G and H.

Theorem 6. *Assume two episodes G and H. Assume that $lab(G) \subset lab(H)$ and for each event e in G only one event occurs in H with the same label. Let π be the unique*

mapping from episode events in G to episode events in H honouring the labels. Then $G \preceq H$ if and only if

1. $nd(e) = nd(f)$ *implies that* $nd(\pi(e)) = nd(\pi(f))$,
2. $nd(e)$ *is a proper child of* $nd(f)$ *implies that* $nd(\pi(e))$ *is a proper child of* $nd(\pi(f))$,
3. $nd(e)$ *is a child of* $nd(f)$ *implies that* $nd(\pi(e))$ *is a child of* $nd(\pi(f))$ *or* $nd(\pi(e)) = nd(\pi(f))$,

for any two events e and f in G.

Proof. To prove the theorem we will use the following straightforward lemma.

Lemma 7. *Let G be a transitively closed episode. Let e and f be two nodes. Then there exists a sequence s covering G and a mapping m such that*

1. *if* $nd(e) \neq nd(f)$, *then* $ts(m(e)) \neq ts(m(f))$,
2. *if* $nd(e)$ *is not a child of* $nd(f)$, *then* $ts(m(e)) < ts(m(f))$,
3. *if* $nd(e)$ *is not a proper child of* $nd(f)$, *then* $ts(m(e)) \leq ts(m(f))$,

The 'if' part is straightforward so we only prove the 'only if' part. Assume that $G \preceq H$ and let e and f be two events in G violating one of the conditions. Apply Lemma 7 with H, $\pi(e)$, and $\pi(f)$ to obtain a sequence s and a mapping m. We can safely assume that m is surjective. Define $m'(x) = m(\pi(x))$ for an event x in G. Since every label is unique in s, m' is the only mapping that will honor the labels in G. Now Lemma 7 implies that m' is not a valid mapping for G, hence $G \not\preceq H$, which proves the theorem. □

If the condition in Theorem 6 does not hold we will have to resort to enumerating the sequences covering H. In order to do that, we need to extend the definition of coverage and subset relationship to the set of episodes.

Definition 40. *A sequence s covers an episode set $\mathcal{G}$ if there is an episode $G \in \mathcal{G}$ such that s covers G. Given two episode sets $\mathcal{G}$ and $\mathcal{H}$ we define $\mathcal{G} \preceq \mathcal{H}$ if every sequence that covers $\mathcal{H}$ also covers $\mathcal{G}$.*

We also need a definition of a prefix subgraph.

Definition 41. *Given a graph G, a* prefix subgraph *is a non-empty induced subgraph of G with* no *proper edges such that if a node n is included then the parents of n are also included. Given a multiset of labels L and an episode G we define $pre(G, L)$ to be the set of all maximal prefix subgraphs such that $lab(V) \subseteq L$ for each $V \in pre(G, L)$. We define $tail(G, L) = \{G - V \mid V \in pre(G, L)\}$ to be the episodes with the remaining nodes. Finally, given an episode set $\mathcal{G}$ we define $tail(\mathcal{G}, L) = \bigcup_{G \in \mathcal{G}} tail(G, L)$.*

Example 25. *Consider episodes given in Figure 3.5. We have $tail(G, ab) = \{H_1, H_2, H_3\}$ and $tail(G, a) = \{H_1\}$.*

The main motivation for our recursion is given in the following theorem.

Figure 3.5: Toy episodes demonstrating $tail(G, ab)$.

Theorem 8. *Given an episode set $\mathscr{G}$ and an episode H, $\mathscr{G} \preceq H$ if and only if for each prefix subgraph V of H, we have $tail(\mathscr{G}, lab(V)) \preceq H - V$.*

Proof. Assume that the condition in the theorem holds and let s be a sequence covering H and let m be the corresponding mapping. We can safely assume that m is surjective. Let t be the smallest time stamp in s and E the events in s having the time stamp t. Let V be the corresponding prefix subgraph,

$$V = \{n \in V(H) \mid ts(m(n)) = t\}.$$

By assumption, we have $tail(\mathscr{G}, lab(V)) \preceq H - V$, hence there is $G \in \mathscr{G}$ and $W \in pre(G, L)$ such that $G - W \preceq H - V$ and $lab(W) \subseteq lab(V)$. Let s' be the sequence obtained from s by removing E. Since s' covers $H - V$, there is a map m' mapping $G - W$ to s'. We can extend this map to G by mapping W to E. Thus s covers G, s covers $\mathscr{G}$, and by definition $\mathscr{G} \preceq H$.

To prove the other direction, assume that $\mathscr{G} \preceq H$. Let V be a prefix subgraph of H. Let s' be a sequence covering $H - V$. We can extend this sequence to s by adding the events with labels $lab(V)$ in front of s'. Let us denote these events by E. Sequence s covers H, and it therefore covers $\mathscr{G}$.

By assumption, there is a mapping m' from G to s for some $G \in \mathscr{G}$. Let W be the (possibly empty) prefix graph of G mapping to E. We can assume that $W \in pre(G, lab(V))$, that is, W is maximal, otherwise let $W' \in pre(G, lab(V))$ such that $W \subset W'$. We can modify m' by remapping the nodes in $W' - W$ to the events in E. This will make W equal to W'. Since s' covers $G - W$, it follows that s' covers $tail(\mathscr{G}, lab(V))$ which proves the theorem. □

We focus for the rest of this section on implementing the recursion in Theorem 8. We begin by an algorithm, GENERATE, given in Algorithm 14, that, given a graph without proper edges, discovers all prefix subgraphs.

Algorithm 14: GENERATE(G, V), recursive routine for iterating the nodes of all prefix subgraphs of G

input : graph G, nodes V discovered so far
output : list of nodes of all prefix subgraphs

```
1 𝒪 ← ∅;
2 foreach n ∈ src(G) do
3 |   𝒪 ← 𝒪 ∪ {V + n} ∪ GENERATE(G − n, V + n);
4 |   Remove n and its descendants from G;
5 return 𝒪;
```

Given a prefix subgraph V of H, our next step is to discover all maximal prefix subgraphs of G whose label sets are subsets of $L = lab(V)$. The algorithm, CONSUME, creating this list, is given in Algorithm 15. CONSUME enumerates over all sources. For each source n such that $lab(n) \subseteq L$, the algorithm tests if there is another node sharing a label with n. If so, the algorithm creates an episode without n and its descendants and calls itself. This call produces a list of prefix graphs $\mathscr{W}$ not containing node n. The algorithm removes all graphs from $\mathscr{W}$ that can be augmented with n (since in that case they are not maximal). Finally, CONSUME adds n to the current prefix subgraph, removes n from G, and removes $lab(n)$ from L.

Algorithm 15: CONSUME(G, L, V), recursive routine for iterating the nodes of all prefix subgraphs of G whose labels are contained in L

input : graph G, nodes V discovered so far, L label set
output : list of nodes of all maximal prefix subgraphs

```
1  𝒪 ← ∅;
2  while src(G) ≠ ∅ do
3      n ← a node from src(G);
4      if lab(n) ⊈ L then
5          Remove n and its descendants from G;
6          continue;
7      if there is m ∈ V(G) s.t. lab(n) ∩ lab(m) ≠ ∅ then
8          H ← G with n and its descendants removed;
9          𝒲 ← CONSUME(H, L, V);
10         𝒪 ← 𝒪 ∪ {W ∈ 𝒲 | lab(W) ∪ lab(n) ⊈ L};
11     V ← V + n; L ← L − lab(n);
12     Remove n from G;
13 return 𝒪 ∪ {V};
```

We are now ready to describe the recursion step of Theorem 8 for testing the subset relationship. The algorithm, STEP, is given in Algorithm 16. Given an episode set $\mathscr{G}$ and a graph H, the algorithm computes $\mathscr{G} \preceq H$. First, it tests whether we can apply Theorem 6. If this is not possible, then the algorithm first removes all nodes from H not carrying a label from $lab(G)$ for $G \in \mathscr{G}$. This is allowed because of the following lemma.

Lemma 9. *Let G and H be two episodes. Let $n \in V(H)$ be a node such that $lab(n) \cap lab(G) = \emptyset$. Let H' be the episode obtained from H by removing n. Then $G \preceq H$ if and only if $G \preceq H'$.*

Proof. The 'if' part is trivial. To prove the 'only if' part, assume that $G \preceq H$. Let s' be a sequence covering H' and let m' be the mapping. If we can show that s' can be extended to s by adding events with the labels $lab(n)$ such that s covers G, then the corresponding mapping m will also be a valid mapping from G to s', which will prove the lemma.

If n is a source or a sink (a node without children), then we can extend s' by adding the event with the label $lab(n)$ either at the beginning or at the end of s'.

Algorithm 16: STEP($\mathcal{G}, H$), recursion solving $\mathcal{G} \preceq H$

input : episode set $\mathcal{G}$ and an episode H
output : $\mathcal{G} \preceq H$

foreach $G \in \mathcal{G}$ **do**
  if Theorem 6 guarantees that $G \preceq H$ **then**
    return true;
  if Theorem 6 states that $G \not\preceq H$ **and** $|\mathcal{G}| = 1$ **then**
    return false;
$V(H) \leftarrow \{n \in V(H) \mid lab(n) \cap lab(G) \neq \emptyset, G \in \mathcal{G}\}$;
$Y \leftarrow$ subgraph of H with only proper sources;
$\mathcal{V} \leftarrow$ GENERATE($Y, \emptyset$);
foreach $V \in \mathcal{V}$ **do**
  $F \leftarrow H - V$; $\mathcal{T} \leftarrow \emptyset$;
  foreach $G \in \mathcal{G}$ **do**
    $X \leftarrow$ subgraph of G with only proper sources;
    $\mathcal{W} \leftarrow$ CONSUME($X, lab(V), \emptyset$);
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{G - W \mid W \in \mathcal{W}\}$;
  if STEP($\mathcal{T}, F$) = **false then**
    return false;
return true;

Assume that n has parents and children. Let t_1 be the largest time stamp of the parents of n and let t_2 be the smallest time stamp of the children of n. Extend s' to s by adding an event f with the label $lab(n)$ and the time stamp $t = 1/2(t_1 + t_2)$. Let x be a child of n and y a parent of n. Since H is transitively closed, x is a child of y in H'. Since this holds for any x and y, we have $t_1 \leq t_2$. Hence $ts(y) \leq t \leq ts(x)$. If x is proper descendant of n, then it is also a proper descendant of any parent of n, thus $t_1 < ts(x)$ and consequently $t < ts(x)$. The same holds for y. This implies that s covers G. □

STEP continues by creating a subgraph Y of H containing only proper sources. The algorithm generates all prefix subgraphs $\mathcal{V}$ of Y and tests each one. For each subgraph $V \in \mathcal{V}$, STEP calls CONSUME and builds an episode set $\mathcal{T} = tail(\mathcal{G}, lab(V))$. The algorithm then calls itself recursively with STEP($\mathcal{T}, H - V$). If at least one of these calls fails, then we know that $\mathcal{G} \not\preceq H$, otherwise $\mathcal{G} \preceq H$.

Finally, to compute $G \preceq H$, we simply call STEP($\{G\}, H$).

3.8 Experiments

We begin our experiments with a synthetic dataset. Our goal is to demonstrate the need for using the closure. In order to do that, we created sequences with a planted pattern $(s_1 s_2)(s_3 s_4) \cdots (s_{2N-1} s_{2N})$. We added this pattern 100 times 50 time units apart from each other. We added 500 noise events uniformly spreading over the whole sequence. We sampled the labels for the noise events uniformly from 900

different labels. The labels for the noise and the labels of the pattern were mutually exclusive. We varied N from 1 to 7. We ran our miner using a window size of $\rho = 10$ and varied the support threshold σ. The results are given in Table 3.1.

$\lvert V(P)\rvert$	σ	$\lvert\mathscr{C}\rvert$	i-closed	frequent	scans
1	100	1	3	3	46
2	100	3	15	27	200
3	100	6	63	729	744
4	100	10	255	59049	3964
5	100	15	1023	14348907	11123
6	100	21	4095	$\approx 10^{10}$	48237
7	100	28	16383	$\approx 2\times 10^{13}$	191277
7	50	29	16384	$> 2\times 10^{13}$	191243
7	40	32	16387	$> 2\times 10^{13}$	191298
7	30	39	16394	$> 2\times 10^{13}$	191463
7	20	127	16488	$> 2\times 10^{13}$	197773
7	10	684	52297	$> 2\times 10^{13}$	480517

Table 3.1: Results from synthetic sequences with a planted episode P. The columns show the number of nodes in P, the support threshold, the size of the final output, the number of instance-closed episodes, the number of subepisodes of P, and the number of sequence scans, respectively.

With a support threshold of 100, the only closed frequent patterns are the planted pattern and its subpatterns of form $(s_i s_{i+1})\cdots(s_j s_{j+1})$, since the frequency of these subpatterns is slightly higher than the frequency of the whole pattern. The number of instance-closed episodes (given in the 4^{th} column of Table 3.1) grows more rapidly. The reason for this is that the instance-closure focuses on the edges and does not add any new nodes. However, this ratio becomes a bottleneck only when we are dealing with extremely large serial episodes, and for our real-world datasets, discussed further below, this ratio is relatively small.

The need for instance-closure becomes apparent when the number of instance-closed episodes is compared to the number of all possible general subepisodes (including those that are not transitively closed) of a planted pattern, given in the 5^{th} column of Table 3.1. We see that had we not used instance-closure, a single pattern having 6 nodes and 12 events renders pattern discovery infeasible.

As we lower the threshold, the number of instance-closed episodes and closed episodes increases, however the ratio between instance-closed and closed episodes improves. The reason for this is that the output contains episodes other than our planted episode, and those mostly contain a small number of nodes.

We also measured the number of sequence scans, namely the number of calls made to SUPPORT, to demonstrate how fast the negative border is growing. Since our miner is a depth-first search and the computation of frequency is based on instances, a single scan is fast, since we do not have to scan the whole sequence.

Our second set of experiments was conducted on real-world data. The dataset consists of alarms generated in a factory, and contains 514 502 events of 9 595 different types, stretching over 18 months. An entry in the dataset consists of a time stamp and an event type. Once again, we tested our algorithm at various thresholds, varying the window size from 3 to 15 minutes. Here, too, as shown in Table 3.2, we can see that the number of i-closed episodes does not explode to the level of all frequent episodes, demonstrating the need for using the i-closure as an intermediate step. Furthermore, we see that in a realistic setting, the number of i-closed episodes stays closer to the number of closed episodes than in the above-mentioned synthetic dataset. This is no surprise, since real datasets tend to contain a lot of patterns containing a small number of events.

As the discovery of all frequent episodes is infeasible, we estimated their number as follows. An episode G has $a(G) = 3^{|PE(G)|}2^{|WE(G)|}$ subepisodes (including those that are not transitively closed) with the same events and nodes. From the discovered i-closed episodes we selected a subset $\mathscr{G}$ such that each $G \in \mathscr{G}$ has a unique set of events and a maximal $a(G)$. Then the lower bound for the total number of frequent episodes is $\sum_{G\in\mathscr{G}} a(G)$. Using such a lower bound is more than enough to confirm that the number of frequent episodes explodes much faster than the number of closed and i-closed episodes, as can be seen in Table 3.2.

Furthermore, our output contained a considerable number of episodes with simultaneous events — patterns that no existing method would have discovered. The runtimes ranged from just under 2 seconds to 90 seconds for the lowest reported thresholds.

win (s)	$\sigma/10^3$	$\lvert\mathscr{C}\rvert$	i-closed	freq.(est)	scans
180	500	6	6	6	194
180	400	8	8	8	220
180	300	12	12	12	282
180	240	23	26	26	792
600	2 000	4	4	4	128
600	1 000	24	27	39	374
600	500	90	137	493	1 196
600	280	422	698	2 321	8 758
900	2 000	24	26	40	350
900	1 000	52	58	94	745
900	500	280	426	1 997	4 604
900	350	1 845	9 484	190 990	63 735

Table 3.2: Results from the *alarms* dataset.

Our third dataset consisted of trains delayed at a single railway station in Belgium. The dataset consists of actual departure times of delayed trains, coupled with train numbers, and contains 10 115 events involving 1 280 different train IDs, stretching over a period of one month. A window of 30 minutes was chosen by a domain

expert. The time stamps were expressed in seconds, so a single train being delayed on a particular day would be found in 1800 windows. Therefore, the frequency threshold for interesting patterns had to be set relatively high. The results are shown in Table 3.3, and were similar to those of the alarm dataset. The runtimes ranged from a few milliseconds to 55 minutes for the lowest reported threshold. The largest discovered pattern was of size 10, and the total number of frequent episodes at the lowest threshold was at least 33 million, once again demonstrating the need for both outputting only closed episodes and using instance-closure.

σ	$\lvert\mathscr{C}\rvert$	i-closed	freq.(est)	scans
30000	141	141	141	9575
20000	1994	1995	2593	219931
17000	8352	8416	22542	812363
15000	26170	26838	172067	2231360
13000	94789	101882	3552104	6865877
12000	189280	211636	33660094	12966895

Table 3.3: Results from the *trains* dataset.

3.9 Conclusions

In this chapter we introduced a new type of sequential pattern, a general episode that takes into account simultaneous events, and provide an efficient depth-first search algorithm for mining such patterns.

This problem setup has two major challenges. The first challenge is the pattern explosion which we tackle by discovering only closed episodes. Interestingly enough, we cannot define closure based on frequency, hence we define closure based on instances. While it holds that all frequency-closed episodes are also instance-closed, the opposite is not true. However, in practice, instance-closure reduces search space dramatically so we can mine all instance-closed episodes and discover frequency-closed episodes as a post-processing step.

The second challenge is to correctly compute the subset relationship between two episodes. We argue that using a subset relationship based on graphs is not optimal and will lead to redundant output. We define a subset relationship based on coverage and argue that this is the correct definition. This definition turns out to be **NP**-hard. However, this is not a problem since in practice most of the comparisons can be done efficiently.

3.10 Acknowledgments

We wish to thank Toon Calders for providing the proof that checking whether a sequence covers an episode is **NP**-hard on a small piece of paper.

CHAPTER 4

Using Cohesion for Mining Patterns in Sequences

Discovering interesting patterns in long sequences, and finding confident association rules within them, is a popular area in data mining. Most existing methods define patterns as interesting if they occur frequently enough in a sufficiently cohesive form. Based on these frequent patterns, association rules are mined in the traditional manner. In this chapter[1]*, we introduce a new interestingness measure, combining cohesion and frequency of a pattern, whereby patterns are deemed interesting if encountering one event from the pattern implies with a high probability that the rest of the pattern can be found nearby. In a dataset consisting of a single sequence, frequency is defined as the probability of observing an event included in the pattern, while the cohesion is defined as the average length of the smallest intervals containing the pattern for each occurrence of its events.*

We further propose to introduce the concept of association rules into this problem setting. The confidence of such an association rule tells us how far on average from a particular event, or a set of events, one has to look, in order to find the rest of the pattern. We present algorithms to efficiently identify the interesting itemsets and association rules within them, given a dataset and user-defined thresholds. After applying our methods to both synthetic and real-life data, we conclude that they indeed give intuitive results in a number of applications.

[1]This chapter is based on work published in SDM 2009 as "A new constraint for mining sets in sequences" by Boris Cule, Bart Goethals and Céline Robardet [13] and in PAKDD 2010 as "Mining Association Rules in Long Sequences" by Boris Cule and Bart Goethals [12].

4.1 Introduction

Discovering interesting episodes [23] is a popular area in temporal or sequential data mining, examples of which are mining text or protein sequences. In such data, the order in which the events appear is being analysed and the user's goal is to identify the regularities that may appear in the dataset, consisting of one or more sequences. The usual approach to episode discovery is to look for episodes consisting of events that frequently appear close to each other. Most of the current state-of-the-art methods first use a window of fixed length to find sufficiently cohesive episodes and then retrieve those that occur in more windows (or sequences) than a given minimum threshold [23]. The use of a window of fixed length is a major limitation of such approaches as no episodes longer than this window can ever be discovered. A different method that increases the window length proportionally to the size of the candidate set has been proposed in order to remove this limitation [8]. Still, in this proposal, the window length remains fixed for a particular candidate when counting its frequency in the sequence. Therefore, when the episode occurs in the sequence, but in a time frame larger than the window size, such occurrences will be disregarded. The high frequency of a set of events appearing close together gives no guarantee that a subset of that set will not sometimes appear far away from the rest of the set.

In this chapter we propose a new constraint to select interesting sets of events. We focus on parallel episodes which are unordered sets of events, and these can thus be considered as sets of items, or itemsets. When looking at one sequence, we define the interestingness of an itemset based on a combination of how often the items in the candidate set appear in the sequence and how close to each other they appear on average. Later on, we present a similar definition for datasets consisting of many sequences. In this way, our approach does not use a fixed window length but instead also takes the occurrences of the items far from the rest of the set into account. It is precisely such occurrences that might sufficiently lower the cohesion of an itemset to render it uninteresting.

As we will show, the introduced interestingness measure has useful properties that allow us to develop an efficient algorithm to search for interesting itemsets, depending on a user-defined threshold. We present a divide-and-conquer method and prune the search space by computing an upper bound on the interestingness measure for potential candidate itemsets. Furthermore, we propose an alternative algorithm that prunes less, but actually runs faster, by utilising a looser, but much faster, pruning technique.

We also introduce the concept of association rules into this setting. We wish to find itemsets that imply the occurrence of other, larger, itemsets nearby. We present an efficient algorithm to mine such rules, taking advantage of two factors that lead to its efficiency — we can mine itemsets and rules in parallel, and we only need to compute the confidence of a simple class of rules, and use them to evaluate all other rules.

We apply our approach to various synthetic and real-life datasets to test the intuitiveness of our method and the efficiency of our algorithms. To test our ability of discovering interesting itemsets, we first use a Markov chain to generate synthetic datasets for which we know that they contain interesting itemsets and test the ability of our method to retrieve them. We also generate datasets using a uniform

distribution of items to test whether our method would output spurious itemsets. Finally, we apply the algorithm on two real-life datasets, an amino-acid sequence of an enzyme and a note sequence of a piece of music, and we analyse the results. To demonstrate the fact that our algorithm for mining association rules runs efficiently and produces meaningful results, we also used three text datasets, generated from books written in three different languages.

The chapter is organised as follows. In Section 4.2, we present the most relevant related work, demonstrating the weaknesses of the main existing methods on a couple of toy examples. Section 4.3 gives the full description of our interestingness constraint for datasets consisting of a single sequence and, using the same examples, demonstrates its relevance and intuitiveness. Section 4.4 presents a short sketch of our algorithm including a pruning method. We propose an even faster algorithm for mining itemsets in Section 4.5, before introducing our algorithm for mining association rules in Section 4.6. In Section 4.7 we experimentally demonstrate the effectiveness of our algorithm applied on synthetic and real-life datasets. In Section 4.8 we extend our definitions to multiple input sequences before presenting an algorithm for this setting, too. Finally, in Section 4.9 we end the chapter with concluding remarks and highlight some possibilities for future work.

4.2 Related Work

Looking for frequent episodes in an event sequence was introduced by Mannila et al. [23]. The proposed WINEPI algorithm finds all episodes that occur in a sufficient number of windows of fixed length. This window length is chosen by the user indicating how close to each other the events of an interesting episode should be. The frequency of an episode is defined as the number of windows in which the episode occurs divided by the total number of possible windows of chosen length.

As was pointed out by Garriga [8], WINEPI suffers from bias against longer episodes. We illustrate this problem with the following example.

Example 26. *Figure 4.1 shows a sequence, in which episode cde appears twice, as does its subepisode cd. Clearly, to a user, a longer episode would be more interesting than a shorter one, but applying* WINEPI *(with a window length greater than 1) to this sequence will always result in $\{c,d\}$ having a larger frequency than $\{c,d,e\}$. For example, using a window of length 3 gives*

$$\begin{aligned} fr(\{c,d\}) &= \frac{4}{14}, \\ fr(\{c,d,e\}) &= \frac{2}{14}. \end{aligned}$$

This is clearly unintuitive. Garriga [8] proposes to solve this problem by increasing the window length proportionally to the episode length, using a parameter *tus* equal to the maximal gap allowed between two events in an episode.

Example 27. *Returning to the sequence given in Figure 4.1, using* tus = 2 *(equivalent to using a window of length 3 for an episode made of two event types) would result in a window of length 5 for episodes made of three event types. The frequencies of*

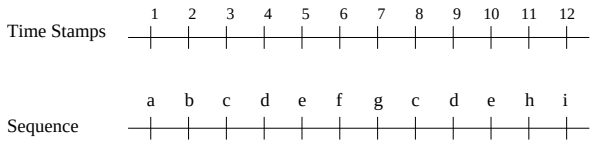


Figure 4.1: An illustrative example.

episodes $\{c,d\}$ *and* $\{c,d,e\}$ *are in this context*

$$\begin{aligned} fr(\{c,d\}) &= \frac{4}{14}, \\ fr(\{c,d,e\}) &= \frac{8}{16}. \end{aligned}$$

For episodes of equal length, WINEPI and Garriga methods give equal results when the chosen *tus* results in the window length chosen for WINEPI. However, these results are also not always intuitive.

Example 28. *The sequence given in Figure 4.2 illustrates that frequency alone is not sufficient to estimate an episode's interestingness. For example, event types c and d appear relatively close to each other more often than event types a and b, and yet episode* $\{a,b\}$ *seems to be more interesting than* $\{c,d\}$ *since its event types always occur close to each other. At the same time, event types e and f also always occur close to each other but their low frequency makes episode* $\{e,f\}$ *less interesting than* $\{c,d\}$.

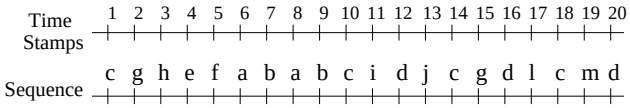


Figure 4.2: A second example

The above mentioned methods of discovering interesting episodes do not take this into account. If we apply the WINEPI *method on the example of Figure 4.2 with a window of length 3, we obtain*

$$\begin{aligned} fr(\{c,d\}) &= \frac{5}{22}, \\ fr(\{a,b\}) &= \frac{4}{22}, \\ fr(\{e,f\}) &= \frac{2}{22}. \end{aligned}$$

The ranking will be the same for all windows of length greater than 3. The same

method with a window of length 2 would give:

$$\begin{aligned} fr(\{c,d\}) &= \frac{0}{21}, \\ fr(\{a,b\}) &= \frac{3}{21}, \\ fr(\{e,f\}) &= \frac{1}{21}. \end{aligned}$$

We can observe that no chosen window length gives the desired result. Garriga's method would give the same results if we chose the corresponding tus *parameter.*

These problems are not surprising, since both WINEPI and Garriga are trying to find how likely we are to encounter an episode, while it may be more interesting to find how likely we are to encounter an event from an episode, and, once we encountered it, how likely we are to encounter the other events of the episode nearby.

As a result, both WINEPI and Garriga ignore the first c in the example in Figure 4.2, while precisely that c plays a pivotal role in our method, indicating that, once we encountered a c, we are not always likely to find a d nearby.

Mannila et al. propose another method, MINEPI, to search for frequent itemsets based on their minimal occurrences [23]. In this method, a minimal occurrence of an episode is defined as a window in which the episode occurs such that the episode does not occur in any of its proper subwindows. The support of an episode is then defined as the number of such windows.

Two main disadvantages of MINEPI are the use of support instead of a frequency ratio making it difficult for the user to make the correct choice, as well as the fact that the method completely disregards the distance between the items in an episode. On top of that, despite the original claim that the introduced support measure is anti-monotonic, this does not hold, as illustrated by Example 29.

Example 29. *Consider sequence $s = aba$. Clearly, episode $X = \{b\}$ has just one minimal occurrence in s, while parallel episode $Y = \{a,b\}$ has two minimal occurrences in s.*

The WINEPI method proposed by Mannila et al. can be extended to also detect association rules within frequent itemsets. As the frequency of an itemset is defined as the proportion of all windows that contain it, the confidence of an association rule $X \Rightarrow Y$, denoted $c(X \Rightarrow Y)$, is defined as the ratio of the frequency of $X \cup Y$ and the frequency of X. Once the frequent itemsets have been found, rules between them are generated in the traditional manner [1]. We have already shown that WINEPI is far from the best method for discovering interesting itemsets, but the following example demonstrates its unintuitiveness of the WINEPI method for evaluating association rules, too.

Example 30. *Consider the sequence given in Figure 4.2. We see that for each occurrence of event a, there is an occurrence of event b right next to it. Similarly, for each g, there is a c right next to it. Logically, association rules $a \Rightarrow b$ and $g \Rightarrow c$ should be equally confident (to be precise, their confidence should be equal to 1). If we choose a window size of 2, the number of possible windows is 21, as the first window*

starts one time stamp before the sequence, and the last one ends one time stamp after the sequence. WINEPI*, therefore, results in* $fr(\{a\}) = \frac{4}{21}$, $fr(\{a,b\}) = \frac{3}{21}$, $fr(\{g\}) = \frac{4}{21}$, $fr(\{c,g\}) = \frac{2}{21}$. *and thus* $c(\{a\} \Rightarrow \{b\}) = 0.75$, $c(\{g\} \Rightarrow \{c\}) = 0.5$. *Any larger chosen window size always gives similar results, namely* $c(\{g\} \Rightarrow \{c\}) < c(\{a\} \Rightarrow \{b\}) < 1$.

In the MINEPI context, association rules are limited to the form $X[win_1] \Rightarrow Y[win_2]$, meaning that if itemset X has a minimal occurrence in a window W_1 of size win_1, then $X \cup Y$ has a minimal occurrence in a window W_2 of size win_2 that fully contains W_1. As we plan to develop rules that tell us how likely we are to find some items nearby, we do not wish to use any fixed window lengths, so these are precisely the sort of rules we wish to avoid.

Generating association rules based on a maximum gap constraint, as defined by Garriga [8], was done by Méger and Rigotti [24], but only for serial episodes X and Y, where X is a prefix of Y. Most other related work has been based on the WINEPI definitions, and only attempted to find the same rules (or a representative subset) more efficiently [15, 21].

4.3 Problem Setting

As mentioned in the introduction, parallel episodes can also be looked at as itemsets. We therefore consider an event to be a couple consisting of an item and a time stamp (i,t) where $i \in I$, the set of all possible items, and $t \in \mathbb{N}$. For the purposes of this chapter, we consider all consecutive events to be equidistant and we assume that two events can never occur at the same time. In our notation, therefore, without any loss of generality, we consider the set of time stamps of an event sequence to be an uninterrupted sequence of natural numbers. We denote a sequence of such events by $\mathscr{S}$.

Definition of the Constraint

In this section, we introduce our interestingness measure that makes it possible to find itemsets that appear frequently in the sequence and consist of items that, on average, appear close to each other. The interestingness of an itemset will therefore depend on two factors: its coverage and its cohesion. Coverage measures how often an item of the itemset appears in the sequence, while cohesion measures how close together the items making up the itemset appear on average.

Definition 42. *Given an input sequence s, and an itemset X, we denote the set of all occurrences of its items in s as*

$$N(X) = \{t \mid (i,t) \in \mathscr{S} \text{ and } i \in X\}.$$

The coverage *of X, denoted $P(X)$, can now be defined as*

$$P(X) = \frac{|N(X)|}{|\mathscr{S}|}.$$

In order to compute the cohesion, we must first evaluate the length of the shortest interval containing the itemset X for each position of $N(X)$.

Definition 43. *Given an itemset X, and a time stamp t, the* length of the minimal window around t containing X, *denoted $W(X, t)$, is defined as*

$$W(X, t) = \min\{t_2 - t_1 + 1 \mid t_1 \leq t \leq t_2 \text{ and } \forall i \in X, \exists (i, t') \in \mathscr{S}, t_1 \leq t' \leq t_2\}.$$

The average length of such shortest intervals is computed as

$$\overline{W}(X) = \frac{\sum_{t \in N(X)} W(X, t)}{|N(X)|}.$$

It is clear that $\overline{W}(X)$ is greater than or equal to the number of items in X. Furthermore, for a fully cohesive itemset, where no other events ever occur between the items making the itemset, $\overline{W}(X) = |X|$. To normalise, we want the cohesion of a fully cohesive itemset to be equal to 1, and the cohesion of other itemsets to be lower.

Definition 44. *Given an input sequence s, and an itemset X, the* cohesion *of X, denoted $C(X)$, is defined as*

$$C(X) = \frac{|X|}{\overline{W}(X)}.$$

Definition 45. *Given an input sequence s and an itemset X, the* interestingness *of X, denoted $I(X)$, is defined as*

$$I(X) = C(X)P(X) = \frac{|N(X)| \times |N(X)| \times |X|}{\sum_{t \in N(X)} W(X, t) \times |\mathscr{S}|}.$$

Given a user defined threshold min_int, an itemset X is considered interesting if $I(X) \geq min\_int$.

Note that both coverage and cohesion of an itemset are real numbers between 0 and 1, and the same is therefore true for its interestingness. This makes it possible to apply the same interestingness threshold to any kind of dataset.

In principle, the user can enter a coverage threshold (*min_cov*) and/or a cohesion threshold (*min_coh*) as separate parameters. In such a case, an interesting itemset must satisfy all of the following:

$$P(X) \geq min\_cov,$$

$$C(X) \geq min\_coh,$$

$$I(X) \geq min\_int.$$

An optional parameter, *max_size*, can be used to limit the output only to itemsets with a size smaller than or equal to *max_size*.

Example 31. *In the sequence shown in Figure 4.1, itemsets $\{c, d\}$ and $\{c, d, e\}$ are both fully cohesive, but $\{c, d, e\}$ clearly covers more of the sequence than $\{c, d\}$, and will therefore be considered more interesting. According to our definition, $P(\{c, d\}) = \frac{4}{12}$ and $P(\{c, d, e\}) = \frac{6}{12}$. As $\{c, d\}$ and $\{c, d, e\}$ are both fully cohesive, $C(\{c, d\}) = 1$ and $C(\{c, d, e\}) = 1$. Therefore, $I(\{c, d\}) = \frac{4}{12}$ and $I(\{c, d, e\}) = \frac{6}{12}$.*

In general, an itemset will always have a lower coverage than any of its supersets, and, provided that they are equally cohesive, will always have a lower interestingness than the superset. On the other hand, we should note that the cohesion of an itemset alone is also not suitable for estimating its interestingness. This is why our interestingness measure consists of combining both coverage and cohesion. The following example illustrates this issue.

Example 32. *In the sequence shown in Figure 4.2, itemset* $\{e, f\}$ *is just as cohesive as itemset* $\{a, b\}$*, but its frequency is too small to be considered more interesting than itemset* $\{c, d\}$*.*

We shall now illustrate how our method computes the interestingness of itemset $\{c, d\}$*. The positions of the items* c *and* d *are in this case*

$$N(\{c, d\}) = \{1, 10, 12, 14, 16, 18, 20\},$$

and the coverage of itemset $\{c, d\}$ *is therefore*

$$P(cd) = \frac{7}{20}.$$

We proceed by computing the minimal interval around each position in $N(\{c, d\})$ *containing both* c *and* d*.*

$$\begin{aligned}
W(\{c,d\},1) = length([1,12]) + 1 &= 12,\\
W(\{c,d\},10) = length([10,12]) + 1 &= 3,\\
W(\{c,d\},12) = length([10,12]) + 1 &= 3,\\
W(\{c,d\},14) = length([12,14]) + 1 &= 3,\\
W(\{c,d\},16) = length([14,16]) + 1 &= 3,\\
W(\{c,d\},18) = length([16,18]) + 1 &= 3,\\
W(\{c,d\},20) = length([18,20]) + 1 &= 3.
\end{aligned}$$

The average length of these intervals is

$$\overline{W}(\{c, d\}) = \frac{30}{7}.$$

The cohesion is thus equal to

$$C(\{c, d\}) = \frac{2 \times 7}{30} = \frac{7}{15},$$

and, finally, the interestingness is equal to

$$I(\{c, d\}) = \frac{7}{15} \times \frac{7}{20} = \frac{49}{300}.$$

The summary of the computation of the interestingness for all three itemsets discussed in this example is given in Table 4.1.

Comparing these results with those obtained in Section 4.2, it can be observed that our method gives more intuitive results than WINEPI and Garriga.

itemset	$\lvert N(X)\rvert$	$\overline{W}(X)$	$C(X)$	$P(X)$	$I(X)$
$\{a,b\}$	4	2	1	0.2	0.2
$\{c,d\}$	7	4.29	0.47	0.35	0.16
$\{e,f\}$	2	2	1	0.1	0.1

Table 4.1: The evaluation of itemsets $\{a,b\}$, $\{c,d\}$ and $\{e,f\}$ in the sequence shown in Figure 4.2.

Association Rules

We are now ready to define the concept of association rules in this setting. The aim is to generate rules of the form *if X occurs, Y occurs nearby*, where $X \cap Y = \emptyset$ and $X \cup Y$ is an interesting itemset. We denote such a rule by $X \Rightarrow Y$, and we call X the *body* of the rule and Y the *head* of the rule. Clearly, the closer Y occurs to X on average, the higher the value of the rule. In other words, to compute the confidence of the rule, we must now use the average length of minimal windows containing $X \cup Y$, but only from the point of view of items making up itemset X. This new average can be computed as

$$\overline{W}(X,Y) = \frac{\sum_{t \in N(X)} W(X \cup Y, t)}{|N(X)|}.$$

Definition 46. *Given itemsets X and Y, the* confidence *of the association rule $X \Rightarrow Y$ is defined as*

$$c(X \Rightarrow Y) = \frac{|X \cup Y|}{\overline{W}(X,Y)}.$$

An association rule $X \Rightarrow Y$ is considered confident if its confidence exceeds a given threshold, min_conf.

The following example shows that our method gives the desired results for the examples discussed in Section 4.2.

Example 33. *We once again return to the sequence given in Figure 4.2. Looking at itemset $\{c,d\}$, we see that the occurrence of a c at time stamp 1 will reduce the value of rule $\{c\} \Rightarrow \{d\}$, but not of rule $\{d\} \Rightarrow \{c\}$. Indeed, we see that $W(\{c,d\},1) = 12$, and the minimal window containing $\{c,d\}$ for the other three occurrences of c is always of size 3. Therefore, $\overline{W}(\{c\},\{d\}) = \frac{21}{4} = 5.25$, and $c(\{c\} \Rightarrow \{d\}) = \frac{2}{5.25} = 0.38$. Meanwhile, the minimal window containing $\{c,d\}$ for all occurrences of d is always of size 3. It follows that $\overline{W}(\{d\},\{c\}) = \frac{9}{3} = 3$ and $c(\{d\} \Rightarrow \{c\}) = \frac{2}{3} = 0.67$. We can conclude that while an occurrence of a c does not highly imply finding a d nearby, when we encounter a d we can be reasonably certain that a c will be found nearby. We also note that, according to our definitions, $c(\{a\} \Rightarrow \{b\}) = 1$ and $c(\{g\} \Rightarrow \{c\}) = 1$, as desired.*

4.4 Algorithm Sketch

In order to find all interesting itemsets, our algorithm generates candidates and uses a pruning technique based on an upper-bound of the constraint to reduce, as soon as possible, the search space. These two main steps are described in detail below.

Candidate Generation

We generate candidates by applying the classical "divide-and-conquer" algorithm [6, 17, 10]. In a nutshell, the algorithm recursively enumerates, in a depth-first manner, all itemsets containing an element, say a, and then all itemsets not containing a. During the enumeration process, a candidate $\langle X, Y \rangle$ is composed of two sets of items: one, denoted X, that contains the items contained in the candidate and its descendants in the search tree (the ones obtained by recursive calls) and another one, denoted Y, that contains the items that still have to be enumerated.

The pseudo code of the INIT algorithm we have implemented is given in Algorithm 17. At the first line, the pruning function presented further in this section is called to check if the recursion process can be stopped. If the recursion is not stopped, the second test evaluates if there are still elements to be enumerated. If not, the candidate is an interesting itemset and is added to the output. Otherwise, an element a is picked from Y and the function is recursively called twice: once with a in X, and once without. In both calls, a is removed from Y. For the first call, X is empty and Y is equal to the set of all items appearing in the sequence.

Algorithm 17: INIT($\langle X, Y \rangle$) finds interesting itemsets

```
1 if UBI(⟨X, Y⟩) ≥ min_int and size(X) ≤ max_size then
2     if Y = ∅ then
3         output X ;
4     else
5         Choose a in Y ;
6         INIT(⟨X ∪ {a}, Y \ {a}⟩) ;
7         INIT(⟨X, Y \ {a}⟩) ;
```

For efficiency, we sort the items with respect to their frequency in descending order. The logic behind this decision is that to prune a candidate $\langle X, Y \rangle$ most efficiently, we would need to encounter the set $X' \subseteq X \cup Y$ such that $\sum_{t \in N(X')} W(X', t)$ is sufficiently high to trigger the pruning. A heuristic to increase the chance of encountering it early is to generate candidates using items sorted with respect to their frequency in descending order, because if the frequency of an itemset is higher than that of another itemset then its corresponding sum of interval lengths is also likely to be higher.

This algorithm satisfies the following useful properties:

1. The time needed to generate an interesting itemset is, in the worst case, bounded by $O(T \times |I|^2)$, where I is the set of all possible items and T is the time complexity of computing the pruning function $UBI(\langle X, Y \rangle)$ explained below. This function can be computed in $O(|\mathcal{S}|^2)$.

2. It supports pruning of monotonic and anti-monotonic functions, that is to say that additional monotonic constraints can be easily handled.

In addition to these advantageous features, the algorithm is easy to understand and implement.

Pruning

Recall that an interesting itemset must satisfy the interestingness constraint,

$$I(X) = \frac{|N(X)| \times |N(X)| \times |X|}{\sum_{t \in N(X)} W(X, t) \times |\mathscr{S}|} \geq min\_int.$$

Starting from a given candidate $\langle X, Y \rangle$, it is clear that, for each itemset Z such that $X \subseteq Z \subseteq X \cup Y$,

$$\begin{aligned} |N(Z)| &\leq |N(X \cup Y)|, \\ |Z| &\leq |X \cup Y|, \\ \sum_{t \in N(X)} W(X, t) &\leq \sum_{t \in N(Z)} W(Z, t). \end{aligned}$$

It therefore follows that

$$I(Z) \leq \frac{|N(X \cup Y)| \times |N(X \cup Y)| \times |X \cup Y|}{\sum_{t \in N(X)} W(X, t) \times |\mathscr{S}|}.$$

Because all such itemsets Z are generated by recursive calls of INIT starting from candidate $< X, Y >$, we can safely prune all itemsets Z that satisfy $X \subseteq Z \subseteq X \cup Y$ if

$$UBI(\langle X, Y \rangle) = \frac{|N(X \cup Y)|^2 \times |X \cup Y|}{\sum_{t \in N(X)} W(X, t) \times |\mathscr{S}|} < min\_int$$

by suppressing further recursive calls.

If minimal cohesion is given as a parameter (*min_coh*), we can achieve further pruning. Using the above inequalities, we find that

$$C(Z) \leq \frac{|N(X \cup Y)| \times |X \cup Y|}{\sum_{t \in N(X)} W(X, t)}.$$

We can therefore prune Z if

$$\frac{|N(X \cup Y)| \times |X \cup Y|}{\sum_{t \in N(X)} W(X, t)} < min\_coh.$$

The crucial step in evaluating $UBI(\langle X, Y \rangle)$ is the computation of the intervals $W(X, t)$ for the relevant time stamps t, i.e., those time stamps at which an item of X occurred. In our implementation, we keep the set of intervals associated with X in a list. When we have generated a candidate $\langle X \cup \{a\}, Y \setminus \{a\} \rangle$ from $\langle X, Y \rangle$, we start its evaluation by updating, if necessary, the set of intervals corresponding to X, i.e., $\{W(X, t) \mid t \in N(X)\}$. In this step, the intervals that do not contain an occurrence of a are removed from the list and new minimal intervals for the corresponding time stamps are computed.

To find the minimal interval around position t containing all elements of $X \cup \{a\}$, we start by looking for the nearest occurrences of elements of X both left and right of position t. We then start reading from the side on which the furthest element is closest to t and continue by removing one element at a time and adding the same element from the other side. This process can stop when the interval on the other side has grown sufficiently to make it impossible to improve on the minimum we

have found so far. When we have found this minimal interval, we can add it to the list.

We then proceed by generating the intervals corresponding to each occurrence of a in the event sequence. These intervals are generated in exactly the same way as the new intervals in the previous step.

As many intervals occur many times, we store the list of distinct intervals together with a list of positions each one applies to.

In the worst case, the number of minimal intervals that need to be found can be equal to the number of items in the sequence, $|\mathscr{S}|$. To find an interval, we might need to read the whole sequence both to the left and to the right of the item. Therefore, the time needed to evaluate the pruning function is $O(|\mathscr{S}|^2)$. It is worth noting that this worst case can actually materialise only if we are evaluating the itemset consisting of all items that appear in the sequence, and even then only if items appearing at each end of the sequence do not appear anywhere else.

4.5 Improved Interesting Itemsets Algorithm

The UBI pruning function presented in Section 4.4 prunes a large number of candidates, but the algorithm still suffers from long runtimes, due to the fact that each time a new itemset X is considered for pruning, $W(X, t)$ needs to be computed for almost each $t \in N(X)$. For large itemsets, this can imply multiple dataset scans just to decide if a single candidate node can be pruned.

In this section, we propose another pruning function in an attempt to balance pruning a large number of candidates with the effort needed for pruning. As the main problem with the original function was computing the exact minimal windows for each candidate, we aim to estimate the length of these windows using a much simpler computation. To do this, we first compute the exact sum of the window lengths for each pair of items, and we then use these sums to come up with a lower bound of the sum of the window lengths for all other candidate nodes.

We first note that

$$\sum_{t \in N(X)} W(X, t) = \sum_{x \in X} \sum_{t \in N(\{x\})} W(X, t).$$

We then note that each window around an item $x \in X$ must be at least as large as the largest such window containing the same item x and any other item $y \in X$. It follows that

$$\sum_{t \in N(\{x\})} W(X, t) \geq \sum_{t \in N(\{x\})} \max_{y \in X \setminus \{x\}} (W(\{x, y\}, t)).$$

Naturally, it also holds that

$$\sum_{t \in N(\{x\})} W(X, t) \geq \max_{y \in X \setminus \{x\}} (\sum_{t \in N(\{x\})} W(\{x, y\}, t)).$$

To simplify our notation, from now on we will denote $\sum_{t \in N(\{x\})} W(\{x, y\}, t)$ by $s(x, y)$. Finally, we see that

$$\sum_{t \in N(X)} W(X, t) \geq \sum_{x \in X} \max_{y \in X \setminus \{x\}} (s(x, y)),$$

giving us a lower bound for the sum of windows containing X around all occurrences of items of X. This gives us a new pruning function:

$$NUBI(\langle X,Y\rangle) = \frac{|N(X\cup Y)|^2\times|X\cup Y|}{\sum_{x\in X}\max_{y\in X\setminus\{x\}}(s(x,y))\times|\mathscr{S}|}.$$

This new pruning function is easily evaluated, as all it requires is that we store $s(x,y)$, the sum of minimal windows containing x and y over all occurrences of x, for each pair of items (x,y), so we can look them up when necessary.

The exact windows will still have to be computed for the leaves of the search tree that have not been pruned, but this is unavoidable. Even for the leaves, it pays off to first check the upper bound, and then, only if the upper bound exceeds the threshold, compute the exact interestingness. The new algorithm, INIT2 uses $NUBI$ instead of UBI, and is given in Algorithm 18.

Algorithm 18: INIT2($\langle X,Y\rangle$) finds interesting itemsets

```
1 if NUBI(⟨X,Y⟩) ≥ min_int and size(X) ≤ max_size then
2     if Y = ∅ then
3         If I(X) ≥ min_int output X ;
4     else
5         Choose a in Y ;
6         INIT2(⟨X ∪ {a}, Y \ {a}⟩) ;
7         INIT2(⟨X, Y \ {a}⟩) ;
```

Example 34. *Let us now return to one of our running examples, based on the sequence given in Figure 4.2, and examine what happens if we encounter node* $\langle\{a,b,c\},\{d,e\}\rangle$ *in the search tree. We denote* $X=\{a,b,c\}$ *and* $Y=\{d,e\}$*. With the original pruning technique, we would need to compute the exact minimal windows containing* X *for each occurrence of* a*,* b *or* c*. After a fair amount of work scanning the dataset many times, in all necessary directions, we would come up with the following:* $\sum_{t\in N(X)} W(X,t) = 7+5+4+3+3+3+7+11 = 43$*. The value of the* UBI *pruning function is therefore:*

$$\frac{|N(X\cup Y)|^2\times|X\cup Y|}{\sum_{t\in N(X)} W(X,t)\times|\mathscr{S}|} = \frac{144\times5}{43\times20} = 0.84.$$

Using the new technique, we would first compute $s(x,y)$ *for all pairs* (x,y)*. The relevant pairs are* $s(a,b)=4$*,* $s(a,c)=8$*,* $s(b,a)=4$*,* $s(b,c)=6$*,* $s(c,a)=27$ *and* $s(c,b)=25$*. We can now compute the minimal possible sum of windows for each item, giving us*

$$\max_{y\in X\setminus\{a\}}(s(a,y))=8,\ \max_{y\in X\setminus\{b\}}(s(b,y))=6,\ \max_{y\in X\setminus\{c\}}(s(c,y))=27,$$

resulting in a sum of $\sum_{x\in X}\max_{y\in X\setminus\{x\}}(s(x,y))=41$*. The value of the NUBI pruning function is therefore*

$$\frac{|N(X\cup Y)|^2\times|X\cup Y|}{\sum_{x\in X}\max_{y\in X\setminus\{x\}}(s(x,y))\times|\mathscr{S}|} = 0.88.$$

We see that by simply looking up a few precomputed values instead of scanning the dataset a number of times, we get a very good estimate of the sum of the window lengths.

4.6 Association Rules Algorithm

Unlike the traditional approaches, which need all the frequent itemsets to be generated before the generation of association rules can begin, we are able to generate rules in parallel with the interesting itemsets. When finding an interesting itemset X, we compute the sum of all minimal windows $W(X, t)$ for each $x \in X$ apart, before adding them up into the overall sum needed to compute $I(X)$. With these sums still in memory, we can easily compute the confidence of all association rules of the form $\{x\} \Rightarrow X \setminus \{x\}$, with $x \in X$, that can be generated from itemset X. In practice, it is sufficient to limit our computations to rules of precisely this form (i.e., rules where the body consists of a single item). To compute the confidence of all other rules, we first note that

$$\sum_{t \in N(X)} W(X \cup Y, t) = \sum_{x \in X} \sum_{t \in N(\{x\})} W(X \cup Y, t).$$

A trivial mathematical property tells us that

$$\sum_{t \in N(\{x\})} W(X \cup Y, t) = \overline{W}(\{x\}, Y \cup X \setminus \{x\})|N(x)|.$$

As a result, we can conclude that

$$\overline{W}(X, Y) = \frac{\sum_{x \in X} \overline{W}(\{x\}, Y \cup X \setminus \{x\})|N(x)|}{|N(X)|},$$

which in turn implies that

$$c(X \Rightarrow Y) = \frac{|X \cup Y||N(X)|}{\sum_{x \in X} \overline{W}(\{x\}, Y \cup X \setminus \{x\})|N(x)|}.$$

Meanwhile, we can derive that

$$c(\{x\} \Rightarrow Y \cup X \setminus \{x\}) = \frac{|X \cup Y|}{\overline{W}(\{x\}, Y \cup X \setminus \{x\})},$$

and it follows that

$$c(X \Rightarrow Y) = \frac{|N(X)|}{\sum_{x \in X} \frac{|N(x)|}{c(\{x\} \Rightarrow Y \cup X \setminus \{x\})}}. \tag{4.1}$$

As a result, once we have evaluated all the rules of the form $\{x\} \Rightarrow X \setminus \{x\}$, with $x \in X$, we can then evaluate all other rules $Y \Rightarrow X \setminus Y$, with $Y \subset X$, without further dataset scans. The AR algorithm for finding both interesting itemsets and confident association rules is given in Algorithm 19.

Example 35. *Looking back at the sequence given in Figure 4.2, let us compute the confidence of rule* $\{a, b\} \Rightarrow \{c\}$. *First, we compute* $c(\{a\} \Rightarrow \{b, c\})$. *There are two occurrences of event a in the sequence, one at time stamp 6, and the other at time stamp 8. The nearest occurrence of event c to both these time stamps is at time stamp 10. Therefore, the two corresponding minimal windows containing* $\{a, b, c\}$ *have to stretch to time stamp 10, and have a length of 5 and 3 respectively. Their average length is equal to 4. As a result,* $c(\{a\} \Rightarrow \{b, c\}) = \frac{3}{4} = 0.75$. *We then compute* $c(\{b\} \Rightarrow \{a, c\})$. *Event b can be found at time stamps 7 and 9, and the two corresponding minimal windows containing* $\{a, b, c\}$ *have a length of 4 and 3, respectively. The average length of such a window is therefore 3.5, and* $c(\{b\} \Rightarrow \{a, c\}) = \frac{3}{3.5} = 0.86$. *From Eq. 4.1, it follows that* $c(\{a, b\} \Rightarrow \{c\}) = \frac{4}{\frac{2}{0.75} + \frac{2}{0.86}} = \frac{4}{5} = 0.8$. *It is easy to check that this corresponds to the value as defined in Section 4.3.*

Algorithm 19: AR($\langle X, Y \rangle$) finds interesting itemsets and confident association rules within them

```
1  if NUBI(⟨X, Y⟩) ≥ min_int and size(X) ≤ max_size then
2      if Y = ∅ then
3          if I(X) ≥ min_int then
4              output X ;
5              foreach x ∈ X do
6                  compute and store c({x} ⇒ X \ {x}) ;
7                  if c({x} ⇒ X \ {x}) ≥ min_conf then
8                      output {x} ⇒ X \ {x} ;
9              foreach Y ⊂ X with |Y| ≥ 2 do
10                 if c(Y ⇒ X \ Y) ≥ min_conf then
11                     output Y ⇒ X \ Y ;
12     else
13         Choose a in Y ;
14         AR(⟨X ∪ {a}, Y \ {a}⟩) ;
15         AR(⟨X, Y \ {a}⟩) ;
```

4.7 Experiments

In this section, we present the results of our experiments performed on various synthetic and real-life datasets, and analyse their implications.

Synthetic Datasets

In our first batch of experiments, we tested the usefulness and the efficiency of the itemset mining algorithm presented in Section 4.4. Fully random datasets would not be suitable for our purposes because all itemsets of equal length would be likely to have similar interestingness values. A more suitable dataset would be one generated using a Markov chain model, which enables us to increase the likelihood of certain items appearing close together and thus forming interesting itemsets.

In our experiments, we use a memoryless model, also called a simple random walk, in which the next item depends only on its immediate predecessor. We can therefore easily describe such a model using a transition matrix. The transition matrix used to generate the dataset is given in Table 4.2. Our goal is to generate a sequence in which itemset $\{a, b, c\}$ would be an interesting pattern hidden among occurrences of several other items denoted x in the table. When a transition leads to an occurrence of x, a random item is picked from a group of 25 items, not including a, b or c. Doing this ensures that none of these 25 items will form interesting itemsets.

Using this model, we generated a sequence of 2000 events. We then ran our algorithm varying the interestingness threshold *min_int* from 0.9 down to 0.2, 0.05 at a time. In all experiments no thresholds were set for coverage and cohesion separately. The results of our experiments can be seen in Figures 4.3(a) and 4.3(b).

	a	b	c	x
a	0.2	0.35	0.35	0.1
b	0.35	0.2	0.35	0.1
c	0.35	0.35	0.2	0.1
x	0.05	0.05	0.05	0.85

Table 4.2: Transition matrix defining a Markov model.

Figure 4.3(a) shows that as long as there are few interesting itemsets to be found, our pruning method works very efficiently. The most interesting itemset $\{a,b,c\}$ is found using a relatively high interestingness threshold of 0.45. No further interesting itemsets can be found without lowering the threshold below 0.3, which confirms the intuitiveness of our method. With the interestingness level of 0.25, itemsets $\{a,b\}$, $\{a,c\}$ and $\{b,c\}$ are also discovered as interesting. Because of the uniform distribution of the remaining 25 items, the number of considered candidates grows significantly if the threshold is lowered further.

We can observe in Figure 4.3(b) that the execution time grows proportionally with the number of generated candidates indicating that the most interesting itemsets can be discovered in a reasonable amount of time. To determine the most appropriate interestingness threshold value for a given dataset, an inexperienced user can start with a high value and decrease it until the first results come through. Running the algorithm with a high interestingness threshold takes only a few seconds.

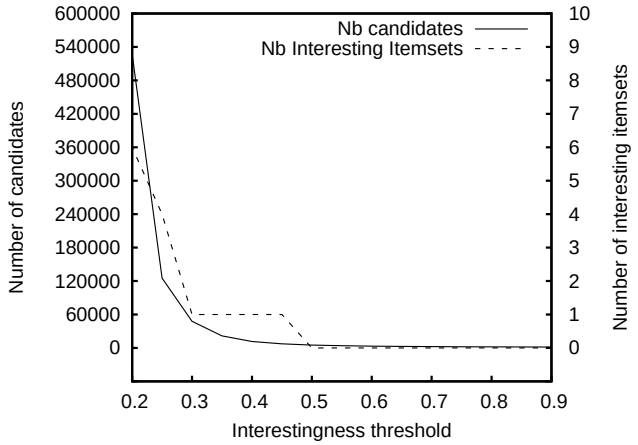
Analysing the generated dataset, we observed that the coverage of $\{a,b,c\}$ was around 0.6. Its interestingness was around 0.46, which means its cohesion was around 0.77. In order to show that the high interestingness of $\{a,b,c\}$ was not only due to its coverage, we investigate what is obtained in a randomly ordered sequence in which items have the same frequency as in the Markov Chain dataset. We generated such a sequence of 2000 events made of 28 items.

As we can see in Figure 4.4(a), we have to lower the interestingness threshold much further in order to get the first results. Itemset $\{a,b,c\}$ is found to have interestingness of around 0.3 indicating a cohesion of 0.5.

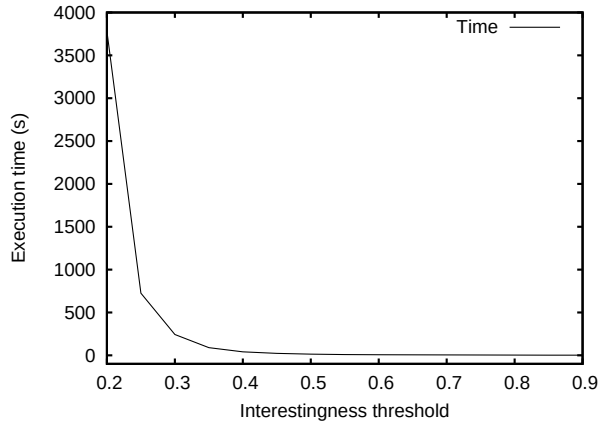
Figure 4.4(b) shows that, once again, the execution time is fully proportional with the number of generated candidates.

To test how much the execution time depends on the size of the sequence, we generated several Markov chain datasets of different sizes, varying between 1000 and 25000. For each size, we generated 10 datasets and applied our algorithm with $min\_int = 0.3$, because it is around this threshold that the first results usually appeared.

Figure 4.5 shows the obtained results. Each vertical line corresponds to one sequence length showing the mean execution time of the 10 algorithm runs and their standard deviation. It can be observed that the execution time depends much more on the structure of the dataset than on its size, even for such similarly structured datasets (all the datasets contained exactly the same number of different items that were distributed according to the same transition matrix).



(a) number of candidates and found itemsets



(b) execution time

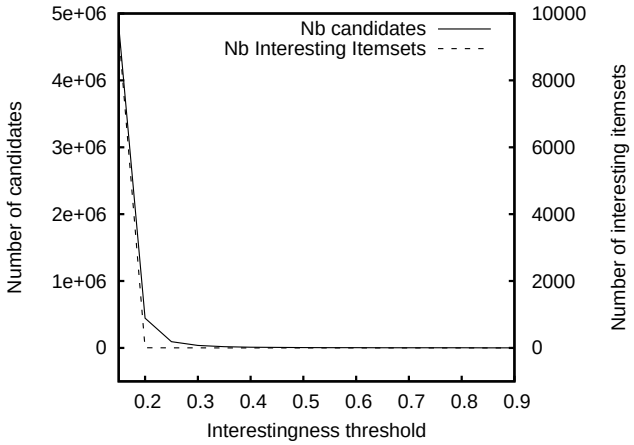
Figure 4.3: (a) Number of generated candidates and interesting itemsets for the Markov chain generated data, with varying *min_int*. (b) Execution time in seconds for the Markov chain generated data, with varying *min_int*.

Real-Life Datasets

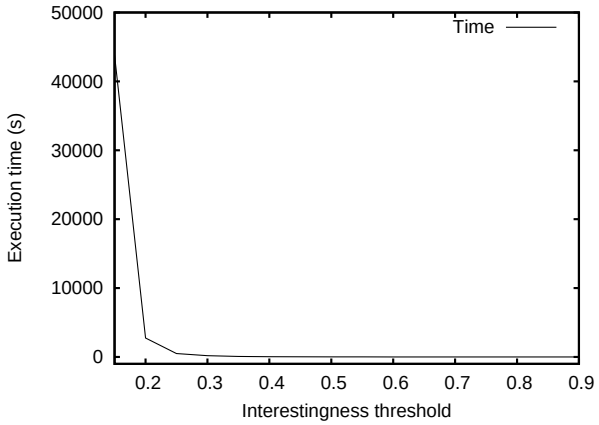
The first real-life dataset we used is one obtained from the amino-acid sequence from human alcohol dehydrogenase [33]. This enzyme has been heavily researched and its genomic sequence is widely available and therefore reliable[2]. The sequence consists of 20 different amino-acids making a chain of 375 items.

In Figure 4.6(a), we can see that we get some results with an interestingness value below 0.4, but no itemset really sticks out, suggesting a uniform distribution of items in the sequence. It turns out that the most interesting itemsets are quite large and

[2]http://www.expasy.org/cgi-bin/sprot-ft-details.pl?P07327@SEQUENCE@2@375



(a) number of candidates and found itemsets



(b) execution time

Figure 4.4: (a) Number of generated candidates and interesting itemsets for the random sequence, with varying *min_int*. (b) Execution time in seconds for the random sequence, with varying *min_int*.

can be found in a reasonable amount of time. These results were not surprising, as biologists confirmed that items were distributed in such a way. To double-check, we once again created a synthetic dataset, 375 items long, where items were distributed randomly, using their frequency in the DNA dataset to determine their probability of occurrence. Figure 4.6(b) shows that the results obtained on this dataset are very similar to those in Figure 4.6(a).

Our second dataset consisted of music notes, collected from one track of a midi-file. We used the notes of the song Abide With Me[3], having first converted the midi-file into a text file, and then pruned all other elements (pitch, channel, velocity,

[3]http://www.tc.umn.edu/~sorem002/hymn_midi.html

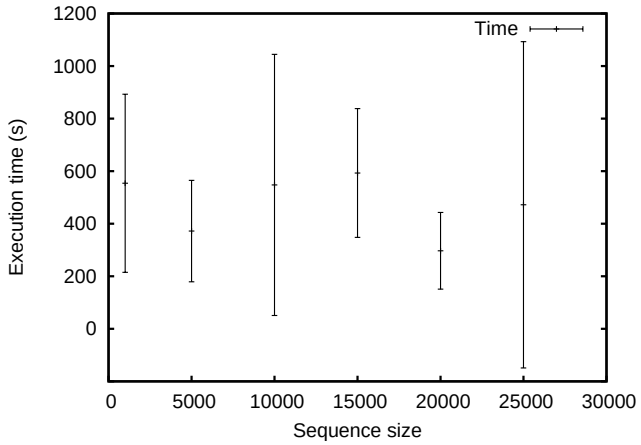


Figure 4.5: Execution time in seconds for the Markov chain generated data, with *min_int* = 0.3 and varying sequence length.

etc.[4]) from the file. This left us with a sequence of 400 notes, made of 18 distinct notes.

Figure 4.7 shows that no interesting itemsets were found using high interesting thresholds, but what was encouraging was the fact that the execution time was minimal. We found the first three interesting itemsets at the level of 0.2: one consisting of four notes, one of five, and the third was a singleton, which was also included in the other two sets. This indicated that this note appeared most often in the sequence, and that the other four notes always appeared relatively close to it (certainly closer than the other 13 notes).

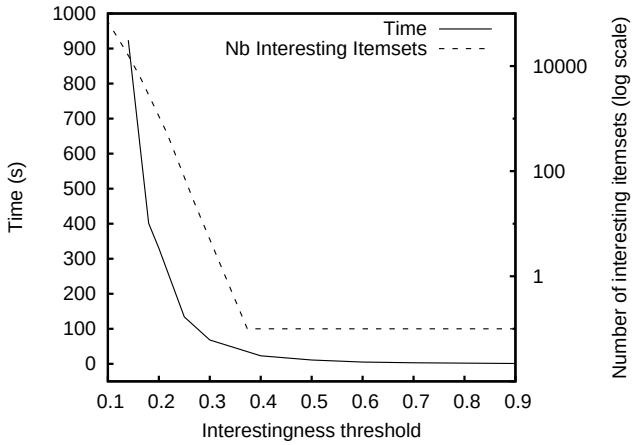
A larger number of itemsets was found to be interesting as we lowered the threshold further, growing consistently even for small decreases in the threshold.

Comparison of the Two Algorithms

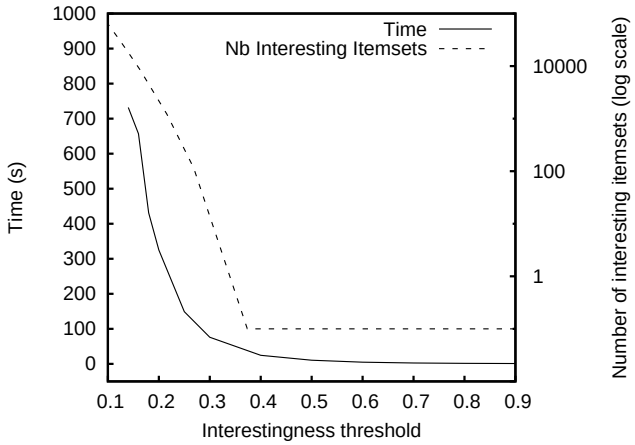
In this batch of experiments, we aimed to demonstrate that the algorithm for finding interesting itemsets presented in Section 4.5 works faster than the one given in Section 4.4 and can handle much longer sequences. However, we immediately designed a dataset that could later be used to test the usefulness of our association rules algorithm, too. To generate a sequence in which some association rules would certainly stand out, we used the Markov chain model given in Table 4.3. Our dataset consisted of items a, b, c, and 22 other items, randomly distributed whenever the transition table led us to item x. We fine tuned the probabilities in such a way that all items apart from c had approximately the same frequency, while c appeared approximately twice as often.

We used the above Markov model to generate ten sequences of 10 000 items and ran the two algorithms on each sequence, varying *min_int*, at first choosing *max_size*

[4] http://www.fourmilab.ch/webtools/midicsv/



(a) DNA dataset



(b) random DNA-like dataset

Figure 4.6: (a) Execution time in seconds and number of extracted itemsets (log-scale) for the DNA dataset, with varying *min_int*. (b) The same for the random DNA-like dataset.

of 4. Figures 4.8(a) and 4.9 show the average runtimes and the number of evaluated candidate nodes for each algorithm, as well as the actual number of identified interesting itemsets. While INIT2 pruned less (Figure 4.9), it ran much faster than INIT, most importantly at the thresholds where the most interesting itemsets could be found (see Figure 4.8(b) for a zoomed-in version). Naturally, if *min_int* = 0, the algorithms take equally long, as all itemsets are identified as interesting, and their exact interestingness must be computed. In short, we see that the runtime of the INIT algorithm is proportional to the number of candidates, while the runtime of the INIT2 algorithm is proportional to the number of interesting itemsets.

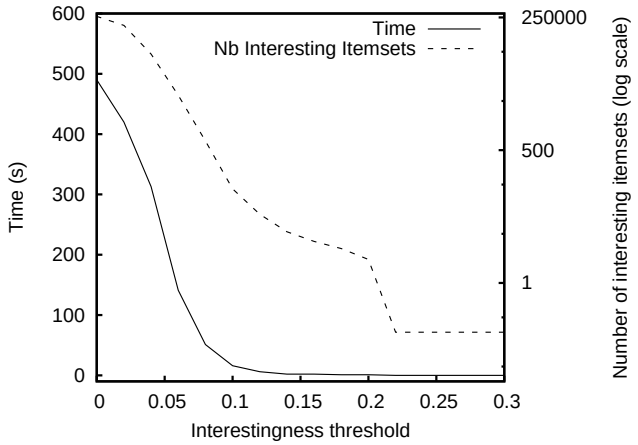


Figure 4.7: Execution time in seconds and number of interesting itemsets (log-scale) for the music dataset, with varying *min_int*.

	a	b	c	x
a	0	0.45	0.45	0.1
b	0.45	0	0.45	0.1
c	0	0	0.1	0.9
x	0.025	0.025	0.05	0.9

Table 4.3: A transition matrix defining a Markov model

Association Rules

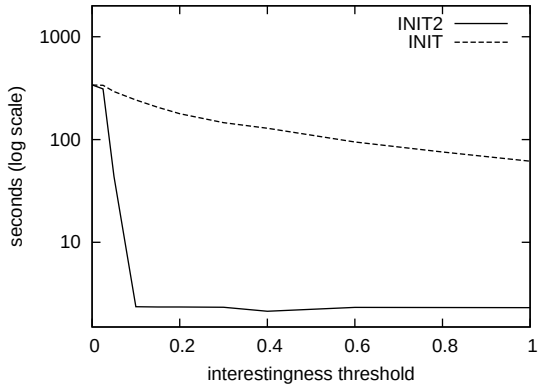
Finally, we tested our association rules algorithm to demonstrate the following two claims:

1. our algorithm for finding association rules gives meaningful results, without generating spurious output.

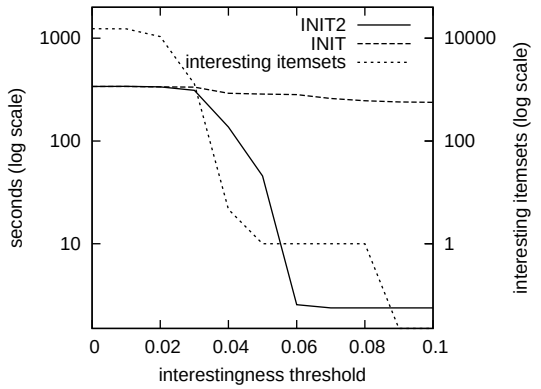
2. our algorithm for finding association rules runs very efficiently, even with a confidence threshold set to 0.

Once again, we used the dataset generated using the Markov model given in Table 4.3. The high probability of transitions from a to b and c, and from b to a and c should result in rules such as $\{a\} \Rightarrow \{c\}$ and $\{b\} \Rightarrow \{c\}$ having a high confidence. However, given that c appears more often, sometimes without an a or a b nearby, we would expect rules such as $\{c\} \Rightarrow \{a\}$ and $\{c\} \Rightarrow \{b\}$ to be ranked lower.

To support the first claim, we ran our association rules algorithm with both *min_int* and *min_conf* equal to 0. We set *max_size* to 4. We now simply had to check which rules had the highest confidence. In repeated experiments, with various 500 000 items long datasets, the results were very consistent. The most interesting



(a)



(b)

Figure 4.8: (a) Runtime comparison of the two algorithms for discovering interesting itemsets with *max_size* set to 4. (b) Zoomed-in version of Figure 4.8(a).

rules were $\{a\} \Rightarrow \{c\}$ and $\{b\} \Rightarrow \{c\}$, with a confidence of 0.52. Then followed rules $\{a\} \Rightarrow \{b,c\}$, $\{b\} \Rightarrow \{a,c\}$ and $\{a,b\} \Rightarrow \{c\}$, with a confidence of 0.34. Rules $\{a\} \Rightarrow \{b\}$ and $\{b\} \Rightarrow \{a\}$ had a confidence of 0.29, while rules $\{a,c\} \Rightarrow \{b\}$ and $\{b,c\} \Rightarrow \{a\}$ had a confidence of 0.2. Rule $\{c\} \Rightarrow \{a,b\}$ had a confidence of around 0.17, while rules not involving a, b or c had a confidence between 0.13 and 0.17. We can safely conclude that our algorithm gave the expected results.

To confirm this claim, we ran our algorithm on three text datasets: *English*[5], *Italian*[6] and *Dutch*[7]. In each text, we considered the letters of the alphabet and the space between words (denoted $\sqcup$) as items, ignoring all other symbols. In all three languages, rule $\{q\} \Rightarrow \{u\}$ had a confidence of 1, as q is almost always followed by

[5] a selection from http://www.gutenberg.org/files/1999/1999.txt
[6] a selection from http://www.gutenberg.org/dirs/etext04/7clel10.txt
[7] a selection from http://www.gutenberg.org/files/18066/18066-8.txt

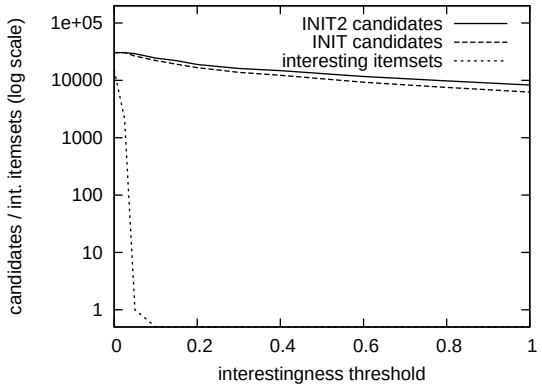


Figure 4.9: Pruning comparison of the two algorithms for discovering interesting itemsets with *max_size* set to 4.

u in all these languages. In all three languages, there followed a number of rules with $\sqcup$ in the head, but the body varied. In Italian, a space is often found near f, as f is mostly found at the beginning of Italian words, while the same is true for j in English. Rules involving two letters revealed some patterns inherent in these languages. For example, rule $\{h\} \Rightarrow \{c\}$ was ranked high in Italian (where h appears very rarely without a c in front of it), while rule $\{j\} \Rightarrow \{i\}$ scored very high in Dutch (where j is often preceded by an i). A summary of the results is given in Table 4.4.

	English	*Italian*	*Dutch*
two letters	$c(\{q\} \Rightarrow \{u\}) = 1$	$c(\{q\} \Rightarrow \{u\}) = 1$	$c(\{q\} \Rightarrow \{u\}) = 1$
	$c(\{z\} \Rightarrow \{i\}) = 0.61$	$c(\{h\} \Rightarrow \{c\}) = 0.75$	$c(\{j\} \Rightarrow \{i\}) = 0.75$
	$c(\{v\} \Rightarrow \{e\}) = 0.58$	$c(\{h\} \Rightarrow \{e\}) = 0.51$	$c(\{d\} \Rightarrow \{e\}) = 0.61$
	$c(\{x\} \Rightarrow \{e\}) = 0.53$	$c(\{z\} \Rightarrow \{a\}) = 0.5$	$c(\{g\} \Rightarrow \{e\}) = 0.57$
three letters	$c(\{q\} \Rightarrow \{e,u\}) = 0.63$	$c(\{q\} \Rightarrow \{e,u\}) = 0.6$	$c(\{q\} \Rightarrow \{i,u\}) = 1$[8]
	$c(\{z\} \Rightarrow \{a,i\}) = 0.52$	$c(\{h\} \Rightarrow \{c,e\}) = 0.57$	$c(\{j\} \Rightarrow \{e,i\}) = 0.46$
	$c(\{q\} \Rightarrow \{n,u\}) = 0.4$	$c(\{q\} \Rightarrow \{a,u\}) = 0.52$	$c(\{g\} \Rightarrow \{e,n\}) = 0.44$
with $\sqcup$	$c(\{y\} \Rightarrow \{\sqcup\}) = 0.85$	$c(\{q\} \Rightarrow \{\sqcup\}) = 0.84$	$c(\{z\} \Rightarrow \{\sqcup\}) = 0.78$
	$c(\{w\} \Rightarrow \{\sqcup\}) = 0.84$	$c(\{f\} \Rightarrow \{\sqcup\}) = 0.7$	$c(\{w\} \Rightarrow \{\sqcup\}) = 0.74$
	$c(\{j\} \Rightarrow \{\sqcup\}) = 0.83$	$c(\{a\} \Rightarrow \{\sqcup\}) = 0.7$	$c(\{v\} \Rightarrow \{\sqcup\}) = 0.73$
	...	...	...
	$c(\{\sqcup\} \Rightarrow \{e\}) = 0.35$	$c(\{\sqcup\} \Rightarrow \{a\}) = 0.39$	$c(\{\sqcup\} \Rightarrow \{n\}) = 0.35$

Table 4.4: A selection of association rules found in three different languages.

[8] This is due to a short dataset, rather than an actual rule in the Dutch language.

To prove our second claim, we ran the AR algorithm on ten Markov chain datasets (each 10 000 items long), using *min_int* of 0.025 and *max_size* of 4, each time varying *min_conf*. The average runtimes and number of found association rules are shown in Figure 4.10. We can see that the exponential growth in the number of generated rules had virtually no effect on the runtime of the algorithm.

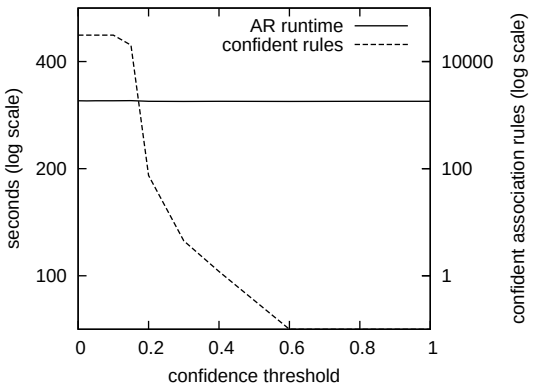


Figure 4.10: Runtimes and the size of the output of the *AR* algorithm with a varying confidence threshold.

4.8 Multiple Sequences

Another setting in which discovering interesting patterns can be applied is that of multiple sequences. Here, the data again consists of items, this time coming in many separate sequences. The frequency of an occurrence of a pattern would now be the number of different sequences in which the pattern occurs, regardless of how many times the pattern occurs in any single sequence. In other words, when looking for the frequency of a pattern alone, we can stop looking at a sequence as soon as we have encountered the first occurrence of the pattern.

To determine the interestingness of a pattern, however, it is not enough to know that the items making the pattern occur in a given sequence. We would also like to know how close they appear to each other. A possible method to do this would be to define a fixed window length as a measure of how close to each other the items need to occur before they can count as an occurrence of the pattern. We could then slide this window over each sequence and find whether the pattern appears in it or not. If we then count in how many sequences the pattern occurs within such a window, we can say that a pattern is interesting if the number of such sequences is high enough. In other words, a pattern is interesting if its items occur close enough to each other in a high enough number of sequences.

Once again, we would like to do more than that. We would like to give a guarantee not only that the items making up a pattern will appear close to each other in a high number of sequences, but also that if they do all appear in a sequence, then they will appear close to each other. To do this, we will once again define interesting patterns

in terms of both frequency and cohesion. Below, we present some preliminary work on generalising the approach from the previous sections to this setting.

Definition of the Constraint

Formally, a single sequence S_j is still defined as in Section 4.3, and we now denote the set of all sequences by $\mathscr{S}$. We can now redefine coverage, cohesion and interestingness given this new problem setting. This time, coverage will measure in how many sequences the itemset appears, while cohesion will measure how close to each other the items making up the itemset appear on average in their minimal occurrences for a given sequence.

Definition 47. *For a given itemset X, we denote the set of all sequences in which all items of X can be found as*

$$N_m(X) = \{j \mid \forall i \in X, \exists (i,t) \in \mathscr{S}_j\}$$

The coverage *of X, denoted $P_m(X)$ can now be defined as:*

$$P_m(X) = \frac{|N_m(X)|}{|\mathscr{S}|}$$

In order to calculate the cohesion, we must first evaluate the length of the shortest interval containing the itemset X for each sequence in $N(X)$:

Definition 48. *Given an itemset X, and a sequence index j, the* length of the minimal window in S_j containing *X, denoted $W_m(X, j)$, is defined as*

$$W_m(X, j) = \min\{t_2 - t_1 + 1 \mid t_1 \leq t_2 \text{ and } \forall i \in X, \exists (i,t) \in \mathscr{S}_j,\ t_1 \leq t \leq t_2\}.$$

The average length of such shortest intervals is computed as

$$\overline{W}_m(X) = \frac{\sum_{j \in N_m(X)} W_m(X, j)}{|N_m(X)|}.$$

Once again, it is clear that $\overline{W}_m(X)$ is greater than or equal to the number of items in X. Furthermore, for a fully cohesive itemset, $\overline{W}_m(X) = |X|$. This allows us to define cohesion in a normalised manner.

Definition 49. *Given an itemset X, the* cohesion *of X, denoted $C_m(X)$, is defined as*

$$C_m(X) = \frac{|X|}{\overline{W}_m(X)}.$$

Definition 50. *Given a set of sequences $\mathscr{S}$ and an itemset X, the* interestingness *of X, denoted $I_m(X)$, is defined as*

$$I_m(X) = C_m(X)P_m(X) = \frac{|N_m(X)| \times |N_m(X)| \times |X|}{\sum_{j \in N_m(X)} W_m(X, j) \times |\mathscr{S}|}.$$

Given a user defined threshold min_int, an itemset X is considered interesting if $I_m(X) \geq$ min_int.

Sequence ID	Sequence
s_1	$abcdefgcdehi$
s_2	$cghefababcidjkdlcmd$

Table 4.5: A dataset consisting of two sequences.

Again, minimal coverage and minimal cohesion can be used as separate thresholds.

Example 36. *To illustrate our interestingness measure, we will now apply it on the sequences given in Table 4.5.*

We start by looking at itemsets $\{a,b\}$, $\{c,d\}$ *and* $\{e,f\}$. *Items a and b can be found next to each other in both sequences, therefore* $P_m(\{a,b\}) = 1$, $C_m(\{a,b\}) = 1$, *and, as a result,* $I_m(\{a,b\}) = 1$. *Exactly the same is true for* $\{e,f\}$, *as this time the number of occurrences in any given sequence is of no importance. c and d, meanwhile, appear next to each other in the first sequence, but the minimal interval containing both c and d in the second sequence,* $W_m(\{c,d\},2)$, *is of length 3. Therefore,* $P_m(\{c,d\}) = 1$, $\overline{W}_m(\{c,d\}) = 2.5$, $C_m(\{c,d\}) = 0.8$, *and* $I_m(\{c,d\}) = 0.8$.

In other words, $\{a,b\}$ *and* $\{e,f\}$ *are in this context equally interesting, while* $\{c,d\}$ *is slightly less interesting, as there exists a sequence in which both c and d appear, but never next to each other.*

Let us now consider itemset $\{e,g\}$. *This itemset appears in both sequences, so* $P_m(\{e,g\}) = 1$. *Further, we find that* $W_m(\{e,g\},1) = 3$, $W_m(\{e,g\},2) = 3$, *and therefore* $\overline{W}_m(\{e,g\}) = 3$. *As a result,* $C_m(\{e,g\}) = 0.67$, *and* $I_m(\{e,g\}) = 0.67$.

It may be worth noting that a sliding window method would either conclude that $\{a,b\}$ *and* $\{e,g\}$ *are equally interesting (if the window size was greater than 2) or that* $\{e,g\}$ *is not interesting at all (if the window size was 2).*

Finally, let's take a look at two more itemsets, $\{d,h\}$ *and* $\{c,k\}$. *Items d and h appear in both sequences, but never near each other, while items c and k appear together only in one sequence, but then next to each other. Intuitively, itemset* $\{c,k\}$ *seems to be more interesting. The results are given in Table 4.6.*

itemset	$P_m(X)$	$W_m(X,1)$	$W_m(X,2)$	$\overline{W}_m(X)$	$C_m(X)$	$I_m(X)$
$\{d,h\}$	1	8	11	9.5	0.21	0.21
$\{c,k\}$	0.5	-	2	2	1	0.5

Table 4.6: Computation of interestingness for itemsets $\{d,h\}$ and $\{c,k\}$.

Once again, all these results confirm the intuitiveness of our method.

Algorithm Sketch

We use the same divide-and-conquer algorithm presented in Section 4.4 to generate candidates, but now the pruning technique is different. Starting from a given

candidate $\langle X, Y\rangle$, for each Z such that $X \subseteq Z \subseteq X \cup Y$, we see that

$$\begin{aligned} |N_m(Z)| &\leq |N_m(X)|, \\ |Z| &\leq |X \cup Y|, \\ \sum_{j \in N_m(X \cup Y)} W_m(X, j) &\leq \sum_{j \in N_m(Z)} W_m(Z, j). \end{aligned}$$

It therefore follows that

$$I_m(Z) \leq \frac{|N_m(X)| \times |N_m(X)| \times |X \cup Y|}{\sum_{j \in N_m(X \cup Y)} W_m(X, j) \times |\mathscr{S}|}.$$

Because all such itemsets Z are generated by recursive calls of the INIT_m algorithm, given in Algorithm 20, starting from candidate $< X, Y >$, we can safely prune all itemsets Z that satisfy $X \subseteq Z \subseteq X \cup Y$ if

$$UBI_m(\langle X, Y\rangle) = \frac{|N_m(X)|^2 \times |X \cup Y|}{\sum_{j \in N_m(X)} W_m(X, j) \times |\mathscr{S}|}$$

by suppressing further recursive calls.

Algorithm 20: $\text{INIT}_m(\langle X, Y\rangle)$ finds interesting itemsets

```
1 if UBI_m(⟨X, Y⟩) ≥ min_int and size(X) ≤ max_size then
2     if Y = ∅ then
3         output X ;
4     else
5         Choose a in Y ;
6         INIT_m(⟨X ∪ {a}, Y \ {a}⟩) ;
7         INIT_m(⟨X, Y \ {a}⟩) ;
```

Note that the first inequality is different from its counterpart in Section 4.4, as an occurrence of an itemset X in a sequence directly implies an occurrence of all its subsets in that same sequence, and each sequence appears only once in $N_m(X)$ regardless of how many times items making up the itemset appear in it. The coverage of a subset can thus never be smaller than the coverage of the set itself. As a result, if minimal coverage is given as a parameter (*min_cov*), we can achieve further pruning using

$$P_m(Z) \leq P_m(X),$$

meaning we can prune Z if

$$P_m(X) < min\_cov.$$

However, since

$$P(Z) \leq P(X) \leq I(X),$$

we can also prune Z if

$$P_m(X) < min\_int,$$

even if minimal coverage is not given as an extra parameter. This gives us an alternative pruning method, computationally less costly than UBI_m presented above.

If minimal cohesion is given as a parameter (*min_coh*), we can achieve even more pruning. Using the above inequalities, we find that

$$C_m(Z) \leq \frac{|N_m(X)| \times |X \cup Y|}{\sum_{t \in N_m(X \cup Y)} W_m(X, t)}.$$

We can therefore prune Z if

$$\frac{|N_m(X)| \times |X \cup Y|}{\sum_{t \in N_m(X \cup Y)} W_m(X, t)} < min\_coh.$$

4.9 Conclusions

In this chapter we presented a new constraint to identify interesting itemsets in one or many event sequences. For one sequence, such itemsets consist of frequent items that on average appear close to each other. Unlike previous approaches that only look for items that appear close to each other frequently enough, our constraint gives a guarantee that when an item from an interesting itemset is encountered, the remainder of the set will be found nearby.

We also provide a sound and complete algorithm to extract itemsets having an interestingness value above a user given threshold. This algorithm takes advantage of certain properties of the defined interestingness measure that allow us to efficiently prune large numbers of candidates. We propose two pruning methods — one that prunes more, but slower, and another that prunes less, but much more efficiently.

Additionally, we present a new way of identifying association rules in sequences. We base ourselves on interesting itemsets and look for association rules within them. When we encounter a part of an itemset in the sequence, our measure of the confidence of the rule tells us how likely we are to find the rest of the itemset nearby. Due to being able to generate association rules while mining interesting itemsets, the cost of finding confident association rules is negligible.

We ran experiments on both synthetic and real-life datasets and the results proved the intuitiveness of our measure. The synthetically generated Markov chain datasets confirmed that our algorithm detects the expected interesting itemsets and, unless the threshold is set too low, does not output any spurious itemsets. A random dataset confirmed that there is no spurious output. The experiments made on real-life datasets provided further proof of the efficiency and usefulness of our pruning technique.

We extended our definitions to a situation where the dataset consists of multiple sequences. Here, an itemset is deemed interesting if its items appear in many sequences close to each other. We presented another algorithm that efficiently looks for such sets in multiple sequences, once again proposing two different pruning methods.

CHAPTER 5

Conclusions

In this thesis we explored several topics related to identifying interesting patterns in long event sequences. Episodes, the most general patterns that are typically represented by directed acyclic graphs (DAGs), are the richest form of pattern, but their detection is also computationally the most costly. For this reason, simpler patterns, such as itemsets (or parallel episodes) and sequences (or serial episodes), have proved more popular. However, other attempts to impose restrictions on the considered episodes have also been proposed.

5.1 Main Contributions

In Chapter 2, we try to reduce the search space by introducing the concept of strict episodes, where two nodes in a DAG carrying the same label must be connected by an edge. This reduces the complexity of the problem without a great loss in the quality of the output. Further reduction of the output was obtained by mining only closed episodes.

The exponential output size is not only a problem in episode mining, it is even more of an issue when it comes to association rules. Nevertheless, the MARBLES algorithm presented in Chapter 2 proved capable of efficiently mining all non-redundant association rules present in a long input sequence. We eliminate redundancy in two ways.

- We define a class of closed association rules, consisting of a free episode on the left-hand side, and a closed episode on the right-hand side.

- We allow the user to impose a confidence boost threshold on the rules, thus eliminating the less informative rules.

The traditional definition of frequency in similar problems leaves a lot to be desired. We illustrate this issue with a number of examples throughout the thesis. Therefore, along with providing algorithms that allow the search for general episodes adopting the traditional approach towards defining frequency in both Chapter 2 and Chapter 3, we also attempt to introduce more intuitive interestingness measures for both episodes and itemsets. In Chapter 2, our MARBLES algorithm works with three different definitions of the confidence of an association rule, based on three different definitions of the frequency of an episode. Along with the traditional definition based on sliding windows of fixed length, we introduce confidence based on minimal windows, as well as weighted minimal windows. We discuss the advantages and disadvantages of all three methods, and conclude that they all have their respective merits.

For a method to be useful, it is not enough for it to be intuitive. It is just as important that the method is efficient. This is the reason why most of the existing quality measures attempt to be anti-monotonic, i.e., they aim to ensure that smaller patterns always score higher than the larger patterns. It is not difficult to see that this is not always desirable from the point of view of the user. In Chapter 4, we propose an interestingness measure for itemsets in a sequence that is not anti-monotonic, and we argue that the resulting complexity need not be entirely prohibitive. The method demonstrably gives more intuitive results, and the runtimes, while longer, can still be kept within reasonable limits, by using efficient pruning techniques.

Finally, while an episode certainly allows us to represent a rich class of patterns, it is not without limits itself. In Chapter 3, we extend the usual definition of an episode, and allow the DAG to carry multiple labels in each node. By doing so, we can depict a situation in which multiple events reoccur in the input sequence simultaneously, which was not possible before. This, naturally, brings a whole new level of complexity to the problem, but we provide an algorithm that can mine this new class of patterns efficiently. An interesting side-effect of the new definition of an episode is the fact that an episode depicted by a DAG consisting of two nodes can now be a subset of an episode depicted by a DAG consisting of just one node. We tackle this, and similar anomalies, by introducing an entirely new subset relationship between episodes, based on the sets of sequences that cover the episodes, rather than on graphs depicting them.

5.2 Outlook

As is true for all scientific research, the work presented in this thesis poses as many new questions as it answers.

In Chapter 2, as well as in Chapter 4, we mine association rules in restricted versions of the general problem setup, thus not using the full richness of the episode representation. In Chapter 3, we even extended the already huge number of patterns that can be depicted by an episode. It remains an open question whether we can efficiently mine association rules in this setting, too.

Furthermore, in Chapter 2 we proposed new confidence measures for association rules, both of them not anti-monotonic. In Chapter 4 we went even fur-

ther, proposing a new interestingness measure for itemsets that, too, is not anti-monotonic. It would certainly be worth investigating if these measures could be applied to the more general problem setup introduced in Chapter 3. For this to happen, these measures would first need to be defined in the new setting, and new efficient pruning techniques would need to be discovered and implemented.

Finally, in Chapter 4 we touched upon the issue of adopting our methods in situations where the input consists of multiple sequences, rather than of just one long sequence. It would be interesting to see if episodes, based on the three different definitions of frequency, could be discovered in this setting. Could we take it further and identify confident association rules, again using all three definitions of frequency as presented in Chapter 2? Could we, perhaps, apply the interestingness, defined in Chapter 4 using coverage and cohesion, to episodes capable of depicting simultaneous events introduced in Chapter 3, and produce association rules between episodes as proposed in Chapter 2? And could we then, to top it all off, adapt this approach to a setting where the input consists of many sequences, too?

These are just some of the remaining open challenges that will surely keep us busy in years to come.

Bibliography

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, pages 487–499, 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. *11th International Conference on Data Engineering (ICDE 1995)*, 0:3–14, 1995.

[4] J. L. Balcázar. Formal and computational properties of the confidence boost of association rules. *CoRR*, abs/1103.4778, 2011.

[5] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Computational Logic*, pages 972–986, 2000.

[6] J. Besson, C. Robardet, J.-F. Boulicaut, and S. Rome. Constraint-based concept mining and its application to microarray data analysis. *Intelligent Data Analysis*, 9(1):59–82, 2005.

[7] T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007)*, pages 83–92, 2007.

[8] G. Casas-Garriga. Discovering unbounded episodes in sequential data. In *Knowledge Discovery in Databases: PKDD 2003, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 83–94, 2003.

[9] G. Casas-Garriga. Summarizing sequential data with closed partial orders. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2005)*, pages 380–391, 2005.

[10] L. Cerf, J. Besson, C. Robardet, and J.-F. Boulicaut. Data peeler: Contraint-based closed pattern mining in n-ary relations. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2008)*, pages 37–48, 2008.

[11] G. Chen, X. Wu, and X. Zhu. Sequential pattern mining in multiple streams. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 585–588, 2005.

[12] B. Cule and B. Goethals. Mining association rules in long sequences. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2010)*, pages 300–309, 2010.

[13] B. Cule, B. Goethals, and C. Robardet. A new constraint for mining sets in sequences. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2009)*, pages 317–328, 2009.

[14] B. Cule, N. Tatti, and B. Goethals. Marbles: Mining association rules buried in long event sequences. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2012)*, pages 248–259, 2012.

[15] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD 1998)*, pages 16–22, 1998.

[16] M. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):530–552, 2002.

[17] A. Gély. A generic algorithm for generating closed sets of a binary relation. In *Proceedings of the Third International Conference on Formal Concept Analysis (ICFCA 2005)*, pages 223–234, 2005.

[18] R. Gwadera, M. J. Atallah, and W. Szpankowski. Markov models for identification of significant episodes. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2005)*, pages 404–414, 2005.

[19] R. Gwadera, M. J. Atallah, and W. Szpankowski. Reliable detection of episodes in event sequences. *Knowledge and Information Systems*, 7(4):415–437, 2005.

[20] R. Gwadera and F. Crestani. Discovering significant patterns in multi-stream sequences. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008)*, pages 827–832, 2008.

[21] S. K. Harms, J. S. Deogun, J. Saquer, and T. Tadesse. Discovering representative episodal association rules from event sequences using frequent closed episode sets and event constraints. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2001)*, pages 603–606, 2001.

[22] S. Laxman, P. S. Sastry, and K. P. Unnikrishnan. 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 (KDD 2007)*, pages 410–419, 2007.

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

[24] N. Méger and C. Rigotti. Constraint-based mining of episode rules and optimal window sizes. In *Knowledge Discovery in Databases: PKDD 2004, 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 313–324, 2004.

[25] T. Oates and P. R. Cohen. Searching for structure in multiple streams data. In *Proceedings of the 13th International Conference on Machine Learning (ICML 1996)*, pages 346–354, 1996.

[26] J. Pei, H. Wang, J. Liu, K. Wang, J. Wang, and P. S. Yu. Discovering frequent closed partial orders from strings. *IEEE Transactions on Knowledge and Data Engineering*, 18(11):1467–1481, 2006.

[27] N. Tatti. Significance of episodes based on minimal windows. In *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM 2009)*, pages 513–522, 2009.

[28] N. Tatti and B. Cule. Mining closed episodes with simultaneous events. In *Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2011)*, pages 1172–1180, 2011.

[29] N. Tatti and B. Cule. Mining closed strict episodes. *Data Mining and Knowledge Discovery*, 25(1):34–66, 2012.

[30] P. Tzvetkov, X. Yan, and J. Han. Tsp: Mining top-k closed sequential patterns. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, pages 347–354, 2003.

[31] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. *20th International Conference on Data Engineering (ICDE 2004)*, 0:79, 2004.

[32] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: some preliminary results. *ACM SIGMOD Record*, 23(2):115–125, 1994.

[33] G. Wu. Frequency and markov chain analysis of the amino-acid sequence of human alcohol dehydrogenase alpha-chain. *Alcohol and Alcoholism*, 35(3):302–306, 2000.

[34] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large datasets. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2003)*, pages 166–177, 2003.

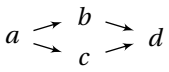
[35] W. Zhou, H. Liu, and H. Cheng. Mining closed episodes from event sequences efficiently. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining(1)*, pages 310–318, 2010.

Samenvatting

Door de toenemende kracht en snelheid van computers is het verzamelen van data gemakkelijker dan ooit. Overheden, bedrijven en ook onderzoekscentra zijn momenteel in het bezit van enorme hoeveelheden data, die ze op een systematisch manier willen analyseren. Het proces dat nuttige informatie haalt uit een grote verzameling data heet data mining.

Een belangrijke deel van data mining is het zoeken naar patronen in data. Een patroon is typisch een verzameling van gebeurtenissen die vaak samen te vinden zijn in de data. In dit proefschrift zoeken we naar patronen in lange sequenties van data. Een patroon bestaat in dit geval uit gebeurtenissen die vaak dicht bij elkaar verschijnen.

De meest expressieve patronen geschikt voor sequentiële data zijn episodes. Een episode wordt voorgesteld door een gerichte acyclische graaf. De knopen van de graaf stellen de gebeurtenissen van de episode voor, terwijl de pijlen tussen de knopen de (partiële) volgorde van deze gebeurenissen aangeven. Figuur 1 geeft een voorbeeld van een episode. Als deze episode frequent in de data voorkomt, wil dat zeggen dat gebeurtenis a vaak vóór gebeurtenissen b and c verschijnt, en dat b en c dan regelmatig door gebeurtenis d werden gevolgd. Het feit dat er geen pijl tussen b en c bestaat betekent dat b soms vóór, en soms na c verschijnt. Als de graaf helemaal geen pijlen bevat, dan spreken we over een parallele episode. In zo een geval, mogen de gebeurtenissen in om het even welke volgorde verschijnen. Als de pijlen in de graaf een totale ordening opleggen, dan hebben we het over een seriële episode, en moeten de gebeurtenissen altijd in één bepaalde volgorde verschijnen.



Figuur 1: Een episode.

Na een korte inleiding in hoofdstuk 1, gaan we in hoofdstuk 2 associatie regels tussen episodes proberen te vinden. Een associatie regel van de vorm *episode1* ⇒ *episode2* wil zeggen dat als we *episode1* in de sequentie vinden, dan is de kans groot dat *episode2* ook in de buurt te vinden is. Bovendien definiëren we drie verschillende manieren om de waarde van een associatie regel te meten. Als laatste, proberen we de output van ons algoritme te minimaliseren door alle redundante regels weg te laten.

In hoofdstuk 3 breiden we de definitie van een episode uit, door toe te laten dat een knoop in de graaf meerdere gebeurtenissen kan bevatten. Dit laat ons toe om patronen te identificeren waarin bepaalde gebeurtenissen vaak tegelijkertijd gebeuren. De complexiteit van het probleem wordt hierdoor uiteraard verhoogd. We beschrijven daarom de nodige stappen om dit probleem toch nog binnen een haalbare tijdspanne op te lossen.

In hoofdstuk 4 beschrijven we een nieuwe, meer intuïtieve, maat om de waarde van een episode te meten. Hierbij eisen we dat de gebeurtenissen binnen een interessante episode frequent zijn en gemiddeld dicht bij elkaar moeten verschijnen. De berekening van deze nieuwe interessantheidsmaat is toch veel ingewikkelder dan de berekening van de bestaande, minder intuïtieve, maten. Hierdoor moesten we, om te beginnen, onze aanpak alleen tot de parallele episodes beperken. Binnen deze context zoeken we ook associatie regels gebaseerd op de nieuwe interessantheidsmaat. Verder geven we ook een preliminaire analyse van hoe deze methode aangepast moet worden indien de data in de vorm van meerdere sequenties komt, in plaats van één lange sequentie.

We eindigen het proefschrift met een samenvatting van onze conclusies in hoofdstuk 5.