Noname manuscript No.
(will be inserted by the editor)

Efficiently Mining Cohesion-based Patterns and Rules in Event Sequences[1]

Boris Cule[2] · Len Feremans[2] · Bart Goethals[2,3]

Received: date / Accepted: date

Abstract Discovering patterns in long event sequences is an important data mining task. Traditionally, research focused on frequency-based quality measures that allow algorithms to use the anti-monotonicity property to prune the search space and efficiently discover the most frequent patterns. In this work, we step away from such measures, and evaluate patterns using cohesion — a measure of how close to each other the items making up the pattern appear in the sequence on average. We tackle the fact that cohesion is not an anti-monotonic measure by developing an upper bound on cohesion in order to prune the search space. By doing so, we are able to efficiently unearth rare, but strongly cohesive, patterns that existing methods often fail to discover. Furthermore, having found the occurrences of cohesive itemsets in the input sequence, we use them to discover the representative sequential patterns and the dominant partially ordered episodes, without going through the computationally expensive candidate generation procedures typically associated with sequential pattern and episode mining. Experiments show that our method efficiently discovers important patterns that existing state-of-the-art methods fail to discover.

Keywords Cohesive Itemsets, Sequential Patterns, Episodes, Association Rules

1 Introduction

Pattern discovery in sequential data is a well-established field in data mining. The earliest attempts focused on the setting where data consisted of many (typically short) sequences, where a pattern was defined as a (sub)sequence that re-occurred in a high enough number of such input sequences (Srikant and Agrawal 1996). The first attempt to identify patterns in a single long sequence of data was proposed by Mannila et al. (1997). The presented WINEPI

[1] A preliminary version appeared as "Efficient Discovery of Sets of Co-occurring Items in Event Sequences" (Cule et al. 2016). Sections 2.4 and 3.6 are based on "Mining Association Rules in Long Sequences" (Cule and Goethals 2010).

[2] University of Antwerp, Antwerp, Belgium,
[3] Monash University, Melbourne, Australia
E-mail: {boris.cule, len.feremans, bart.goethals}@uantwerpen.be

method uses a sliding window of a fixed length to traverse the sequence, and a pattern is then considered frequent if it occurs in a high enough number of these sliding windows. The paper describes algorithms for mining various pattern types — parallel episodes, which are essentially itemsets (with the possibility of some items re-occurring), serial episodes, which are equivalent to sequential patterns, and general episodes, which are partially ordered patterns. Here, we use the term *pattern* when we talk of any pattern type, and use more specific terms when appropriate.

An often-encountered critique of this method is that the obtained frequency is not an intuitive measure, since it does not correspond to the actual number of occurrences of the pattern in the sequence. For example, given sequence $axbcdbya$, and a sliding window length of 3, the frequency of itemset $\{a, b\}$ will be equal to 2, as will the frequency of itemset $\{c, d\}$. However, itemset $\{a, b\}$ occurs twice in the sequence, and itemset $\{c, d\}$ just once, and while the method is motivated by the need to reward c and d for occurring right next to each other, the reported frequency values remain difficult to interpret.

Laxman et al. (2007) attempted to tackle this issue by defining the frequency as the maximal number of non-overlapping minimal windows of the pattern in the sequence. In this context, a minimal window of the pattern in the sequence is defined as a contiguous subsequence of the input sequence that contains the pattern, such that none of its smaller contiguous subsequences also contains the pattern. However, while the method uses a relevance window of a fixed length, and disregards all minimal windows that are longer than the relevance window, the length of the minimal windows that do fit into the relevance window is not taken into account at all. For example, given sequence $axyzbcd$, with a relevance window larger than 4, the frequency of both itemset $\{a, b\}$ and itemset $\{c, d\}$ would be equal to 1, which would not reflect that the items making up the second pattern occur much closer to each other than those making up the first pattern.

Cule et al. (2014) proposed an amalgam of the two approaches (MARBLES$_{\mathrm{W}}$), defining the frequency of a pattern as the maximal sum of weights of a set of non-overlapping minimal windows of the pattern, where the weight of a window is defined as the inverse of its length. However, this method, too, struggles with the interpretability of the proposed measure. For example, given sequence $axbcdbya$ and a relevance window larger than 3, frequency of $\{a, b\}$ would be $2/3$, while frequency of $\{c, d\}$ would be $1/2$. Additionally, as the input sequence grows longer, the sum of these weights will grow, and the defined frequency can take any real positive value, giving the user no idea how to set a sensible frequency threshold.

All the techniques mentioned above use frequency measures that satisfy the so-called APRIORI property (Agrawal and Srikant 1994). This property implies that the frequency of a pattern is never smaller than the frequency of any of its superpatterns (in other words, frequency is an *anti-monotonic* measure). While this property is computationally very desirable, since large candidate patterns can be generated from smaller frequent patterns, the undesirable side effect is that larger patterns, which are often more useful to the end users, will never be ranked higher than any of their subpatterns. On top of this, all these methods focus solely on how often certain items occur near each other, and do not take occurrences of these items far away from each other into account. Consequently, if two items occur frequently, and through pure randomness often occur near each other, they will form a frequent itemset, even though they are, in fact, in no way correlated.

In another work, Cule et al. (2009) proposed a method that steps away from anti-monotonic quality measures, and introduce a new interestingness measure that combines

the coverage of an itemset with its cohesion. Cohesion is defined as a measure of how near each other the items making up an interesting itemset occur on average. However, the authors defined the coverage of an itemset as the sum of frequencies of all items making up the itemset, which results in a massive bias towards larger patterns instead. Furthermore, this allows for a very infrequent item making its way into an interesting itemset, as long as all other items in the itemset are very frequent and often occur near the infrequent item. As a result, the method is not scalable for any sequence with a large alphabet of items, which makes it unusable in most realistic data sets.

In this work, we use the cohesion introduced by Cule et al. (2009) as a single measure to evaluate cohesive itemsets. We consider itemsets as potential candidates only if each individual item contained in the itemset is frequent in the dataset. This allows us to filter out the infrequent items at the very start of our algorithm, without missing out on any cohesive itemsets. However, using cohesion as a single measure brings its own computational problems. First of all, cohesion is not an anti-monotonic measure, which means that a superset of a non-cohesive itemset could still prove to be cohesive. However, since the search space is exponential in the number of frequent items, it is impossible to evaluate all possible itemsets. We solve this by developing an upper bound on the maximal possible cohesion of all itemsets that can still be generated in a particular branch of the depth-first-search tree. This bound allows us to prune large numbers of potential candidate itemsets, without having to evaluate them. Furthermore, we present an efficient method to identify the minimal windows that contain a particular itemset, which is necessary to evaluate its cohesion.

Having discovered the cohesive itemsets, we move on to the problem of finding cohesive sequential patterns and partially ordered episodes. Due to the combinatorial explosion, the number of possible candidate patterns that potentially must be generated and evaluated quickly becomes prohibitive for typical sequential pattern or episode mining algorithms. We avoid this problem by taking the already discovered cohesive itemsets as a starting point, and then only evaluating those total and partial orders that actually occur in the data. More concretely, each discovered minimal window of an itemset represents one or more possible sequential patterns, so all we need to do is go through the list of such windows and update the frequency of each encountered sequential pattern. Once that is done, we report the representative sequential patterns and the dominant episodes (which we obtain simply by intersecting the discovered sequential patterns).

Finally, we show that cohesive itemsets can also form a basis for mining association rules between items. A cohesive itemset tells us that the items forming the itemset occur close to each other on average, but in some cases it may happen that an occurrence of a particular item implies, with a high probability, that some other items will occur nearby, but the implication may not hold in the other direction. For example, it is possible that item a always occurs near item b, but not vice versa (i.e., there may be many occurrences of item b far from any occurrence of item a). In this case, itemset $\{a, b\}$ would not be very cohesive, but an association rule $\{a\} \Rightarrow \{b\}$ would still be informative. We present an efficient algorithm that generates such rules starting from the discovered cohesive itemsets, using a cohesion-based confidence measure. Unlike the traditional frequency-based 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. Furthermore, we present

an important mathematical property that allows us to very quickly compute the confidence of most association rules, without having to revisit the data at all.

Our experiments show that our method discovers important patterns that existing methods struggle to rank highly, while dismissing obvious patterns consisting of items that co-occur frequently, but are not at all correlated. We further show that we achieve these results quickly, thus demonstrating the efficiency of our algorithm.

The remainder of the paper is organised as follows. In Sect. 2 we formally describe the problem setting and define the patterns we aim to discover. Sect. 3 provides a detailed description of our algorithms, while in Sect. 4 we present a thorough experimental evaluation of our method, in comparison with a number of existing state-of-the-art methods. We present an overview of the most relevant related work in Sect. 5, before summarising our main conclusions in Sect. 6.

2 Problem Setting

The dataset consists of a single event sequence $S = (e_1, \ldots, e_n)$. Each event e_k is represented by a pair (i_k, t_k), with i_k an event type (coming from the domain of all possible event types) and t_k an integer time stamp. For any $1 < k \leq n$, it holds that $t_k > t_{k-1}$. We use $S_{[j,l]}$, with $j < l$, to denote subsequence $(e_j, e_{j+1}, \ldots, e_{l-1}, e_l)$. The *length* of sequence S, denoted $|S|$, is equal to $t_n - t_1 + 1$, and the length of a subsequence $S_{[j,l]}$, denoted $|S_{[j,l]}|$, is equal to $t_l - t_j + 1$. For simplicity, we omit the time stamps from our examples, and write sequence $(e_1, \ldots, e_n)$ as $i_1 \ldots i_n$, implicitly assuming that the time stamps are consecutive integers starting with 1. In further text, we refer to event types as *items*, and sets of event types as *itemsets*.

2.1 Frequent Cohesive Itemsets

For an itemset $X = \{i_1, \ldots, i_m\}$, we denote the set of occurrences of items making up X in a sequence S by $N(X) = \{t \mid (i, t) \in S, i \in X\}$. For an item i, we define the *support* of i in an input sequence S as the number of occurrences of i in S, $sup(i) = |N(\{i\})|$. Given a user-defined support threshold *min_sup*, we say that an itemset X is *frequent* in a sequence S if for each $i \in X$ it holds that $sup(i) \geq min\_sup$.

To evaluate the cohesiveness of an itemset X in a sequence S, we must first identify minimal occurrences of the itemset in the sequence. More specifically, for each occurrence of an item in X, we will look for the minimal window within S that contains that occurrence and the entire itemset X. Formally, given a time stamp t, such that $(i, t) \in S$ and $i \in X$, we define the *size of a minimal occurrence of X around t* as

$$W_t(X) = \min\{|S_{[s,e]}| | t_s \leq t \leq t_e \wedge \forall i \in X \ \exists (i, t') \in S : t_s \leq t' \leq t_e\}.$$

We further define the *size of the average minimal occurrence of X in S* as

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

Finally, we define the *cohesion* of itemset X, with $|X| > 0$, in a sequence S as

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

If $|X| = 0$, we define $C(X) = 1$.

Given a user-defined cohesion threshold *min_coh*, we say that an itemset X is *cohesive* in a sequence S if it holds that $C(X) \geq min\_coh$.

Note that the cohesion is higher if the minimal occurrences are smaller. Furthermore, a minimal occurrence of itemset X can never be smaller than the size of X, so it holds that $C(X) \leq 1$. If $C(X) = 1$, then every single minimal occurrence of X in s is of length $|X|$.

The cohesion of a single item is always equal to 1. For singletons, therefore, our approach is equivalent to discovering frequent items. Since we are interested in mining frequent cohesive itemsets, we will from now on consider only itemsets consisting of 2 or more items. Formally, we say that an itemset X is a *frequent cohesive itemset* if

1. $|X| > 1$,
2. $\forall i \in X : sup(i) \geq min\_sup$,
3. $C(X) \geq min\_coh$.

An optional parameter, *max_size*, can be used to limit the size of the discovered patterns.

Cohesion is not an anti-monotonic measure. In other words, a superset of a non-cohesive itemset could turn out to be cohesive. For example, given sequence *abcxacbybac*, we can see that $C(\{a, b\}) = C(\{a, c\}) = C(\{b, c\}) = 6/7$, while $C(\{a, b, c\}) = 1$. While this allows us to eliminate bias towards smaller patterns, it also brings additional computational challenges which will be addressed in Sect. 3.

Finally, note that the definition of cohesion makes it potentially sensible to outliers. For example, if two items a and b are strongly correlated, just one random occurrence of an a far from any b could very negatively affect the cohesion of itemset $\{a, b\}$. However, this is somewhat mitigated by the fact that a and b must be frequent items to begin with, so it is unlikely that there would be an occurrence of either item arbitrarily far from all occurrences of the other one. Additionally, the more occurrences there are, the less of an effect one outlier will have on the average window size. Nevertheless, in data known to contain outliers, a user could avoid this risk by dividing the input sequence into segments (overlapping or not), and then mining cohesive itemsets in each segment. A single outlier would then affect the value of a pattern in one segment, but the pattern would still be ranked highly in all other segments where it is present.

2.2 Representative Sequential Patterns

In this section, we show how frequent cohesive itemsets can be used as a basis for discovering representative sequential patterns in the data, while avoiding computationally expensive candidate generation steps typical for sequential pattern mining. Since we use cohesive itemsets as a starting point, we are only able to find sequential patterns in which no event can re-occur. This is an inherent cost of choosing for simplicity of itemset mining over the complexity of sequential pattern mining. Naturally, this does mean our method is not suitable for data where important patterns often contain multiple instances of the same

item (such as, for example, DNA sequences, where the number of distinct items is limited). We define the necessary concepts below and present our mining algorithm in Sect. 3.4.

Given a frequent cohesive itemset $X = \{i_1, \ldots, i_n\}$, we can generate a sequential pattern $sp = s_1 s_2 \ldots s_n$ from X if $s_k \in X$, for $k \in \{1, \ldots, n\}$. In this case, we call X the *underlying itemset* of sp, denoted X_{sp}. Having found the minimal occurrences of a frequent cohesive itemset, our goal here is to find in which order do the items making up the itemset most often occur in those minimal occurrences. Any such frequently occurring order uniquely defines a sequential pattern.

Given a sequence $S = (e_1, \ldots, e_m)$, we say sequential pattern $sp = s_1 s_2 \ldots s_n$ *occurs* in S, denoted $sp \subseteq S$, if there exist integers $1 \leq k_1 < \ldots < k_n \leq m$, such that $s_j = e_{k_j}$ for $j \in \{1, \ldots, n\}$.

We define the *set of occurrences* of sp in an input sequence S as

$$occ_{se}(sp) = \{t \in N(X_{sp}) | \exists j, k : j \leq t \leq k, sp \subseteq S_{[j,k]}, |S_{[j,k]}| = W_t(X_{sp})\}.$$

Note that this formal definition corresponds to the above intuition, and says that a sequential pattern sp is considered to occur at time stamp t if its occurrence *around* time stamp t is as long as the minimal occurrence of its underlying itemset X_{sp} around time stamp t.

Finally, we define the *occurrence ratio* of sp within the occurrences of X_{sp} as

$$occ\_ratio_{se}(sp) = \frac{|occ_{se}(sp)|}{|N(X_{sp})|}.$$

Intuitively, the occurrence ratio of a sequential pattern sp measures the likelihood that the items making up its underlying itemset X_{sp} appear in a minimal occurrence of X_{sp} in the order defined by sp. Given a user-defined minimal occurrence ratio threshold min_or, we say a sequential pattern sp is *representative* if $occ\_ratio_{se}(sp) \geq min\_or$.

It is interesting to note that multiple sequential patterns can be contained within the same minimal occurrence of their underlying itemset X_{sp}. For example, given itemset $X = \{a, b, c, d\}$ and input sequence *abcbd*, we can see that the entire sequence is a minimal occurrence of X, and it contains occurrences of sequential patterns *abcd* and *acbd*. In this case, both *abcd* and *acbd* would have an occurrence ratio of 1. Moreover, a minimal occurrence of an itemset at a given time stamp is not necessarily unique. For example, in an input sequence *abcda*, at time stamp 2, the minimal occurrence of itemset $\{a, b, c, d\}$ could be both $S_{[1,4]}$ and $S_{[2,5]}$. In this case, sequential patterns *abcd* and *bcda* both occur at time stamp 2, and at time stamps 3 and 4, but only one of them occurs at time stamps 1 and 5. As a result, in this example, both *abcd* and *bcda* would have an occurrence ratio of 4/5, or 0.8.

2.3 Dominant Episodes

The approach described above can also be extended to finding dominant partial orders, or episodes, within the occurrences of frequent cohesive itemsets. In episode mining literature, an episode is typically represented by a directed acyclic graph $G = (V(G), E(G))$. Here $V(G) = (v_1, \ldots, v_m)$ is the set of nodes, where each node v_i corresponds to an item, and $E(G)$ is the set of directed edges between items, where an edge (v_i, v_j) means that item v_i

occurs before item v_j in any occurrence of G. We say an episode G is *transitively closed* if for any $v_i, v_j, v_k \in V(G)$ it holds that if $(v_i, v_j) \in E(G)$ and $(v_j, v_k) \in E(G)$ then $(v_i, v_k) \in E(G)$. To avoid redundancy, we only consider transitively closed episodes. However, in our examples and illustrations, we omit the edges whose presence is implied by other edges.

Formally, given a sequence $S = (e_1, \ldots, e_n)$ and an episode G we say that G *occurs* in S, denoted $G \preceq S$, if there exists a map f mapping each node $v_i \in V(G)$ to an index in $\{1, \ldots n\}$, such that $v_i = e_{f(v_i)}$ for $i = \{1, \ldots, m\}$, and that if there is an edge (v_i, v_j) in $E(G)$, then we must have $t_{f(v_i)} < t_{f(v_j)}$.

As with sequential patterns, we use frequent cohesive itemsets as a starting point (as a result, no item can re-occur in any episode we discover). Given an itemset X, we can generate episode G from X if $V(G) = X$. Again, in such a case, we call X the underlying itemset of G, denoted X_G. For two episodes, G_1 and G_2, with $X_{G_1} = X_{G_2}$, we say G_1 is a *subepisode* of G_2, denoted $G_1 \subseteq G_2$, if $E(G_1) \subseteq E(G_2)$. Given a sequential pattern sp, we denote its equivalent episode G_{sp}, whereby $V(G_{sp}) = X_{sp}$, and $E(G_{sp})$ imposes a total order on the items as defined by sp.

We define, as above, the *set of occurrences* of G in an input sequence S as

$$occ_{po}(G) = \{t \in N(X) | \exists j, k : j \leq t \leq k, G \preceq S_{[j,k]}, |S_{[j,k]}| = W_t(X_G)\}.$$

Finally, we define the *occurrence ratio* of G within the occurrences of X_G as

$$occ\_ratio_{po}(G) = \frac{|occ_{po}(G)|}{|N(X_G)|}.$$

Intuitively, $occ\_ratio_{po}(G)$ measures the likelihood that the items making up the underlying itemset X_G will appear in a minimal occurrence of X_G in the order defined by G.

Naturally, given an itemset X, the episode with the highest occurrence ratio will always be the itemset itself (i.e., the episode with $V(G) = X$ and $E(G) = \emptyset$), since it will, per definition, occur in every minimal occurrence of X. In fact, the occurrence ratio is an anti-monotonic measure, in the sense that if $G_1 \subseteq G_2$ then $occ\_ratio_{po}(G_1) \geq occ\_ratio_{po}(G_2)$. As a result, the episodes with the highest occurrence ratio will arguably be the least interesting ones (those with no edges will score the highest, followed by those with a single edge, etc.). We therefore choose to search for interesting episodes differently from our approach outlined for sequential patterns above. We propose to discover *exactly one* episode for each frequent cohesive itemset, namely, the most specific episode (i.e., as many edges as possible) that describes a sufficient number of occurrences of the itemset itself, given a user-defined minimal occurrence ratio threshold *min_por*. We conclude this section by formalising the above concepts.

Given a set of episodes $\{G_1, \ldots, G_n\}$, with $V(G_i) = X$ for $i \in \{1, \ldots, n\}$, we define the *intersecting episode* of $\{G_1, \ldots, G_n\}$, denoted $\bigcap_{i \in \{1,\ldots,n\}} G_i$, as the episode with nodes $V(\bigcap_{i \in \{1,\ldots,n\}} G_i) = X$ and edges $E(\bigcap_{i \in \{1,\ldots,n\}} G_i) = \bigcap_{i \in \{1,\ldots,n\}} E_i$. Given a frequent cohesive itemset X, we define its *dominant episode* as

$$dpo_X = \bigcap_{\substack{X_{sp}=X \\ occ\_ratio_{se}(sp) \geq y}} G_{sp},$$

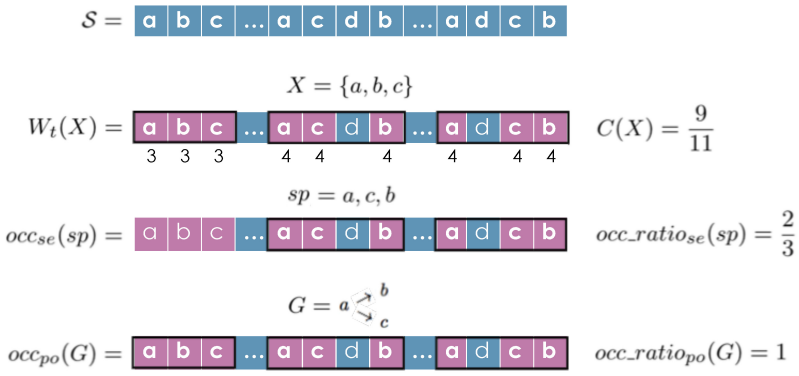


Fig. 1 Example of a cohesive itemset, a representative sequential pattern and a dominant episode.

with

$$y = \max\{z \in [0,1] | occ\_ratio_{po}(\bigcap_{\substack{X_{sp}=X \\ occ\_ratio_{se}(sp)\geq z}} G_{sp}) \geq min\_por\}.$$

In other words, we generate the dominant episode of an itemset X by taking the intersection of the minimal necessary number (as guaranteed by the value y above) of the top ranked sequential patterns (with respect to the occurrence ratio) generated from X. In practice, as will be discussed in detail in Sect. 3.5, we keep on adding the top sequential patterns until the occurrence ratio of the resulting intersection becomes high enough to satisfy the *min_por* threshold. Here, too, our main goal is to provide a simple, quick, method, based on cohesive itemsets and avoiding the high computational cost of generating exponentially many partial orders and then comparing them all to each other using some interestingness measure, that can still produce very satisfactory results. However, due to the simplicity of the approach, there is an inherent risk that some interesting partial orders may be missed.

We illustrate the above concepts with a simple example, shown in Fig. 1. Consider input sequence $abc \ldots acdb \ldots adcb$. In this case, $X = \{a, b, c\}$ is a cohesive itemset with cohesion $C(X) = \frac{3}{\frac{33}{9}} \approx 0.818$. Sequential pattern $sp = acb$ has an occurrence ratio $occ\_ratio_{se}(acb) = \frac{6}{9} = \frac{2}{3}$. Episode G, shown in the figure, and indicating that event a occurs first, followed by events b and c in unspecified order, however, has an occurrence ratio of 1. If the *min_por* threshold was set higher than $2/3$, G would be the dominant episode of itemset X, otherwise G_{sp} would be the dominant episode. For generation of the dominant episode G we would start with sequential patterns $sp_1 = acb$, with $occ\_ratio_{se}(sp_1) = \frac{2}{3}$, and $sp_2 = abc$, with $occ\_ratio_{se}(sp_2) = \frac{1}{3}$. Then we convert both sequential patterns to their equivalent episodes $E(G_{sp_1}) = \{a \to c, c \to b, a \to c\}$ and $E(G_{sp_2}) = \{a \to b, b \to c, a \to c\}$ where we added one edge to both episodes by computing the transitive closure. The intersection is then given by $E(G_{sp_1}) \cap E(G_{sp_2}) = \{a \to b, a \to c\}$.

2.4 Association Rules

Finally, we are also able to use the frequent cohesive itemsets as a basis for discovering 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 a frequent cohesive itemset. We denote such a rule by $X \Rightarrow Y$, and we call X the antecedent of the rule and Y the consequent of the rule. Intuitively, 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. We therefore define this new average as

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

We define the *confidence* of the association rule as

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

Note that if Y always occurs right next to a fully cohesive itemset X, the average minimal window will be exactly equal to the size of itemset $X \cup Y$, and the confidence of the rule $X \Rightarrow Y$ will be equal to 1. Intuitively, the inverse of the confidence tells us how large the average minimal window containing $X \cup Y$ for each occurrence of X is, compared to the minimal possible window (the size of $X \cup Y$), as illustrated in the example below. A rule $X \Rightarrow Y$ is considered confident if its confidence exceeds a given threshold, *min_conf*, i.e., if $c(X \Rightarrow Y) \geq$ *min_conf*.

Consider, for example, input sequence *abcabdbxyzb*. We can see that every a has a b right next to it, but not the other way around. As a result, itemset $X = \{a, b\}$ has a cohesion of $C(X) = 2/3$, which is high, but far from perfect. However, if we look at the two possible association rules between the two items, we can gain more insight into the data. For the two occurrences of item a, the minimal occurrence of itemset X nearby is always of size 2. Therefore, $c(\{a\} \Rightarrow \{b\}) = 1$. However, for the four occurrences of item b, the minimal occurrences of X are of size 2, 2, 4 and 8, respectively, with an average of 4. Therefore, $c(\{b\} \Rightarrow \{a\}) = 0.5$. From this information, we can conclude that if we encounter item a, we can be quite certain that item b will occur nearby, while the inverse implication is less likely.

Just like the cohesion of an itemset, the confidence of an association rule can also be sensitive to outliers in the data. Again, if two items a and b are strongly correlated, just one random occurrence of an a far from any b could negatively affect the confidence of rule $\{a\} \Rightarrow \{b\}$. However, in this case, the confidence of rule $\{b\} \Rightarrow \{a\}$ would not be affected. Once again, to reduce the effect of outliers, data could be divided into segments and rules discovered to hold in many segments could be considered to be the most reliable.

3 Algorithm

In this section we present a detailed description of our algorithms. We first show how we generate candidates in a depth-first manner, before explaining how we can prune large numbers of potential candidates by computing an upper bound of the cohesion of all itemsets

that can be generated within a branch of the search tree. Next we provide an efficient method to compute the sum of minimal windows of a particular itemset in the input sequence. Finally, we discuss algorithms for finding representative sequential patterns and dominant episodes, as well as confident association rules based on cohesive itemsets.

3.1 Depth-First-Search

The main routine of our FCI_{SEQ} algorithm is given in Algorithm 1. We begin by scanning the input sequence, identifying the frequent items, and storing their occurrence lists for later use. We then sort the set of frequent items on support in ascending order (line 2), initialise the set of frequent cohesive itemsets FC as an empty set (line 3), and start the depth-first-search process (line 4). Once the search is finished, we output the set of frequent cohesive itemsets FC, representative sequential patterns FC_{seq}, dominant episodes FC_{epi} and association rules FC_{rules} (line 5), that are computed depending on the status of the parameters *findSeq*, *findEpi* and *findRule*.

Algorithm 1: FCI_{SEQ}(*min_sup*, *max_size*, *min_coh*, *findSeq*, *min_or*, *findEpi*, *min_por*, *findRule*, *min_conf*) finds cohesive itemsets, representative sequential patterns, dominant episodes and confident association rules in a single sequence

1 FI = all items i where $N(\{i\}) \geq min\_sup$;
2 sort FI on support in ascending order;
3 $FC = FC_{seq} = FC_{epi} = FC_{rules} = \emptyset$;
4 DFS($\emptyset$, FI, *max_size*, *min_coh*, *findSeq*, *min_or*, *findEpi*, *min_por*, *findRule*, *min_conf*);
5 **return** $\langle FC, FC_{seq}, FC_{epi}, FC_{rules} \rangle$

The recursive DFS procedure is shown in Algorithm 2. In each call, X contains the candidate itemset, while Y contains items that are yet to be enumerated. In line 1, we evaluate the pruning function $C_{max}(X, Y)$ to decide whether to search deeper in the tree or not. This function will be described in detail in Sect. 3.2. If the branch is not pruned, there are two possibilities. If we have reached a non-leaf node (line 3) and there are more items to be enumerated, we pick the first such item a (line 4) and make two recursive calls to the DFS function — the first with a added to X (this is only executed if $X \cup \{a\}$ satisfies the *max_size* constraint), and the second with a discarded. Alternatively, if a leaf node is encountered (line 9), we add the discovered cohesive itemset to the output (provided its size is greater than 1). We then call FIND_SEQUENTIAL_PATTERNS (discussed in Sect. 3.4) to find sequential patterns based on X, if either *findSeq* or *findEpi* is true (note that we need the sequential patterns in order to find the dominant episodes, even if we do not need to output the sequential patterns themselves). We report the representative sequential patterns if *findSeq* is true, and then call FIND_DOMINANT_EPISODE (discussed in Sect. 3.5) to find the dominant episode based on X if *findEpi* is true. Finally, we also find and report association rules based on X by calling FIND_RULES (discussed in Sect. 3.6) if *findRule* is true.

Algorithm 2: DFS(X, Y, *max_size*, *min_coh*, *findSeq*, *min_or*, *findEpi*, *min_por*, *findRule*, *min_conf*) depth-first search to find cohesive itemsets, representative sequential patterns and dominant episodes, and confident association rules

```
1  if C_max(X, Y) < min_coh then
2  |  return;
3  else if Y ≠ ∅ then
4  |  a = first(Y);
5  |  if |X ∪ {a}| ≤ max_size then
6  |  |  DFS(X ∪ {a}, Y \
   |  |    {a}, max_size, min_coh, findSeq, min_or, findEpi, min_por, findRule, min_conf);
7  |  end
8  |  DFS(X, Y \ {a}, max_size, min_coh, findSeq, min_or, findEpi, min_por, findRule, min_conf);
9  else
10 |  if |X| > 1 then
11 |  |  FC = FC ∪ {X};
12 |  |  if findSeq or findEpi then
13 |  |  |  sps ← FIND_SEQUENTIAL_PATTERNS(X);
14 |  |  |  if findSeq then
15 |  |  |  |  for sp ∈ sps do
16 |  |  |  |  |  if occ_ratio_se(sp) ≥ min_or then
17 |  |  |  |  |  |  FC_seq ← FC_seq ∪ sp;
18 |  |  |  |  end
19 |  |  |  end
20 |  |  |  if findEpi then
21 |  |  |  |  G ← FIND_DOMINANT_EPISODE(X, sps, min_por);
22 |  |  |  |  FC_epi ← FC_epi ∪ G;
23 |  |  |  end
24 |  |  end
25 |  |  if findRule then
26 |  |  |  rules ← FIND_RULES(X, min_conf);
27 |  |  |  FC_rules ← FC_rules ∪ rules;
28 |  |  end
29 |  end
30 end
```

3.2 Pruning

At any node in the search tree, X denotes all items currently making up the candidate itemset, while Y denotes all items that are yet to be enumerated. Starting from such a node, we can still generate any itemset Z, such that $X \subseteq Z \subseteq X \cup Y$ and $|Z| \leq max\_size$. In order to be able to prune the entire branch of the search tree, we must therefore be certain that for every such Z, the cohesion of Z cannot satisfy the minimum cohesion threshold.

In the remainder of this section, we first define an upper bound for the cohesion of all itemsets that can be generated in a particular branch of the search tree, before providing a detailed proof of its soundness. Given itemsets X and Y, with $|X| > 0$ and $X \cap Y = \emptyset$, the $C_{max}(X, Y)$ pruning function used in line 1 of Algorithm 2 is defined as

$$C_{max}(X, Y) = \frac{|X \cup Y_{max}||N(X \cup Y_{max})|}{\sum_{t \in N(X)} W_t(X) + |X \cup Y_{max}||N(Y_{max})|},$$

where

$$Y_{max} = \{Y_i | \max_{\substack{Y_i \subseteq Y, \\ |Y_i| \leq \ max\_size - |X|}} |N(Y_i)|\}. \quad (1)$$

For $|X| = 0$, we define $C_{max}(X, Y) = 1$. Note that if $Y = \emptyset$, $C_{max}(X, Y) = C(X)$, which is why we do not need to evaluate $C(X)$ before outputting X in line 4 of Algorithm 2.

Before proving that the above upper bound holds, we will first explain the intuition behind it. When we find ourselves at node $\langle X, Y \rangle$ of the search tree, we will first evaluate the cohesion of itemset X. If X is cohesive, we need to search deeper in the tree, as supersets of X could also be cohesive. However, if X is not cohesive, we need to evaluate how much the cohesion can still grow if we go deeper into this branch of the search tree. Logically, starting from $C(X) = \frac{|X|}{W(X)} = \frac{|X||N(X)|}{\sum_{t \in N(X)} W_t(X)}$, the value of this fraction will grow maximally if the numerator is maximised, and the denominator minimised. Clearly, as we add items to X, the numerator will grow, and it will grow maximally if we add as many items to X as possible. However, as we add items to X, the denominator must grow, too, so the question is how it can grow minimally. In the worst case, each new window added to the sum in the denominator will be minimal (i.e., its length will be equal to the size of the new itemset), and the more such windows we add to the sum, the higher the overall cohesion will grow. Note that *the worst case* from the point of view of our pruning capability actually refers to the highest-scoring candidate that could yet be produced, and therefore corresponds to *the best case* in terms of pattern discovery.

For example, given sequence *acb* and a cohesion threshold of 0.8, assume we find ourselves in node $\langle \{a, b\}, \{c\} \rangle$ of the search tree. We will then first find the smallest windows containing $\{a, b\}$ for each occurrence of a and b, i.e., $W_1(\{a, b\}) = W_3(\{a, b\}) = 3$. Note that we do this by going through the lists of time stamps of a and b, and not by revisiting the entire input sequence. In other words, to compute the upper bound of the cohesion of all itemsets that can still be generated from this node, we can only use the information about the occurrences of a and b. It turns out that $C(\{a, b\}) = \frac{2 \times 2}{3+3} = \frac{2}{3}$, which is not cohesive enough. However, if we add c to itemset $\{a, b\}$, we know that the size of the new itemset will be 3, we know the number of occurrences of items from the new itemset will be 3, and the numerator will therefore be equal to 9. For the denominator, we have no such certainties, but we know that, in the worst case, the windows for the occurrences of a and b will not grow (i.e., each smallest window of $\{a, b\}$ will already contain an occurrence of c), and the windows for all occurrences of c will be minimal (i.e., of size 3). Indeed, when we evaluate the above upper bound, we obtain $C_{max}(\{a, b\}, \{c\}) = \frac{3 \times (2+1)}{6+3 \times 1} = \frac{9}{9} = 1$. We see that even though the cohesion of $\{a, b\}$ is $\frac{2}{3}$, the cohesion of $\{a, b, c\}$ could, in the worst case, be as high as 1. And in our sequence *acb*, that is indeed the case. The above example also demonstrates the tightness of our upper bound, as the computed value can, in fact, turn out to be equal to the actual cohesion of a superset yet to be generated.

We now present a full formal proof of the soundness of the proposed upper bound. In order to do this, we will need the following lemma.

Lemma 1 *For any six positive numbers a, b, c, d, e, f, with $a \leq b$, $c \leq d$ and $e \leq f$, it holds that*

1. *if $\frac{a+c+e}{b+e} < 1$ then $\frac{a+c+e}{b+e} \leq \frac{a+d+f}{b+f}$,*

2. if $\frac{a+c+e}{b+e} \geq 1$ *then* $\frac{a+d+f}{b+f} \geq 1$.

Proof We begin by proving the first claim. To start with, note that if $\frac{a+c+e}{b+e} < 1$, then $\frac{a}{b} < 1$. It follows that $\frac{a+e}{b+e} < 1$, and for any positive number f, with $e \leq f$, it holds that $\frac{a+e}{b+e} \leq \frac{a+f}{b+f}$. Subsequently, it holds that $\frac{a+c+e}{b+e} \leq \frac{a+c+f}{b+f}$. Finally, for any positive number d, with $c \leq d$, it holds that $\frac{a+c+f}{b+f} \leq \frac{a+d+f}{b+f}$, and therefore $\frac{a+c+e}{b+e} \leq \frac{a+d+f}{b+f}$. For the second claim, it directly follows that if $\frac{a+c+e}{b+e} \geq 1$, then $\frac{a+c}{b} \geq 1$, $\frac{a+d}{b} \geq 1$, and $\frac{a+d+f}{b+f} \geq 1$. □

Theorem 1 *Given itemsets X and Y, with $X \cap Y = \emptyset$, for any itemset Z, with $X \subseteq Z \subseteq X \cup Y$ and $|Z| \leq max\_size$, it holds that $C(Z) \leq C_{max}(X, Y)$.*

Proof We know that $C(Z) \leq 1$, so the theorem holds if $|X| = 0$. Assume now that $|X| > 0$. First, recall that $C(Z) = \frac{|Z|}{W(Z)} = \frac{|Z||N(Z)|}{\sum_{t \in N(Z)} W_t(Z)}$. We can rewrite this expression as

$$C(Z) = \frac{(|X| + |Z \setminus X|)(|N(X)| + |N(Z \setminus X)|)}{\sum_{t \in N(X)} W_t(Z) + \sum_{t \in N(Z \setminus X)} W_t(Z)}.$$

Further note that for a given time stamp in $N(X)$, the minimal window containing Z must be at least as large as the minimal window containing only X, and for a given time stamp in $N(Z \setminus X)$, the minimal window containing Z must be at least as large as the size of Z. It therefore follows that

$$\sum_{t \in N(X)} W_t(Z) \geq \sum_{t \in N(X)} W_t(X),$$
$$\sum_{t \in N(Z \setminus X)} W_t(Z) \geq |Z||N(Z \setminus X)|,$$

and, as a result,

$$C(Z) \leq \frac{|X||N(X)| + |Z \setminus X||N(X)| + |Z||N(Z \setminus X)|}{\sum_{t \in N(X)} W_t(X) + |Z||N(Z \setminus X)|}.$$

Finally, we note that, per definition,

$$|Z \setminus X| \leq \min(max\_size, |X \cup Y|) - |X|,$$

and, since Z is generated by adding items from Y to X, until either the size of Z reaches *max_size* or there are no more items left in Y,

$$|N(Z \setminus X)| \leq |N(Y_{max})|.$$

At this point we will use Lemma 1 to take the proof further. Note that, per definition, $C(X) = \frac{|X||N(X)|}{\sum_{t \in N(X)} W_t(X)} \leq 1$. We now denote

$$\begin{aligned}
a =& |X||N(X)|, \\
b =& \sum_{t \in N(X)} W_t(X), \\
c =& |Z \setminus X||N(X)|, \\
d =& (\min(max_size, |X \cup Y|) - |X|)|N(X)| = |Y_{max}||N(X)|, \\
e =& |Z||N(Z \setminus X)|, \\
f =& |X \cup Y_{max}||N(Y_{max})|.
\end{aligned}$$

Since a, b, c, d, e and f satisfy the conditions of Lemma 1, we know that it holds that

1. if $\frac{|X||N(X)|+|Z\setminus X||N(X)|+|Z||N(Z\setminus X)|}{\sum_{t \in N(X)} W_t(X)+|Z||N(Z\setminus X)|} < 1$ then
   $\frac{|X||N(X)|+|Z\setminus X||N(X)|+|Z||N(Z\setminus X)|}{\sum_{t \in N(X)} W_t(X)+|Z||N(Z\setminus X)|} \leq \frac{|X||N(X)|+|Y_{max}||N(X)|+|X\cup Y_{max}||N(Y_{max})|}{\sum_{t \in N(X)} W_t(X)+|X\cup Y_{max}||N(Y_{max})|}$,
2. if $\frac{|X||N(X)|+|Z\setminus X||N(X)|+|Z||N(Z\setminus X)|}{\sum_{t \in N(X)} W_t(X)+|Z||N(Z\setminus X)|} \geq 1$ then
   $\frac{|X||N(X)|+|Y_{max}||N(X)|+|X\cup Y_{max}||N(Y_{max})|}{\sum_{t \in N(X)} W_t(X)+|X\cup Y_{max}||N(Y_{max})|} \geq 1$.

Finally, note that
$\frac{|X||N(X)|+|Y_{max}||N(X)|+|X\cup Y_{max}||N(Y_{max})|}{\sum_{t \in N(X)} W_t(X)+|X\cup Y_{max}||N(Y_{max})|} =$
$\frac{|X\cup Y_{max}|(|N(X)|+|N(Y_{max})|)}{\sum_{t \in N(X)} W_t(X)+|X\cup Y_{max}||N(Y_{max})|} = C_{max}(X,Y)$.
From the first claim above, it follows that if $\frac{|X||N(X)|+|Z\setminus X||N(X)|+|Z||N(Z\setminus X)|}{\sum_{t \in N(X)} W_t(X)+|Z||N(Z\setminus X)|} < 1$, then $C(Z) \leq C_{max}(X,Y)$. From the second claim, it follows that if $\frac{|X||N(X)|+|Z\setminus X||N(X)|+|Z||N(Z\setminus X)|}{\sum_{t \in N(X)} W_t(X)+|Z||N(Z\setminus X)|} \geq 1$, then $C_{max}(X,Y) \geq 1$, and since, per definition, $C(Z) \leq 1$, it follows that $C(Z) \leq C_{max}(X,Y)$. This completes the proof. □

Since an important feature of computing an upper bound for the cohesion of all itemsets in a given branch of the search tree is to establish how much cohesion could grow *in the worst case*, we need to figure out which items from Y should be added to X to reach this worst case. As has been discussed above, the worst case is actually materialised by adding as many as possible items from Y, and by first adding those that have the most occurrences. However, if the *max_size* parameter is used, it is not always possible to add all items in Y to X. In this case, we can only add $max\_size - |X|$ items to X, which is why we defined Y_{max} as we did in Equation 1. Clearly, if $|X \cup Y| \leq max\_size$, $Y_{max} = Y$. If not, at first glance it may seem computationally very expensive to determine $|N(Y_i)|$ for every possible Y_i. However, we solve this problem by sorting the items in Y on support in descending order. In other words, if $Y = \{y_1, \ldots, y_n\}$, with $sup(y_i) \geq sup(y_{i+1})$ for $i \in \{1, \ldots, n-1\}$, then we can compute $|N(Y_{max})|$ as

$$|N(Y_{max})| = \sum_{i \in \{1,\ldots,\ max\_size - |X|\}} |N(\{y_i\})|.$$

As a result, the only major step in computing $C_{max}(X,Y)$ is that of computing $\sum_{t \in N(X)} W_t(X)$, as the rest can be computed in constant time. The procedure for computing $\sum_{t \in N(X)} W_t(X)$

is explained in detail in Sect. 3.3. We end this section with an example illustrating our pruning technique. Given an input sequence $acdebbfgha$ and thresholds $min\_coh = 0.8$ and $max\_size = 3$, assume we are visiting the $\langle X, Y\rangle$ node of the search tree, with $X = \{a, b\}$ and $Y = \{c, d, e, f, g, h\}$. At this point, we will compute the sizes of the minimal occurrences of itemset $\{a, b\}$ for time stamps 1, 5, 6 and 10, and find that they all equal 5. As a result, the cohesion of $\{a, b\}$ will be equal to 0.4. However, we cannot be certain if we can prune this branch of the tree unless we know that none of the itemsets that can be generated within it cannot be cohesive. Therefore, we need to evaluate our upper bound for the cohesion of all such itemsets, $C_{max}(X, Y)$. We first compute

$$Y_{max} = \{Y_i | \max_{\substack{Y_i \subseteq Y, \\ |Y_i| \leq \ max\_size - |X|}} |N(Y_i)|\} = \{c\}.$$

As discussed above, by sorting the items in Y on frequency, we know that Y_{max} can be obtained by picking items in order from Y until we have either reached the *max_size* constraint (as in this case) or run out of items. We then compute

$$C_{max}(X, Y) = \frac{|X \cup Y_{max}||N(X \cup Y_{max})|}{\sum_{t \in N(X)} W_t(X) + |X \cup Y_{max}||N(Y_{max})|}$$

$$= \frac{|\{a, b, c\}||N(\{a, b, c\})|}{\sum_{t \in N(X)} W_t(X) + |\{a, b, c\}||N(\{c\})|} = \frac{3 \times 5}{20 + 3 \times 1} = \frac{15}{23} \approx 0.65.$$

In other words, no itemset that can be generated within this branch of the search tree can have a cohesion higher than 0.65, and since the cohesion threshold is set to 0.8, we can safely prune the entire branch.

3.3 Computing the Sum of Minimal Windows

The algorithm for computing the sum of minimal windows is shown in Algorithm 3. For a given itemset X, the algorithm keeps a list of all time stamps at which items of X occur in the *positions* variable. The *nextpos* variable keeps a list of next time stamps for each item, while *lastpos* keeps a list of the last seen occurrences for each item. Since we need to compute the minimal window for each occurrence, we keep on doing this until we have either computed them all, or until the running sum has become large enough to safely stop, knowing that the branch can be pruned (line 7). Concretely, by rewriting the definition of $C_{max}(X, Y)$, we know we can stop if we are certain that the sum will be larger than

$$W_{max}(X, Y) = \frac{|X \cup Y_{max}||N(X)| + |N(Y_{max})|(1 - min\_coh)}{min\_coh}.$$

When a new item comes in, we update the working variables, and compute the first and last position of the current window (line 18). If the smallest time stamp of the current window has changed, we go through the list of active windows and check whether a new shortest length has been found. If so, we update it (line 24). We then remove all windows for which we are certain that they cannot be improved from the list of active windows (line 26), and update the overall sum (line 27). Finally, we add the new window for the current time stamp to the list of active windows (line 31).

Algorithm 3: SUM_MIN_WINS(X, Y) sums minimal windows of X

```
   /* Maintain last, current and next visited position for each item  */
 1 smw ← 0; index ← 0;
 2 positions ← positions for every item in X;
 3 nextpos ← {positions[i_1][0], positions[i_2][0], positions[i_3][0], ...};
 4 lastpos ← {−∞, −∞, −∞, ...};
 5 prev_min ← −∞; active_windows ← ∅;
   /* Main loop over each occurrence */
 6 while index < |N(X)| do
       /* Abandon if running sum is already too high to be cohesive */
 7     if smw + (|N(X)| + |active_windows| − index) × |X| > W_max(X, Y) then
 8         return ∞;
 9     end
       /* Update last and next position of next item */
10     current_pos ← ∞;
11     current_item ← ∅;
12     for i in X do
13         if current_pos > nextpos[i] then
14             current_pos ← nextpos[i];
15             current_item ← i;
16         end
17     end
       /* Compute current window */
18     lastpos[current_item] ← current_pos;
19     nextpos[current_item] ← next(positions[current_item], current_pos);
20     minpos ←min(lastpos); maxpos ←max(lastpos);
       /* If new minimum in current window */
21     if minpos ≠ −∞ and minpos > prev_min then
           /* Inner loop over previous occurrences for non-final windows */
22         for window ∈ active_windows do
               /* Update window size */
23             newwidth ← maxpos−min(minpos, window.pos) + 1;
24             window.width ←min(window.width, newwidth);
               /* Make window final */
25             if window.pos < minpos or window.width == |X| or
                 window.width < (maxpos − window.pos + 1) then
26                 active_windows ← active_windows \ {window};
27                 smw ← smw + window.width;
28             end
29         end
30     end
31     active_windows ← active_windows ∪ {window(current_pos, maxpos − minpos + 1)};
32     prev_min ← minpos; index ← index + 1;
33 end
34 smw ← smw+sum(window.width|window ∈ active_windows);
35 return smw;
```

Note that the sum of minimal windows is independent of Y, the items yet to be enumerated. Therefore, if the branch is not pruned, the recursive DFS procedure shown in Algorithm 2 will be called twice, but X will remain unchanged in the second of those calls (line 9), so we will not need to recompute the sum of windows, allowing us to immediately evaluate the upper bound in the new node of the search tree.

We illustrate how the algorithm works on the following example. Assume we are given the input sequence *aabccccacb*, and we are evaluating itemset $\{a, b, c\}$. Table 1 shows the values of the main variables as the algorithm progresses. As each item comes in, we update the values of *nextpos* and *lastpos* (other variables are not shown in the table). In each iteration, we compute the current best minimal window for the given time stamp as $\max(lastpos) - \min(lastpos) + 1$. We also update the values of any previous windows that might have changed for the better (this can only happen if $\min(lastpos)$ has changed), using either the current window above if it contains the time stamp of the window's event, or the window stretching from the relevant time stamp to $\max(lastpos)$. Finally, before proceeding with the next iteration, we remove all windows for which we are certain that they cannot get any smaller from the list of active windows.

Table 1 Computation of minimal windows.

t	item	*nextpos*	*lastpos*	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0	-	$(1,3,4)$	(∞,∞,∞)	-	-	-	-	-	-	-	-	-	-
1	a	$(2,3,4)$	$(1,\infty,\infty)$	∞	-	-	-	-	-	-	-	-	-
2	a	$(8,3,4)$	$(2,\infty,\infty)$	∞	∞	-	-	-	-	-	-	-	-
3	b	$(8,10,4)$	$(2,3,\infty)$	∞	∞	∞	-	-	-	-	-	-	-
4	c	$(8,10,5)$	$(2,3,4)$	**4**	**3**	**3**	**3**	-	-	-	-	-	-
5	c	$(8,10,6)$	$(2,3,5)$	-	-	-	-	4	-	-	-	-	-
6	c	$(8,10,7)$	$(2,3,6)$	-	-	-	-	4	5	-	-	-	-
7	c	$(8,10,9)$	$(2,3,7)$	-	-	-	-	4	5	6	-	-	-
8	a	$(\infty,10,9)$	$(8,3,7)$	-	-	-	-	**4**	5	6	6	-	-
9	c	$(\infty,10,\infty)$	$(8,3,9)$	-	-	-	-	-	5	6	6	7	-
10	b	(∞,∞,∞)	$(8,10,9)$	-	-	-	-	-	**5**	**4**	**3**	**3**	**3**

In the table, windows that are not active are marked with '-', while definitively determined windows are shown in bold. We can see that, for example, at time stamp 4, we have determined the value of the first four windows. Window w_1 cannot be improved on, since time stamp 1 has already dropped out of *lastpos*, while the other three windows cannot be improved since 3 is the absolute minimum for a window containing three items. At time stamp 8, we know that the length of w_5 must be equal to 4, since any new window to come must stretch at least from time stamp 5 to a time stamp in the future, i.e., at least 9. Finally, once we have reached the end of the sequence, we mark all current values of still active windows as determined.

Note that the goal of this algorithm is to determine whether we can prune a branch of the search tree or not. We know that a branch can be pruned if the computed sum of windows is large enough. Therefore, we optimise the algorithm to stop computing minimal windows once the running sum is already large enough, since, in that case, we can prune without computing the exact sum.

3.4 Representative Sequential Patterns

In this section we describe our algorithm for discovering representative sequential patterns based on frequent cohesive itemsets. For example, when mining patterns consisting of words

used in the *On the Origin of Species by Means of Natural Selection* by *Charles Darwin* (see Sect. 4 for more details about the dataset) we discover that itemset {*tierra, del, fuego*} has a cohesion of 1. However, we also find that sequential pattern *tierra del fuego* has an occurrence ratio of 1 within the occurrences of its underlying itemset {*tierra, del, fuego*}. In other words, in every minimal occurrence of itemset {*tierra, del, fuego*}, the word *tierra* occurs before the word *del*, followed by *fuego*. In this section we describe how, starting from a frequent cohesive itemset and its minimal occurrences, we discover representative sequential patterns.

3.4.1 Computing minimal windows for sequential patterns

Computing the number of occurrences of sequential patterns, as defined in Sect. 2.2, brings with it additional complexity. So far we were only interested in the size of the minimal window at each occurrence. However, at first glance, given a cohesive itemset $X = \{i_1, \ldots, i_n\}$, we must now count each occurrence of up to $n!$ sequential permutations in the worst case. In order to compute the number of occurrences correctly we must also deal with the fact that *more than one* sequential pattern can occur within one minimal occurrence of the itemset. For example, given sequence *aba* we need to take into account that both sequential patterns *ab* and *ba* occur at time stamp 2 since the occurrences of both are equally long as the minimal occurrence of itemset $\{a, b\}$. Algorithm SUM_MIN_WINS$_{\text{seq}}$ for computing the *minimal windows for sequential patterns* is shown in Algorithm 4. This algorithm is an adaptation of Algorithm 3, and we therefore omit unmodified code for brevity. The main difference is that we now return a list of *minimal windows*, where for each occurrence t we maintain possibly *multiple instances* of the minimal windows at each occurrence. The list of *final windows* is first initialised (line 2) and returned (line 30) together with the sum of minimal windows. As a result, within the inner loop we now not only update the minimal width of active windows, but maintain, where necessary, multiple instances of windows with the same minimal length (lines 9 to 17). If a new window is found with a lower minimal width, we *reset* the instances of active windows (line 12). If a new window is found with the same minimal width, we *add* that window to the instances (line 15). For example, given sequence $\ldots bxaba \ldots$, for itemset $\{a, b\}$, we will find one minimal window for the two occurrences of a, and the first occurrence of b, but we will find two minimal windows for the second occurrence of b. Note that, compared to Algorithm 3, we omitted the condition of removing windows where $window.width == |X|$ from the list of active windows (line 17), because we now want to enumerate all instances of the minimal window.

3.4.2 Finding Representative Sequential Patterns

We now describe the algorithm FIND_SEQUENTIAL_PATTERNS, shown in Algorithm 5, which returns the set of all sequential patterns that occur within the minimal windows of underlying itemset X. In the main algorithm (see Algorithm 2) we only report sequential patterns that occur often enough, that is, we remove sequential patterns where the occurrence ratio is lower than *min_or*. Remark that we have to deal with *multiple permutations* within the same window caused by *duplicate* items. For example, given itemset $\{a, b, c, d\}$ and a minimal window $a\underline{b}\underline{c}bd$, both sequential patterns *abcd* and *acbd* occur within the same minimal

Algorithm 4: SUM_MIN_WINS$_{\text{seq}}(X, Y)$ adaptation of Algorithm 3 for maintaining possibly multiple minimal windows at each occurrence of X

```
   /* Maintain last, current and next visited position for each item  */
1  ...
2  final_windows ← ∅;
   /* Main loop over each occurrence */
3  while index < |N(X)| do
       /* Abandon if running sum is already to high to be cohesive */
4      ...
       /* Update last and next position of next item */
5      ...
       /* Compute current window */
6      ...
       /* If new minimum in current window */
7      if minpos ≠ −∞ and minpos > prev_min then
           /* Inner loop over previous occurrences for non-final windows */
8          for window ∈ active_windows do
               /* Update window size and maintain 1 or more instances of minimal windows at
                  each occurrence */
9              newwidth ← maxpos−min(minpos, window.pos) + 1;
10             if newwidth < window.width then
11                 window.width ← newwidth;
12                 window.instances = {instance(minpos, maxpos)};
13             end
14             if newwidth == window.width then
15                 window.instances = window.instances ∪ {instance(minpos, maxpos)};
16             end
               /* Make window final */
17             if window.pos < minpos or window.width < (maxpos − window.pos + 1) then
18                 active_windows ← active_windows \ {window};
19                 final_windows ← final_windows ∪ {window};
20                 smw ← smw + window.width;
21             end
22         end
23     end
24     curr_window ← window(current_pos, maxpos − minpos + 1, instance(minpos, maxpos));
25     active_windows ← active_windows ∪ {curr_window};
26     prev_min ← minpos; index ← index + 1;
27 end
28 smw ← smw+sum(window.width|window ∈ active_windows);
29 final_windows ← final_windows ∪ active_windows;
30 return ⟨smw, final_windows⟩;
```

window since b occurs more than once. In general, our algorithm should discover any permutation of $|X|$ elements. A naïve approach would enumerate all $|X|!$ candidate sequential patterns and check for each minimal window of X if the candidate occurs. Our algorithm, however, only generates candidate sequential patterns that occur in at least one window, making the method feasible even if $|X|$ is large.

FIND_SEQUENTIAL_PATTERNS starts by defining an empty multiset (line 2) which is returned (line 23) and updated with the discovered sequential patterns in each window (line 21). The idea here is that we count each permutation that occurs at each occurrence, and return this multiset, for example returning $\{abcd : 5, acbd : 4\}$. The outer loop (lines 3

Algorithm 5: FIND_SEQUENTIAL_PATTERNS(X) finds sequential patterns based on cohesive itemset X

```
1  ⟨smw, min_windows_seq⟩ ← SUM_MIN_WINS_seq(X, ∅);
2  sps ← multiset();
   /* Main loop over each occurrence. */
3  for win ∈ min_windows_seq do
4      win_occurrences ← ∅;
       /* Inner loop over each minimal window instance */
5      for win_ins ∈ win do
6          positions ← {⟨i, t⟩| i ∈ X ∧ win_ins.min ≤ t ≤ win_ins.max};
           /* Remove positions not relevant for enumerating sequential patterns. */
7          for n = 1; n < |positions|; n = n + 1 do
8              ⟨i_n, t_n⟩ ← positions[n];
9              ⟨i_{n+1}, t_{n+1}⟩ ← positions[n + 1];
10             if i_n = i_{n+1} or i_n = i_0 or i_n = i_{|positions|} then
11                 positions ← positions \ positions[n];
12         end
           /* Enumerate sequential patterns using the cartesian product of positions. */
13         for i ∈ X do
14             pos_i ← {⟨i, t⟩ | ⟨i, t⟩ ∈ positions};
15         end
16         cart_prod ← pos_1 × pos_2 × ... × pos_{|X|};
           /* Update occurrences in window */
17         for X_pos ∈ cart_prod do
18             win_occurrences ← win_occurrences ∪ to_sequence(X_pos);
19         end
20     end
       /* Update total occurrences in sequence. */
21     sps ← sps ⊎ win_occurrences;
22 end
23 return sps;
```

to 21) loops over all occurrences of the itemset. We create an empty set (line 4) to count any distinct *sequential pattern* found within the current occurrence. Note that adding each sequential pattern directly to the multiset would result in potentially doubly counting a sequential pattern occurrence. Next, in the inner loop (lines 5 to 19) we loop over possibly multiple minimal window instances at each occurrence. In the inner loop we generate sequential patterns occurring within each minimal window instance. We first fetch the positions for each item of X within the window (line 6) from the input sequence and retrieve a list of $\langle i, t\rangle$ positions. Next we *filter* out positions that are not relevant for generating candidate sequential patterns: For the boundary items at the minimal and maximal positions, we know (by definition of a minimal window) that no other position of those items will result in a smaller window, thus each sequential pattern must start with the item found at the left boundary and end with the item on the right boundary. Therefore, we can safely ignore any other positions of the boundary items within the minimal window (line 11). We also remove any position where two or more occurrences of the same item directly follow each other (line 11). Next we enumerate all possible permutations using a *cartesian product* (line 16) of the filtered position lists of each item. In the simplest case, all elements will occur just once, and the cartesian product will result in exactly one sequential pattern. If we do have multiple positions for some items, the cartesian product will combine them

with positions of other items and generate multiple sequential patterns. We then add these sequential patterns to the current set of already discovered sequential patterns within the window.

3.4.3 Example

In order to illustrate FIND_SEQUENTIAL_PATTERNS we provide an example. Assume an itemset $X = \{a, b, c, d\}$ and input sequence

$$\begin{array}{cccccccccc} \mathbf{a} & z & \mathbf{b}_1 & \mathbf{c}_1 & \mathbf{b}_2 & z & \mathbf{b}_3 & \mathbf{c}_2 & \mathbf{d}. \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$

Note that, in this case, the entire input sequence is also a minimal window of X. The position list is then $\{\langle a, 1\rangle \langle b_1, 3\rangle \langle c_1, 4\rangle \langle b_2, 5\rangle, \langle b_3, 7\rangle, \langle c_2, 8\rangle \langle d, 9\rangle\}$. We filter out b_2 since it it is directly followed by b_3. We then take the cartesian product of the position lists of each item, thereby generating all candidate sequential patterns within this window:

$$\begin{array}{cccccccccccc} a & \times & b & \times & c & \times & d & & a & \times & b & \times & c & \times & d & & \text{to\_sequence} \\ 1 & & 3 & & 4 & & 9 & & 1 & & 3 & & 4 & & 9 & & \text{a b c d} \\ & & 7 & & 8 & & & = & 1 & & 7 & & 4 & & 9 & = & \text{a c b d} \\ & & & & & & & & 1 & & 3 & & 8 & & 9 & & \text{a b c d} \\ & & & & & & & & 1 & & 7 & & 8 & & 9 & & \text{a b c d} \end{array}$$

We then convert the result of the cartesian product trivially into distinct sequential patterns, that is *abcd* and *acbd*, and increment the support for both sequential patterns.

For a second example, assume $X = \{b, c\}$ and the input sequence is the same as above. SUM_MIN_WINS$_{\text{seq}}$ would then generate a minimal window at $t \in \{3, 4, 5, 7, 8\}$ of length 2 and two minimal window *instances* at $t = 4$, namely b_1, c_1 and c_1, b_2. We find two sequential patterns, $sp_1 = \langle b, c\rangle$ occurring at $t \in \{3, 4, 7, 8\}$ and $sp_2 = \langle c, b\rangle$ occurring at $t = \{4, 5\}$. Given a minimal occurrence ratio threshold $min\_or = 0.8$, we conclude that sp_1 is representative, and sp_2 is not.

3.5 Dominant Episodes

In this section we describe our algorithm for finding dominant episodes within the minimal occurrences of cohesive itemsets. In the previous section, given an itemset X, we enumerated all representative sequential patterns. Here we find the single most dominant episode (or partial order) of X. We take a more direct approach, and compute the *intersection* of the *top-k* sequential patterns (or total orders) of X. The value of k is specific to a particular itemset X, and is determined by iteratively using more sequential patterns until the resulting episode satisfies the minimal occurrence ratio threshold *min_por*.

FIND_DOMINANT_EPISODE, our algorithm for finding the dominant episode of a frequent cohesive itemset, is shown in Algorithm 6. Three parameters must be provided: a cohesive itemset X, a set of sequential patterns *sps* that is computed using FIND_SEQUENTIAL_PATTERNS, and the minimal occurrence ratio threshold *min_por*. We start by computing the absolute minimal support required (line 1). Then we sort the sequential patterns on descending support. Next we initialise the candidate episode with all nodes of itemset X, and no edges.

We then start our main loop (lines 5 to 23) by generating candidate episodes based on the intersection of the top k total orders (or sequential patterns). For each sequential pattern, we compute the transitive closure of the total order. For the first sequential pattern we then initialise the candidate episode with this set of edges (line 14). For the remaining sequential patterns we iteratively take the intersection (line 16) with the previous partial order, thereby keeping only the edges between items that hold for top-2, top-3 and finally top-k sequential patterns. As we remove edges, based on the intersection, the occurrence ratio of the resulting episode grows (or remains unchanged). We return the candidate episode if its occurrence ratio has reached the required minimal occurrence ratio (line 22). Note that if the set of edges becomes empty, we break the main loop (line 18) and return the itemset itself (an episode that imposes no order at all on its events).

Algorithm 6: FIND_DOMINANT_EPISODE(X, sps, min_por) finds dominant episode based on cohesive itemset X

```
1  min_sup_G ← ⌈|N(X)| × min_por⌉;
2  sort sps descending on occurrence ratio;
3  G ← G(X, ∅);
4  support_G ← 0;
   /* Main loop: generates candidate episodes by iteratively taking the intersection of
      total orders of the top k sequential patterns */
5  for k = 1; k ≤ |sps|; k = k + 1 do
6  |   sp ← sps[k];
7  |   G_sp ← G(X, ∅);
   |   /* compute transitive closure current total order */
8  |   for i ← 1 to |sp| − 1 do
9  |   |   for j ← i + 1 to |sp|) do
10 |   |   |   E(G_sp) ← E(G_sp) ∪ ⟨sp[i], sp[j]⟩;
11 |   |   end
12 |   end
13 |   if E(G) = ∅ then
14 |   |   E(G) ← E(G_sp);
15 |   else
16 |   |   E(G) ← E(G) ∩ E(G_sp);
17 |   end
18 |   if E(G) = ∅ then
19 |   |   break;
20 |   support_G ← COMPUTE_SUPPORT_EPISODE(G);
21 |   if support_G ≥ min_sup_G then
22 |   |   break;
23 |   end
24 end
25 return ⟨G, support_G⟩;
```

To compute the support of an episode based on minimal windows of an itemset X, we use the COMPUTE_SUPPORT_EPISODE procedure shown in Algorithm 7. We first loop over each occurrence (or minimal window) and then over each minimal window instance. As discussed in the previous section, it is important to loop over multiple minimal window instances since there can be more than one minimal window instances for a single occurrence of an item. We optimise our computation in two cases: When any edge is not covered by

Algorithm 7: COMPUTE_SUPPORT_EPISODE(X, G) compute support of candidate episode in sequence, based on minimal windows of itemset X

```
1  ⟨smw, min_windows_seq⟩ ← SUM_MIN_WINS_seq(X, ∅);
2  support_G ← 0;
3  for win ∈ min_windows_seq do
       /* Test if partial order holds in current minimal window */
4      covers_window ← false;
5      for win_ins ∈ win do
6          covers_instance ← true;
7          positions ← {⟨i, t⟩| i ∈ X ∧ win_ins.min ≤ t ≤ win_ins.max};
8          for ⟨i1, i2⟩ ∈ E(G) do
9              covers_edge ← ∃⟨i1, t1⟩, ⟨i2, t2⟩ ∈ positions : t1 < t2;
10             if not covers_edge then
11                 covers_instance ← false;
12                 break;
13             end
14         end
15         if covers_instance then
16             covers_window ← true;
17             break;
18         end
19     end
20     if covers_window then
21         support_G ← support_G + 1;
22     end
23 end
24 return support_G;
```

the current instance (line 12) we do not check remaining edges for the current instance, and when an instance is covered we do not check other instances (line 17).

3.5.1 Example

In order to illustrate FIND_DOMINANT_EPISODE we provide the following example. Suppose we have a cohesive itemset $X = \{a, b, c\}$ and input sequence $abc \ldots abc \ldots acb$. FIND_SEQUENTIAL_PATTERNS(X) finds two sequential patterns: $sp_1 = abc$ occurs 6 times, and $sp_2 = acb$ occurs 3 times. Next we execute FIND_DOMINANT_EPISODE$(X, sps = \{abc : 6, acb : 3\}, min_por = 0.8)$. We first compute the transitive closure $G_{sp_1} = G(X, \{a \to b, b \to c, a \to c\})$ which is our first candidate G_1. $support(G_1) = 6$ which is less than 0.8×9. Next we compute $G_{sp_2} = G(X, \{a \to c, c \to b, a \to b\}$ and compute the intersection to get our second candidate $G_2 = G(X, \{a \to b, b \to c, a \to c\} \cap \{a \to c, c \to b, a \to b\}) = G(X, \{a \to b, a \to c\})$. We find that $support(G_2) = 9$, and we have thus found our dominant episode, in which a occurs before b and c, but no order is imposed between b and c, with $occ\_ratio_{po}(G_2) = 1$.

3.6 Association Rules

We conclude this section with an algorithm for *efficiently* discovering *confident association rules* based on cohesive itemsets. Before presenting the algorithm, we introduce a theorem

that we use to compute the confidence of rules $Y \Rightarrow X \setminus Y$, with $Y \subset X$ and $|Y| \geq 2$, without additional dataset scans.

3.6.1 Efficiently computing confidence

When discovering a cohesive itemset X, we need to compute the exact minimal window $W_t(X)$ containing X for each time stamp t at which an item $x \in X$ occurs. In fact, we compute the sum of all such windows for each $x \in X$ in Sum_Min_Wins, before adding them up into the overall sum needed to compute $C(X)$. 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. Formally, the confidence of such a rule is equal to

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

where

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

and these are precisely the sums of windows we have already computed when discovering itemset X itself. We now show that, in practice, it is sufficient to limit our computations to rules of precisely this form (i.e., rules where the antecedent consists of a single item), as the confidence of all rules $Y \Rightarrow Z$, with $|Y| \geq 2$, can be derived from the confidence values of rules of this form.

Theorem 2 *Given a frequent cohesive itemset X and its two disjunct subsets Y and Z (i.e., $X = Y \cup Z$ and $Y \cap Z = \emptyset$), such that $|Y| \geq 2$ and $|Z| \geq 1$, it holds that*

$$c(Y \Rightarrow Z) = \frac{|N(Y)|}{\sum_{y \in Y} \frac{|N(\{y\})|}{c(\{y\} \Rightarrow Z \cup Y \setminus \{y\})}}.$$

Proof We begin the proof by noting that

$$\sum_{t \in N(Y)} W_t(Y \cup Z) = \sum_{y \in Y} \sum_{t \in N(\{y\})} W_t(Y \cup Z).$$

A trivial mathematical property tells us that the sum of some numbers is equal to their average multiplied by their quantity, and therefore

$$\sum_{t \in N(\{y\})} W_t(Y \cup Z) = \overline{W}(\{y\}, Z \cup Y \setminus \{y\}) |N(\{y\})|.$$

As a result, we can conclude that

$$\overline{W}(Y, Z) = \frac{\sum_{t \in N(Y)} W_t(Y \cup Z)}{|N(Y)|} = \frac{\sum_{y \in Y} \overline{W}(\{y\}, Z \cup Y \setminus \{y\}) |N(\{y\})|}{|N(Y)|},$$

which in turn implies that

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

Meanwhile, from the definition of confidence, we can derive that

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

and therefore it holds that

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

from which it directly follows that

$$c(Y \Rightarrow Z) = \frac{|N(Y)|}{\sum_{y \in Y} \frac{|N(\{y\})|}{c(\{y\} \Rightarrow Z \cup Y \setminus \{y\})}}.$$

□

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 $Y \subset X$ and $|Y| \geq 2$, without further dataset scans.

3.6.2 Finding association rules

The algorithm for discovering association rules is shown in Algorithm 8. First, we compute the confidence of all rules where the left-hand-side consists of a single item, and cache this result in memory (lines 1 to 12). Then we generate the powerset of X (line 14) and for each proper subset Y of X we compute the confidence based on Theorem 2 (lines 15 to 21). We return all rules that exceed the *min_conf* threshold (line 23). Note that, unlike traditional approaches, which first find all frequent itemsets and only then start generating association rules, we generate rules in parallel with the cohesive itemsets as shown in Algorithm 2.

Finally, remark that the pseudocode of Algorithms 5, 7 and 8 begins by computing the minimal windows of itemset X. While this is included for the sake of formal completeness, this line is not actually executed at this point. These minimal windows are computed already when evaluating $C_{max}(X, Y)$ in line 1 of Algorithm 2, and are then stored and reused later in Algorithms 5, 7 and 8.

3.7 Parameter tuning and top-k mining

Given the exponential nature of the itemset candidate space, FCI_{SEQ} can take a very long time to complete, depending on the parameters. It is therefore crucial to set the parameters sensibly. This is especially the case for the cohesion threshold (*min_coh*), which determines whether parts of the search space can be pruned or not. Here we provide some insight into how a user could come up with a good parameter setting.

First, we remark that setting *min_sup* might be done by a domain expert, for example by looking at the possibly long tail of infrequent items, and deciding if itemsets consisting of these infrequent items are worth exploring. The *max_size* parameter is optional, since normally no dataset will contain any cohesive patterns longer than a certain size due to data characteristics (we show this experimentally in Sect. 4.4), but if a user is only interested in shorter patterns, this parameter can be set according to personal choice (it can also be used to reduce runtimes, if necessary).

Algorithm 8: FIND_RULES(X, *min_conf*) returns all confident association rules generated from itemset X

```
⟨smw, min_windows⟩ ← SUM_MIN_WINS(X, ∅);
/* Compute confidence of rules i → X \ {i} where i is a single item. */
conf_items ← ∅;
for i ∈ X do
    positions_i ← {t | ⟨i, t⟩ ∈ S};
    smw_i ← 0;
    for win ∈ min_windows do
        if win.pos ∈ positions_i then
            smw_i ← smw_i + win.width;
        end
    end
    conf_items[i] ← |X| / (smw_i/|N({i})|);
end
rules ← ∅;
pset ← powerset(X);
/* Main loop that generates candidate rules of form Y → X \ Y */
for Y ∈ pset do
    if |Y| > 0 and |Y| < |X| then
        /* Compute confidence Y → X \ Y based on conf_items */
        smw_Y ← 0;
        for i ∈ Y do
            smw_Y ← smw_Y + |N({i})|/conf_items[i];
        end
        conf_Y ← |N(Y)|/smw_Y;
        if conf_Y ≥ min_conf then
            rules ← rules ∪ {Y → X \ Y};
        end
    end
end
return rules;
```

However, if no domain expert is available, we propose the following procedure for setting each parameter. The idea of the procedure is to start with parameter values that only require a short time, typically minutes, to mine patterns. New parameter settings can be explored in small steps, thereby lengthening the runtime gradually to re-run FCI_{SEQ} to find more patterns. We start with a relatively high value of *min_sup* (depending on the size of the dataset), a small value for *max_size*, e.g., 4, and a high value of *min_coh*, e.g., 0.9. or 1. This first run should execute very fast, with high potential for pruning, and a limited candidate space. After this initial run, the user can incrementally decrease *min_coh*, in steps of 0.1, until a sufficient number of patterns is found. After this step, *min_sup* can be decreased to smaller values to see if any new patterns involving less frequent items appear high in the ranking. We remark that, unlike frequent pattern mining, low values for *min_sup* do not seem to significantly increase runtime in our experiments (see Sect. 4.2). This is because patterns consisting of low-frequent items are often easily pruned, as the mean minimal window size of supersets consisting of other low-frequent and high-frequent items are often large, allowing us to effectively prune most candidates. Similarly, *max_size* can be increased to see if any new large patterns appear high in the ranking. The user can stop decreasing

min_sup and increasing *max_size* if no new interesting patterns are found or if the runtimes become prohibitive.

The two occurrence ratio thresholds, used for mining sequential patterns and episodes, should be set high, since the idea is to discover *representative* total and partial orders that capture most of the itemset's occurrences. If the occurrence ratio is low, the order of items (total or partial) can hardly be considered representative. Similarly, the confidence threshold should be set high in order to produce reliable association rules.

Finally, we remark that our algorithm can quite easily be adapted to allow the mining of *top-k* most cohesive patterns without the need to set *min_coh* in advance. To do this, we could maintain a *heap* of patterns during depth-first search. We would only add patterns to the heap if fewer than k patterns are in the heap, or if the cohesion of the current candidate pattern is higher than the minimal cohesion of a pattern currently in the heap (in which case the pattern with minimal cohesion would be removed from the heap). After k candidates are added, we can use the minimal value of cohesion in the heap of current candidates as a dynamic value for *min_coh* and use it for pruning using $C_{max}(X, Y)$ as before. As more patterns are discovered, the lowest value for cohesion in the heap increases, thereby pruning more candidates. Moreover, using this dynamic value as an *increasing* threshold for the upper bound would satisfy the invariant that at each step all pruned candidates are safely pruned using $C_{max}(X, Y) < min\_heap$, and thus have a lower cohesion than the pattern with the minimal cohesion the in final top-k set of patterns.

4 Experiments

In this section we compare our method with related state-of-the-art mining algorithms that take a *single* event sequence as input — WINEPI, LAXMAN[1], MARBLES$_W$ and Compact Minimal Windows (CMW) (Tatti 2014). As discussed in Sect. 1, these algorithms use a variety of frequency-based quality measures to evaluate the patterns, or, in the case of CMW, re-rank the output according to the difference, or leverage, between actual and expected minimal window lengths.

Since the available implementations[2,3] were made with the goal of discovering partially ordered episodes, we had to post-process the output in order to filter out only itemsets. Additionally, in some cases, we had to slightly amend the implementations to generate not only closed, but all frequent patterns. Therefore, making any kind of runtime comparisons would be unfair on these methods, since general episode mining requires the generation of many more candidate patterns than itemset mining. Consequently, in this section we limit ourselves to a qualitative analysis of the output.

In Sect. 4.1 we use a Hidden Markov Model-based generator (Zimmermann 2014) to create several *synthetic* data sets[4]. We use this generator to reproduce the benchmark study of Zimmermann (2014) thereby creating synthetic sequences and embedded patterns under

[1]The algorithm was given no name by its authors.

[2]The implementations of WINEPI, LAXMAN and MARBLES$_W$ are available at http://users.ics.aalto.fi/ntatti/software/closedepisodeminer.zip.

[3]The implementation of CMW was kindly provided by the author, but is not publicly available.

[4]The implementation of the generator is available at https://zimmermanna.users.greyc.fr/software.html

different assumptions. We then compare the performance of the state-of-the-art methods with FCI_{SEQ}, by reporting the rank of the discovered embedded patterns. In Sect. 4.2 we use different *real-world* datasets and compare the top-k episodes of different state-of-the-art methods from a quality perspective. In Sect. 4.3 we compare the mining of *association rules* on real-world datasets. In Sect. 4.4 we end the experimental section with a *performance* analysis of FCI_{SEQ}. We remark that our implementation, datasets and experimental scripts are all publicly available[5].

4.1 Synthetic Data

4.1.1 Varying noise probability

For the first experiment we investigate the effect noise probability has on discovering a single embedded pattern in a synthetic sequence. The synthetic sequence has a length of 5000 events consisting of 20 different events. The maximum delay between two events is between 0 and 20. We embed a single serial episode with 4 elements. We then generate 10 variations, while varying the chance of a random event between each pattern embedding, that is p, between 0.05 and 0.95 in steps of 0.1. More details can be found in the original work of Zimmermann (2014).

For FCI_{SEQ} we set *max_size* to 5 and *min_coh* to a value lower than the cohesion of the embedded pattern. For WINEPI, LAXMAN, $\text{MARBLES}_{\text{W}}$ and CMW we set the *window* to be large enough for any pattern embedding, that is 50. For WINEPI, LAXMAN, $\text{MARBLES}_{\text{W}}$ and CMW we set the *threshold* small enough that at least thousands of patterns are reported in a reasonable time, that is $t = 80$ for WINEPI, $t = 5$ for LAXMAN and CMW (which uses LAXMAN for episode generation), and $t = 1$ for $\text{MARBLES}_{\text{W}}$. In addition, we set *alpha*, that controls the scaling of the ranking based on compactness of window sizes, to 0.5 for CMW, and split the sequence in two equal parts: one for training and candidate generation, and one for testing if patterns are significant as done in the original paper (Tatti 2014).

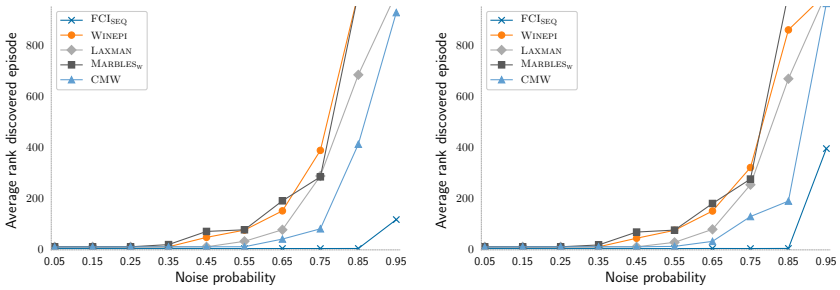
We run two variations: in the first we assume that noise events and delays between both pattern and noise events are sampled using a uniform distribution, and for the second variation we generate noise events based on the Poisson distribution. The average rank of the discovered pattern within a synthetic sequence with varying noise probability is shown in Fig. 2. Note that if an embedded pattern is not ranked within the top-1000 or is not found at all, we cap the rank to 1000, which is why we do not show these scores on the plots, as they would be misleading).

Due to the inherent randomness of the Zimmermann generator, we run each experiment 5 times, and report the rank of the discovered episode(s) averaged over these 5 runs. We see that FCI_{SEQ} consistently outperforms the other methods and always ranks the embedded patterns very highly, while other methods underperform in the presence of noise.

4.1.2 Varying the size of alphabet

In the second experiment, we investigate the effect of an increasing number of event types, denoted by M. We vary M between 4 and 40 in steps of 4. The synthetic sequences have a

[5]The implementation of FCI_{SEQ} is available at https://bitbucket.org/len_feremans/fci_public

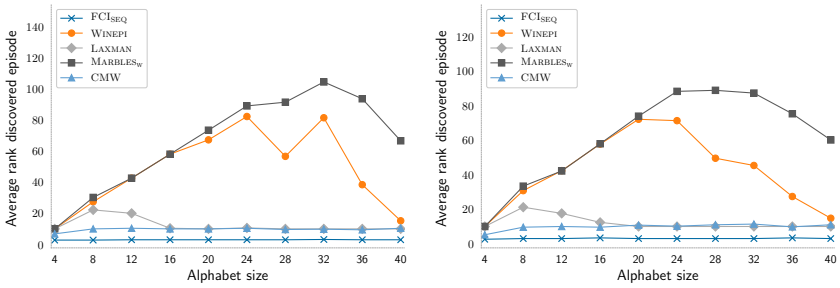


(a) (b)

Fig. 2 Impact of varying noise on the discovery of a single episode in synthetic data. (a) Uniformly distributed noise. (b) Poisson-distributed noise.

length of 5000 events, and the maximum delay between two events is uniformly between 0 and 20. We also embed a single serial episode with 4 elements. The uniform noise probability is fixed to 0.5. For mining the patterns we use the same parameters as in our previous experiment.

The average rank of the embedded pattern for varying alphabet size is shown in Fig. 3. For a second variation of this experiment, we generate noise that is distributed using the Poisson distribution. Once again, we see that our algorithm outperformed the other methods on every single experiment.



(a) (b)

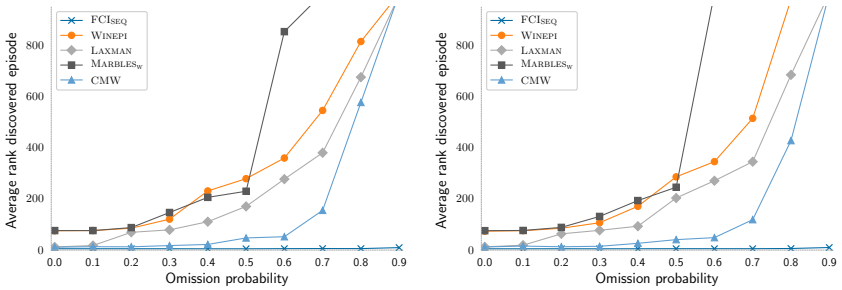
Fig. 3 Impact of varying alphabet size on the discovery of a single episode in synthetic data. (a) Uniformly distributed noise. (b) Poisson-distributed noise.

4.1.3 Varying probability of omissions

In the third experiment, we vary the probability that a source event of the embedded pattern is *not* included, in order to mimic this type of failure that occurs in real-world datasets. We vary the failure probability o between 0 and 0.9 in steps of 0.1. As before, the synthetic

sequences have a length of 5000 events consisting of 20 different events, the maximum delay between two events is distributed uniformly between 0 and 20, we embed a single serial episode with 4 elements, and the uniform noise probability is 0.5. For mining patterns we use the same parameters as in our previous experiments.

The average rank of the pattern embedding for varying the probability of omissions is shown in Fig. 4. Here, too, we perform a second variation of the experiment where the noise is distributed using the Poisson distribution. We see that our algorithm is not affected much by the omission probability, while all other methods begin to struggle as the omission probability rises.



(a) (b)

Fig. 4 Impact of varying the probability of omitting events of source episodes, on the discovery of a single episode in synthetic data. (a) Uniformly distributed noise. (b) Poisson-distributed noise.

4.1.4 Varying maximum time delay

In the fourth experiment we study the effect of a larger time delay between consecutive events, and we therefore vary the maximal delay g between 20 and 100 in steps of 10. We vary this delay both for regular sequence events, and for the events belonging to pattern embeddings. For other generator parameters we use the same settings as before, that is a sequence length of 5000, alphabet size of 20, noise probability of 0.5, omission probability of 0 and a single serial episode with 4 elements. We also use the same parameters for mining as above, except for *window* which we increase to 100.

The average rank of the pattern embedding for varying the maximal time delay is shown in Fig. 5. For a second variation we generate noise that is distributed using the Poisson distribution. As discussed in Zimmermann (2014), we also consider a third variant where the maximum delay between two consecutive events of an embedded pattern is not constrained by g. As in the previous experiments, FCI$_{\text{SEQ}}$ consistently discovers the embedded pattern in the top-10. Unlike the previous experiments the performance of CMW is also comparable.

4.1.5 Varying the number of patterns

In our fifth and final variation we study the effect of increasing the number of patterns, and we therefore vary the number of patterns n between 1 and 5. All other parameters (both

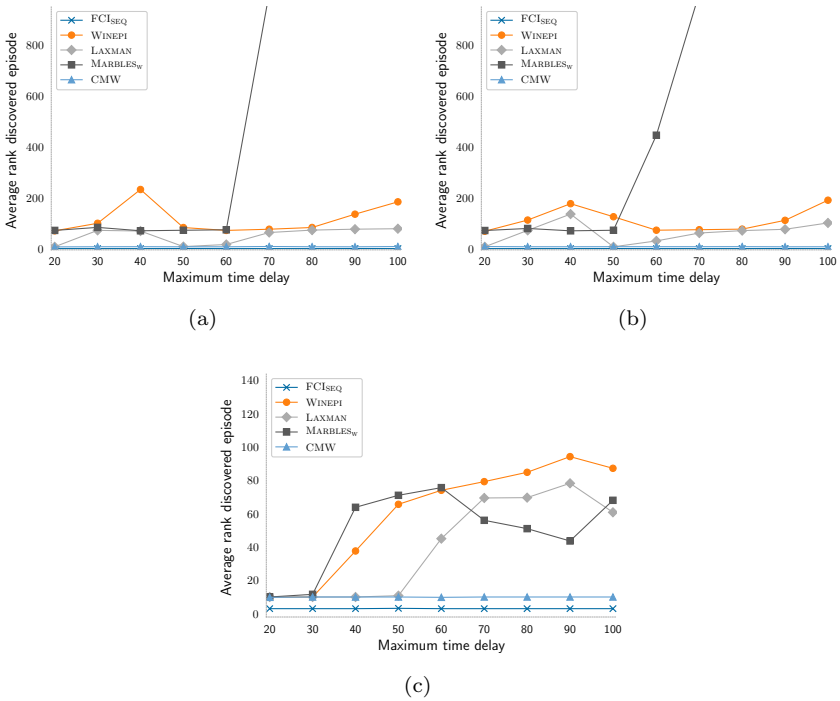


Fig. 5 Impact of varying maximum time delay on the discovery of a single episode in synthetic data. (a) Uniformly distributed noise. (b) Poisson-distributed noise. (c) Uniformly distributed noise, with no maximum delay between pattern events.

for sequence generation and pattern mining) were set as in the previous experiments. The average rank for the discovered patterns is shown in Fig. 6. In addition to experimenting with both uniformly and Poisson-distributed noise, we also consider two other variations (assuming uniformly distributed noise) where embedded episodes are interleaved (occurrences of two episodes may overlap) and events are shared (two embedded patterns may contain the same event) (Zimmermann 2014). We see that the number of different embedded patterns affects the performance of our algorithm more adversely than the parameters discussed previously, but FCI_{SEQ} still outperforms all other methods by quite a margin.

4.1.6 Discussion

When looking at the results of the above experiments we can conclude that, on the Zimmermann benchmark, FCI_{SEQ} clearly outperforms the state-of-the-art methods. Cohesion seems to be a more robust measure to a variety of artificially induced types of noise that occur in real-world settings. Compared to the frequency-based methods, this is not surprising, since frequency is a poor proxy for interestingness. While CMW seems to perform generally better than frequency-based methods, it also seems far less robust to the different types of variation than FCI_{SEQ}.

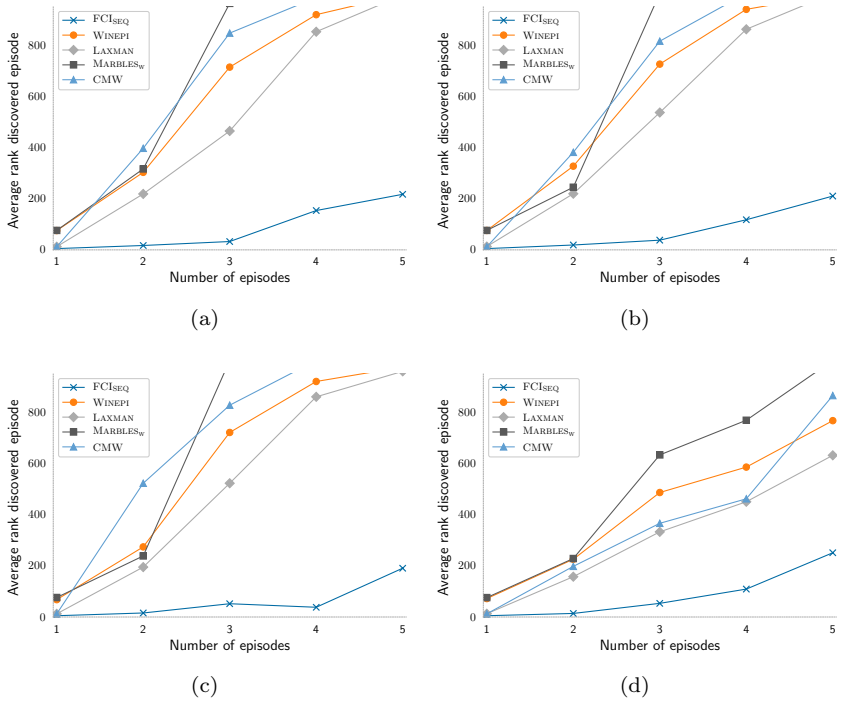


Fig. 6 Impact of varying the number of episodes, and the discovery of them in synthetic data. (a) Uniformly distributed noise. (b) Poisson-distributed noise. (c) Interleaved embedded episodes. (d) Embedded episodes with shared events.

While it would be tempting to conclude FCI_{SEQ} is superior on any dataset, we also acknowledge that our method has drawbacks. More specifically, when we consider the different parameters of the Zimmermann generator, we see two parameters, for which FCI_{SEQ} would have trouble with respect to recovering patterns. The first parameter r controls if event types are *repeated* in an embedded pattern. Since FCI_{SEQ} first mines itemsets, we do not generate any patterns containing repeating items. A consequent issue is that we are unable to differentiate different, more complex, patterns, if the alphabet size is very small, e.g., given a DNA sequence with only 4 distinct items. A second parameter s controls if events are *shared* between multiple source episodes. Since we always consider the minimal window length at each item occurrence, having two (or more) embedded patterns share the same item will result in some occurrences of the shared item being far from the occurrences of other items in both patterns, and therefore in smaller values for cohesion for both patterns. As a consequence, the subset of the embedded patterns without the shared item would have a larger value for cohesion, and it is possible that the full pattern would not be recovered.

4.2 Real-world data

4.2.1 Datasets

We selected two text datasets in which the discovered patterns can be easily discussed and explained. *Species* contains the complete text of *On the Origin of Species by Means of Natural Selection* by Charles Darwin[6]. *Trump* contains all tweets of president Trump[7] between January 1, 2017 and March 1, 2018. We preprocessed both sequences using the Porter Stemmer, removed stop words, transformed words to lower case, and removed any special characters. Since the *Trump* dataset consists of many short sequences, we appended all tweets to create a single long sequence. In addition we also removed HTML content, such as URLs and entities. After preprocessing, the *Species* dataset has a sequence length of $|S| = 85\,447$ and contains 5 547 distinct items. For *Trump* the *single* sequence length is $|S| = 27\,837$ and contains 4 061 distinct items.

4.2.2 Cohesive Itemsets

For each of the existing methods, we set the frequency threshold low enough in order to generate thousands of patterns. For FCI_{SEQ}, we did the same with the cohesion threshold. We then sorted the output on the respective quality measures — the sliding window frequency for WINEPI, the non-overlapping minimal window frequency for LAXMAN, the weighted window frequency for $\text{MARBLES}_{\text{W}}$, the leverage-based score for CMW, and cohesion for FCI_{SEQ}. For FCI_{SEQ}, we used the sum of support of individual items making up an itemset as the second criterion for ranking. For other methods pattern size is used as a second criterion for ranking. Finally, we use alphabetical order of patterns to break ties in all four methods. The frequency threshold was set to 30 for WINEPI, 5 for LAXMAN, and 1 for $\text{MARBLES}_{\text{W}}$ in both datasets, with the sliding window size set to 15. For CMW we set *alpha* to 0.5 and split the sequence in two: the first half is used for discovering patterns while the second half is used for testing whether patterns are significant (Tatti 2014). CMW uses the frequent episodes produced by LAXMAN as input. We run FCI_{SEQ} with the cohesion threshold set to 0.015 for *Species* and 0.02 for *Trump*, and we set the support threshold to 5 for both datasets. Since none of the state-of-the-art methods produced any itemsets consisting of more than 6 items, we set the *max_size* parameter to 6 to reduce runtimes.

The top 5 patterns discovered by the different methods are shown in Table 2. We can see that there are clear differences between the patterns discovered by FCI_{SEQ} and CMW, and those discovered by the frequency-based methods, which produce very similar results. First of all, the patterns ranked first and second in our output for the *Species* dataset are of size 3, which would be theoretically impossible for WINEPI and $\text{MARBLES}_{\text{W}}$, and highly unlikely for LAXMAN, since all three use anti-monotonic quality measures. Second, we observe that the patterns we discover are in fact quite rare in the dataset, but they are very strong, since all occurrences of these patterns are highly cohesive. Concretely, the phrase *tierra del fuego* occurs seven times in the book, and none of these words occur anywhere else in the book. The value of this pattern is therefore quite clear — if we encounter any one of these

[6] http://www.gutenberg.org/

[7] http://www.trumptwitterarchive.com/

three words, we can be certain that the other two can be found nearby. A similar argument holds for the expression "Natura non facit saltus", Latin for "nature does not make jumps", of which "non" is considered a stopword, and removed during preprocessing. CMW also discovers interesting, albeit *different* itemsets. CMW ranks *tierra del fuego* 32$^{\text{nd}}$ and *natura facit saltus* 27$^{\text{th}}$.

Table 2 Top 5 itemsets discovered by the different methods.

	FCI_{SEQ}	Winepi	Laxman	$\text{Marbles}_{\text{W}}$	CMW
Species	del, fuego, tierra	natur, select	natur, select	natur, select	absenc, island, mammal, ocean, terrestri
	facit, natura, saltum	speci, varieti	form, speci	speci, varieti	altern, glacial, north, period, south
	del, fuego	form, speci	speci, varieti	distinct, speci	bat, island, mammal, ocean, speci, terrestri
	del, tierra	natur, speci	natur, speci	form, speci	bat, island, mammal, ocean, terrestri
	facit, saltum	distinct, speci	distinct, speci	condit, life	cross, fertil, hybrid, mongrel, offspr, varieti
Trump	puerto, rico	fake, new	fake, new	fake, new	ab, japan, minist, prime
	hunt, witch	cut, tax	cut, tax	cut, tax	high, hit, market, stock, time
	harbor, pearl	america, great	america, great	america, great	abc, cnn, fake, nbc, new
	lago, mar	great, make	great, peopl	america, make	alabama, big, luther, strang, vote
	arabia, saudi	america, make	great, make	great, make	lowest, market, stock, unemploy, year

The top pattern of CMW relates to chapter XIII in the book, with a subsection titled "absence of bathracians and terrestrial mammals on oceanic islands". FCI_{SEQ} is unable to find this pattern, as the cohesion is very low overall since *absence, terrestri, mammal, ocean, island* co-occur infrequently together in the book. If we would however segment this very long book, and run FCI_{SEQ} on each individual chapter, the cohesion, *local* to chapter XIII, would also be high for this itemset.

In the *Trump* dataset the top-5 patterns produced by FCI_{SEQ} all have a cohesion of 1, which indicates that they always occur next to each other in tweets, in this case in 27, 23, 8, 7 and 7 tweets, respectively. Like *tierra del fuego*, these are examples of itemsets that are highly correlated, but occur too infrequently to rank highly in existing state-of-the-art methods. For example *pearl harbor*, *saudi arabia* and *mar (a) lago* do not occur in the top-300 of the existing state-of-the art methods.

We conclude that in order to find less frequent, but strongly correlated patterns such as *tierra del fuego* or *mar (a) lago* with existing state-of-the-art methods, the user would need to wait a long time before a huge output was generated, and would then need to trawl through thousands of itemsets in the hope of finding them. FCI_{SEQ}, on the other hand, ranks them at the very top. From the perspective of the frequency-based methods, top patterns typically consist of words that occur very frequently in the dataset, regardless of whether the occurrences of the words making up the itemset are correlated or not. The top patterns

reported by CMW also seem very interesting, but quite different from those produced by FCI_{SEQ}. A disadvantage of CMW is that only half of the sequence is available for training, causing the method to miss out on any patterns that only occur in the test part. A second disadvantage are the so-called *free-rider episodes* where an item added independently to an existing high-leverage episode also has a high score. For a more complete picture, we provide the top-25 itemsets discovered by all five methods in both datasets in the Appendix.

While the top patterns are different, there is still some overlap between the output generated by the various methods. For example, the pattern *natur(al) select(ion)*, ranked first in the *Species* dataset by the existing methods, was ranked 16$^{\text{th}}$ by FCI_{SEQ}, which shows that our method is also capable of discovering very frequent patterns, as long as they are also cohesive. Table 3 shows the size of the overlap between the itemsets discovered by FCI_{SEQ} and those discovered by the other methods. We compute the size of the overlap within the top k itemsets for each method, for varying values of k.

Table 3 Overlap in the top k itemsets discovered by FCI_{SEQ} and other methods.

	k	Winepi	Laxman	$\text{Marbles}_{\text{W}}$	CMW
Species	100	12	13	11	2
	500	28	35	25	2
	1 000	45	53	38	4
Trump	100	15	14	16	0
	500	37	47	34	18
	1 000	54	67	50	30

4.2.3 Representative Sequential Patterns

For evaluating representative sequential patterns we show the top-5 sequential patterns discovered by FCI_{SEQ} with *findSeq* set to *true*, *min_coh* = 0.015 (0.02 for *Trump*), *min_sup* = 5 and *max_size* = 6, and compare the discovered representative sequential patterns with total orders reported by state-of-the-art methods. We additionally set the occurrence ratio threshold for sequential patterns, *min_or*, to 0.7, and thus only report sequential patterns for itemsets where the sequential pattern occurs in at least 70% of itemset occurrences. The results for FCI_{SEQ} for *Species* are shown in Table 4. The patterns are first sorted on cohesion, then on occurrence ratio, then on support, and finally alphabetically, if all other measures are equal. These results show that the items making up the most cohesive itemsets always occur in a specific order. Therefore, when this is the case, the representative sequential patterns form a more informative way to represent the most interesting patterns.

For FCI_{SEQ}, Winepi, Laxman, $\text{Marbles}_{\text{W}}$ and CMW the top-5 sequential patterns are shown in Table 5 for both datasets. Due to the *min_or* threshold of 0.7, we only report a representative sequential pattern for those itemsets that actually have one. In fact, out of the 1 130 discovered cohesive itemsets in *Species*, only 21 have a representative sequential pattern, for all others there is no single specific order that would be representative for the occurrences of the itemset. In *Trump*, only 22 out of 16 372 itemsets had a representative sequential pattern. For both *Species* and *Trump*, the top-5 sequential patterns had an

Table 4 Top 5 sequential patterns discovered by FCI_{SEQ} for *Species* with values for both *cohesion* and *support* of the underlying cohesive itemset X, and the occ_ratio_{se} of the representative sequential pattern sp for X.

$C(X)$	$sup(X)$	occ_ratio_{se}	sp
1.0	21	1.0	tierra, del, fuego
1.0	18	1.0	natura, facit, saltum
1.0	14	1.0	del, fuego
1.0	14	1.0	tierra, del
1.0	12	1.0	facit, saltum

occ_ratio_{se} of 1, indicating that *all* occurrences of the underlying itemset came in the order defined by the sequential pattern, clearly demonstrating the usefulness of outputting these patterns. Conversely, for itemsets where the order is not important, our method outputs no sequential pattern at all. Unlike FCI_{SEQ}, the frequency-based methods again rank many spurious patterns highly. Note, for example, that all three methods output both *variety speci(es)* and *speci(es) variety* in the top-10. This clearly demonstrates that, while the two words often co-occur due to them both being very frequent, there is no sequential relationship between them. Furthermore, similarly to itemsets, we remark that the top-1000 of all frequency-based methods contained very few ($< 5\%$) sequential patterns consisting of more than two items, and none of them ranked sequential patterns *tierra del fuego* and *natura (non) facit saltum* in the top-1000. On the other hand, the most interesting sequential patterns found by the frequency-based methods are ranked highly by FCI_{SEQ}, too. For example, *natur(al) select(ion)*, *fake new(s)* and *tax cut(s)* can all be found in our top-20 sequential patterns for the respective dataset.

For sequential patterns, we see that the re-ranking of CMW for sequential patterns candidates generated by Laxman, produces a more interesting set of patterns. The top sequential pattern *struggl(e) (for) exist(ence) geometr(ic) ratio (of) increas(e)* appears in chapter III. FCI_{FCI} does not rank this pattern highly, since *struggle* also co-occurs often with *life*, or on its own as verb, causing large minimal windows for these occurrences. Likewise, the word *increase* (or variants like *increasing* with the same stem) are very common, and occurs frequently in chapter III, without any instance of the other words nearby. Accordingly, the interestingness of this pattern is very low concerning our proposed cohesion measure. In *Species*, CMW also reports both *tierra del fuego* and *natura (non) facit saltum* in the top-25. For *Trump*, the situation is different, in that some top patterns found by FCI_{SEQ} are ranked rather low by CMW, e.g. *pearl harbor* is ranked 9921^{st}.

4.2.4 Dominant Episodes

To discover dominant episodes we run FCI_{SEQ} with *findEpi* set to *true*, $min\_coh = 0.015$ (0.02 for *Trump*), $min\_sup = 5$ and $max\_size = 5$. Furthermore, we set the minimum occurrence ratio for episodes, *min_por*, to 0.7, and thus only report episodes $G = (V(G), E(G))$ for itemsets X, where $V(G) = X$ and the partial order defined by $E(G)$ occurs in at least 70% of itemset occurrences. In *Species*, the dominant episode for 658 out of 1 130 cohesive itemsets was either a sequential pattern (a total order) or the itemset itself (no order). For the remaining 472 itemsets, the dominant episode was a partial order. Note that for item-

Table 5 Top 5 sequential patterns discovered by the different methods.

	FCI_{SEQ}	Winepi	Laxman	$\text{Marbles}_{\text{W}}$	CMW
	tierra, del, fuego	natur, select	natur, select	natur, select	struggl, exist, geometr, ratio, increas
	natura, facit, saltum	varieti, speci	varieti, speci	distinct, speci	variat, superven, earli, ag, inherit
Species	del, fuego	speci, varieti	speci, form	varieti, speci	form, life, chang, simultan, world
	tierra, del	distinct, speci	speci, varieti	condit, life	variat, superven, earli, inherit, ag
	facit, saltum	speci, form	form, speci	speci, varieti	steril, speci, cross, hybrid, offspr
	puerto, rico	fake, new	fake, new	fake, new	stock, market, hit, time, high
	witch, hunt	tax, cut	tax, cut	tax, cut	job, stock, market, time, high
Trump	pearl, harbor	america, great	america, great	america, great	greatest, witch, hunt, histori
	mar, lago	make, america	make, great	make, america	stock, market, hit, high
	saudi, arabia	make, great	make, america	unit, state	presid, moon, south, korea

sets of size 2, the only possible episodes represent either an itemset or a sequential pattern. As a result, all the partial orders of interest consisted of three or more items. We therefore exclude itemsets and sequential patterns from the episode output.

The results of FCI_{SEQ} for *Species* are shown in Table 6. Note that the episodes are ranked on cohesion of the underlying itemset, after filtering. The episodes containing $\{hexagon(al), prism, pyramid, rhomb, sphere\}$ are due to a section in the book discussing the making of the honeycomb structure of bees, while the episode containing $\{leptali(s), ithomia, mimick\}$ is due to a section where the similarities of these two butterfly species are discussed. While the specific semantics are less of interest in text datasets, what is interesting is that these are partial orders that hold for more than 70% of occurrences.

Episodes reported by state-of-the-art methods on *Species* are shown in Table 7. Here, too, we omit episodes defining either a total order or no order at all. While the episode output of FCI_{SEQ} and CMW provides additional information about partial orders present in the occurrences of itemsets that have no representative sequential pattern, the three frequency-based methods produce various combinations of very frequent items, which is not very informative. Finally, note that by using the *min_por* threshold, we ensure that we produce a single dominant episode per underlying itemset, which is representative of the occurrences of that itemset. Other methods produce many partial orders for the same itemset which can result not only in spurious patterns being discovered, but also in an undesirably large output size (e.g., for CMW the top-100 episodes *all* consist of variations (with different edges) of the two underlying itemsets of episodes shown in Table 7).

Table 6 Five dominant episodes discovered by FCI_{SEQ} for *Species* with values for *cohesion*, *support* and occ_ratio_{po} for dominant episodes G.

$C(X)$	$sup(X)$	occ_ratio_{po}	G
0.096	20	0.75	leptali → ithomia, mimick
0.022	38	0.92	hexagon → prism, sphere
0.021	44	0.86	prism → pyramid, rhomb; hexagon, sphere
0.020	4666	0.76	gener → speci, natur, form, distinct
0.020	29	0.86	prism → pyramid, rhomb; hexagon

Table 7 Top 5 episodes discovered by state-of-art methods on *Species*.

Winepi	Laxman	$\text{Marbles}_{\text{W}}$	CMW
natur → select, speci	natur → select, speci	natur → select, speci	glacial → period → north; altern → south
natur → select, varieti	natur → select, case	natur → select, case	glacial → period → north; altern → south
natur → select, case	natur → select, varieti	natur → select, varieti	glacial → period → north; altern → south
close → allied, speci	select → speci, natur	close → allied, speci	glacial → period → north; altern → south
select → speci, natur	natur → speci, select	natur → select, theori	absenc → terrestri → mammal; ocean → island

4.3 Association rules

4.3.1 Text Datasets

Using FCI_{SEQ} with *findRule* set to *true*, we generate association rules for both datasets with the confidence threshold *min_conf* set to 0.7, and parameters *min_coh, max_size* and *min_sup* as defined in the previous section. We again compare rules with Winepi, $\text{Marbles}_{\text{W}}$, and with $\text{Marbles}_{\text{M}}$, which uses a confidence measure based on non-overlapping minimal windows (as defined by Laxman). CMW is only used for re-ranking episodes, not mining association rules. We set *min_conf* = 0.7 for the state-of-art methods, and the frequency threshold to 40 for Winepi, 10 for $\text{Marbles}_{\text{M}}$, and 1 for $\text{Marbles}_{\text{W}}$ in both datasets, with the sliding window size set to 15.

We slightly adjusted the underlying implementation of the state-of-art methods, namely CLOSEPI, and made the modified code publicly available in our repository. Since CLOSEPI mines *general episodes*, instead of only parallel episodes, or itemsets, and then generates rules with potentially both general episodes on the left- and right-hand side, this causes an order-of-magnitude more rules, and requires additional computing resources. Therefore, we made sure that parallel episodes, closed by a partial episode, are not removed, and instead removed all non-parallel episodes before mining association rules.

The top-5 rules for *Species* and *Trump* for all methods are shown in Table 8. Rules were ranked first on the confidence measure related to each method, and then on support of the antecedent to break ties. For *Species* and *Trump* the top-5 rules for FCI_{SEQ} have a confidence of 1.0. Most of these top rules consist of itemsets that are fully cohesive, meaning that if one of the items occurs, the others always occur next to them. Only *migrat(ation)*→ *chain* is different, since the underlying itemset is not fully cohesive, but the association rule with *migrat(ation)* as antecedent is fully confident. Interestingly, $\text{MARBLES}_{\text{W}}$ and $\text{MARBLES}_{\text{M}}$ also report rules of fully cohesive itemsets ranked in the top-5, such as *tierra del fuego* and *natura facit saltum*. Unlike itemsets ranked on frequency, the ranking on confidence produces quite different results between all methods, especially between the three different frequency-based approaches. While all methods find very interesting rules, a disadvantage of the three state-of-the-art methods is that they often rank rules with frequent items such as *speci(es), natur(al)* or *select(ion)* as the consequent very highly, when this is, in most cases, due to these items accidentally occurring in the vicinity of the antecedent, and not due to an actual association. On the other hand our method fails to discover rules such as $divis(ion), kingdom \rightarrow anim(al)$. The phrase "division of the animal kingdom" occurs about 10 times in *Species*. FCI_{SEQ} does not report the itemset or any subset or resulting rules, because when we consider all occurrences of items in the antecedent — both *kingdom* and *divis(ion)* — and then compute the average minimal window lengths, in this instance, cohesion is very low (< 0.001), or, in other words, the majority of the occurrences of the three items do not co-occur anywhere. Once again, we provide the top-25 rules of both datasets in the Appendix.

4.3.2 Character Sequence Datasets

As a second experiment we run association rule mining on text split on each character. We are interested in finding association rules between letters that are specific within each language — in this case, English, French and Dutch. We use the complete text of *David Copperfield* by Charles Dickens[6] in English and translations in French and Dutch. We removed special characters, transformed words to lower case, tokenised the text on individual characters and added the '_' symbol to denote spaces between words. We limited the three sequences to the first $|S| = 500\,000$ characters. The dictionary consists of 26 letters and '_' .

We run FCI_{SEQ} with $min\_coh = 0$, $max\_size = 4$, $minsup = 50$ and $min\_conf = 0.3$. In Table 9 we show the top-5 rules discovered for each language, split into three categories. We first show rules consisting of two letters, then rules consisting of three letters, and, finally, rules containing the space between words, or '_'. We see that some reported patterns are common in all three languages, and some patterns are discriminative for a specific language. For example $q \rightarrow u$ is a typical combination found in all three languages, as q is almost

Table 8 Top 5 rules, ranked on confidence, discovered by the different methods.

	FCI_{SEQ}	Winepi	$\text{Marbles}_{\text{M}}$	$\text{Marbles}_{\text{W}}$
	fuego, tierra → del	divis, kingdom → anim	hive → bee	divis, kingdom → anim
	del, fuego → tierra	averag, genera → speci	mivart → mr	fuego, tierra → del
Species	del, tierra → fuego	cuckoo, lai, nest → egg	case, select, structur → natur	independ, ordinari → view
	facit, natura → saltum	varieti, zone → intermedi	candol → de	natura, saltum → facit
	natura, saltum → facit	genera, present, varieti → speci	case, organ, select → natur	inherit, superven → earli
	puerto→ rico	hit, stock→ market	cut, reform→ tax	honor, minist→ prime
	rico→ puerto	high, hit, stock→ market	puerto→ rico	confer, joint→ press
Trump	hunt→ witch	bill, reform, tax→ cut	rico→ puerto	immigr, merit→ base
	witch→ hunt	biggest, cut, histori→ tax	witch→ hunt	greatest, hunt→ witch
	migrat→ chain	ab, prime→ minist	hunt→ witch	donald, proclaim→ trump

always followed by a u. A typical Dutch rule is $j \rightarrow i$, where ij is a very common combination of letters, while rule $j \rightarrow e$ is very specific for French. Rule $y \rightarrow \_$ is typical for English and Dutch, where y often occurs either at the start or at the end of a word, and rarely in the middle. The same holds for $j \rightarrow \_$ in French, where j is mostly found at the start of the word. This experiment confirms that our method finds valuable association rules, and tends not to rank spurious rules highly.

4.4 Performance Analysis

We tested the behaviour of our itemset mining algorithm when varying the cohesion, support and size thresholds on the two text datasets (*findSeq*, *findEpi* and *findRule* were all set to *false*). The results are shown in Fig. 7. As expected, we see that the number of patterns increases as the cohesion and support thresholds are lowered. In particular, when the cohesion threshold is set too low, the size of the output explodes, as even random combinations of frequent items become cohesive enough. However, as the support threshold decreases, the number of patterns stabilises, since rarer items typically only make up cohesive itemsets with each other, so only a few new patterns are added to the output (when we lower the support threshold to 2, we see another explosion as nearly the entire alphabet is considered frequent).

In all settings, it took no more than a few minutes to find tens of thousands of patterns. Note that with reasonable support and cohesion thresholds, we could even set the *max_size* parameter to ∞ without encountering prohibitive runtimes, allowing us to discover patterns

Table 9 Top 5 rules, ranked on confidence, for different types of character associations in English, French and Dutch. The input dataset is *David Copperfield*.

Category	English	French	Dutch
	$c(q \to u) = 0.984$	$c(q \to u) = 0.980$	$c(q \to u) = 0.661$
	$c(v \to e) = 0.686$	$c(j \to e) = 0.609$	$c(j \to i) = 0.625$
Two letters	$c(x \to e) = 0.616$	$c(g \to e) = 0.589$	$c(b \to e) = 0.605$
	$c(r \to e) = 0.432$	$c(d \to e) = 0.542$	$c(n \to e) = 0.603$
	$c(z \to e) = 0.416$	$c(r \to e) = 0.540$	$c(g \to e) = 0.594$
	$c(q \to i, u) = 0.524$	$c(q \to u, e) = 0.678$	$c(q \to i, u) = 0.667$
	$c(q \to u, e) = 0.520$	$c(b \to a, e) = 0.357$	$c(q \to i, n) = 0.626$
Three letters	$c(q \to t, u) = 0.399$	$c(q \to u, i) = 0.348$	$c(q \to u, n) = 0.576$
	$c(v \to a, e) = 0.362$	$c(l \to a, e) = 0.346$	$c(q \to u, e) = 0.485$
	$c(q \to i, e) = 0.357$	$c(v \to a, e) = 0.346$	$c(q \to r, e) = 0.483$
	$c(y \to \_) = 0.918$	$c(j \to \_) = 0.924$	$c(y \to \_) = 0.979$
	$c(w \to \_) = 0.863$	$c(q \to u, \_) = 0.906$	$c(q \to \_) = 0.894$
Letters near	$c(d \to \_) = 0.835$	$c(q \to \_) = 0.824$	$c(x \to \_) = 0.846$
word boundary	$c(q \to \_, u) = 0.794$	$c(d \to \_) = 0.790$	$c(z \to \_) = 0.800$
	$c(b \to \_) = 0.790$	$c(q \to u, e, \_) = 0.785$	$c(m \to \_) = 0.781$

of arbitrary size (in practice, the size of the largest pattern is limited due to the characteristics of the data, so output size stops growing at a certain point). Since the methods we compare with use a relevance window, defining how far apart two items may be in order to still be considered part of a pattern, they can never find patterns of arbitrary size. For example, using a window of size 15 implies that no pattern consisting of more than 15 items can ever be discovered.

In a second performance experiment we vary the maximum size of patterns, and report the number of candidates visited by the main DFS routine (Algorithm 2) versus the number of candidates that is *theoretically possible*. Given an alphabet of $|\Omega|$ items, the number of itemsets of length 2 up to max_size that is theoretically possible is given by $\sum_{k=2}^{max\_size} \binom{|\Omega|}{k}$. We run FCI_{SEQ} on *Species* and set min_sup to 5, resulting in $|\Omega| = 5547$ different words, and set min_coh to 0.5. In Table 10, we report the number of candidates visited versus the number of possible candidates for varying max_size. Remark that running up to $max\_size = 48$ took only 41 minutes on a laptop. We conclude that pruning on cohesion is effective in narrowing the search in an otherwise intractable search space.

The runtime of additionally mining representative sequential patterns, dominant episodes and association rules is shown in Fig. 8. Since sequential pattern, episode and association rule mining is triggered for each cohesive itemset, additional runtime costs for mining other types of patterns is relative to the number of reported cohesive itemsets. If the number of cohesive itemsets is small, such as for *Species* with 1 339 itemsets, the additional time required for finding sequential patterns or episodes is small, compared to the time for only mining cohesive itemsets. For *Trump* the number of itemsets is larger, that is 16 393, and the additional time needed for finding sequential patterns is naturally higher, and even higher for dominant episodes (note, however, that episode mining requires the execution of the sequential pattern mining phase, even if sequential patterns are not required for output). For mining rules, the additional runtime cost for mining confident rules based on cohesive itemsets is relatively small for both datasets. This is mainly due to the efficient computation

Table 10 Impact of varying *max_size* on the number of candidates enumerated (with pruning) by FCI_{SEQ} versus the theoretically possible number of candidates (without pruning). Experiment run on *Species* with parameters $min\_sup = 5$ ($|\Omega| = 5547$) and $min\_coh = 0.5$.

max_size	Number of candidates visited by FCI_{SEQ}	Theoretically possible number of candidates
8	4.8×10^6	1.3×10^{22}
16	5.0×10^6	1.3×10^{40}
24	5.4×10^6	2.4×10^{56}
32	8.0×10^6	2.8×10^{71}
40	2.3×10^7	4.4×10^{85}
48	1.3×10^8	1.3×10^{99}

of confidence, as described in Sect. 3.6. We conclude that our algorithms perform efficiently in a variety of settings.

5 Related Work

We have examined the most important related work in Sect. 1, and experimentally compared our work with the existing state-of-the-art methods in Sect. 4. Here, we place our work into the wider context of sequential pattern mining.

At the heart of most pattern mining algorithms is the need to reduce the exponential search space into a manageable subspace. When working with an anti-monotonic quality measure, such as frequency, the Apriori property can be deployed to generate candidate patterns only if some or all of their subpatterns have already proved frequent. This approach is used in both breadth-first-search (BFS) and depth-first-search (DFS) approaches, such as Apriori (Agrawal and Srikant 1994), Eclat (Zaki 2000) and FP-growth (Han et al. 2004) for itemset mining in transaction databases, GSP (Srikant and Agrawal 1996), SPADE (Zaki 2001), BIDE (Wang and Han 2004) and PrefixSpan (Pei et al. 2004) for sequential pattern mining in sequence databases, or Winepi (Mannila et al. 1997) and Marbles (Cule et al. 2014) for episode mining in event sequences.

For computational reasons, non-anti-monotonic quality measures are rarely used, or are used to re-rank the discovered patterns in a post-processing step. Tatti proposed a way to measure the significance of an episode by comparing the lengths of its occurrences to expected values of these lengths if the occurrences of the patterns' constituent items were scattered randomly (Tatti 2014). In a later work, Tatti introduced the EpiRank algorithm (2015) to re-rank episodes in a dataset consisting of multiple sequences based on *leverage* (Webb 2010). Both methods, however, use the output of an existing frequency-based episode miner (Tatti and Cule 2012), and then compute the new measures for the discovered patterns. In this way, the rare patterns, such as those discussed in Sect. 4, will once again not be found. Our FCI_{SEQ} algorithm falls into the DFS category, and the proposed quality measure is not anti-monotonic, but rather than evaluating it in a post-processing step, we rely on an alternative pruning technique to reduce the size of the search space. We believe the additional computational effort to be justified, as we manage to produce

intuitive results, with the most interesting patterns, which existing state-of-the-art methods sometimes fail to discover at all, ranked at the very top.

Petitjean et al. (2016) proposed an alternative measure of interestingness for ranking sequential patterns, which does not satisfy the anti-monotonicity property either, and an algorithm (SKOPUS) to *directly* enumerate candidate sequential patterns satisfying the interestingness measure. This measure, like EPIRANK, is based on leverage and compares the support against the expected support assuming independence. Unlike our approach, SKOPUS takes as input a database of many, typically short, sequences, rather than a single sequence.

Feremans et al. (2018) proposed an alternative interestingness measure for evaluating sequential patterns in a single long sequence. They evaluate what percentage of the pattern's minimal occurrences are small enough, where small enough is defined by multiplying a user-defined parameter with the size of a pattern. For example, if this parameter is set to 2, a window of size 6 will be small enough for a pattern of size 3, but not for a pattern of size 2. However this method takes the sizes of the minimal windows of a pattern not into account, as long as they are small enough. For example, if any window of size 10 or smaller is considered small enough, then a window of size 2 will score just as much as a window of size 9.

In other related work, various authors incorporated *temporal constraints* in pattern mining (Méger and Rigotti 2004; Pei et al. 2007). Two types of constraints are either a maximal window constraint, i.e., the maximum elapsed time between the first and last event of an occurrence of the pattern, and a maximal gap constraint, i.e., the maximum elapsed time between any two consecutive events in each occurrence. The main difference between our work and these approaches is that we always consider *all* event occurrences, instead of counting only pattern occurrences satisfying the temporal constraints. Furthermore, we do not only count occurrences, but use the window length of each occurrence as a weight used in ranking the *top* patterns. Finally, we remark that a recent benchmarking study that compared different episode miners (Zimmermann 2014), whose framework we follow in Sect. 4.1, reported experimental results of the gap constraint techniques, and these results were considerably worse than the results of FCI_{SEQ} with respect to recovering patterns embedded in the data.

Multiple authors have also stepped away from mining all frequent patterns, and rather tried to reduce the number of patterns often based on information theoretic approaches such as the *Minimal Description Length* (Grünwald 2007), thereby producing a smaller set of patterns that covers the sequence, or, in most cases, a database of many sequences. Methods such as SQS (Tatti and Vreeken 2012), GOKRIMP (Lam et al. 2014), and ISM (Fowkes and Sutton 2016) follow this approach. These methods take a database of typically short sequences as input, which is different from our approach. Another key difference is that rather than enumerating as few candidates as possible and then selecting the best candidates according to an interestingness measure, they employ *heuristic search* to incrementally build a set of non-redundant patterns, instead of trying to enumerate an exact set of patterns, based on a definition of interestingness.

Finding interesting pairs (or n-grams) of co-occurring words in natural language has also been extensively studied by the Natural Language Processing community (Manning and Schütze 1999). Here the goal is to find *collocations*, that is co-occurring words that are typical in a corpus, such as *strong tea* or *the rich and the famous*. Specific to the Natural

Language domain, collocations are also characterised by the semantic concept of limited compositionality, that is the meaning of the collocation of words is only weakly related to the meaning of the individual words, e.g., *strong* has a different meaning in the phrase *strong tea* than in *strong man*. In the context of mining collocations, different methods start with counting all frequent bi-grams or n-grams within a fixed window, after filtering words based on part of speech tags (Justeson and Katz 1995). Different authors have proposed ranking collocations based on various statistics, such as mean and variance of word distances, and based on various hypothesis tests, such as the t test, χ^2 test, and likelihood ratios, or using pointwise mutual information (Church and Mercer 1993; Manning and Schütze 1999). In essence, the different methods are frequency-based methods using fixed windows with re-ranking based on different statistics. Beside the domain-specific analysis of this problem, there are major technical differences with our approach. We are also interested in mining co-occurrences of potentially larger sets of items efficiently, while the focus by the NLP community is more on defining interestingness measures on bi-grams (or smaller sets of items). We remark that, for mining smaller sets of items within a small fixed window, it is relatively straightforward to design an algorithm that generates all possible n-grams in reasonable time. However, for larger itemsets and window sizes any algorithm would need to at least prune the search space of possible candidates in some way, to overcome the combinatorial explosion induced by larger itemsets and window sizes.

6 Conclusion

In this paper, we present a novel method for finding valuable patterns in event sequences. First of all, we evaluate the quality of the discovered itemsets using cohesion, a measure of how far apart the items making up the itemset are on average. In this way, we reward strong patterns that are not necessarily very frequent in the data, which allows us to discover patterns that existing frequency-based algorithms fail to find. Since cohesion is not an anti-monotonic measure, we rely on an alternative pruning technique, based on an upper bound of the cohesion of candidate patterns that have not been generated yet. We show both theoretically and empirically that the method is sound, the upper bound tight, and the algorithm efficient, allowing us to discover large numbers of patterns reasonably quickly.

Based on the discovered cohesive itemsets, we then search for representative sequential patterns and dominant episodes, which offer additional information about the order in which the items making up the itemsets occur. If no order is representative of the occurrences of an itemset, we report no sequential pattern or partial order. Furthermore, we mine association rules, with a confidence measure based on the cohesion of the antecedent and consequent, rather than the frequency-based definition common in literature. We integrate the mining process of all four pattern types into a single efficient algorithm.

Experimental results demonstrate that our approach produces a more intuitive ranking of patterns than existing frequency-based state-of-the-art methods. For all pattern types, we rank interesting patterns highly, while avoiding spurious patterns that consist of unrelated items that often co-occur purely because they all occur very frequently. For sequential patterns and, particularly, episodes, we limit the number of patterns that can be generated from a single itemset, thus avoiding a pattern explosion common for existing algorithms.

Our experiments confirm both the high quality of our output and the efficiency of our algorithm.

Acknowledgements

The authors would like to thank the VLAIO SBO HYMOP project for funding this research.

References

Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *International Conference on Very Large Data Bases*, pages 487–499, 1994.

Kenneth W Church and Robert L Mercer. Introduction to the special issue on computational linguistics using large corpora. *Computational linguistics*, 19(1):1–24, 1993.

Boris Cule and Bart Goethals. Mining association rules in long sequences. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 300–309. Springer, 2010.

Boris Cule, Bart Goethals, and Céline Robardet. A new constraint for mining sets in sequences. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 317–328, 2009.

Boris Cule, Nikolaj Tatti, and Bart Goethals. Marbles: Mining association rules buried in long event sequences. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 7(2):93–110, 2014.

Boris Cule, Len Feremans, and Bart Goethals. Efficient discovery of sets of co-occurring items in event sequences. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 361–377. Springer, 2016.

Len Feremans, Boris Cule, and Bart Goethals. Mining top-k quantile-based cohesive sequential patterns. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 90–98. SIAM, 2018.

Jaroslav Fowkes and Charles Sutton. A subsequence interleaving model for sequential pattern mining. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 835–844. ACM, 2016.

Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

John S Justeson and Slava M Katz. Technical terminology: some linguistic properties and an algorithm for identification in text. *Natural language engineering*, 1(1):9–27, 1995.

Hoang Thanh Lam, Fabian Mörchen, Dmitriy Fradkin, and Toon Calders. Mining compressing sequential patterns. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 7(1):34–52, 2014.

Srivatsan Laxman, PS Sastry, and KP 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*, pages 410–419. ACM, 2007.

Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

Nicolas Méger and Christophe Rigotti. Constraint-based mining of episode rules and optimal window sizes. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 313–324. Springer, 2004.

Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE transactions on knowledge and data engineering*, 16(11): 1424–1440, 2004.

Jian Pei, Jiawei Han, and Wei Wang. Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems*, 28(2):133–160, 2007.

François Petitjean, Tao Li, Nikolaj Tatti, and Geoffrey I Webb. Skopus: Mining top-k sequential patterns under leverage. *Data Mining and Knowledge Discovery*, 30(5):1086–1111, 2016.

Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer, 1996.

Nikolaj Tatti. Discovering episodes with compact minimal windows. *Data Mining and Knowledge Discovery*, 28(4):1046–1077, 2014.

Nikolaj Tatti. Ranking episodes using a partition model. *Data Mining and Knowledge Discovery*, 29(5):1312–1342, 2015.

Nikolaj Tatti and Boris Cule. Mining closed strict episodes. *Data Mining and Knowledge Discovery*, 25(1):34–66, 2012.

Nikolaj Tatti and Jilles Vreeken. The long and the short of it: summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 462–470. ACM, 2012.

Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *IEEE international conference on data engineering*, pages 79–90, 2004.

Geoffrey I Webb. Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM Transactions on Knowledge Discovery from Data*, 4 (1):3, 2010.

Mohammed J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3):372–390, 2000.

Albrecht Zimmermann. Understanding episode mining techniques: Benchmarking on diverse, realistic, artificial data. *Intelligent Data Analysis*, 18(5):761–791, 2014.

Appendix

Top-25 patterns

Tables 11 and 12 show the top-25 itemsets for *Species* and *Trump*, Tables 13 and 14 show the top-25 sequential patterns, and Tables 15 and 16 the top-25 association rules. Note that, as discussed in Sect. 4, FCI_{SEQ} produced fewer than 25 sequential patterns per dataset, due to the usage of the minimal occurrence ratio threshold. A lower threshold would naturally result in more patterns, but we argue that these patterns are better omitted from the output, since they are in fact not representative of the occurrences of the underlying itemset. Patterns for FCI_{SEQ} in bold are not reported by any other state-of-the-art method in the top-1000, likewise, patterns in bold for other methods are not reported by FCI_{SEQ} in the top-1000. Note that since we only produce fewer than 25 sequential patterns, nearly all of the patterns found by other methods are in bold.

Table 11 Top 25 itemsets for *Species*.

FCI_{SEQ}	Winepi	Laxman	$Marbles_W$	CMW
del, fuego, tierra	natur, select	natur, select	natur, select	**absenc, island, mammal, ocean, terrestri**
facit, natura, saltum	**speci, varieti**	form, speci	**speci, varieti**	**altern, glacial, north, period, south**
del, fuego	form, speci	**speci, varieti**	distinct, speci	**bat, island, mammal, ocean, speci, terrestri**
del, tierra	natur, speci	natur, speci	form, speci	**bat, island, mammal, ocean, terrestri**
facit, saltum	distinct, speci	distinct, speci	condit, life	**cross, fertil, hybrid, mongrel, offspr, varieti**
facit, natura	gener, speci	gener, speci	natur, speci	**differ, endow, incident, special, steril**
fuego, tierra	**differ, speci**	**differ, speci**	**anim, plant**	**ag, earli, inherit, success, superven, variat**
natura, saltum	case, speci	case, speci	**genu, speci**	**glacial, northern, period, southern, temper**
ithomia, leptali	condit, life	case, natur	**differ, speci**	**ag, earli, inherit, period, superven, variat**
leptali, mimick	**genu, speci**	**natur, organ**	**be, organ**	**inhabit, island, mainland, nearest, relat**
ithomia, leptali, mimick	**anim, plant**	**select, speci**	**group, speci**	**fertil, mongrel, offspr, univers, varieti**
ithomia, mimick	**group, speci**	**group, speci**	gener, speci	**cross, fertil, mongrel, offspr, varieti**
hexagon, sphere	**number, speci**	**number, speci**	**genera, speci**	**mountain, northern, southern, temper**
forcep, urchin	case, natur	**genera, speci**	**cross, speci**	**ag, earli, inherit, superven, variat**
rufescen, sanguinea	**genera, speci**	**charact, speci**	**individu, speci**	**absenc, island, ocean, terrestri**
pyramid, rhomb	**cross, speci**	**plant, speci**	case, speci	**crop, fantail, pouter, tail**
natur, select	**be, organ**	**form, varieti**	**alli, speci**	**cross, differ, incident, system, unknown**
sur, tom	**close, speci**	**anim, plant**	**number, speci**	**exist, geometr, increas, ratio, struggl**
natur, speci	**charact, speci**	form, natur	**form, life**	**absenc, mammal, ocean, terrestri**
prism, pyramid, rhomb	**descend, speci**	condit, life	**descend, speci**	**connect, exist, intermedi, lesser, number, varieti**
busk, chela	**individu, speci**	**natur, variat**	**alli, close**	**fittest, man, natur, select, surviv**
form, speci	**form, varieti**	**organ, speci**	case, natur	**absenc, island, mammal, terrestri**
gener, speci	**natur, variat**	**individu, speci**	**exist, speci**	**altern, glacial, north, period**
matthew, vol	**natur, organ**	natur, select, speci	**close, speci**	**cross, differ, incident, reproduct, system**
saint, sur	**select, speci**	**anim, speci**	**produc, speci**	**endow, incident, special, steril**

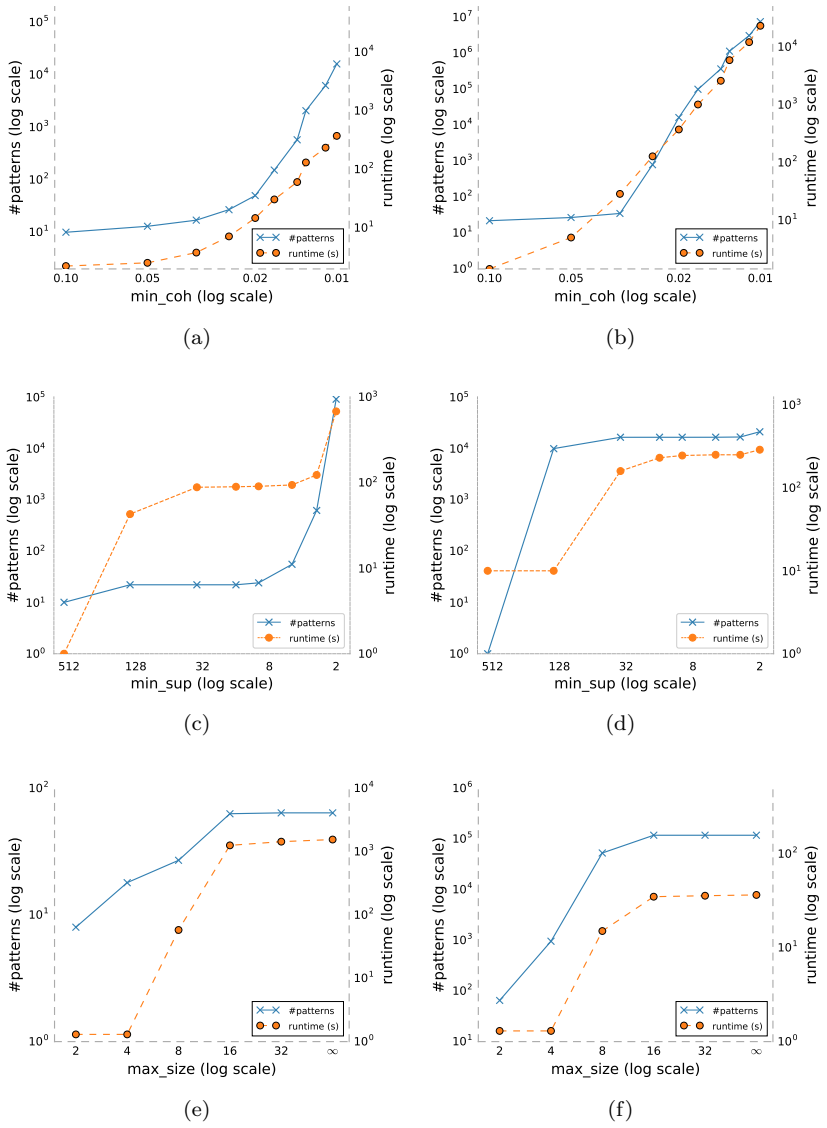


Fig. 7 Impact of various thresholds on output size and runtime. (a) Varying *min_coh* on *Species*, *min_sup* = 5, *max_size* = 4. (b) Varying *min_coh* on *Trump*, *min_sup* = 4, *max_size* = 5. (c) Varying *min_sup* on *Species*, *min_coh* = 0.02, *max_size* = 4. (d) Varying *min_sup* on *Trump*, *min_coh* = 0.02, *max_size* = 5. (e) Varying *max_size* on *Species*, *min_sup* = 350, *min_coh* = 0.02. (f) Varying *max_size* on *Trump*, *min_sup* = 150, *min_coh* = 0.02.

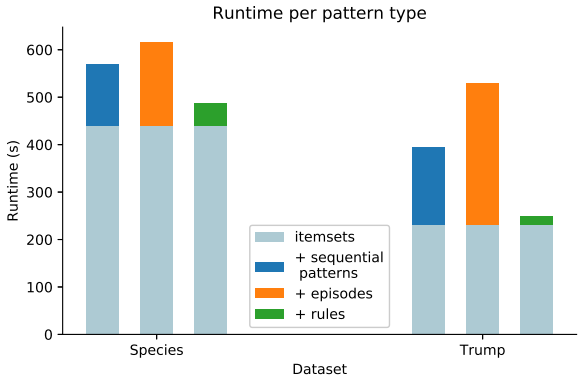


Fig. 8 Impact on runtime of mining different types of patterns and rules. For *Species*, parameters were $min\_coh = 0.015$, $min\_sup = 4$, $max\_size = 5$, $min\_or = 0.5$, $min\_por = 0.5$ and $min\_conf = 0.7$. For *Trump*, parameters were $min\_coh = 0.02$, $min\_sup = 4$, $max\_size = 5$, $min\_or = 0.5$, $min\_por = 0.5$ and $min\_conf = 0.7$.

Table 12 Top 25 itemsets for *Trump*.

FCI_{SEQ}	WINEPI	LAXMAN	$MARBLES_W$	CMW
puerto, rico	fake, new	fake, new	fake, new	**ab, japan, minist, prime**
hunt, witch	cut, tax	cut, tax	cut, tax	**high, hit, market, stock, time**
harbor, pearl	america, great	america, great	america, great	**abc, cnn, fake, nbc, new**
lago, mar	great, make	great, peopl	america, make	**alabama, big, luther, strang, vote**
arabia, saudi	america, make	great, make	great, make	**lowest, market, stock, unemploy, year**
jong, kim	great, peopl	great, job	**state, unit**	**base, immigr, merit, system**
davo, switzerland	america, great, make	countri, great	**great, honor**	**high, hit, job, market, stock**
davo, wef	great, job	great, todai	korea, north	**abc, cb, cnn, fake, new**
davo, switzerland, wef	**great, honor**	america, make	great, job	**greatest, histori, hunt, witch**
switzerland, wef	countri, great	**great, tax**	america, great, make	**alabama, great, luther, state, strang**
unga, usaatunga	fake, media	great, state	great, peopl	**high, hit, market, stock**
prstrong, ricardorossello	great, state	big, great	**media, new**	**donald, presid, proclaim, trump**
rex, tillerson	**state, unit**	great, new	**hard, work**	**korea, moon, presid, south**
fake, new	**great, tax**	america, great, make	fake, media	**bail, compani, democrat, insur**
minist, prime	great, todai	**great, honor**	market, stock	**fail, fake, media, new, nytim**
christma, merri	**great, work**	**great, work**	great, state	**high, market, stock, unemploy**
korea, north	**media, new**	fake, media	**hous, white**	**abc, cnn, nbc, new**
cut, tax	great, new	**cut, great**	**republican, senat**	**high, job, market, stock, time**
america, make	**dai, great**	**great, presid**	countri, great	**alabama, luther, strang, vote**
chain, migrat	big, great	**american, great**	**dai, great**	**high, market, record, stock, time**
great, peopl	**hard, work**	great, year	great, new	**big, luther, senat, strang**
market, stock	korea, north	**dai, great**	**great, meet**	**court, state, suprem, unit**
america, great	fake, media, new	**great, republican**	great, todai	**cnn, fail, fake, new, nytim**
fake, great, new	**cut, great**	**cut, great, tax**	**cut, reform**	**high, market, stock, time**
america, great, make	**american, great**	**fake, great**	minist, prime	**alabama, great, luther, senat, strang**

Table 13 Top 25 sequential patterns for *Species*.

FCI_{SEQ}	Winepi	Laxman	$Marbles_W$	CMW
tierra, del, fuego	natur, select	natur, select	natur, select	**struggl, exist, geometr, ratio, increas**
natura, facit, saltum	**varieti, speci**	**varieti, speci**	**distinct, speci**	**variat, superven, earli, ag, inherit**
del, fuego	**speci, varieti**	**speci, form**	**varieti, speci**	**form, life, chang, simultan, world**
tierra, del	**distinct, speci**	**speci, varieti**	**condit, life**	**variat, superven, earli, inherit, ag**
facit, saltum	**speci, form**	**form, speci**	**speci, varieti**	**steril, speci, cross, hybrid, offspr**
natura, facit	**form, speci**	**speci, natur**	**organ, be**	**wide, diffus, speci, larger, genera, vari**
tierra, fuego	**condit, life**	**natur, speci**	**speci, genu**	**success, variat, superven, earli, inherit**
natura, saltum	**speci, natur**	**distinct, speci**	**speci, form**	**inhabit, island, nearest, mainland**
natur, select	**speci, genu**	**speci, gener**	**form, speci**	**chapter, geolog, success, organ, be**
vol, matthew	**natur, speci**	**case, speci**	**individu, speci**	**varieti, exist, lesser, number, intermedi**
avicularia, vibracula	**organ, be**	**gener, speci**	**close, alli**	**glacial, period, north, south**
eject, foster, brother	**speci, gener**	**speci, differ**	**speci, natur**	**incident, differ, reproduct, system**
eject, foster	**differ, speci**	**differ, speci**	**anim, plant**	**natur, system, genealog, arrang**
oviger, frena	**speci, differ**	**speci, case**	**group, speci**	tierra, del, fuego
movabl, zooid	**case, speci**	**condit, life**	**speci, genera**	**seiz, place, economi, natur**
inter, se	**speci, genera**	**select, natur**	**alli, speci**	**superven, earli, ag, inherit**
sphere, prism	**gener, speci**	**speci, distinct**	**differ, speci**	natura, facit, saltum
foster, brother	**individu, speci**	**number, speci**	**form, life**	**instinct, slave, make, ant**
sown, mix	**group, speci**	**speci, genu**	**speci, gener**	**varieti, exist, lesser, number, connect**
sphere, hexagon, prism, rhombic	**number, speci**	**speci, genera**	**natur, speci**	**direct, action, extern, condit**
hexagon, prism	**speci, distinct**	**group, speci**	**speci, differ**	**ocean, island, terrestri, mammal**
	anim, plant	**speci, group**	**number, speci**	**natur, select, extinct, diverg, charact**
	alli, speci	**case, natur**	**speci, group**	**exist, geometr, ratio, increas**
	speci, group	**charact, speci**	**fresh, water**	**revers, long, lost, charact**
	speci, case	**individu, speci**	**case, speci**	**form, naturalist, rank, distinct, speci**

Table 14 Top 25 sequential patterns for *Trump*.

FCI_{SEQ}	WINEPI	LAXMAN	$MARBLES_W$	CMW
puerto, rico	fake, new	fake, new	fake, new	**stock, market, hit, time, high**
witch, hunt	tax, cut	tax, cut	tax, cut	**job, stock, market, time, high**
pearl, harbor	**america, great**	**america, great**	**america, great**	**greatest, witch, hunt, histori**
mar, lago	**make, america**	**make, great**	**make, america**	**stock, market, hit, high**
saudi, arabia	**make, great**	**make, america**	**unit, state**	**presid, moon, south, korea**
kim, jong	**make, america, great**	**great, peopl**	**make, great**	**presid, donald, trump, proclaim**
davo, switzerland	**unit, state**	**make, america, great**	**great, honor**	**fake, new, fail, nytim, cnn**
switzerland, wef	**great, honor**	**great, job**	north, korea	**market, hit, time, high**
usaatunga, unga	fake, media	**great, honor**	**make, america, great**	**stock, market, time, high**
ricardorossello, prstrong	**great, job**	**peopl, great**	**new, media**	**stock, market, hit, time**
rex, tillerson	**great, peopl**	**great, state**	stock, market	**stock, hit, time, high**
fake, new	**new, media**	fake, media	fake, media	**massiv, tax, cut, reform**
prime, minist	north, korea	**great, countri**	**white, hous**	**healthcar, tax, cut, reform**
merri, christma	stock, market	**great, tax**	**work, hard**	**fake, new, cnn, abc**
north, korea	**great, state**	**unit, state**	**great, job**	**republican, senat, work, hard**
tax, cut	**fake, new, media**	**tax, great**	**great, peopl**	**radic, islam, terror**
chain, migrat	**white, hous**	**great, todai**	**great, state**	**fake, new, cnn, nbc**
stock, market	**work, hard**	**great, work**	**republican, senat**	**new, media, fail, nytim**
luther, strang	**great, countri**	**countri, great**	prime, minist	**make, america, great, fake, new**
fake, media	**tax, reform**	north, korea	**tax, reform**	**greatest, witch, hunt**
berni, sander	**peopl, great**	**great, america**	**fake, new, media**	**joint, press, confer**
radic, islam	**republican, senat**	**great, american**	**crook, hillari**	**prime, minist, ab**
	great, tax	**great, presid**	**cut, reform**	**behalf, flotu, melania**
	great, american	**new, great**	**great, countri**	**stock, market, hit, high, great**
	great, meet	**new, media**	**men, women**	**m, gang, member**

Table 15 Top 25 rules for *Species*.

FCI_{SEQ}	WINEPI	$MARBLES_M$	$MARBLES_W$
fuego, tierra → del	**divis, kingdom → anim**	**hive → bee**	**divis, kingdom → anim**
del, fuego → tierra	**averag, genera → speci**	**mivart → mr**	fuego, tierra → del
del, tierra → fuego	**cuckoo, lai, nest → egg**	**case, select, structur → natur**	**independ, ordinari → view**
facit, natura → saltum	**varieti, zone → intermedi**	**candol → de**	natura, saltum → facit
natura, saltum → facit	**genera, present, varieti → speci**	**case, organ, select → natur**	**inherit, superven → earli**
facit, saltum → natura	**cape, hope → good**	**distinct, rank, varieti → speci**	**accumul, act, natur → select**
fuego → del	**accumul, act, natur → select**	**breed, rock → pigeon**	**rang, vari → speci**
del → fuego	**inherit, superven → earli**	**diverg, select → natur**	**economi, seiz → place**
tierra → del	fuego, tierra → del	**wallac → mr**	**ag, inherit, superven → earli**
del → tierra	**differ, genera, varieti → speci**	**humbl → bee**	**exist, select, theori → natur**
fuego → del, tierra	**genu, greater → speci**	**function, select → natur**	**inherit, superven, variat → earli**
tierra → del, fuego	**select, structur, theori → natur**	**independ, select → natur**	**act, natur, sole → select**
del → fuego, tierra	**genera, smaller, varieti → speci**	**life, physic → condit**	**newli, varieti → form**
facit → saltum	**independ, ordinari → view**	**charact, secondari → sexual**	**exist, varieti, zone → intermedi**
saltum → facit	**charact, secondari, speci → sexual**	**favour, select, speci → natur**	**ask, distinct → speci**
natura → facit	**creat, independ, view → speci**	**select, speci, theori → natur**	**ag, inherit, superven, variat → earli**
facit → natura	**inherit, superven, variat → earli**	**charact, diverg, select → natur**	**bottom, rest → side**
natura → facit, saltum	**genera, larger, number → speci**	**malai → archipelago**	**end, mean → gain**
facit → natura, saltum	**exist, select, theori → natur**	**genu, manner → speci**	**rang, vari, wide → speci**
saltum → facit, natura	**action, diverg, natur → select**	**genu, produc → speci**	**incident, reproduct → differ**
prism → hexagon	**action, diverg, select → natur**	**fittest → surviv**	**english, face → short**
fuego → tierra	natura, saltum → facit	**incipi, varieti → speci**	**economi, natur, seiz → place**
tierra → fuego	**ag, inherit, superven → earli**	**cell, hive → bee**	**larger, relat → genera**
natura → saltum	**rang, vari → speci**	**genera, larger, varieti → speci**	**manner, mivart → mr**
saltum → natura	**bird, cuckoo, lai, nest → egg**	**fritz → muller**	**life, organ, physic → condit**

Table 16 Top 25 rules for *Trump*.

FCI_{SEQ}	WINEPI	$MARBLES_M$	$MARBLES_W$
puerto → rico	**hit, stock → market**	**cut, reform → tax**	**honor, minist → prime**
rico → puerto	**high, hit, stock → market**	puerto → rico	**confer, joint → press**
hunt → witch	**bill, reform, tax → cut**	rico → puerto	**immigr, merit → base**
witch → hunt	**biggest, cut, histori → tax**	witch → hunt	**greatest, hunt → witch**
migrat → chain	**ab, prime → minist**	hunt → witch	**donald, proclaim → trump**
pearl → harbor	**abc, fake → new**	**great, white → hous**	**hit, record, stock → market**
harbor → pearl	**honor, minist → prime**	**high, market → stock**	**immigr, merit, system → base**
merri → christma	**behalf, melania → flotu**	**cut, dem → tax**	**law, offic → enforc**
saudi → arabia	**hit, stock, time → market**	**cut, pass → tax**	**liyuan, madam → peng**
arabia → saudi	**massiv, reform, tax → cut**	**biggest, cut → tax**	**donald, presid, proclaim → trump**
lago → mar	**abc, cnn, fake → new**	**market, record → stock**	**chain, visa → migrat**
mar → lago	**confer, joint → press**	**alabama, strang → luther**	**great, honor, minist → prime**
kim → jong	**immigr, merit → base**	**suprem → court**	**high, hit, record, stock → market**
jong → kim	**abc, fake, nbc → new**	**great, prime → minist**	**cut, massiv, work → tax**
sander → berni	**high, hit, stock, time → market**	**great, minist → prime**	**great, high, stock → market**
usaatunga → unga	**biggest, reform → tax**	**great, puerto → rico**	**histori, witch → hunt**
rex → tillerson	**biggest, reform → cut**	**hous, tax → cut**	**jame, leak → comei**
switzerland → davo	**biggest, reform → cut, tax**	**cut, hous → tax**	**bill, massiv, tax → cut**
wef → davo	**hit, record, stock → market**	**fake, nytim → new**	**famili, thought → prayer**
davo → switzerland	**biggest, reform, tax → cut**	**big, cut, great → tax**	**men, protect → women**
switzerland → davo, wef	**biggest, cut, pass → tax**	**birthdai → happi**	**jone, pelosi, puppet → schumer**
switzerland, wef → davo	**biggest, cut, reform → tax**	**xi → china**	**great, job, market → stock**
wef → davo, switzerland	**immigr, merit, system → base**	**premium → obamacar**	**american, cut, massiv → tax**
switzerland → wef	**alabama, big, strang → luther**	**great, strang → luther**	**high, stock, unemploy → market**
davo, switzerland → wef	**budget, cut → tax**	**north, presid → korea**	**anthem, great, stand → nation**