SEQRET: Mining Rule Sets from Event Sequences

Aleena Siji,[1]° Joscha Cüppers,[2] Osman Ali Mian,[3]° Jilles Vreeken[2]

[1] Helmholtz AI, Munich
[2] CISPA Helmholtz Center for Information Security
[3] Institute for AI in medicine, UK Essen
aleena.siji@helmholtz-munich.de, joscha.cueppers@cispa.de, osman.mian@uk-essen.de, vreeken@cispa.de

Abstract

Summarizing event sequences is a key aspect of data mining. Most existing methods neglect conditional dependencies and focus on discovering sequential patterns only. In this paper, we study the problem of discovering both conditional and unconditional dependencies from event sequences. We do so by discovering rules of the form $X \rightarrow Y$ where X and Y are sequential patterns. Such rules are simple to understand and provide a clear description of the relation between the antecedent and the consequent. To discover a succinct and non-redundant set of rules we formalize the problem in terms of the Minimum Description Length principle. As the search space is enormous and does not exhibit helpful structure, we propose the SEQRET method to discover high-quality rule sets in practice. Through extensive empirical evaluation we show that unlike the state of the art, SEQRET ably recovers the ground truth on synthetic datasets and finds useful rules from real datasets.

1 Introduction

In many applications data naturally takes the form of events happening over time. Examples include industrial production logs, the financial market, device failures in a network, medical treatment plans etc. Existing methods for analyzing event sequences primarily focus on mining unconditional, frequent sequential patterns (Agrawal and Srikant 1995; Mannila and Meek 2000; Tatti 2009). Loosely speaking, these are subsequences that appear more often in the data than we would expect. Real world processes are often more complex than this, as they often include conditional dependencies. For example, the formation of tropical cyclones (C) in the Bay of Bengal is often but not always followed by heavy rainfall (R) on the coast. Knowing such a relationship is helpful both in predicting events and in understanding the underlying data generating mechanisms.

In this paper, we are interested in discovering rules of the form $X \rightarrow Y$ from long event sequences, where X and Y are sequential patterns. Existing methods for mining such rules either suffer from pattern explosion, i.e. are prone to returning orders of magnitude more results than we can possibly analyze (Fournier-Viger et al. 2021; Chen et al. 2021), or have very limited expressivity, e.g. require the constituent events to occur in contiguous order (Bourrand et al. 2021).

We aim to discover succinct sets of rules that generalize the data well and explicitly allow for gaps between the occurrences of the constituent events of these rules. To ensure compact and non-redundant results, we formalize the problem using the Minimum Description Length (MDL) principle (Grünwald 2007). Loosely speaking, we are after that set of sequential rules that together compresses the data best.

However, the problem we so arrive at is computationally challenging. For starters, there exist exponentially many rules, exponentially many rule sets, and then again exponentially many ways to describe the data given a set of rules. Moreover, the search space does not exhibit structure we can use to efficiently obtain the optimal result. To mine good rule sets from data we therefore propose the greedy SEQRET algorithm. We introduce two variants. SEQRET-CANDIDATES constructs a good rule set from a set of candidate patterns by splitting them into high-quality rules. SEQRET-MINE, on the other hand, only requires the data and mines a good rule set from scratch. Starting from a model of singleton rules, it iteratively extends them into more refined rules. To avoid testing all possible extensions, we consider only those extensions that occur significantly more often than expected.

Through extensive evaluation, we show that both variants of SEQRET work well in practice. On synthetic data we show that they are robust to noise and recover the ground truth well. On real-world data, we show that SEQRET returns succinct sets of rules that give clear insight into the data generating process. This is in stark contrast to existing methods which either return many thousands of rules (Fournier-Viger et al. 2021) or are restricted to rules where events occur contiguously (Bourrand et al. 2021).

To summarize, the main contributions are as follows:

(a) We define a pattern language for fully ordered sequential rules that accommodates for gaps.
(b) We present SEQRET-CANDIDATES for constructing a high-quality rule set, given a set of sequential patterns.
(c) We present SEQRET-MINE for mining a high-quality rule set given a databse of event sequences.
(d) We extensively evaluate SEQRET on synthetic and real-world datasets, comparing it to the state-of-the-art.

°Work done while at CISPA Helmholtz Center for Information Security.

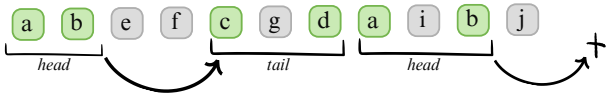

Figure 1: Toy example of rule $ab \rightarrow cd$ in a sequence. Head ab triggers the rule twice. Tail cd follows only once. Hence, $supp(ab \rightarrow cd) = 1$ and $conf(ab \rightarrow cd) = 0.5$.

2 Preliminaries

In this section we introduce basic notation and give a short introduction to the MDL principle.

Notation

As data we consider a database D of $|D|$ event sequences. A sequence $S \in D$ consists of $|S|$ events drawn from a finite alphabet Ω of discrete events $e \in \Omega$. We denote the total number of events in the data as $||D||$. We write S_t for the t^{th} sequence in D. We omit the subscript whenever clear from context. We write $S[i]$ to refer to the i^{th} event in sequence S, and $S[i, j]$ for the subsequence from the i^{th} up to and including the j^{th} event. We denote an empty sequence by ϵ.

A serial episode X is a sequence of $|X|$ events drawn from Ω. A sequential rule r captures the conditional dependence between serial episodes X and Y. Intuitively, it expresses that whenever we see X in the data it is more likely that Y will follow. We refer to X as the *head* of r, denoted $head(r)$, and to Y as the *tail* of r, denoted $tail(r)$. If X is an empty pattern, $X = \epsilon$, we call $X \rightarrow Y$ an *empty head rule*. Empty head rules where $|Y| = 1$ are called *singleton* rules.

A subsequence $S[i, j]$ is a window of pattern X iff X is a subsequence of $S[i, j]$, and we say $S[i, j]$ *matches* X and that X occurs in $S[i, j]$. A pattern window $S[i, j]$ is *minimal* for X iff no proper sub-window of $S[i, j]$ matches X. A rule window of r is a tuple of two pattern windows $S[i, j]$ and $S[k, l]$ when $S[i, j]$ matches $head(r)$, $j < k$, and $S[k, l]$ matches $tail(r)$, we denote it by $S[i, j; k, l]$.

We say a window $S[i, j]$ *triggers* rule r when it is a minimal window of $head(r)$. A rule window $S[i, j; k, l]$ *supports* a rule r if $S[i, j]$ triggers $head(r)$ and $S[k, l]$ matches $tail(r)$. We call the number of events that occur in a rule window between the rule head and the rule tail, $k-j-1$, the *delay* of the rule instance. We give an example in Fig. 1. We denote the number of windows over all sequences $S \in D$ that trigger a rule r as the trigger count $trigs(r)$. We define the *support* of a rule r as the number of rule windows $S[i, j; k, l]$ in D where $S[k, l]$ is a minimal window of $tail(r)$ and follows the head with minimum delay. Finally, we define the confidence of a rule r as its support relative to its trigger count, formally $conf(r) = supp(r)/trigs(r)$.

Minimum Description Length Principle

The Minimum Description Length (MDL) (Grünwald 2007) is a computable and statistically well-founded approximation of Kolmogorov complexity (Li and Vitányi 1997). For a given model class $\mathcal{M}$ it identifies the best model $M \in \mathcal{M}$ as the one that minimizes the number of bits for describing both model and data without loss. Formally, minimize

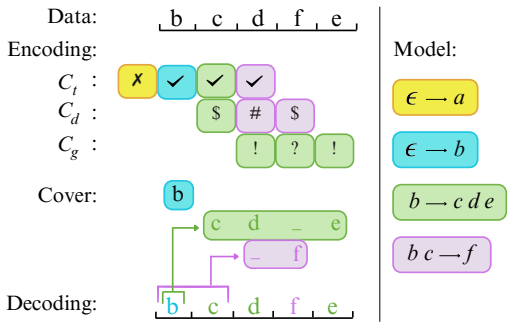

Figure 2: Toy example of encoding sequence S using rule set R. C_t encodes if a triggered rule *hits* or *misses*. C_d encodes the delay between the trigger and the rule tail. C_g encodes the gaps in the rule tail. Together, they form a *cover* C.

$L(M) + L(D \mid M)$ where $L(M)$ is the length in bits of model M and $L(D \mid M)$ is the length in bits of data D given M. This is known as two-part, or crude MDL—in contrast to one-part, or refined MDL, which is not computable for arbitrary model classes. We use two-part MDL because we are interested in the model, i.e. the set of rules that describes the data best. In MDL we are never concerned with materialized codes but only code lengths. To use MDL, we have to define a model class and code length functions, we do so next.

3 MDL for Sequential Rules

We now formally define the problem. We consider sets of sequential rules as our model class $\mathcal{R}$. By MDL, we are interested in that set $R \in \mathcal{R}$ that best describes data D.

Decoding an Event Sequence

Before we formally define how we *encode* models and data, we give the main intuition by *decoding* an already encoded sequence. We give an example in Fig. 2.

To decode a symbol, we consider the rules from R that are currently triggered. For those, we read codes from the trigger stream C_t. Initially, the context is empty and hence only empty-head rules trigger. The first trigger code is a *miss* for singleton rule $\epsilon \rightarrow a$, meaning that the rule tail does not follow. The second trigger code is a *hit* for rule $\epsilon \rightarrow b$. As empty-head rules do not incur delays, we can immediately write b as the first symbol of the sequence.

This triggers rule $b \rightarrow cde$. We hence read a code from the trigger stream C_t, and find that it is a hit. As this rule does not have an empty head, there may be a delay between the head and its tail and we read a code from the delay stream C_d. It is a start code, so we write the first symbol of the tail (c), this creates a minimal window of bc.

This triggers rule $bc \rightarrow f$. We read from C_t to find that it is a hit, then from C_d to find that its tail is delayed. To determine if we write the next symbol from tail cde, we read from the gap stream C_g. It has a fill code, meaning no gap, so we write d. This time, no new rule triggers. Tail cde is not yet completely decoded and f is delayed. For each delayed tail we read a code from C_d, and for each incomplete tail we

read a code from C_g. Here, we read a start code for tail f and a gap code for tail cde, we hence write f. Again, no new rule is triggered. Now only tail cde is not yet fully decoded. We read a fill code from C_g and write e as the last symbol.

To summarize, sequences are encoded from left to right, and rules automatically trigger whenever we observe a minimal window of the head. For each trigger, we encode whether the tail follows using a *hit* or *miss* code. When a rule hits, we encode whether its tail follows immediately or later, using a *start* resp. *delay* code. We encode whether *gaps* occur in the rule tail using *fill* and *gap* codes. Empty-head rules never incur a delay. To avoid unnecessary triggers, we only encode those empty-head rules if no other rule encodes the current symbol (e.g. all active tails say 'gap').

Computing the Description Lengths

Now that we have the intuition, we can formally describe how to encode a model, respectively the data given a model.

Encoding a Model A model $R \in \mathcal{R}$ is a set of rules. To reward structure such as chains where the tail of one rule is the head of another (e.g. $r_1 = \epsilon \to AB$, and $r_2 = AB \to CD$), we first encode the set P of all non-empty, non-singleton heads and all non-singleton tails. Formally,

$$P = \{head(r) \mid \forall r \in R\} \cup \{tail(r) \mid \forall r \in R\} \setminus (\Omega \cup \epsilon) \ .$$

The encoded length $L(P)$, is defined as

$$L(P) = L_{\mathbb{N}}(|P| + 1) + \sum_{p \in P} L_{\mathbb{N}}(|p|) + |p| \log_2(|\Omega|) \ ,$$

where we first encode the number of these patterns using $L_{\mathbb{N}}$, the MDL-optimal encoding for integers (Rissanen 1983). It is defined for $z \geq 1$ as $L_{\mathbb{N}}(z) = \log^* z + \log c_0$ where $\log^* z$ is the expansion $\log z + \log\log z + \cdots$ including only the positive terms. To ensure this satisfies the Kraft inequality, we set $c_0 = 2.865064$ (Rissanen 1983). Since P can be empty and $L_{\mathbb{N}}$ is only defined for numbers ≥ 1 we offset it by one. Next, we encode each pattern $p \in P$ where we use $L_{\mathbb{N}}$ to encode its length and then choose each subsequent symbol $e \in p$ out of alphabet Ω.

Now that we have the set of all heads and tails, we have

$$L(R \mid P) = L_{\mathbb{N}}(|R| + 1) + |R| \cdot (\log_2(|P| + |\Omega| + 1) + \log_2(|P| + |\Omega|)) \ ,$$

as the encoded length in bits for a set of rules. We first encode the number of rules. As R can be empty, we again offset by one. Next, for each rule $r \in R$, we choose its head from $P \cup \Omega$, and then its tail from $P \cup \Omega$.

Putting this together, the number of bits to describe a rule set $R \in \mathcal{R}$ without loss is

$$L(R) = L(P) + L(R \mid P) \ .$$

Encoding Data given a Model As we saw in the example, reconstructing the data requires three code streams C_t, C_d, and C_g. For an arbitrary database we additionally need to know how many sequences it includes, and their lengths. Formally, the description length of data D given model R is

$$L(D \mid R) = L_{\mathbb{N}}(|D|) + \sum_{S \in D} L_{\mathbb{N}}(|S|) + L(C_t) + L(C_d) + L(C_g) \ .$$

To encode the code streams C_t, C_d, C_g we use prequential codes (Grünwald 2007). These codes are asymptotically optimal without requiring us to make arbitrary choices in how to encode the code distributions. Formally, we have

$$L(C_j) = -\sum_{i=1}^{|C_j|} \log_2 \frac{usg_i(C_j[i] \mid C_j) + c}{i + unique(C_j) \cdot c} \quad ,$$

where $usg_i(C_j[i] \mid C_j)$ denotes the number of times $C_j[i]$ has been used in C_j up to the i^{th} position, $unique(C_j)$ denotes the number of unique symbols in C_j, and c is a small constant. As common in prequential coding, we set c to 0.5.

The Problem, Formally

We can now formalize the problem we aim to solve.

The Sequential Rule Set Mining Problem *Given a sequence database D over alphabet Ω, find the smallest rule set $R \in \mathcal{R}$ and cover C such that the total encoded size*

$$L(R) + L(D \mid R)$$

is minimal.

The resulting search space is enormous. To begin with, there exist super-exponentially many covers of D given R. The optimal cover depends on the code lengths, which in turn depend on code usages. Even if the optimal cover is given, the problem of finding the optimal rule set is super-exponential: there exist exponentially many patterns p in the size of alphabet Ω, exponentially many rules r in the number of patterns, and exponentially many sets of rules. None of these sub-problems exhibit substructure, e.g. monotonicity or submodularity, that we can exploit to find the optimal solution. Hence, we resort to heuristics.

4 The Seqret Algorithm

In this section we introduce SEQRET, for discovering high-quality **seq**uential **r**ule-s**et**s from data. We break the problem down into two parts: optimizing the description of the data given a rule set, and mining good rule sets. For the latter we propose two variants, SEQRET-CANDIDATES for doing so given a set of candidate patterns, and SEQRET-MINE for mining rule sets directly from data.

Selecting a Good Cover

A lossless description of D using rules R correspond to a set of rule windows such that each event e in D is *covered* by exactly one window. We are after that cover which minimizes $L(D \mid R)$. Finding the optimal cover is infeasible, hence we settle for a good cover and find one greedily.

The idea is to define an order over the rule windows and greedily select the next best window until the data is completely covered. To minimize the encoding length, we prefer covering as many events as possible with a single rule. Therefore, we prefer rules with long tails, high confidence, and high support. Among windows of otherwise equally good rules, we prefer those with lower delays and fewer gaps. We consider the starting position of the rule tail as

a final tie breaker. Combining this, we define WINDOW ORDER as descending on $|tail(r)|$, $conf(r)$, and $supp(r)$, then ascending on $l - j - |tail(r)|$ and k, where r is a rule and $S[i, j; k, l]$ is a rule window.

To find a good cover, we start with an empty set and greedily add rule windows to it in WINDOW ORDER. To avoid searching for all possible rule windows in the beginning, we only consider the best window per rule trigger – defined as the one with the fewest gaps in its rule tail window, followed by lowest delay – and look for the next best only if we do not select the former due to conflicts, i.e. when its constituent events are already covered by a previously selected window. To avoid evaluating hopeless windows, we limit ourselves to those within user-set *max delay* ratio and *max gap* ratio. We give the pseudocode of COVER algorithm with worst-case time complexity in the appendix.

Selecting Good Rule Sets

Next, we consider the problem of discovering good rule sets. We first propose an approach for doing so given a set of sequential patterns as input. The intuition is that, when the ground truth includes a sequential rule $X \rightarrow Y$, a good sequential pattern miner will return XY. Then we can reconstruct the ground truth by considering splits of candidate patterns XY into candidate rules $X \rightarrow Y$.

We propose SEQRET-CANDIDATES, for which we give the pseudocode and worst-case time complexity in the appendix. We first initialize the rule set with all singleton rules to ensure that the data can be encoded losslessly. We then iterate over the candidate patterns in descending order of their contribution to compression (Tatti and Vreeken 2012). We split each pattern into candidate rules – for example, pattern *abc* generates candidate rules $\epsilon \rightarrow abc$,$a \rightarrow bc$, and $ab \rightarrow c$ – and choose the one whose addition to the rule set minimizes the total description length, with $L(D|R)$ determined by the COVER procedure. We keep it in our model if it improves the score and discards otherwise. We iterate this process for all candidate patterns.

Generating Good Rules

Next, we move our attention to mining good rule sets directly from data. The first step is to generate good candidate rules. Suppose we know a rule r. We consider extending it with events $e \in \Omega$ that occur significantly more often than expected by chance, either within or directly adjacent to the rule windows of r. A rule has $|r|+1$ gap positions, i.e. before its first event, between its constituent events, and after its last event. For example, rule $ab \rightarrow cde$ has 6 gap positions,

$$\underset{g_0}{\wedge}\ a\ \underset{g_1}{\wedge}\ b\ \underset{g_2}{\wedge}\ \rightarrow\ c\ \underset{g_3}{\wedge}\ d\ \underset{g_4}{\wedge}\ e\ \underset{g_5}{\wedge}\ .$$

For each rule r, we test for every gap position g_i, if $e \in \Omega$ is more frequent than expected in its rule windows. Our null hypothesis is

$$H_0 : \sum_{w \in B} \mathbb{1}(e \in g(i, w)) \leq \sum_{w \in B} \Pr(e \in g(i, w))$$

where B is the set of best rule windows of r as used in COVER procedure, $w \in B$ is one such window of rule r, and $g(i, w)$ a function that returns gap i from window w. The probability of e occurring in gap g_i with length $|g(i, w_p)|$ in a rule window w_p, is given by

$$\Pr(e \in g(i, w_p)) = 1 - \left(1 - \frac{supp(\epsilon \rightarrow e)}{||D||}\right)^{|g(i,w_p)|} .$$

To test for statistical significance, we model the expected neighborhood as Poisson binomial distribution (Wang 1993), where trials are the rule windows, and the success probability per trial is decided by the length of a specific gap poisiton. Computing the CDF however is expensive (Le Cam 1960; Volkova 1996). We use the fast normal approximation with continuity correction (Hong 2013) for cases where the number of trials, i.e $supp(r)$, is greater than 10. If less than or equals 10, we simply check if the actual count of occurrences is greater than the expected count by more than one.

If event e is measured to be significantly more frequent in g_i than expected, we generate a new rule by inserting e at the position of g_i in the rule. We show the pseudo-code for the CANDRULES procedure in the appendix.

Mining Good Rule Sets

Finally, we describe SEQRET-MINE for mining good rule sets directly from data. We provide the pseudocode as Algorithm 1. We initialize rule set R with all the singleton rules (line 1). Next, we consider adding candidates based on the rules already in the model (line 3). As we want to generate the most promising candidate rules first, we start with rules with high support and high confidence. We define a greedy EXTEND ORDER as 1) $\uparrow$ $supp(r)$, 2) $\uparrow$ $conf(r)$, 3) $\uparrow$ $|tail(r)|$ and 4) $\uparrow$ $|head(r)|$, where $\uparrow$ indicates that higher values are preferred. For each, we generate a set of candidate rules as described previously, we test them for addition in the order of their p-values (line 4).

We add those rules into the model whose inclusion results in significant reduction in total encoded size (line 5). We use the no-hypercompression inequality (Bloem and de Rooij 2020; Grünwald 2007) to test for significance at level α, writing $\ll_\alpha$ for "significantly less".[1] When adding a candidate rule r' to the model does not improve compression, we test if replacing the rule r we generated it from leads to a better compression (line 7). To ensure we can always describe the data without loss, we never remove singleton rules.

After adding a new rule, SEQRET-MINE performs a pruning step to remove existing rules that may have become redundant or obsolete (line 10). The PRUNE method iterates over the non-singleton rules in the model and removes those whose exclusion reduces the total encoded size. We do so in PRUNE ORDER where we consider rules in order of lowest usage, highest encoded size, and lowest tail length.

We repeat generating candidate rules, adding them, and pruning redundant rules until convergence. Convergence is guaranteed as our score is lower bounded by 0. We discuss the worst-case time complexity in the appendix.

[1] We set $\alpha = 0.05$, corresponding to a minimum gain of 5 bits.

Algorithm 1: SEQRET-MINE

Input: Sequence database D, Significance level α
Output: Rule set R

$R \leftarrow \{\epsilon \rightarrow e \mid \forall e \in \Omega\}$;
do
  for $r \in R$ in EXTEND ORDER **do**
    for $r' \in \text{CANDRULES}(D, r)$ in order of p-value **do**
      if $L(D, R \cup \{r'\}) \ll_\alpha L(D, R)$ **:**
        $R \leftarrow R \cup \{r'\}$;
      else if $r \notin \{\epsilon \rightarrow e \mid \forall e \in \Omega\}$ **and** $L(D, R \cup \{r'\} \setminus \{r\}) \ll_\alpha L(D, R)$ **:**
        $R \leftarrow (R \setminus \{r\}) \cup \{r'\}$;
      if R updated **:**
        $R \leftarrow \text{PRUNE}(D, R)$;
        continue with next r
while R updated;
return R

5 Related Work

Classical methods for sequential pattern mining focus on extracting all frequent patterns and suffer from pattern explosion leading to excessively many, largely redundant, and often spurious results (Fournier-Viger et al. 2017). Mining closed frequent patterns alleviates this, but is sensitive to noise (Yan, Han, and Afshar 2003; Wang and Han 2004).

Pattern set mining avoids the pattern explosion by instead scoring *sets* of patterns. The Minimum Description Length principle has been shown to be a robust criterion for identifying good pattern sets in practice (Galbrun 2022; Cüppers et al. 2024; Tatti and Vreeken 2012). Different pattern languages, scores, and methods have been proposed, such as those allowing gaps (Tatti and Vreeken 2012) and interleaved patterns (Bhattacharyya and Vreeken 2017; Cüppers, Krieger, and Vreeken 2024). Related, but inherently different, is process mining (Van Der Aalst 2016), it is concerned with describing the entire process, whereas we describe conditional dependencies in event sequence.

Classical methods for rule mining in event sequences operate similar to frequent pattern mining, but in addition to the frequency requirement also impose a minimum confidence threshold. Various approaches have been proposed to address different data modalities, such as rules over itemsets ordered by time (Fournier-Viger et al. 2014, 2012), or over events in sequences (Dalmas, Fournier-Viger, and Norre 2017; Zaki 2001; Cule and Goethals 2010). The former generally count the number of sequences containing a rule as its support, whereas the latter use sliding or minimal windows to count rule occurrences within a sequence. Rules can be further categorized into partially ordered rules (Fournier-Viger et al. 2021; Chen et al. 2021) and fully sequential rules (Zaki 2001). These methods discover probabilistic rules, in contrast temporal logic learning methods (Roy et al. 2023) discover strict formal logic rules. As we discover probabilistic rules we focus our discussion on these.

The previous discussed, probabilistic, approaches all consider the quality of individual patterns and hence suffer from pattern explosion. To address this, Fournier-Viger and Tseng propose TNS (Fournier-Viger and Tseng 2013), a method that reports top-k non redundant rules, but its notion of redundancy is unable to detect semantically redundant rules.

Most closely related to SEQRET are existing methods which use MDL to select or mine rules. Existing rule set miners for event sequences either filter down an existing set of rules (Chen et al. 2022), do not allow for gaps (Bourrand et al. 2021) or is a supervised method requiring a target (Cüppers, Kalofolias, and Vreeken 2022). Thus, no existing methods directly addresses the problem we consider.

6 Experiments

In this section we empirically evaluate SEQRET-CANDIDATES and SEQRET-MINE.

We compare to POERMA (Fournier-Viger et al. 2021) and POERMH (Chen et al. 2021) as representative frequent rule miners, to TNS (Fournier-Viger and Tseng 2013) as a top k non-redundant rule set miner, to COSSU (Bourrand et al. 2021) as an MDL-based rule set miner, and to SQS (Tatti and Vreeken 2012) and SQUISH (Bhattacharyya and Vreeken 2017) as MDL-based sequential pattern miners.

As candidate patterns for SEQRET-CANDIDATES we use the output of SQS (Tatti and Vreeken 2012). For TNS we set k to the number of rules in the ground truth, and for POERMA and POERMH, the minimum support and minimum confidence values according to the ground truth. For real datasets where the ground truth is unknown, we set k as the number of rules returned by SEQRET-MINE. Further, we use a minimum support threshold of 10 where feasible, and 20 otherwise. We allow all methods a maximum runtime of 24 hours, except for COSSU, which generally took longer and is allowed upto 48 hours. We provide details in the appendix.

Experiments on Synthetic Data

We first consider data with known ground truth. For a given alphabet size, number of rules, rule sizes, and confidence, we first generate a rule set R by selecting events from alphabet Ω uniformly at random, with replacement, to form the rule.

Data Generation Given a randomly generated rule set R, we next generate an event sequence S. We first generate background noise by sampling uniformly at random from Ω. Next, we plant patterns, i.e. empty-head rules in R by sampling them unifromly and writing to S at random positions. We make sure to not overwrite existing rules. Finally, we go over the generated sequence and wherever a non-empty-head rule is triggered, we sample as per the desired rule confidence whether the trigger is a hit or a miss. If it is a hit, we sample the delay and insert the corresponding rule tail. We provide further details in the appendix. Unless stated otherwise, we generate sequences of length $10\,000$ over alphabets of size 500, rule sets of size 20 with rule confidence 0.75. We generate 20 datasets per configuration in each experiment.

Evaluation Metric As evaluation metric, we consider the $F1$ score. To reward partial discovery, we base the recall and precision on the similarity between discovered rules and

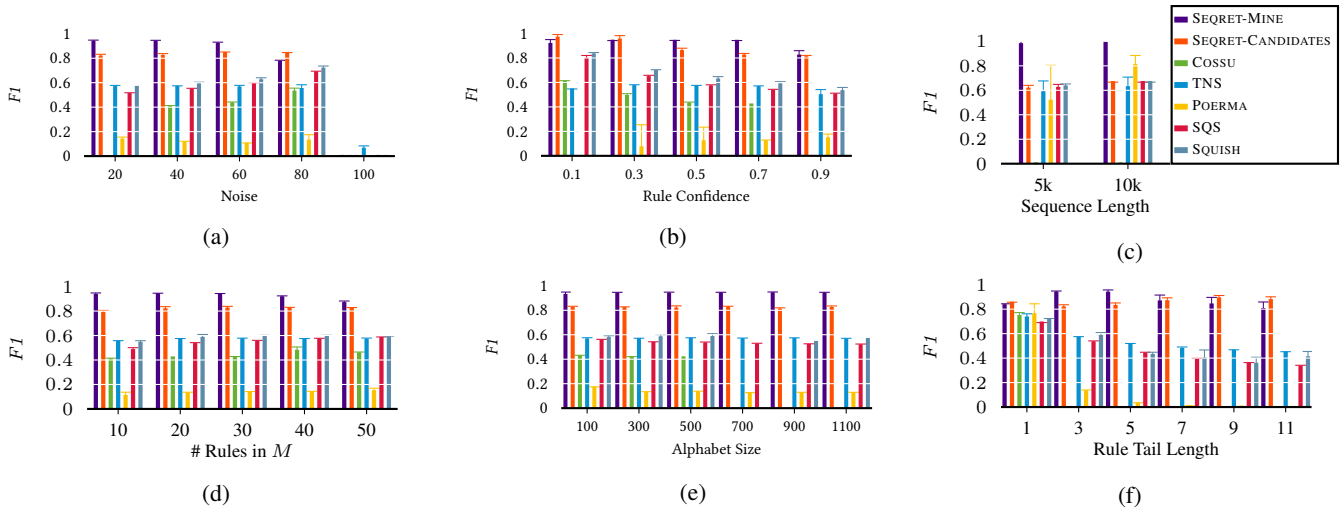

Figure 3: [Higher is better] $F1$ scores for synthetic data. We observe that SEQRET is robust against (a) high noise, (b) low rule confidence, (d) varying number of true rules, (e) large alphabets, and (f) varying rule tail lengths. In (c) we evaluate rule recovery where heads and tails are only as frequent as by chance, only SEQRET-MINE still picks up the ground truth reliably.

ground truth rules, computed using the Levenshtein edit distance without substitution, i.e the longest common subsequence distance (Navarro 2001). The similarity between two rules is a weighted average of the simialrity between their respective heads, tails and complete rules. We then compute recall and precision using the similarity scores following Cüppers and Vreeken (2020). We provide full details of the evaluation metric in the appendix.

Destructive Noise We first evaluate robustness of SEQRET against destructive noise. To this end, we add noise to generated data by flipping individual events $e \in S$ with probability betwee 0.2 and 1. We show the results in Figure 3a. We observe that SEQRET is robust against noise and recovers the ground truth well even at 80% noise. At 100% noise, there is no structure in the data and all methods except TNS correctly discovers no rules. In all experiments, POERMA performs better than or comparable to POERMH, we hence omit POERMH from results. Further, as SQUISH regularly crashes, we report averages over finished runs only.

Rule Confidence Next, we evaluate recovery under different rule confidence levels. We vary the rule confidence from 0.1 to 0.9 and show the results in Figure 3b. We observe that SEQRET is robust against low confidence rules.

Random Rule Triggers We evaluate whether SEQRET recovers conditional dependencies even when rule heads and tails are infrequent. To test this, we generate data where no rule heads are planted, instead they occur only by chance. To ensure that the rule heads indeed do occur, we limit its size to 1. We then insert rule tails wherever the corresponding rules have triggered. We also limit the size of the tails to 1 to ensure they do not stand out as patterns by themselves. We show the results in Figure 3c. SEQRET-MINE consistently recovers the rules while SEQRET-CANDIDATES performs worse because SQS does not find good seed patterns.

Scaling Parameters Next, we evaluate how SEQRET performs for ground truth models, alphabets, resp. rule tails of different sizes. First, we consider data with 10 to 50 ground truth rules for which we show the results in Fig. 3d. Next, we vary the alphabet from 100 to 1000 unique events and report the results in Fig. 3e Finally, we vary the length of the rule tails from 1 to 11 and show the results in Figure 3f. We observe for all three scenarios that SEQRET recovers the ground truth well and visibly outperforms the competition.

Experiments on Real Datasets

Next, we evaluate SEQRET on real world data.

Datasets We use nine datasets from different domains. *JMLR* and *Presidential* are two text datasets (Tatti and Vreeken 2012). *POS* contains parts-of-speech tags for a book (Pokou, Fournier-Viger, and Moghrabi 2016). *Ordonez* and *Lifelog* contain daily activities of a person logged over several days (Ordónez, De Toledo, and Sanchis 2013). *Rolling Mill* contains process logs from a steel manufacturing plant (Wiegand, Klakow, and Vreeken 2021). *Sepsis* contains treatment logs for sepsis cases from a hospital (Wiegand, Klakow, and Vreeken 2022). *Ecommerce* contains purchase data of users of an online store spanning across 7 months, obtained from Kaggle. Finally, *Lichess* contains sequences of moves from online chess games. Further details are provided in the appendix. In Table 1 we provide statistics on datasets, as well as the results for different methods.

General Observations Overall, we observe that frequency-based methods like POERMA and POERMH discover a high number of rules, making interpretation difficult to impossible. TNS produces largely redundant rules. COSSU is limited by its restrictive rule language and discovers very few rules. SQS is a sequential pattern miner that identifies meaningful patterns, but does not capture conditional dependencies that SEQRET successfully models.

			SEQRET-CANDS			SEQRET-MINE			SQS		POERMA	POERMH	TNS	COSSU	
Dataset	$\|\|D\|\|$	$\|\Omega\|$	$\|P\|$	$\|R\|$	$\%L$	$\|P\|$	$\|R\|$	$\%L$	$\|P\|$	$\%L$	$\|R\|$	$\|R\|$	$\|R\|$	$\|R\|$	$\%L$
JMLR	14501	1920	62	9	1.9	2	203	3.3	116	1.6	127	1282442	205	–	–
Presidential	62010	3973	30	4	0.5	2	129	1.0	58	0.4	57	90552	131	–	–
POS	45531	36	65	6	18.4	38	33	18.1	160	12.6	–	–	71	5	-0.5
Ordonez	739	10	2	0	11.5	1	5	16.9	2	11.5	95923	113337	6	2	-2.4
Lifelog	40520	78	36	6	8.9	16	79	10.0	59	6.5	2001521	1932296	95	5	-1.7
Rollingmill	18416	446	158	50	52.2	9	313	56.3	247	50.5	–	–	247	46	20.6
Sepsis	13114	11	19	5	35.7	11	14	31.2	42	23.5	849299	–	20	9	-1.1
Ecommerce	30875	127	77	10	13.7	89	130	27.9	95	13.6	43513	231824	219	6	-0.1
Lichess	20012	2273	81	18	2.4	11	326	4.6	113	2.1	4326	2068	348	–	–

Table 1: Results on real-world data. We report the number of discovered patterns (non-empty-head rules) P and rules R. For SEQRET, SQS and COSSU, we report the percentage of bits saved against the SEQRET null model as $\%L$. Failed runs, e.g. because of excessive runtime (COSSU) or out-of-memory errors (POERMA and POERMH), are indicated by '–'.

This difference is evident when comparing the compression achieved by different methods: SQS compresses the data less effectively than SEQRET, likely because SEQRET is more expressive. SEQRET-CANDIDATES improves upon SQS but still compresses worse than SEQRET-MINE.

Case Studies Next, we present illustrative examples to highlight how results from SEQRET differ from those of state-of-the-art methods. Consider a phrase from the *JMLR* dataset, '*support vector machine*', that all methods identify in some form. SEQRET-MINE discovers the rules $\langle \epsilon \rightarrow$ *support, vector*$\rangle$ and $\langle$*support, vector* $\rightarrow$ *machine*$\rangle$, which expresses that *support*, *vector* is a pattern, and that whenever it occurs it increases the probability of but is *not necessarily* followed by *machine*. In contrast, SEQRET-CANDIDATES and SQS both treat the entire phrase as a single pattern, $\langle$*support, vector, machine*$\rangle$, failing to capture the independent existence of *support, vector*. On the other end of the spectrum, POERMA and TNS discover 12 resp. 14 mainly redundant rules involving either *support* or *vector*.

In the *POS* dataset, SEQRET discovers common sentence structures, e.g. the pattern $\langle \epsilon \rightarrow$ *determiner*, *cardinal number*$\rangle$ capturing phrases such as "the first". SEQRET also captures rules, e.g. $\langle$*to*, *verb-base-form* $\rightarrow$ *personal-pronoun*$\rangle$ and $\langle$*to*, *verb-base-form* $\rightarrow$ *possessive-pronoun*$\rangle$, correctly identifying how either personal pronouns or possessive pronouns tend to follow phrases like "to tell" or "to give". COSSU, the method closest to our approach, fails to find any of the rules discussed. Meanwhile, SQS finds these structures as several independent patterns disregarding the conditional dependencies.

For the *Lichess* dataset, SEQRET finds the well-known "King's Pawn Game" opening as $\langle \epsilon \rightarrow$ *white:e4*, *black:e5*$\rangle$. It further discovers 12 rules with *white:e4*, *black:e5* as the head, capturing the different variations that often follow, e.g. the rule $\langle$*white:e4*, *black:e5* $\rightarrow$ *white:Nf3*$\rangle$, shown in Figure 4a. Diving deeper into results, SEQRET discovers rules involving "King's side castling" (shown in Figure 4b) Examples are $\langle$*black:O-O* $\rightarrow$ *black:Re8*$\rangle$ and $\langle$*black:O-O* $\rightarrow$ *white:Qe2*$\rangle$. The former captures black moving its rook to e8, a position originally occupied by the King and made available only after castling. The latter captures white moving its queen to e2 following black castling, as a deterrence to black rook. The frequency-based methods find between 128 and 1370 mostly redundant rules involving castling.

(a) (b)

Figure 4: In (a) we show the rule $\langle$*white:e4 (red arrow), black:e5 (green arrow)* $\rightarrow$ *white:Nf3 (blue arrow)*$\rangle$. In (b) we show black castling, *black:O-O*.

7 Conclusion

We considered the problem of mining a succinct and non-redundant set of rules from long event sequences. We formalized the problem in terms of the MDL principle and presented SEQRET-CANDIDATES and SEQRET-MINE algorithms. We evaluated both on synthetic and real-world data. On synthetic data we saw that SEQRET recovers the ground truth well and is robust against noise, low rule confidence, varying alphabet resp. rule set sizes. On real-world data SEQRET provided insights that existing methods could not.

As future work, an interesting direction is a differentiable approach to sequential rule mining. Existing work (Gao et al. 2025) focuses on the supervised classification setting, but not unsupervised rules that describe conditional dependencies. Additional, we consider it highly interesting to study the causal aspects of sequential rules. Our approach lends itself to a causal framework by mapping the rule heads and tails to temporal variables and re-modeling the rules as structural equations involving these variables. Moreover, the MDL principle as used in this paper nicely maps to the Algorithmic Markov Condition (Janzing and Schölkopf 2010) criterion to choose a plausible causal model.

References

Agrawal, R.; and Srikant, R. 1995. Mining sequential patterns. In *ICDE*, 3–14. Los Alamitos, CA, USA: IEEE Computer Society.

Bhattacharyya, A.; and Vreeken, J. 2017. Efficiently Summarising Event Sequences with Rich Interleaving Patterns. In Chawla, N. V.; and Wang, W., eds., *Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017*, 795–803. SIAM.

Bloem, P.; and de Rooij, S. 2020. Large-scale network motif analysis using compression. *Data Mining and Knowledge Discovery*, 34: 1421–1453.

Bourrand, E.; Galarraga, L.; Galbrun, E.; Fromont, E.; and Termier, A. 2021. Discovering Useful Compact Sets of Sequential Rules in a Long Sequence. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 1295–1299. Los Alamitos, CA, USA: IEEE Computer Society.

Chen, X.; Gan, W.; Wan, S.; and Gu, T. 2022. MDL-based Compressing Sequential Rules. *ArXiv*, abs/2212.10252.

Chen, Y.; Fournier-Viger, P.; Nouioua, F.; and Wu, Y. 2021. Mining partially-ordered episode rules with the head support. In *Big Data Analytics and Knowledge Discovery: 23rd International Conference, DaWaK 2021, Virtual Event, September 27–30, 2021, Proceedings 23*, 266–271. Springer.

Cule, B.; and Goethals, B. 2010. Mining association rules in long sequences. In *Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part I 14*, 300–309. Springer.

Cüppers, J.; Kalofolias, J.; and Vreeken, J. 2022. Omen: discovering sequential patterns with reliable prediction delays. *Knowl. Inf. Syst.*, 64(4): 1013–1045.

Cüppers, J.; Krieger, P.; and Vreeken, J. 2024. Discovering Sequential Patterns with Predictable Inter-event Delays. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, 8346–8353. AAAI Press.

Cüppers, J.; Schoen, A.; Blanc, G.; and Gimenez, P.-F. 2024. FlowChronicle: Synthetic Network Flow Generation through Pattern Set Mining. *Proceedings of the ACM on Networking*, 2(CoNEXT4): 1–20.

Cüppers, J.; and Vreeken, J. 2020. Just Wait For It... Mining Sequential Patterns with Reliable Prediction Delays. In Plant, C.; Wang, H.; Cuzzocrea, A.; Zaniolo, C.; and Wu, X., eds., *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, 82–91. IEEE.

Dalmas, B.; Fournier-Viger, P.; and Norre, S. 2017. TWINCLE: A constrained sequential rule mining algorithm for event logs. *Procedia computer science*, 112: 205–214.

Fournier-Viger, P.; Chen, Y.; Nouioua, F.; and Lin, J. C.-W. 2021. Mining partially-ordered episode rules in an event sequence. In *Intelligent Information and Database Systems: 13th Asian Conference, ACIIDS 2021, Phuket, Thailand, April 7–10, 2021, Proceedings 13*, 3–15. Springer.

Fournier-Viger, P.; Faghihi, U.; Nkambou, R.; and Nguifo, E. M. 2012. CMRules: Mining sequential rules common to several sequences. *Knowledge-Based Systems*, 25(1): 63–76.

Fournier-Viger, P.; Gueniche, T.; Zida, S.; and Tseng, V. S. 2014. ERMiner: sequential rule mining using equivalence classes. In *Advances in Intelligent Data Analysis XIII: 13th International Symposium, IDA 2014, Leuven, Belgium, October 30–November 1, 2014. Proceedings 13*, 108–119. Springer.

Fournier-Viger, P.; Lin, J. C.-W.; Kiran, R. U.; Koh, Y. S.; and Thomas, R. 2017. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1): 54–77.

Fournier-Viger, P.; and Tseng, V. S. 2013. TNS: mining top-k non-redundant sequential rules. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 164–166.

Galbrun, E. 2022. The minimum description length principle for pattern mining: A survey. *Data mining and knowledge discovery*, 36(5): 1679–1727.

Gao, K.; Inoue, K.; Cao, Y.; Wang, H.; and Feng, Y. 2025. Differentiable Rule Induction from Raw Sequence Inputs. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Grünwald, P. 2007. *The Minimum Description Length Principle*. MIT Press.

Hong, Y. 2013. On computing the distribution function for the Poisson binomial distribution. *Computational Statistics & Data Analysis*, 59: 41–51.

Janzing, D.; and Schölkopf, B. 2010. Causal Inference Using the Algorithmic Markov Condition. *IEEE Transactions on Information Theory*, 56(10): 5168–5194.

Le Cam, L. 1960. An approximation theorem for the Poisson binomial distribution.

Li, P.; and Vitányi, M. 1997. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer.

Mannila, H.; and Meek, C. 2000. Global Partial Orders From Sequential Data. In *KDD*, 161–168.

Navarro, G. 2001. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1): 31–88.

Ordónez, F. J.; De Toledo, P.; and Sanchis, A. 2013. Activity recognition using hybrid generative/discriminative models on home environments using binary sensors. *Sensors*, 13(5): 5460–5477.

Pokou, Y. J. M.; Fournier-Viger, P.; and Moghrabi, C. 2016. Authorship attribution using small sets of frequent part-of-speech skip-grams. In *The Twenty-Ninth International Flairs Conference*.

Rissanen, J. 1983. A Universal Prior for Integers and Estimation by Minimum Description Length. *Annals Stat.*, 11(2): 416–431.

Roy, R.; Gaglione, J.-R.; Baharisangari, N.; Neider, D.; Xu, Z.; and Topcu, U. 2023. Learning interpretable temporal properties from positive examples only. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 6507–6515.

Tatti, N. 2009. Significance of Episodes Based on Minimal Windows. In *ICDM*, 513–522.

Tatti, N.; and Vreeken, J. 2012. 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*, KDD '12, 462–470. New York, NY, USA: Association for Computing Machinery. ISBN 9781450314626.

Van Der Aalst, W. 2016. Data science in action. In *Process mining: Data science in action*, 3–23. Springer.

Volkova, A. Y. 1996. A refinement of the central limit theorem for sums of independent random indicators. *Theory of Probability & Its Applications*, 40(4): 791–794.

Wang, J.; and Han, J. 2004. BIDE: Efficient Mining of Frequent Closed Sequences. In *ICDE*, 79–90.

Wang, Y. H. 1993. On the number of successes in independent trials. *Statistica Sinica*, 295–312.

Wiegand, B.; Klakow, D.; and Vreeken, J. 2021. Mining Easily Understandable Models from Complex Event Data. *SIAM International Conference on Data Mining (SDM), SIAM*.

Wiegand, B.; Klakow, D.; and Vreeken, J. 2022. Discovering interpretable data-to-sequence generators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 4237–4244.

Yan, X.; Han, J.; and Afshar, R. 2003. CloSpan: Mining: Closed Sequential Patterns in Large Datasets. In *SDM*, 166–177. SIAM.

Zaki, M. J. 2001. SPADE: An efficient algorithm for mining frequent sequences. *Machine learning*, 42: 31–60.

Appendix

A Supplementary Material: Experiments

Setup

We ran all experiments on an Intel Xeon Gold 6244 @ 3.6 GHz, with 256GB of RAM. For the methods POERMA and POERMH, the JVM max heap size was increased upto 64 GB. Both these methods discover partially ordered rules from long event sequences, albeit with different definitions of rule support. To ensure fair comparison, we constraint our synthetic data generation to rules where constituent events appear in lexicographical order, and re-arrange the partially-ordered rules found to this order. We set a time limit of 24 hours for all methods except COSSU. As COSSU took a very long time to complete across all experiments, we increased the time limit for COSSU alone to 48 hours. Across all synthetic experiments except where the rule tail size was varied, SEQRET-MINE completed within a few seconds to 1 hour max. In experiment varying rule tail size, SEQRET-MINE took upto 3 hours in a few instances indicating that data with highly interleaved rules take up more time in rule search. We report the runtimes on real datasets in Table 2.

Dataset	Runtime
JMLR	4 h
Presidential	8 h
POS	3 h
Lifelog	2 h
Ordonez	3 s
Sepsis	1 h
Ecommerce	11 h
Rollingmill	22 h
Lichess	17 h

Table 2: Runtime of SEQRET-MINE for different datasets.

Synthetic Data Generation

Given an alphabet as input, we first generate a random rule set. We take in as parameters the rule set size, the rule-head size, the rule-tail size and the rule confidences. We also parameterize whether or not the rule heads occur as independent patterns, i.e for a rule $X \to Y$, whether $\epsilon \to X$ exists or not. If $\epsilon \to X$ doesn't exist, then X is only as frequent as expected by chance. Given these parameters, we randomly select events from the alphabet to form the rule heads and the rule tails, and add them to the rule set.

Next, using the rule set as ground truth, we generate the sequence database. We first generate an initial sequence using all the empty-head rules, and then insert the rule tails wherever the non-empty-head rules have triggered. We take in as parameters an initial sequence size and noise percentage. By noise, we mean the events in the sequence that can be covered only using one of the singleton rules. Therefore, given a noise percentage, we uniformly sample from the singleton rules, i.e the alphabet, to generate the stipulated percentage of the initial sequence size. Following this, we uniformly sample from the empty-head non-singleton rules and fill them into random positions to generate the remaining sequence. Finally, we go over the generated sequence, identify rules that have been triggered and insert the corresponding rule tails as per the specified rule confidences.

As for the delays and gaps, we take in as parameters a delay probability, i.e probability with which the data generation algorithm skips positions following a rule trigger, and a gap probability, i.e probability with which the data generation algorithm skips positions within rule tails. Note that the insertion of rule tails will alter the sequence size and the noise percentage. We keep the gap and delay probabilities low at 0.1 and 0.2 respectively. To run SEQRET, we set the *max delay* and the *max gap* both to 2.

Unless stated otherwise, we generate sequences of length $10\,000$ over alphabets of size 500, rule sets of size 20 with rule confidence 0.75. We generate 20 datasets per configuration in each experiment.

Evaluation Metrics

To evaluate a method, we compare the rule set retrieved by it the against the ground truth in terms of recall and precision. To measure the similarity between rules quantitatively, we define a metric based on the LCS distance measure (Navarro 2001). As the LCS distance is upper bounded by the sum of lengths of the patterns and lower bounded by zero, we can compute the similarity between any two patterns A and B as

$$sim\,(A, B) = 1 - \frac{d\,(A, B)}{|A| + |B|}\ ,$$

where $d\,(A, B)$ refers to the LCS distance. Further, we define the similarity measure between two rules $A \to B$ and $C \to D$ where either A or C is non-empty as

Algorithm 2: COVER

Input: Sequence database D, rule set R
Output: Cover C

1 $C \leftarrow \{\}$;
2 $W \leftarrow \{\text{BESTRULEWIN}(r, D) \mid \forall r \in R\}$;
3 **while** $\exists S_t \in D$ where, $\exists e \in S_t$ not covered by C **do**
4 $\quad w \leftarrow$ next $w \in W$ in WINDOW ORDER;
5 $\quad W \leftarrow W \setminus \{w\}$;
6 $\quad$ **if** $\nexists z \in C$ that **conflicts** with w **:**
7 $\quad\quad C \leftarrow C \cup \{w\}$;
8 $\quad$ **else**
9 $\quad\quad W \leftarrow W \cup \{\text{NEXTBESTWIN}(w, C, D)\}$;
10 **return** C

$$\begin{aligned} sim\,(A \to B, C \to D) &= 0.5 * sim\,(AB, CD) \\ &+ 0.25 * sim\,(A, C) \\ &+ 0.25 * sim\,(B, D) \end{aligned}$$

Using this similarity measure, we calculate recall and precision. To compute recall, we sum up the similarity measure of each true rule with respect to the best matching mined rule, and normalize the sum. Formally, given a true model T and a mined model M, we define

$$recall\,(T, M) = \frac{\sum_{t \in T} \max_{m \in M} sim\,(t, m)}{|T|}\ .$$

Similarly, to compute precision, we use the similarity measures of the mined rules with respect to their best matching true rules. To penalize redundancy in the mined model, we should ideally choose a non-redundant subset of the mined model that maximizes the total similarity measure. However, this is not trivial to solve. Therefore, we resort to a heuristic measure as proposed in OMEN (Cüppers and Vreeken 2020), and choose the top $|T|$ maximum similarity measures from the mined model. Given true model T and mined model M,

$$precision\,(T, M) = \frac{\sum_{m' \in M'} m'}{|M|}\ ,$$

where $M' \subseteq \{\max_{t \in T} sim\,(t, m) \mid \forall m \in M\}$ such that M' contains the max $|T|$ elements. Finally, we use precision and recall to compute the F1 score.

Real-world Datasets

We use nine datasets drawn from different domains. We consider two text datasets, *JMLR*, containing abstracts from the JMLR journal, and *Presidential*, containing addresses delivered by American presidents (Tatti and Vreeken 2012). *POS* contains sequences of parts-of-speech tags obtained by the Stanford NLP tagger on the book "History of Julius Caesar" (Pokou, Fournier-Viger, and Moghrabi 2016). *Ordonez* (Ordónez, De Toledo, and Sanchis 2013) and *Lifelog* contains the daily activities logged by a person over several days. *Rolling Mill* contains the process logs from a steel manufacturing plant (Wiegand, Klakow, and Vreeken 2021). *Sepsis* contains the logs of sepsis cases from a hospital (Wiegand, Klakow, and Vreeken 2022). *Ecommerce* contains the purchase data from an online store, for several users across 7 months, obtained from Kaggle. It was transformed into a sequence database of items bought, ordered by the time of purchase, with one sequence per user. Finally, *Lichess* contains sequences of moves from online chess games, also obtained from Kaggle.

B Supplementary Material: Algorithms

Cover Algorithm

We give the pseudocode of COVER as Algorithm 2. We start by initializing cover C with the empty set and window set W with for each rule the best rule windows per trigger (lines 1-2). We then greedily add rule windows to C in order of WINDOW ORDER. If a window conflicts with an already selected window (line 6), we skip it and search for the next best rule window for the corresponding trigger and add it to W (line 9). We continue this process until all events in D are covered. To avoid evaluating hopeless windows, we limit ourselves to those within a user-set *max delay* ratio and *max gap* ratio. We provide further details and pseudo code for BESTRULEWIN and NEXTBESTWIN procedures below.

The worst case time complexity of COVER depends on the number of rules in R, total number of events in D, and the lengths of the heads and tails per rule. In Appx. C we show the complexity of COVER is $\mathcal{O}(|R| \cdot ||D||(h + t^3 + t \log_2(|R| \cdot ||D||t))$, where h and t are the max head resp. tail length.

Figure 5: An illustration of potential rule windows for the rule $ab \rightarrow cd$. We pick the nearest minimal window of the rule tail as our preferred window.

Rule Windows

Algorithm 3: BESTRULEWIN

Input: $rule, D$
Output: $windows$
$windows \leftarrow \{\}$;
$triggers \leftarrow \{S[i,j] \mid S \in D,\ (j-i+1) - |head(rule)| \leq$ *max gap* $* |head(rule)|\}$,
  where $S[i,j]$ is a minimal window of $head(rule)$;
for $S[i,j] \in triggers$ **do**
  $k, l \leftarrow$ indices of first minimal window $S[k,l]$ of $tail(rule)$ such that $(k-j-1) \leq$ *max delay* $* |tail(rule)|$ and $(l-k+1) - |tail(rule)| \leq$ *max gap* $* |tail(rule)|$;
  if $S[k,l]$ exists **:**
    $windows \leftarrow windows \cup \{(rule, S[i,j;k,l])\}$;
return $windows$

Algorithm 4: NEXTBESTWIN

Input: $win, cover, D$
Output: win'
/* win contains a pointer to the rule ($win.rule$) and the window in the format $S[i,j;k,l]$ */
$k', l' \leftarrow$ indices of first minimal window $S[k',l']$ of $tail(win.rule)$ such that $\forall e \in \{e \mid S[k',l']$ covers $e\}, \nexists w \in cover$ where w covers e and $(k'-j-1) \leq$ *max delay* $* |tail(win.rule)|$ and $(l'-k'+1) - |tail(win.rule)| \leq$ *max gap* $* |tail(win.rule)|$;
$win' \leftarrow (win.rule, S[i,j;k',l'])$;
return win'

A rule window $S[i,j;k,l]$ captures the positions at which a rule occurs in a sequence S. Here, $S[i,j]$ is the window within which the rule head occurs and $S[k,l]$ is the window within which the rule tail occurs such that $j \leq k$. To avoid double counting and minimize gaps, we use minimal windows to identify the rule head patterns that trigger the rules. But what about the rule windows? For each rule head, we prefer the nearest minimal window of the rule tail pattern to complete the rule window. We prefer minimal windows as they minimize the gaps and treat the rule tail as a cohesive unit. If multiple minimal windows of the rule tail exist following the trigger, then we pick the nearest one so as to minimize the delay. Further, we restrict the search to windows that follow a user-set *max delay* ratio such that $k - j - 1/|tail(r)| \leq$ *max delay*, and a *max gap* ratio such that $l - k + 1/|tail(r)| \leq$ *max gap* and $j - i - 1/|head(r)| \leq$ *max gap*.

Algorithm 3 outlines the pseudo-code to find the best rule windows for a given rule.

The BESTRULEWIN method, however, assumes that none of the events forming the preferred windows for different rules have already been covered. As we start covering the sequence with these windows, however, it may happen that, at some point, events that are common to multiple rule tails have already been covered. In such cases, we look for the next best rule window. The next best rule window is the nearest minimal window of the rule tail following the trigger such that the events forming the rule tail are not already covered. Algorithm 4 outlines the pseudo-code for the NEXTBESTWIN method.

As an example, consider the sequence $\langle a, a, b, b, c, c, d, d\rangle$ in Figure 5 and rule $ab \rightarrow cd$. The minimal window $S[1,2]$ captures the rule head that triggers the rule. Assuming none of the events in the sequence have already been covered, we pick $S[5,6]$ as the rule tail window. The alternate windows for the rule tail are considered if and only if positions 5 or 6 have already been covered by other rules.

Algorithm 5: SEQRET-CANDIDATES

Input: Sequence database D, set of patterns F
Output: Rule set R

$R \leftarrow \{\epsilon \rightarrow e \mid \forall e \in \Omega\}$
for $p \in F$ ordered descending by $L(D, F \setminus \{p\}) - L(D, F)$ **do**
  $r \leftarrow \arg\max_{r' \in \text{SPLIT}(p)} L(D, R) - L(D, R \cup \{r'\})$
  if $L(D, R \cup \{r\}) < L(D, R)$ **:**
    $R \leftarrow M \cup \{r\}$;
return R;

Algorithm 6: SPLIT

Input: Pattern p
Output: Set of rules R

$R \leftarrow \emptyset$
$i \leftarrow 0$
while $i < |p|$ **do**
  $R \leftarrow R \cup (p[0, i], p[i + 1, |p|])$
  $i \leftarrow i + 1$
return R

Seqret-Candidates Algorithm

The pseudocode for SEQRET-CANDIDATES is given in Algorithm 5. We first initialize the rule set with all singleton rules to ensure that the data can be encoded losslessly. We then iterate over the candidate patterns in descending order of their contribution to compression (Tatti and Vreeken 2012). We split each pattern into candidate rules – for example, pattern *abc* generates candidate rules $\epsilon \rightarrow abc$,$a \rightarrow bc$, and $ab \rightarrow c$ – and choose the one whose addition to the rule set minimizes the total description length, with $L(D|R)$ determined by the COVER procedure. We keep it in our model if it improves the score. We iterate this process for all candidate patterns. The Algorithm 6 shows the pseudo-code for the SPLIT procedure. The run time is dominated by the number of cover computations. We have to compute a new cover for each candidate rule to test, and each pattern p can be split into $|p|$ candidate rules. For a given set of sequential patterns F, the complexity of SEQRET-CANDIDATES is then $\mathcal{O}(|F|(\max_{p \in F} |p|))$.

CandRules Algorithm

The Algorithm 7 shows the pseudo-code for the CANDRULES procedure.

Algorithm 7: CANDRULES

Input: $D, \Omega, rule$
Output: $candidates$

$candidates \leftarrow \{\}$;
$windows \leftarrow$ BESTRULEWIN$(rule, D)$;
for $position \in \{h_0, h_1, ..., h_{|head(rule)|}\} \cup \{t_0, t_1, ..., t_{|tail(rule)|}\}$ **do**
  /* h represents rule head and t represents rule tail */
  for $e \in \Omega$ **do**
    $count \leftarrow |\{w \in windows \mid w$ contains e in the gap at $position\}|$;
    $p_{e^c} \leftarrow 1 - \frac{supp(\epsilon \rightarrow e)}{|D|}$; /* e^c refers to the complement of e */
    $expected \leftarrow |windows| - \sum_{w \in windows} (p_{e^c})^{|g_w|}$;
    /* g_w refers to the gap in w at $position$ */
    if $count$ significantly greater than $expected$ **:**
      /* see section 4 for details of significance test */
      $candidates \leftarrow candidates \cup \{\text{INSERT}(rule, e, position)\}$ /* INSERT inserts e to $rule$ at $position$ */
return $candidates$

Prune Algorithm

The Algorithm 8 shows the pseudo-code for the PRUNE procedure.

Algorithm 8: PRUNE

Input: D, R
Output: pruned R

```
for r ∈ R ordered by PRUNE ORDER do
    if r is not a singleton rule :
        if L(D, R \ {r}) < L(D, R) :
            R ← R \ {r};
return R
```

C Supplementary Material: Time Complexity Analysis

Time Complexity of the Rule-Set Mining Problem

To evaluate the time complexity of the problem, let us split the problem into two parts - one, to find the optimal cover given a rule set, and two, to find the optimal rule set. For simplicity, let us assume a single long sequence S in the database, drawn from the alphabet Ω.

Given a rule set R, we know that a cover is a many-to-one mapping between the events in S to rules in R. In other words, it is a permutation with replacement of the rules in R over $|S|$ events. Therefore, we can compute the number of possible covers as $|R|^{|S|}$. The worst-case time complexity of the first part of our problem is

$$\mathcal{O}(|R|^{|S|}) \ .$$

Now let us compute the time complexity of the second part of our problem. The longest rule that can occur in S would be of length $|S|$. The total number of rules possible would be the sum of the number of rules possible per size, with size ranging from 1 to $|S|$. Considering that rules can be built from sequential patterns, let us first compute the number of sequential patterns possible for size k. A sequential pattern is a permutation of the alphabet with replacement. Therefore, for size k we get $|\Omega|^k$ possible sequential patterns. Now, including the possibility of an empty-head, we can choose k positions to split the pattern into a rule head and a rule tail. Thus, for size k, we can compute the number of rules possible as $k * |\Omega|^k$. The total number of rules possible is then given by

$$\sum_{k=1}^{|S|} k * |\Omega|^k \ .$$

A rule set being a subset of all possible rules, we can compute the number of possible rule sets as the size of the power-set. Since we retain the singleton rules in every possible rule set, we can compute the number of valid rule sets as $2^{\sum_{k=1}^{|S|} k*|\Omega|^k - |\Omega|}$. The worst-case time complexity of the second part of our problem is then given by

$$\mathcal{O}(2^{\sum_{k=1}^{|S|} k*|\Omega|^k - |\Omega|}) \ \approx \ \mathcal{O}(2^{|\Omega|^{|S|}}) \ .$$

Time Complexity of SEQRET-MINE

Let us now analyze the time complexity of our solution. We will analyze each part of the problem separately. We consider the worst-case where all the events in the database occur as a single long sequence S. The set of distinct events form the alphabet Ω.

Time Complexity of COVER Given a rule set R, we first compute the complexity of finding the rule windows. To do so, the method looks for all rule triggers and for each rule trigger, finds the nearest minimal window of the rule tail. Iterating over S and looking for triggers of each rule $r \in R$ results in a worst-case time complexity of $\mathcal{O}(|S| * \sum_{r \in R} |head(r)| * \textit{max gap})$. Ignoring the *max gap* parameter that stays constant irrespective of the problem size and upper bounding the size of any rule head by $\max_{r \in R} |head(r)|$, denoted by max_head_size, we can rewrite the same as

$$\mathcal{O}(|S| * |R| * max\_head\_size) \ .$$

To complete the rule window for each trigger, BESTRULEWIN next looks for the nearest minimal window of the rule tail until the maximum allowed delay. Given a trigger, looking for a rule tail, for any rule r, requires computational time in the order of $\mathcal{O}(|tail(r)| * \textit{max delay} * |tail(r)| * \textit{max gap})$. Once again ignoring the constant parameters and using max_tail_size to upper

bound the size of a rule tail, we can rewrite this as $\mathcal{O}(max\_tail\_size^2)$. Thus, we can compute the worst-case time complexity of BESTRULEWIN as

$$\mathcal{O}\left(|S| * |R| * (max\_head\_size + max\_tail\_size^2)\right) .$$

As triggers are bounded by minimal windows, and only one minimal window can exist per starting or ending position, the number of triggers per rule is upper bounded by $|S|$. Since BESTRULEWIN finds only the one nearest minimal window of the rule tail for each trigger, we can upper bound the total number of rule windows returned to $|R| * |S|$. The next step in COVER is to sort the rule windows in WINDOW ORDER. This incurs a time complexity of

$$\mathcal{O}(|R| * |S| * \log_2(|R| * |S|)) .$$

The final step in COVER is to consider each rule window in the sorted order and cover the sequence. However, there could arise cases where the considered rule windows are in conflict with previous rule windows which already covered the same events. This in turn leads to the execution of NEXTBESTWIN. Each time NEXTBESTWIN is called for a rule trigger, it looks for the next nearest minimal window of the rule tail until the maximum allowed delay. If such a rule window is found, then it is added to the sorted list of rule windows maintaining the order. A single call to NEXTBESTWIN for a trigger of rule r, in the worst-case, incurs computational time in the order of $|tail(r)|^2$ to find the next nearest minimal window of the rule tail (ignoring the parameters *max gap* and *max delay*). Suppose W denotes the sorted list of rule windows at any point in time. Once (if) the next best rule window is found, the method incurs additional computational time in the order of $\log_2(|W|)$ to find the position of insertion using a binary search.

The question is how many such calls to NEXTBESTWIN could happen in the worst case. We could also upper bound the size of W by the same value. To answer this question, let us consider when NEXTBESTWIN is called. It is called whenever an event that participates in a rule window is already covered by a previous rule window. From the point of view of a rule trigger, each event following it until a limit determined by *max delay* and *max gap* times the rule tail, can potentially participate in a rule window. The SEQRET starts with one such rule window and looks for the next best rule window if and only if any of the participating events is already covered. Further, the next best rule window omits the previously covered events. Therefore, the maximum number of times NEXTBESTWIN gets called is limited by the number of events following the rule trigger, given by $|tail(r)| * (\textit{max delay} + \textit{max gap} + 1)$. Ignoring the constants for the purpose of complexity analysis, we can rewrite it as $|tail(r)|$. Over all triggers for all rules, we can then compute the worst-case time complexity of finding the next best windows and adding them to the sorted list of rule windows as

$$\mathcal{O}\left(\sum_{r \in R} |S| * |tail(r)| * \left(|tail(r)|^2 + \log_2(|W|)\right)\right) ,$$

where W is the list of rule windows. Once again, as worst-case, we use max_tail_size to rewrite the same. Further, we can limit the size up to which W can grow by the number of times NEXTBESTWIN gets called, i.e in the order of $|R|*|S|*max\_tail\_size$. Putting it all together, we find the total computational time for all calls to NEXTBESTWIN to be in the order of

$$\mathcal{O}(|R| * |S| * max\_tail\_size * \log_2(|R| * |S| * max\_tail\_size) + |R| * |S| * max\_tail\_size^3)$$

Finally, using the ordered list of rule windows W, we cover the sequence S. As singleton rules are also included in R, it is guaranteed to cover the entire sequence in one iteration over all the rule windows in W (in practice, it will be much lesser as many events get covered by a single rule window). Therefore, we can compute the worst-case time complexity to loop over the list of rule windows and cover the sequence S as

$$\mathcal{O}(|R| * |S| * max\_tail\_size) .$$

Thus, from equations C, C, C and C, we can compute the total worst-case time complexity of COVER as

$$\begin{aligned}
&\mathcal{O}(|R| * |S| * (max_head_size + max_tail_size^2) \\
&\quad + |R| * |S| * \log_2(|R| * |S|) \\
&\quad + |R| * |S| * max_tail_size \\
&\quad + |R| * |S| * max_tail_size * \log_2(|R| * |S| * max_tail_size) \\
&\quad + |R| * |S| * max_tail_size^3) .
\end{aligned}$$

Considering only the dominating terms, we get

$$\begin{aligned}
\mathcal{O}(|R| * |S| * (&max_head_size \\
&+ max_tail_size^3 \\
&+ max_tail_size * \log_2(|R| * |S| * max_tail_size)) .
\end{aligned}$$

Time Complexity of SEQRET-MINE Next, we analyze the time complexity of the greedy miner, SEQRET-MINE. Let us consider a single iteration of the miner. Let R' be the candidate rule set at that time point. Then, SEQRET-MINE grows the rule set by searching for a new rule that improves the encoding cost by extending each rule, at each position, with their significant neighbors. As worst-case, let us assume that the miner had to search over all rules, at all positions. Further, let us assume that all events in the alphabet Ω are significant (although this is impossible). To simplify the computations, we use $\max_{r' \in R'} |head(r')|$ as the max_head_size and $\max_{r' \in R'} |tail(r')|$ as the max_tail_size to upper bound the lengths of $head(r')$ and $tail(r')$ for any $r' \in R'$. Then, the computational time of the search will be in the order of $\mathcal{O}(|R'| * (max\_head\_size + max\_tail\_size) * |\Omega|)$. Once a new rule is added to the rule set, SEQRET-MINE tries to prune the rule set by removing each non-singleton rule. The computational time required by PRUNE will be in the order of $|R'| - |\Omega|$. Considering only the dominating term, we can thus conclude the worst-case time complexity of each iteration as

$$\mathcal{O}(|R'| * (max\_head\_size + max\_tail\_size) * |\Omega|) \ ,$$

where R' denotes the candidate rule set at that time point. Next, we try to analyze the number of iterations possible before the algorithm converges.

We know that in each iteration, SEQRET-MINE adds a new rule to the current rule set only if the addition improves the encoding cost. Similarly, a rule is pruned from the current rule set only if the removal improves the encoding cost. If the encoding cost cannot be improved anymore, then the algorithm halts. In other words, SEQRET-MINE will never revert back to a rule set from which it grew in the past. Therefore, the maximum number of iterations is upper bounded by the number of unique rule sets possible. In Section C, we saw that the number of possible rule sets is in the order of $\mathcal{O}(2^{|\Omega|^{|S|}})$.

Caching for Faster Runtime In practice, however, we observe the number of rule sets considered by the greedy approach to be much smaller. Further, we cache the rule windows found for each rule as and when they are first encountered. Therefore, if the same rule is present in a future rule set, we do not recompute the rule windows. In other words, BESTRULEWIN is invoked only once per rule. The same is true for CANDRULES. We cache the neighbors found for each rule as and when they are first encountered. As a result, the time complexity in practice would be much lower, even if SEQRET-MINE attempted the worst-case possibility of all unique rule sets before converging. Further, we do not reconsider rules once pruned in the future iterations.