Discovering Interpretable Data-to-Sequence Generators

Boris Wiegand[1,2], **Dietrich Klakow**[2], **Jilles Vreeken**[3]
[1] SHS – Stahl-Holding-Saar, Dillingen, Germany
[2] Saarland University, Saarbrücken, Germany
[3] CISPA Helmholtz Center for Information Security, Germany
boris.wiegand@stahl-holding-saar.de, dietrich.klakow@lsv.uni-saarland.de, jv@cispa.de

Abstract

We study the problem of predicting an event sequence given some meta data. In particular, we are interested in learning easily interpretable models that can accurately generate a sequence based on an attribute vector. To this end, we propose to learn a sparse event-flow graph over the training sequences, and statistically robust rules that use meta data to determine which paths to follow. We formalize the problem in terms of the Minimum Description Length (MDL) principle, by which we identify the best model as the one that compresses the data best. As the resulting optimization problem is NP-hard, we propose the efficient ConSequence algorithm to discover good event-flow graphs from data.

Through an extensive set of experiments including a case study, we show that it ably discovers compact, interpretable and accurate models for the generation and prediction of event sequences from data, has a low sample complexity, and is particularly robust against noise.

Introduction

Real-world event sequences are often accompanied by additional meta data. For example, event logs of manufacturing processes contain sequences of production steps with product properties and attributes by the customer's order. Usually, there is a relationship between these attributes and the observed event sequence, like certain product groups require different manufacturing activities. To gain a better understanding of the data generating process, we are often interested in uncovering this mechanism.

In a predictive scenario, for example, production planners want to know the event sequence for a given product in advance, such that they can avoid bottlenecks and optimize the process flow. Rules in production planning systems are often hand-crafted and do not necessarily display the true complexity of the real process. Existing process models tend to show idealized, high-level behavior and thus give a limited picture of the real process (van der Aalst 2016, p. 30).

We are not the first to study sequence prediction based on meta data. Existing neural network approaches (Taymouri, La Rosa, and Erfani 2021; Camargo, Dumas, and González-Rojas 2019; Pasquadibisceglie et al. 2019) can achieve high accuracy given sufficient training data and hyperparameter tuning, however, the resulting models are inherently difficult to interpret. As the key applications, such as optimizing planning, require interpretation, these solutions do not suffice in practice. Surprisingly little work results in interpretable models being accurate and robust to noise.

In this paper, we take a different approach. We propose to model the event sequences as a directed graph with classification rules on the meta data to determine which paths to follow. Such an *event-flow graph* should fit the data well, but at the same time have a low model complexity to increase interpretability by humans. We formalize the problem in terms of the Minimum Description Length (MDL) principle, by which we identify the best model as the one giving the shortest lossless description of the data.

Due to NP-hardness of the resulting optimization problem, we propose the greedy method ConSequence, which first discovers a directed graph for a given set of event sequences and then finds classification rules on the meta data for nodes with multiple successors. While in practice any rule-based classifier can be plugged in, we propose the algorithm GERD, which uses a reliable rule effect estimator to find compact and meaningful rules.

Through extensive experiments including a case study, we show ConSequence discovers compact, interpretable and accurate models for the generation of event sequences from data. Our method has low sample complexity, works well under noise and deals with different real-world data.

The main contributions we make in this paper are

(a) formulate the problem of interpretable yet accurate prediction of event sequences from meta data with MDL,
(b) an efficient heuristic to discover event-flow graphs with rules for sequence prediction
(c) an extensive empirical evaluation.

Our paper is structured as follows. Next, we define necessary notation and concepts. In Section 3, we formally define the problem, and propose our algorithmic solution in Section 4. Before our experiments and a case study in Section 6, we provide an overview of related work in Section 5. Eventually, we draw a conclusion and outline possible future work.

We provide detailed empirical results as well as details for reproducibility in the supplementary. We make all code and data publicly available.[1]

[1]http://eda.mmci.uni-saarland.de/prj/consequence

Preliminaries

Before we formalize the problem, we introduce necessary notation and concepts used in the remainder of the paper.

Notation

We consider datasets of event sequences with meta data. Such a dataset D consists of n instances (x, y), where x is a vector with meta data, and y is a finite event sequence. We write X to refer to all meta data vectors, and Y to refer to all event sequences in the dataset.

The list of attributes A specifies the meta data present in the dataset, where we define the possible values of an attribute $A_i \in A$ by its domain $\operatorname{dom} A_i$, i.e. $x_i \in \operatorname{dom} A_i$. Attributes are either numerical with $\operatorname{dom} A_i \subseteq \mathbb{R}$, or categorical with $\operatorname{dom} A_i = \{c_1, \ldots, c_k\}$. Each event sequence y is drawn from a finite alphabet of possible events $\Omega = \{a, b, \ldots\}$, i.e. a sequence of length l is a sample from Ω^l.

For a given dataset D, we want to find a mapping $X \to \Omega^*$, that given an attribute vector x, generates or predicts the corresponding sequence y. Our model for interpretable sequence prediction is based on a directed graph $G = (V, E)$, where V denotes the set of nodes in the graph and E the set of edges. The set of successors of a node $v \in V$ is given by $\operatorname{succ}(v)$ and its out-degree is specified by $\deg^+(v)$.

MDL

The Minimum Description Length (MDL) principle (Rissanen 1978; Grünwald 2007) is an information theoretic approach for model selection. It identifies the best model as the one that provides the shortest lossless description of the given data. Formally, given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ minimizes $L(M) + L(D \mid M)$, where $L(M)$ is the length of the description of the model in bits, and $L(D \mid M)$ is the length of the data encoded with the model. This is also known as two-part, or crude MDL. Although one-part, or refined MDL, provides stronger theoretical guarantees, it is only computable in specific cases (Grünwald 2007). Since we are especially interested in the model, we use two-part MDL. In MDL, we are only concerned with code lengths, not actual code words.

To apply the MDL principle to our problem of sequence prediction, we will now define our model as well as the encoding of model and data.

Interpretable Sequence Prediction

In this section, we define our model for interpretable sequence prediction and give a formal problem definition.

Event-Flow Graphs

We propose to model the prediction of event sequences from meta data with *event-flow graphs*. An event-flow graph $M = (G, R)$ consists of a directed graph G and a *rule relation* R. As any directed graph, G is defined by a tuple (V, E), where the nodes correspond to events from Ω. Multiple nodes are allowed to refer to the same event. In addition, a valid event-flow graph consists of a source node v_s and a sink node v_e, which do not refer to any event. We use a path from v_s to v_e to represent an event sequence y.

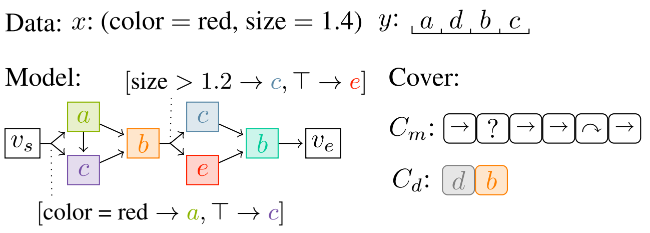

Data: x: (color = red, size = 1.4) y: $a\ d\ b\ c$
Model:
[size > 1.2 → c, ⊤ → e]
v_s
a
c
b
c
e
b
v_e
[color = red → a, ⊤ → c]
Cover:
C_m: → ? → → ⌒ →
C_d: d b

Figure 1: Toy example for a sequence y with meta data x and a cover of the sequence using a simple event-flow graph.

To model the relationship between meta data and event sequence, we assign classification rules to nodes using the rule relation $R : V \to \mathcal{R}$. At a given node v, such a rule predicts the next node to follow. Formally, we denote a rule by $\delta \to c$, where $\delta : X \to \{\top, \bot\}$ is the *condition*, that for a given meta data vector x either evaluates to true ($\top$) or false ($\bot$), and $c \in \operatorname{succ}(v)$ is the *consequence*. A condition consists of multiple terms that we stack together using *and* ($\wedge$) and *or* ($\vee$), e.g. $\delta = (\theta_1 \wedge \theta_2) \vee \theta_3$. A condition term θ always consists of an attribute $a \in A$, an operator from the set $\{>, \leq, =\}$ and a value for comparison $q \in \operatorname{dom} a$. We combine multiple rules to *decision lists* (Rivest 1987), which are ordered rule sets, where the classification output is determined by the first firing rule, i.e. for which $\delta(x) = \top$.

Given an event-flow graph M, we describe or *cover* a sequence $y \in \Omega^*$ by traversing M from v_s to v_e. Since real-world processes and data usually contain noise, we allow errors while traversing the graph to enable succinct models. To reconstruct a sequence from a given cover, we read instructional codes from the *code stream* C, which together with the rules in the graph determine the path through M and correct missed or redundant events. Conceptually, we split C into the *model stream* C_m, which encodes how to traverse the model, and the *disambiguation stream* C_d, which encodes ambiguous choices of events.

For better illustration, we give a toy example for the cover of a sequence using a simple event-flow graph in Figure 1. Starting at the source node v_s, we read the first code from C_m. The $\boxed{\rightarrow}$ tells us to move one step forward in the model and emit the next event we arrive at. Since the color attribute of our example has the value *red*, the rule at v_s only allows us to go to a, which we then emit. We read the next code from C_m, $\boxed{?}$, which means that the next event is not captured by the model. To disambiguate the choice between the events in Ω, we read from C_d and get the code for d. The next code in C_m is $\boxed{\rightarrow}$, so we go forward in the model and emit an event. From a, we can either go to b or to c, and this time, there is no rule telling us, which path to follow. Therefore, we read again from C_d and go to b, which we also emit. We continue by reading $\boxed{\rightarrow}$ from C_m, we evaluate the rule on the size attribute and go to and emit c. The next code in C_m, $\boxed{\curvearrowright}$, tells us to go forward in the model, but not to emit the event, i.e. the event in the model is redundant. From c, we can only go to b. Finally, we read the last $\boxed{\rightarrow}$ and arrive at v_e, which means that we have reconstructed S without loss.

We now take this concept of sequence cover to define an MDL score that will formalize how a good event-flow graph for a given dataset should look like.

MDL for Event-Flow Graphs

A good event-flow graph should fit the data well and at the same time avoid unnecessary complexity. We use the MDL principle to formalize this requirement, i.e. we are looking for a model with low overall encoding cost. First, we define how to compute the length of the data encoding using the cover concept as introduced in the former section.

Data Encoding Let Y be a given set of sequences, X the corresponding attribute vectors and M an event-flow graph. Then, the encoded length of the data using the model is

$$L(Y \mid M, X) = L(C_m) + L(C_d) \ ,$$

i.e. we have to compute the encoded length of the model stream C_m and the disambiguation stream C_d in the cover.

If we knew the distribution of the codes in C_m and C_d beforehand, Shannon entropy would give us the optimal length for prefix-free codes, i.e. code x would have length in bits of $-\log P(x)$. This matches the intuition, that more frequently used codes should have shorter encoded lengths. To avoid any arbitrary choices in the model encoding, we use prequential codes (Grünwald 2007), which are asymptotically optimal without requiring initial knowledge of the code distribution. We start encoding with a uniform distribution and update the counts after every received message, such that we have a valid probability distribution for optimal prefix codes at any point of time (Cover and Thomas 2006).

We condition code lengths on the current node in the event-flow graph while covering a sequence to make maximal use of available information, and to avoid that local changes in the graph change encoding lengths at all nodes. For the model stream C_m, which contains the codes $\boxed{\rightarrow}$, $\boxed{\curvearrowright}$ and $\boxed{?}$, this results in an encoded length of

$$L(C_m) = -\sum_{i=1}^{|C_m|} \log \frac{\mathrm{usg}_i(C_m[i] \mid v_i) + \epsilon}{\sum \mathrm{usg}_i(\,\cdot \mid v_i) + \epsilon} \ ,$$

where $\mathrm{usg}_i(C_m[i] \mid v_i)$ denotes how often the i-th code in C_m has been used before at the current node v_i, and ϵ with standard choice 0.5 is for additive smoothing.

The codes in C_d refer to events in Ω. Which codes are possible at one point of time is dependent on the last code in C_m. If we have to go forward in the model after reading a $\boxed{\rightarrow}$ or $\boxed{\curvearrowright}$ code, only events corresponding to the directly following nodes in the graph are possible, whereas reading a $\boxed{?}$ enables all events in Ω. That is why we conceptually split C_d into three individual streams, with C_c being the stream for correctly modeled events after reading $\boxed{\rightarrow}$, C_r being the stream for redundant events after reading $\boxed{\curvearrowright}$ and C_x for missed events after reading $\boxed{?}$. Then, all three streams follow the same computation scheme as C_m. For example,

$$L(C_x) = -\sum_{i=1}^{|C_x|} \log \frac{\mathrm{usg}_i(C_x[i] \mid v_i) + \epsilon}{\sum \mathrm{usg}_i(\,\cdot \mid v_i) + \epsilon} \ .$$

This gives us a lossless encoding of the data using an event-flow graph.

Model Encoding Since we are using prequential codes for the data encoding, the computation for the encoded length of an event-flow graph $L(M)$ is quite simple. Intuitively, a graph with more nodes, more edges and more rules should have a higher encoding length. Formally, we have

$$L(M) = L_{\mathbb{N}}(|V|+1) + |V| \cdot \log|\Omega| + \log(|V|^2+1) + \binom{|V|^2}{|E|} + \sum_{v \in V} L(v) \ ,$$

where we first encode the number of nodes in the graph, then the events of the nodes, the number and layout of the edges and finally the rules at each node. For the number of nodes, we use the MDL-optimal encoding for integers $z \geq 1$ (Rissanen 1983), defined as $L_{\mathbb{N}}(z) = \log^* z + \log c_0$, where $\log^* z = \log z + \log\log z + \dots$ and we sum only the positive terms, and $c_0 = 2.865064$ is set such that we satisfy the Kraft-inequality – i.e. ensure it is a lossless code. The number of edges is encoded using an index over all possibilities, starting at 0 and with $|V|^2$ as an upper bound. For the edge layout, we use a data-to-model code (Li and Vitányi 1993), which is an index over a canonically ordered set of all directed graphs of $|V|$ nodes and $|E|$ edges.

If a node has less than two successors, no rule is possible, and $L(v) = 0$. Otherwise, we compute the encoded length of the rules at node v with

$$L(v) = L_{\mathbb{N}}(|v|) + \sum_{\delta, c \in v} L(\delta) + \log \deg^+(v) \ ,$$

where we first encode the number of rules and then the conditions and consequences. The encoded length of a condition is computed depending on its type. For a simple term θ that makes a comparison on attribute a with operator o, we have

$$L(\theta) = \log 3 + \log|A| + L(o \mid a) + \log|\operatorname{dom} a| \ ,$$

i.e. we encode the type of the condition with an index over $\{\mathrm{or}, \mathrm{and}, \mathrm{term}\}$, attributes, operators and comparison values. The encoded length of the operator depends on the choice of attribute. For categorical attributes, the only possible operator in our rule language is $(=)$, i.e. $L(o \mid a) = 0$, whereas for numerical attribute, we have to distinguish between $(>)$ and $(\leq)$, i.e. $L(o \mid a) = \log 2 = 1$. Like in many rule mining or decision tree algorithms, we assume a discretization grid for cut points in conditions (Fayyad and Irani 1992), such that we can use $\log|\operatorname{dom} a|$ to compute the encoded length for both categorical and numerical attributes. If a condition consists of multiple subconditions either joined with $(\vee)$ or $(\wedge)$, we compute the encoded length by

$$L_{\wedge}(\delta) = L_{\vee}(\delta) = L_{\mathbb{N}}(|\delta|) + \sum_{i=1}^{|\delta|} L(\delta_i) \ ,$$

where we specify the unbounded number of subconditions, before we encode each subcondition, which is either a simple term θ or again consists of multiple other conditions.

All of this together gives us a lossless encoding of an event-flow graph.

Formal Problem Definition

We now have all ingredients for a formal definition of our data-to-sequence problem.

Minimal Event-Flow Graph Problem *Given a dataset with attributes and event sequences (A, X, Y), find the minimal rule containing event-flow graph M and cover C, such that the total encoded cost $L(M) + L(Y \mid M, X)$ is minimal.*

Solving this problem optimally is impossible in practice. Just finding the optimal cover for a given sequence and event-flow graph is already NP-complete. This is due to the equivalence between event-flow graphs and 1-safe nets, for which computing the optimal alignment with a sequence has been proven as NP-complete, even if the model is acyclic (Cheng, Esparza, and Palsberg 1993; Adriansyah 2014).

In practice, we do not know the event-flow graph to begin with, and the number of possible directed acyclic graphs grows super-exponentially with the number of nodes (Robinson 1973). Last, at every branch in the model, we need to find a rule list, however, mining optimal decision trees or rule sets with minimal model complexity has been proven to be NP-complete, too (Hyafil and Rivest 1976; Andersen and Martinez 1995).

The search space of our problem also does not show any trivially exploitable structure, such as monotonicity or submodularity, hence, we resort to heuristics.

Algorithm

To discover good event-flow graphs in practice, we split the Minimal Event-Flow Graph Problem into multiple parts. Since each part is already hard to solve by itself, we propose greedy solutions to each of the subproblems separately. We discuss the algorithms for these in turn. For a detailed runtime complexity analysis, we refer to the supplementary.

Computation of a Cover

We start by providing an algorithm to find a good cover for a given model M, i.e. a cover C with low $L(Y \mid M, X)$. To compute a cover with near-optimal encoding length, we use the intuition that the better a model fits the data, the shorter the encoded length of the cover should be. This is equivalent to minimizing the number of $\boxed{?}$ and $\boxed{\frown}$ codes in C_m.

This formulation of the problem is equivalent to finding an optimal alignment between a Petri net and a sequence, where the standard approach to solve this problem optimally is to apply an A* search strategy (Adriansyah 2014).

We follow this approach as shown in Algorithm 1 by starting with an empty cover, which we iteratively extend, until we find a complete cover, i.e. after decoding we have reconstructed the whole sequence and have arrived at v_e. The candidates are ranked by their cost, i.e. the number of errors the model makes in terms of $\boxed{?}$ and $\boxed{\frown}$ codes in the cover, plus a heuristic h, which is an estimate of the cost for making the candidate a complete cover. If the heuristic function h is *admissible*, i.e. a lower bound of the true cost, and *consistent*, i.e. non-increasing for following states, A* finds the optimal solution (Hart, Nilsson, and Raphael 1968). One admissible and consistent heuristic for our problem is the number of uncovered events in the sequence for which there is no corresponding node reachable (Adriansyah 2014, p. 66).

Algorithm 1: COVER an Event Sequence

```
input : event sequence $y$, event-flow graph $M$, beam
        size $w$, heuristic $h$
output: a cover for $y$ using $M$
1 $Q \leftarrow$ queue containing the empty cover;
2 while True do
3     $C \leftarrow$ pop top element from $Q$;
4     if $C$ covers both $y$ and $M$ then
5         return $C$
6     foreach possible extension $C'$ to $C$ do
7         $c \leftarrow$ number of $\boxed{?}$ and $\boxed{\frown}$ in $C_m$;
8         insert $c'$ into $Q$ using priority $c + h(C')$;
9     $Q \leftarrow$ the $w$ best candidates in $Q$;
```

Unfortunately, always having the guarantee to find the optimum comes quite with a cost of runtime. Therefore, to cover a sequence in feasible runtime, we modified the A* search in a beam search fashion, where we only keep the w best candidates in each iteration.

Discovering an Event-Flow Graph Without Rules

With the cover algorithm, we can now compute our optimization target $L(M, Y, X) = L(M) + L(Y \mid M, X)$. To reduce the search space, we first try to find a compact event-flow graph without rules. One can interpret this as an event-flow graph with optimal rules emulated by the cover algorithm. After having an event-flow graph, we can learn rules that try to reproduce the routing choices of the cover.

To find a good event-flow graph without rules for a given dataset, we propose a greedy bottom-up search. We start with an empty graph, which just consists of source v_s and sink v_e. Iteratively, we add paths to the model corresponding to the most frequent sequences in the dataset. We only keep paths, that improve the objective score. For reference, we provide pseudo-code in the supplementary.

Finding Classification Rules for Path Prediction

After having found an event-flow graph and a cover, we now discover rules to reproduce the routing decisions by the cover. At each node with more than one successor, we learn a rule-based classifier, that predicts the next node for given meta data. The learning algorithm should produce rules, that fit the data well, which leads to small $L(Y \mid M, X)$, because the rules reduce entropy and thus code lengths in the code stream of the cover. At the same time the rules should not get too complex, which would result in high $L(M)$.

One should notice that each node contains its own classification dataset. If this node corresponds to infrequent but still relevant behavior of the data, the number of instances in the dataset will be low. To deal with datasets containing many attributes, we need a statistically robust learner that needs little data to infer meaningful, well-generalizing rules.

For this, we try to find rules with a high *effect* on predicting the class label $p(c \mid \delta) - p(c \mid \bar{\delta})$. A positive effect means that setting attributes x such that $\delta = \top$ increases chances to observe class label c. The effect is robustly estimated by

$$\hat{e}(\delta, c) = \frac{n_{\delta,c} + 1}{n_\delta + 2} - \frac{n_{\bar{\delta},c} + 1}{n_{\bar{\delta}} + 2} - \frac{\beta}{2\sqrt{n_\delta + 2}} - \frac{\beta}{2\sqrt{n_{\bar{\delta}} + 2}} ,$$

where $n_{\delta,c}$ and $n_{\bar{\delta},c}$ are counts how often class label c is observed if condition δ evaluates to $\top$ or $\bot$, n_δ and $n_{\bar{\delta}}$ are counts how often condition δ in total evaluates to $\top$ or $\bot$, and β is a confidence parameter. Higher values for β require more evidence to compute a positive effect and increase robustness to outliers. (Budhathoki, Boley, and Vreeken 2021)

Maximizing $\hat{e}$ minimizes our MDL defined objective score. Applying the logarithm for an information theoretic interpretation of probability distributions, one can transform $p(c \mid \delta) - p(c \mid \bar{\delta})$ into $-\log\left[p(c, \delta) - p(c)p(\delta)\right] - \log p(\delta)$. This means, rules that have high predictive power on the next node in the event-flow graph, decrease $L(Y \mid M, X)$. The term $-\log p(\delta)$ can be seen as a regularizer for infrequent rules, which has a positive effect on minimizing $L(M)$.

Mining optimal decision trees or rule sets with minimal model complexity has been proven to be NP-complete (Hyafil and Rivest 1976; Andersen and Martinez 1995). Hence, we implemented a greedy approach for finding rules with maximal effect, which we call *greedy effective rule discovery* (GERD). We start with an empty rule list and iteratively add rules until we have covered all instances in the dataset. To greedily find a rule with high effect, we first look at rules with one term and only keep the rule with the highest effect. If this rule does not have a positive effect, we replace it with the default rule, which is a rule that always fires and outputs the class label with the highest support. Otherwise, we try to extend the rule with one additional term using ($\wedge$) and ($\vee$), such that the effect of the rule increases. If this is successful, we again try to extend the rule, else we append the rule to the list of found rules. We repeat this process until the list of rules covers all instances in the dataset. For further details, we provide pseudo-code in the supplementary.

Related Work

Event sequence prediction is a broadly studied topic. Much work deals with the problem of predicting the next event in a sequence based on past events, without considering additional meta data. This includes association rule mining (Rudin et al. 2011), Markov models (Begleiter, El-Yaniv, and Yona 2004) and pattern mining (Wright et al. 2015).

Recent work considering meta data to predict sequences is mostly based on neural networks. The approaches mainly differ in the feature encoding and the concrete network architecture. Proposed methods include Long short-term memory (LSTM) networks (Hochreiter and Schmidhuber 1997) with one-hot encoding (Tax et al. 2017), LSTMs with embedding techniques (Camargo, Dumas, and González-Rojas 2019), Convolutional neural networks (Pasquadibisceglie et al. 2019) and LSTMs with an adversarial training scheme (Taymouri, La Rosa, and Erfani 2021).

Although there is work that applies approaches from explainable artificial intelligence (Mehdiyev and Fettke 2020) on neural networks for business process prediction, there is only limited work focusing more accessible models. One recent exception is given by the data-aware transition system DATS (Polato et al. 2018). First, the observed prefixes of event sequences are used to create a state machine. Which prefixes are mapped to which state is determined by a *state abstraction function*. Examples are the *list* function, where each unique prefix is mapped to its own state, and the *set* function where prefixes with the same set of events are mapped to the same state. For predicting future events given an attribute vector, a Naïve Bayes classifier estimates the transition probabilities between states and the path with the highest probability is predicted.

Inferring state machines from event data is also studied in software engineering with the goal of anomaly detection and test case generation but not sequence prediction (Lorenzoli, Mariani, and Pezzè 2008; Walkinshaw, Taylor, and Derrick 2016). Model complexity and interpretability play a minor role in these approaches and software execution traces are usually much less noisy than business process data.

Data-to-text generation deals with the creation of text, which can be seen as a kind of event sequence, from data. While today much work is based on neural networks, traditional approaches use handcrafted rule-based templates for sentences. (Gatt and Krahmer 2018) First work tries to reduce the effort in creating templates, but human supervision is still required (van der Lee, Krahmer, and Wubben 2018).

Outstanding from the above, CONSEQUENCE enables sequence generation from data with an accessible white-box model while requiring no handcrafted templates or rules and with minimal need for hyper-parameter tuning.

Experiments

In this section, we evaluate CONSEQUENCE on both synthetic and real-world datasets. As a quality measure we use Levenshtein similarity (Levenshtein 1966) between actual and predicted sequence. Let $\rho(y_1, y_2)$ be the minimal number of deletions, insertions and replacements to transform sequence y_1 into sequence y_2, normalized Levenshtein similarity is defined by NLS $= 1 - \frac{\rho(y_1,y_2)}{max(|y_1|,|y_2|)}$, i.e. NLS $\in [0, 1]$ and $y_1 = y_2 \rightarrow$ NLS $= 1$.

We ran all experiments on a server with two Intel(R) Xeon(R) Silver 4110 CPUs, 128 GB of RAM and two NVIDIA Tesla P100 GPUs. Except for LSTM, which uses the GPU, we report wall-clock runtimes for single-thread execution. We provide code and data for research purpose as well as details for reproducibility in the supplementary.[2]

Baseline Methods

We compare CONSEQUENCE to various baselines that show different strengths and weaknesses on our task. First, we compare to a data-to-sequence LSTM.

To show that the order of events matters, i.e. it is not sufficient to just predict occurence and frequency of events, we also compare to the rule-based multilabel classificator BOOMER (Rapp et al. 2020). To get a simple baseline for

[2]http://eda.mmci.uni-saarland.de/prj/consequence

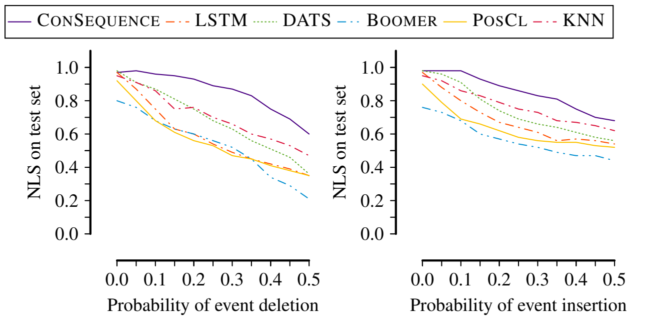


Figure 2: [**ConSequence is noise-robust**] NLS (higher is better) on test set dependent on the amount of destructive (left) and additive noise in the training set (right).

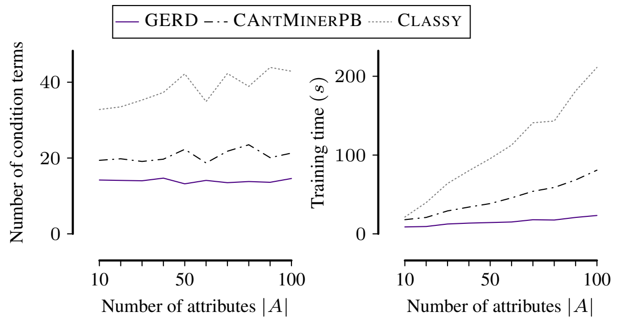


Figure 3: [**GERD is fast and produces small models**] Model complexity in number of condition terms (left) and runtime (right) for GERD, CAntMinerPB and Classy on artificial data with varying number of attributes.

ordering the predicted multiset of events, we put events to positions in the sequence, where we have seen them most frequently in the training set.

As additional baselines, we use KNN that predicts the event sequence with a k-nearest neighbor classifier, and DATS as described in the related work section.

Finally, we use GERD to predict events for each possible position in the sequence. This means, if the longest sequence in the training set has length 100, then we learn 100 independent classifiers. The length of the predicted sequence is determined by the lowest positioned classifier that predicts end-of-sequence. We call this baseline PosCl.

Synthetic Data

First, we evaluate on synthetic data, where we generate data from ground-truth models with varying properties. We report on the noise-robustness of different methods. For this experiment, we sampled instances from an artificial ground-truth model, split the dataset into a training set with 8000 instances and a test set with 2000 instances, and applied a noise model on the training data. In Figure 2, we show the NLS for various sequence predictors dependent on the amount of noise. We consider destructive noise, where for each event in the dataset, we remove this event with some probability α, and additive noise, where at each position in the dataset, we add a random event from Ω with probability α. As one can see, ConSequence performs well under realistic amounts of noise, especially being more robust to noise than its competitors.

While in theory, we allow any classifier for the prediction of successors in the event-flow graph, we propose using GERD for good reason. In Figure 3, we compare the use of ConSequence with three different rule-based classifiers. Besides GERD, we examine CAntMinerPB (Otero and Freitas 2016), which uses an ant colony optimization to mine a decision list with low prediction error, and Classy (Proença and van Leeuwen 2020), which uses MDL to select a set of classification rules. GERD produces models with a lower number of decision terms, which are thus easier to be understood by humans. Furthermore, due to its scaling behavior, it is able to deal with a larger number of attributes in a lower amount of time than its competitors.

Data	n	$\|S\|$	$\hat{m}$	$\|\Omega\|$	$\|A\|$
Production	255	209	13	57	2
Sepsis	782	733	19	18	23
Rolling Mill	49233	1639	39	270	175
Software	500	200	151	24	12

Table 1: [**Statistics for real-world datasets**] Number of sequences n, number of unique sequences $|S|$, average sequence length $\hat{m}$, size of event alphabet $|\Omega|$ and number of attributes $|A|$ for four real-world datasets.

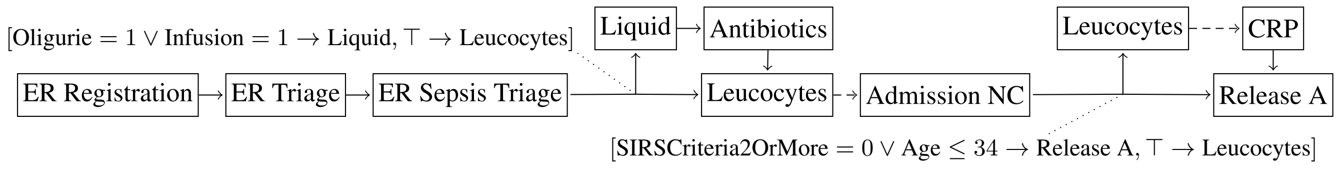
Real-World Data

To show that ConSequence performs well in practice, we now evaluate on data from the real-world. We selected four datasets with different properties as summarized in Table 1. Production (Levy 2014) is a collection of event sequences from a production process, which contains only 255 relatively short sequences, from which 209 are unique. Sepsis (Mannhardt 2016) contains trajectories of Sepsis patients in a Dutch hospital. We filtered out incomplete sequences and only considered attributes which were available at the beginning of a sequence. Rolling Mill is a manufacturing event log of a German steel producer. It stands out with its high number of instances, unique events and attributes. Software, the forth and last dataset in our comparison, is a profiling log of the Java program *density-converter* (Favre-Bulle 2020) that takes image files as input and converts them to different formats and densities, such that they can be used on different target platforms like Android or iOS. The events in this dataset refer to classes called during program execution, and the attributes refer to command line arguments.

We ran Boomer, PosCl, DATS, LSTM and ConSequence ten times on these datasets with a random train-test-split of 80%. In Table 2, we give an overview of the averaged results. ConSequence achieves the highest NLS on the testset for datasets with few instances and especially outperforms other methods on the Software dataset with a large average sequence length. As expected, the black-box LSTM performs well with a large training set, which was only available for the Rolling Mill data, while ConSequence still

	BOOMER			POSCL			DATS			LSTM		CONSEQUENCE			
Data	NLS	$\|M_C\|$	t	NLS	$\|M_C\|$	t	NLS	$\|M_V\|$	t	NLS	t	NLS	$\|M_V\|$	$\|M_C\|$	t
Production	0.28	999	3s	0.28	86	4s	0.35	1712	1s	0.05	82s	0.42	8	2	24s
Sepsis	0.43	539	13s	0.52	99	16s	0.50	47	5s	0.51	363s	0.58	22	16	156s
Rolling Mill	0.74	1778	2.5h	0.90	10^5	14d	0.67	137	13m	0.99	5h	0.94	104	746	5h
Software	0.20	66	5s	0.69	2004	37s	0.22	7	2s	0.52	6m	0.84	929	92	16m

Table 2: [**Results on real-world data**] We give NLS (higher is better), number of condition terms $|M_C|$ in the model if applicable (lower means less complex), number of nodes $|M_V|$ for graph-based models and runtime t on the training set.



[Oligurie = 1 ∨ Infusion = 1 → Liquid, ⊤ → Leucocytes]
Liquid
Antibiotics
Leucocytes
CRP
ER Registration
ER Triage
ER Sepsis Triage
Leucocytes
Admission NC
Release A
[SIRSCriteria2OrMore = 0 ∨ Age ≤ 34 → Release A, ⊤ → Leucocytes]

Figure 4: [**Sepsis Event-Flow Graph**] Excerpt from the model found by CONSEQUENCE on the Sepsis dataset. Dashed arrows indicate skipped nodes. We provide the complete model in the appendix.

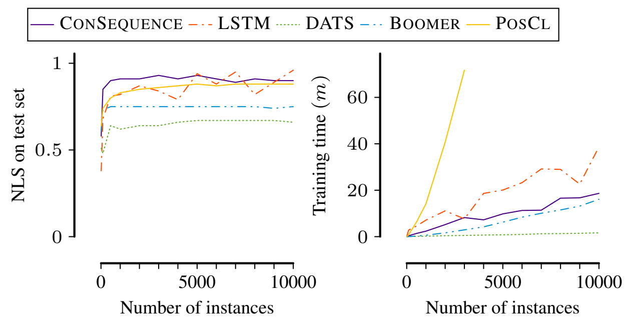


Figure 5: [**CONSEQUENCE has low sample complexity and scales well**] NLS on the test set (left, higher is better) and training runtime (right) depending on the number of instances in the Rolling Mill training set.

beats the other white-box approaches by a large margin.

CONSEQUENCE produces small and thus better understandable models. The models by CONSEQUENCE have less condition terms than found by BOOMER and less nodes than the graphs discovered by DATS. This is because both BOOMER and DATS focus on prediction accuracy and less on model size. While CONSEQUENCE is not the fastest, it still runs within a reasonable amount of time.

In Figure 5, we report how the size of the training set impacts the training time and the NLS on the test set. CONSEQUENCE already achieves its best performance with 1000 training instances in the Rolling Mill dataset. Although there are certainly faster methods such as DATS, CONSEQUENCE shows a linear scaling behavior w.r.t. to the number of instances. Together with the low sample complexity, this enables applicability on a wide range of real-world datasets.

Case Study

Eventually, we want to show with a real-world example that the models by CONSEQUENCE are understandable and meaningful. We start with an excerpt of the discovered event-flow graph for the Sepsis dataset as displayed in Figure 4. One can clearly recognize the typical flow of a Sepsis patient in the hospital. The process starts with the arrival in the emergency room (ER). If the patient has an Oligurie (malfunction of kidneys) or for other reasons needs an infusion, he or she is provided with liquid and antibiotics. In any case, leukocytes are counted for further diagnosis. After admission to normal care (NC), patients without certain symptoms, which are potentially younger, are released soon, while other patients need further treatment.

Evidently, CONSEQUENCE produces inherently accessible and interpretable models. We provide a similar case study for the Rolling Mill dataset in the supplementary.

Conclusion

We studied the problem of accurate yet interpretable sequence prediction from data. For this, we modeled event sequences with directed graphs and discovered classification rules to explain the relationship between attributes in the dataset and paths in the graph. We formalized the problem in terms of the MDL principle, i.e. the best model is the one that compresses the data best. As the resulting optimization problem is NP-hard, we proposed the efficient CONSEQUENCE algorithm to discover good models in practice.

Through an extensive set of experiments including a case study, we showed that our approach indeed produces compact, interpretable and accurate models, is robust against noise and has low sample complexity, which enables applicability on a wide range of real-world datasets.

Future work might extend CONSEQUENCE to more applications like prediction of running cases given an event sequence prefix, where meta data belongs to events instead of the whole sequence. A richer modeling language for event-flow graphs using patterns instead of single event nodes, could result in even more succinct models, that better fit complex behavior like concurrent events.

Acknowledgements

The authors thank SHS – Stahl-Holding-Saar for providing the Rolling Mill dataset and giving valuable input for the practical application of our method in production planning.

References

Adriansyah, A. 2014. *Aligning observed and modeled behavior*. Ph.D. thesis, TU Eindhoven.

Andersen, T.; and Martinez, T. 1995. NP-completeness of minimum rule sets. In *Proceedings of the 10th international symposium on computer and information sciences*, 411–418.

Begleiter, R.; El-Yaniv, R.; and Yona, G. 2004. On prediction using variable order Markov models. *JAIR*, 22: 385–421.

Budhathoki, K.; Boley, M.; and Vreeken, J. 2021. Discovering Reliable Causal Rules. In *SDM*, 1–9.

Camargo, M.; Dumas, M.; and González-Rojas, O. 2019. Learning accurate LSTM models of business processes. In *BPM*, 286–302. Springer.

Cheng, A.; Esparza, J.; and Palsberg, J. 1993. Complexity results for 1-safe nets. In *FSTTCS*, 326–337. Springer.

Cover, T. M.; and Thomas, J. A. 2006. *Elements of Information Theory*. Wiley-Interscience New York.

Favre-Bulle, P. 2020. Density Image Converter Tool for Android, iOS, Windows and CSS. https://github.com/patrickfav/density-converter (accessed on 08-06-2021).

Fayyad, U.; and Irani, K. 1992. On the handling of continuous-valued attributes in decision tree generation. *Mach. Learn.*, 8: 87–102.

Gatt, A.; and Krahmer, E. 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *JAIR*, 61: 65–170.

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

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE TSSC*, 4(2): 100–107.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation*, 9(8): 1735–1780.

Hyafil, L.; and Rivest, R. 1976. Constructing optimal binary decision trees is NP-complete. *Inf. Process. Lett.*, 5(1): 15–17.

Levenshtein, V. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics – Doklady*, volume 10, 707–710.

Levy, D. 2014. Production Analysis with Process Mining Technology. 10.4121/uuid:68726926-5ac5-4fab-b873-ee76ea412399.

Li, M.; and Vitányi, P. 1993. *An Introduction to Kolmogorov Complexity and its Applications*. Springer.

Lorenzoli, D.; Mariani, L.; and Pezzè, M. 2008. Automatic generation of software behavioral models. In *ICSE*, 501–510.

Mannhardt, F. 2016. Sepsis Cases - Event Log. 10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

Mehdiyev, N.; and Fettke, P. 2020. Explainable Artificial Intelligence for Process Mining: A General Overview and Application of a Novel Local Explanation Approach for Predictive Process Monitoring. *CoRR*, abs/2009.02098.

Otero, F.; and Freitas, A. 2016. Improving the Interpretability of Classification Rules Discovered by an Ant Colony Algorithm: Extended Results. *Evolutionary Computation*, 24(3): 385–409.

Pasquadibisceglie, V.; Appice, A.; Castellano, G.; and Malerba, D. 2019. Using convolutional neural networks for predictive process analytics. In *ICPM*, 129–136. IEEE.

Polato, M.; Sperduti, A.; Burattin, A.; and de Leoni, M. 2018. Time and activity sequence prediction of business process instances. *Computing*, 100(9): 1005–1031.

Proença, H.; and van Leeuwen, M. 2020. Interpretable multiclass classification by MDL-based rule lists. *Information Sciences*, 512: 1372–1393.

Rapp, M.; Mencía, E. L.; Fürnkranz, J.; Nguyen, V.-L.; and Hüllermeier, E. 2020. Learning gradient boosted multi-label classification rules. In *ECML PKDD*, 124—140. Springer.

Rissanen, J. 1978. Modeling by shortest data description. *Automatica*, 14(1): 465–471.

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

Rivest, R. 1987. Learning Decision Lists. *Mach. Learn.*, 2(3): 229–246.

Robinson, R. 1973. Counting labeled acyclic digraphs. *New Directions in the Theory of Graphs*, 239–273.

Rudin, C.; Letham, B.; Salleb-Aouissi, A.; Kogan, E.; and Madigan, D. 2011. Sequential event prediction with association rules. In *COLT*, 615–634.

Tax, N.; Verenich, I.; La Rosa, M.; and Dumas, M. 2017. Predictive business process monitoring with LSTM neural networks. In *CAiSE*, 477–492. Springer.

Taymouri, F.; La Rosa, M.; and Erfani, S. 2021. A Deep Adversarial Model for Suffix and Remaining Time Prediction of Event Sequences. In *SDM*, 522–530.

van der Aalst, W. 2016. *Process Mining – Data Science in Action*. Springer, second edition.

van der Lee, C.; Krahmer, E.; and Wubben, S. 2018. Automated learning of templates for data-to-text generation: comparing rule-based, statistical and neural methods. In *INLG*, 35–45.

Walkinshaw, N.; Taylor, R.; and Derrick, J. 2016. Inferring extended finite state machine models from software executions. *Empirical Software Engineering*, 21(3): 811–853.

Wright, A.; Wright, A.; McCoy, A.; and Sittig, D. 2015. The use of sequential pattern mining to predict next prescribed medications. *J Biomed Inf*, 53: 73–80.

Algorithm 2: DISCOVER EVENT-FLOW GRAPH

```
input  : event sequences $Y$
output: event-flow graph $M$
1  $M \leftarrow$ empty graph;
2  foreach unique sequence $y \in Y$ in desc. frequency do
3      $G' \leftarrow M \cup y$;
4      if $L(M', Y) < L(M, Y)$ then
5          $M \leftarrow M'$
6  return $G$
```

Appendix

Here, we include supplementary material which could not be part of our main paper.

Pseudo-Code for Discovering an Event-Flow Graph

In Algorithm 2, we show the pseudo-code for our greedy bottom-up search to find an event-flow graph without rules. We start with an empty event-flow graph, which just consists of the source node v_s and the sink node v_e. Iteratively, we add paths to the event-flow graph corresponding to the most frequent sequences in the dataset. We only keep paths, that improve the objective score.

Pseudo-Code for GERD

In Algorithm 3, we provide pseudo-code for our rule-based classifier GERD. We start with an empty rule list and iteratively add rules until we have covered all instances in the dataset. To greedily find a rule with high effect, we first look at rules with one term and only keep the rule with the highest effect. If this rule does not have a positive effect, we replace it with the default rule, which is a rule that always fires and outputs the class label with the highest support.

Otherwise, we try to extend the rule with one additional term using ($\wedge$) and ($\vee$), such that the effect of the rule increases. If this is successful, we again try to extend the rule, else we append the rule to the list of found rules. We repeat this process until the list of rules covers all instances in the dataset.

Runtime Complexity Analysis

In this section, we conduct a runtime complexity analysis for our algorithms.

Algorithm 1 Let b be the branching factor of the search, i.e. the average number of expansions for nodes in the search, and let d be the depth of the search tree. Without limiting the capacity of the open list, A* has worst-case runtime complexity $\mathcal{O}(b^d)$. With a beam width parameter w, the number of expanded nodes in the worst-case scenario is $\sum_{i=0}^{w-1} b^i + wb(d+1-w)$, i.e. we have a runtime, which is linear in the depth d and exponential in the beam width w.

The branching factor b strongly depends on the average degree in the event-flow graph. When expanding an existing partial cover, we can go to any successor of the current node in the event-flow graph using a $\boxed{\rightarrow}$ or a $\boxed{\frown}$, or we can have a $\boxed{?}$ without moving in the model. This leads to the upper bound $b \leq 2\frac{|E|}{|V|} + 1$.

We find a complete cover by the latest after as many $\boxed{?}$ codes as events in the sequence, and as many $\boxed{\frown}$ codes as the length of the shortest path from v_s to v_e. In other words, we can cover sequence and model independently from each other, which gives us an upper bound on d.

Algorithm 3 Computing the effect of a rule needs a pass over all n instances. Since at least one instance is covered in every iteration, the outer loop is called at most n times. Finding a rule with one term resp. extending a rule with one term scales linearly in the number of attributes $|A|$. Since we require an improvement over $\hat{e}$ in every iteration of the inner loop, the number of newly covered instances by a rule is at least as high as the number of inner iterations. This leads to a worst-case runtime complexity of $\mathcal{O}(n^2 \cdot |A|)$.

Algorithm 2 In the worst-case, every sequence in Y is unique, which means that we need n iterations. In each iteration, the cover is computed to find extension points in the model for an uncovered sequence y and to compute the total encoded length to decide whether the extension is rejected or not. Since the cover computation is needed n times, the cover algorithm is the main bottleneck in the algorithm.

Algorithm 3: Rule Discovery (GERD)

```
input  : attribute vectors $X$, label vector $z$,
         confidence parameter $\beta$
output: a list of rules $R$
1  $R \leftarrow [\,]$;
2  while $|X| > 0$ do
3      $R' \leftarrow$ possible rules with one term;
4      $r^* \leftarrow \arg\max_{r \in R'} \hat{e}(r, X, z, \beta)$;
5      if $\hat{e}(r^*, X, z, \beta) > 0$ then
6          $r^* \leftarrow$ default rule;
7      else
8          repeat
9              $R' \leftarrow$ possible one term extensions of $r^*$;
10             $r^* \leftarrow \arg\max_{r \in R' \cup \{r^*\}} \hat{e}(r, X, z)$;
11         until $r^*$ remains unchanged;
12     append $r^*$ to $R$;
13     remove instances from $X, z$ covered by $r^*$;
14 return $R$
```

Details for Baseline Methods

Here, we provide details for the baseline methods in our experiments and how we selected hyperparameters.

LSTM We tuned hyperparameters by conducting five runs with a random search on the hyperparameter values. The model with highest accuracy on a hold-out validation set was selected. To prevent overfitting, we used dropout and early-stopping. The set of optimized hyperparameters included the size of the network, the batch size and the amount of dropout.

Boomer To show that the order of events matters, i.e. it is not sufficient to just predict occurence and frequency of events, we also compare to the rule-based multilabel classification algorithm BOOMER (Rapp et al. 2020). Each event together with its frequency gets a label in the multilabel classification dataset. For example, the event sequence aba produces the labels $\{a_2, b_1\}$.

We order the predicted multiset of events by putting events to positions in the sequence, where they occurred most frequently in the training set. Formally, we solve the linear program

$$\begin{aligned}
\text{maximize} \sum_{i}^{n} \sum_{j}^{n} c_{ij} x_{ij} & \\
\text{subject to} \sum_{i}^{n} x_{ij} = 1, & \qquad j = 1, \dots, n \\
\sum_{j}^{n} x_{ij} = 1, & \qquad i = 1, \dots, n \\
x_{ij} \in \{0, 1\} \ , &
\end{aligned}$$

where n is the number of events in the multiset, c_{ij} is the number of times event i occurs at position j, $x_{ij} = 1$ means event i is set to position j, and the constraints ensure, that all positions in the sequence will be filled and no event will be set to more than one position.

Since BOOMER uses ensemble learning to infer the set of rules, its focus is on prediction accuracy and less on model complexity. BOOMER has a hyperparameter to prune individual rules, which we activated, such that single rules do not have unnecessarily many condition terms. We implemented a simple post-processing of the rule set, where we remove the rules with the lowest weight in the ensemble until prediction accuracy on a held-out validation set decreases.

Regarding the rest of hyperparameters, we used the default values. The main reason why BOOMER achieves a relatively low NLS is the ordering of events. An extensive hyperparameter tuning of BOOMER would not result in a significant improvement over the reported NLS. Constructing an additional white-box model, that sorts the predicted multiset in a correct order is not trivial and would definitely increase the model complexity. Learning a black-box model such as an LSTM would tremendously decrease interpretability. This is another argument for an integrated approach such as CONSEQUENCE that uses interpretable models to predict event sequences from meta data.

DATS To build the transition system for DATS, we used *list state abstraction*, since it showed best NLS in our experiment setting, which matches the results in the original paper (Polato et al. 2018).

We also observed that DATS struggles with complex event logs, which result in large transition systems. Therefore, we added a frequency filter to the event log to preprocess the data before building the transition system. Let y^* be the most frequent sequence in the dataset and $\operatorname{supp} y$ denote the support or absolute frequency of y. Then, for a given threshold $\alpha \in [0, 1]$, we only keep sequences in the dataset with $\frac{\operatorname{supp} y}{\operatorname{supp} y^*} \geq \alpha$. In our experiments, we tried out $\alpha \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ and reported NLS and $|M_V|$ for the best run.

Generation of Artificial Data and Models

For the experiments on synthetic data, we used a self-implemented generator for the creation of ground-truth models, i.e. event-flow graphs, and datasets. In this section, we provide details about this generation process.

First, we generate the attributes in the dataset. The parameters of this generator include the number of categorical and the number of numerical attributes to be generated. For noise robustness experiment in Figure 2, we generated two categorical and three numerical attributes, and for the experiment in Figure 3, we plugged in values from five to 50 for each parameter. For each categorical attribute we sample a value from a discrete uniform distribution over the possible values, where the number of possible values is determined by another parameter. For numerical attributes, we sample from a uniform distribution over the interval $[0, 1]$.

In the second step, we generate an event-flow graph. For the experiments presented in the paper, we first generated a list of nested if-then-else rules, e.g.

```
IF num-feature-3 $> 0.043$ THEN
  Append 7, 19
ELSE
  Append 2, 0, 6
IF cat-feature-1 $= 3$ THEN
  Append 1, 11, 17
ELSE
  IF cat-feature-2 $= 2$ THEN
    Append 11
Append 1.
```

After generating such a list of rules, we convert the list to an event-flow graph. The complexity of the generated rule list can be controlled via additional parameters. For details, we refer to our shared source code.

Eventually, we use the event-flow graph to generate a sequence for each instance in the dataset. This gives us a synthetic dataset with attributes and event sequences plus an event-flow graph as ground-truth model.

Hyperparameter Sensitivity

In this section, we report on the hyperparameter sensitity of CONSEQUENCE. Since we modeled the problem using MDL, we are almost free of hyperparameters. The cover algorithm as presented in Algorithm 1 contains a beam size parameter, that provides a trade-off between runtime and quality of the A* search. In our classifier GERD as shown in Algorithm 3, we have a confidence parameter β to control robustness of the discovered classification rules.

We first examine the influence of the beam size parameter w in the cover algorithm and report the results in Figure 6. The choice of the beam width has only marginal influence on the prediction accuracy in terms of NLS on the testset.

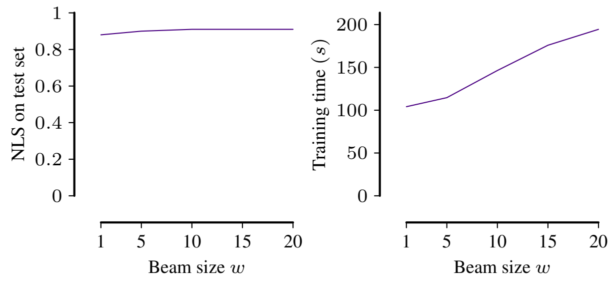


Figure 6: [**Hyperparameter sensitivity of the cover algorithm**] NLS on the testset (left) and runtime (right) for the cover algorithm on 1000 instances of the Rolling Mill dataset with varying beam size parameter w.

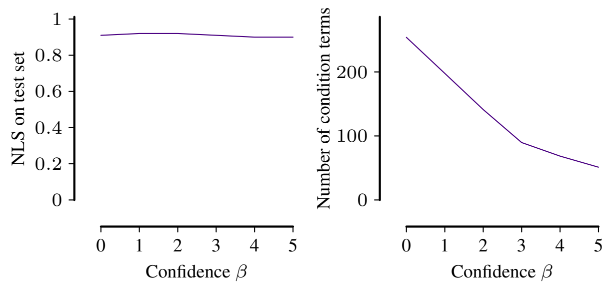


Figure 7: [**Hyperparameter sensitivity of GERD**] NLS on the testset (left) and number of discovered condition terms (right) for CONSEQUENCE using GERD on 1000 instances of the Rolling Mill dataset with varying confidence β.

For the Rolling Mill dataset, we could observe a slight increase of the NLS from 0.88 for $w = 1$ to 0.91 for $w = 20$, however, $w = 10$ was already large enough to reach a NLS of 0.91. Obviously, the choice of w has a larger influence on the training time of CONSEQUENCE, because a greater beam width increases the number of necessary computations in the cover algorithm. Since the relationship between training and beam size in the plot does look exponentially, we conclude that the used heuristic in the cover algorithm does its job, by steering the A* search into the right direction.

In Figure 7, we report on the influence of the confidence parameter β of GERD. Again, the prediction accuracy in terms of normalized Levenshtein similarity on the test set is relatively independent from the choice of β. Higher values of β require more evidence to include condition terms into the model and thus have a slightly worse fit on noise-free data. The choice of β has a huge impact on the number of condition terms in the discovered model. A lower β leads to significantly bigger models.

To produce the results in our experiments, we set $w = 10$ and $\beta = 2.0$. We especially chose $\beta = 2.0$, because the same value is used by the authors who proposed the rule effect estimator $\hat{e}$, where $\beta = 2.0$ corresponds to a 95.45% confidence level (Budhathoki, Boley, and Vreeken 2021).

Case Study for Rolling Mill Dataset

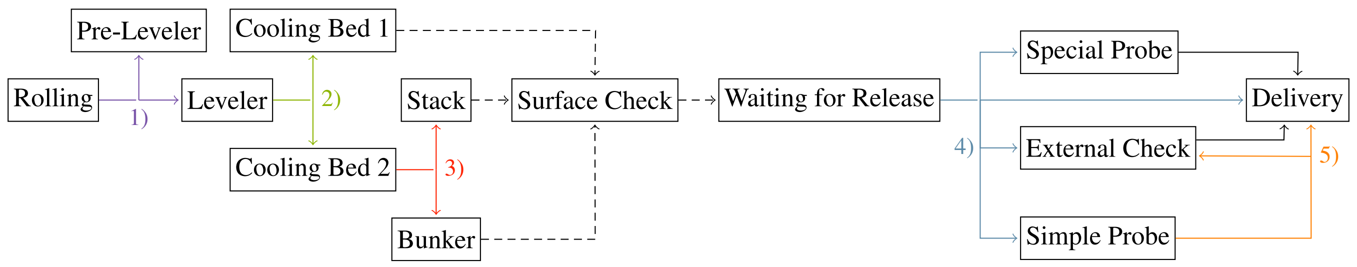
The event-flow graph found for the Rolling Mill dataset is equally understandable than model found for the Sepsis dataset. In Figure 8, we show the graph for the beginning and the end of the process. First, the plates are rolled at rolling-stands to meet their customer defined thickness. This happens at high temperatures and forces, otherwise, thickness reduction would not be possible. Therefore, the plate surface is not completely leveled by the large rolling stands and must be adjusted. Plates with a special accelerated cooling (ACC) treatment or with a special demand on their use, need an additional pre-leveler activity (rule 1).

After leveling, plates need to cool down, before their processing can continue. At this rolling mill, production splits up into a part for thicker plates and a part for thinner plates, which both have their own cooling beds (rule 2). After other processing steps with an intermediate surface check, plates wait at the end of the rolling mill for release. Before a plate can be delivered to the customer, probes must confirm that the plate meets the product quality requirements (rule 4). For some plates, an external inspector conducts additional checks (rule 5). In spite of the relatively high number of attributes, GERD has produced meaningful rules with a clear connection between attributes and activities in the process.

Complete Models for Sepsis Case Study

While we could only show excerpts of the discovered models for real-world data in the main paper, we use the supplementary space to provide more details about the discovered models on the Sepsis public dataset. In Figure 9, we show the complete model found by CONSEQUENCE, which is not remarkably harder to follow than the excerpt in Figure 4.

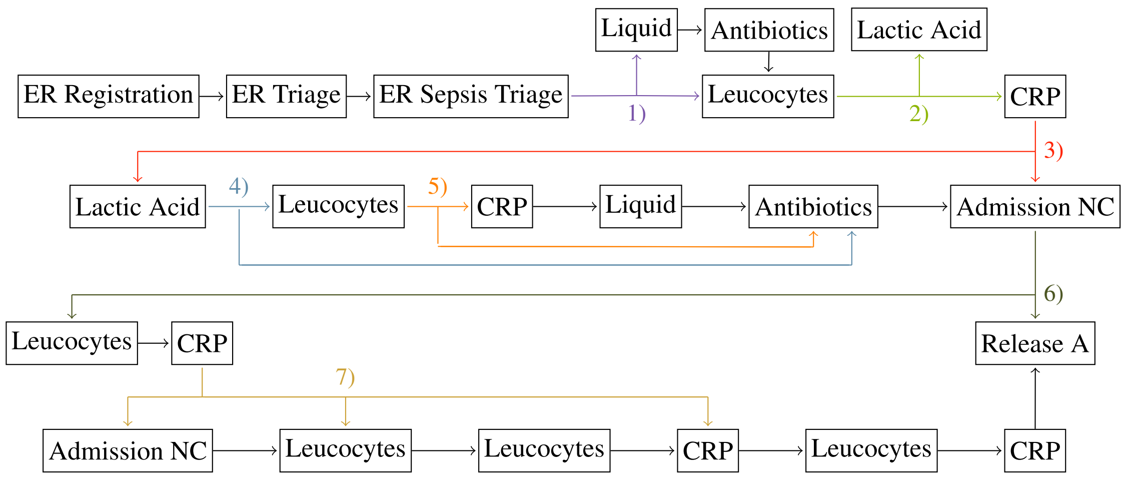
As one can see in Figure 10, independent classification rules as found by POSCL do not give a global picture of the process and are much harder to follow. Furthermore, the resulting rules show unnecessary redundancy, which makes it even more time consuming to analyse the model. A transition system discovered by DATS as shown in Figure 14 can visualize the flow of events in the process, however, the model also shows unnecessary complexity and redundancy in spite of filtering. In contrast to the model found by CONSEQUENCE, the model found by DATS does not contain simple, human understandable rules, that give global explanations for the generation of sequences.



1) [ACC $=$ false $\vee$ Purpose $= 13 \rightarrow$ Leveler, $\top \rightarrow$ Pre-Leveler]
2) [Thickness $> \tau_1 \rightarrow$ Cooling Bed 2, $\top \rightarrow$ Cooling Bed 1]
3) [Thickness $> \tau_2 \rightarrow$ Bunker, $\top \rightarrow$ Stack]

4) [Simple Probe Requested $=$ true $\rightarrow$ Simple Probe,
Special Probe Requested $=$ true $\rightarrow$ Special Probe,
Inspection $\in \{$WWF, WFF, FFF$\} \rightarrow$ External Check,
$\top \rightarrow$ Delivery]
5) [Inspector $=$ Dillinger $\rightarrow$ Delivery, $\top \rightarrow$ External Check]

Figure 8: [**Rolling Mill Event-Flow Graph**] Excerpt from the model found by ConSequence on the Rolling Mill dataset. Dashed arrows indicate skipped nodes.



1) [Oligurie $= 1 \vee$ Infusion $= 1 \rightarrow$ Liquid, $\top \rightarrow$ Leucocytes]
2) [Oligurie $= 1 \vee$ Hypotensie $= 1 \rightarrow$ Lactic Acid, $\top \rightarrow$ CRP]
3) [Infusion $= 1 \rightarrow$ Lactic Acid, $\top \rightarrow$ Admission NC]
4) [Infusion $= 0 \rightarrow$ Antibiotics, $\top \rightarrow$ Leucocytes]
5) [DiagnosticArtAstrup $= 1 \rightarrow$ CRP, $\top \rightarrow$ Antibiotics]

6) [SIRSCriteria2OrMore $= 0 \vee$ Age $\leq 34 \rightarrow$ Release A,
$\top \rightarrow$ Leucocytes]
7) [Age $\leq 27 \rightarrow$ CRP, DisfuncOrg $= 1 \rightarrow$ Leucocytes,
Hypotensie $= 1 \wedge$ Oligurie $= 1 \rightarrow$ Admission NC,
$\top \rightarrow$ Leucocytes]

Figure 9: [**Sepsis Event-Flow Graph**] Complete Model found by ConSequence on the Sepsis dataset. An excerpt of this model was given in the main paper in Figure 4.

1) [$\top \rightarrow$ ER Registration]

2) [$\top \rightarrow$ ER Triage]

3) [$\top \rightarrow$ ER Sepsis Triage]

4) [Infusion $= 1 \rightarrow$ Liquid, Age $\leq 37 \vee$ DiagnosticUrinaryCulture $= 1 \rightarrow$ Lactic Acid, $\top \rightarrow$ Leucocytes]

5) [DiagnosticSputum $= 1 \rightarrow$ Leucocytes, $\top \rightarrow$ CRP]

6) [InfectionSuspected $= 0 \wedge$ SIRSCriteria2OrMore $= 0 \rightarrow$ Admission NC, $\top \rightarrow$ Lactic Acid]

7) [Infusion $= 1 \rightarrow$ Liquid, DiagnosticLacticAcid $= 1 \wedge$ DiagnosticXthorax $= 1 \rightarrow$ Antibiotics,
DiagnosticXthorax $= 1 \wedge$ DiagnosticECG $= 1 \rightarrow$ Admission NC, DiagnosticUrinaryCulture $= 1 \rightarrow$ Antibiotics,
$\top \rightarrow$ Admission NC]

8) [Infusion $= 1 \rightarrow$ Antibiotics, DiagnosticBlood $= 1 \wedge$ SIRSCritTemperature $= 1 \rightarrow$ Admission NC,
$\top \rightarrow$ Leucocytes]

9) [Infusion $= 1 \vee$ SIRSCritLeucos $= 1 \rightarrow$ Admission NC, $\top \rightarrow$ CRP]

10) [DiagnosticIC $= 0 \wedge$ SIRSCriteria2OrMore $= 0 \rightarrow$ END, Age $\leq 34 \rightarrow$ Release A,
Hypotensie $= 1 \wedge$ Hypoxie $= 1 \rightarrow$ Lactic Acid, $\top \rightarrow$ CRP]

11) [SIRSCriteria2OrMore $= 0 \wedge$ DiagnosticIC $= 0 \rightarrow$ END, Hypotensie $= 1 \rightarrow$ LacticAcid, $\top \rightarrow$ CRP]

12) [Infusion $= 0 \rightarrow$ END, Hypotensie $= 1 \rightarrow$ LacticAcid, $\top \rightarrow$ CRP]

13) [Infusion $= 0 \rightarrow$ END, Hypotensie $= 1 \rightarrow$ LacticAcid, $\top \rightarrow$ CRP]

14) [Infusion $= 0 \vee$ Age $\leq 34 \rightarrow$ END, DisfuncOrg $= 1 \wedge$ Hypotensie $= 1 \rightarrow$ LacticAcid,
Hypoxie $= 1 \rightarrow$ Release A, $\top \rightarrow$ Leucocytes]

15) [Infusion $= 0 \rightarrow$ END, Hypoxie $= 1 \wedge$ Hypotensie $= 1 \rightarrow$ LacticAcid,
Hypotensie $= 1 \wedge$ Oligurie $= 1 \rightarrow$ CRP, $\top \rightarrow$ END]

16) [Oligurie $= 0 \wedge$ Hypotensie $= 0 \rightarrow$ END, $\top \rightarrow$ CRP]

17) [Oligurie $= 0 \wedge$ Hypotensie $= 0 \rightarrow$ END, $\top \rightarrow$ Leucocytes]

18) [Hypotensie $= 0 \rightarrow$ END, $\top \rightarrow$ CRP]

19) [Hypotensie $= 0 \rightarrow$ END, $\top \rightarrow$ Leucocytes]

20) [Hypotensie $= 0 \rightarrow$ END, $\top \rightarrow$ CRP]

21) [$\top \rightarrow$ END]

Figure 10: [**Sepsis Classification Rules**] Model found by PosCl on the Sepsis dataset.

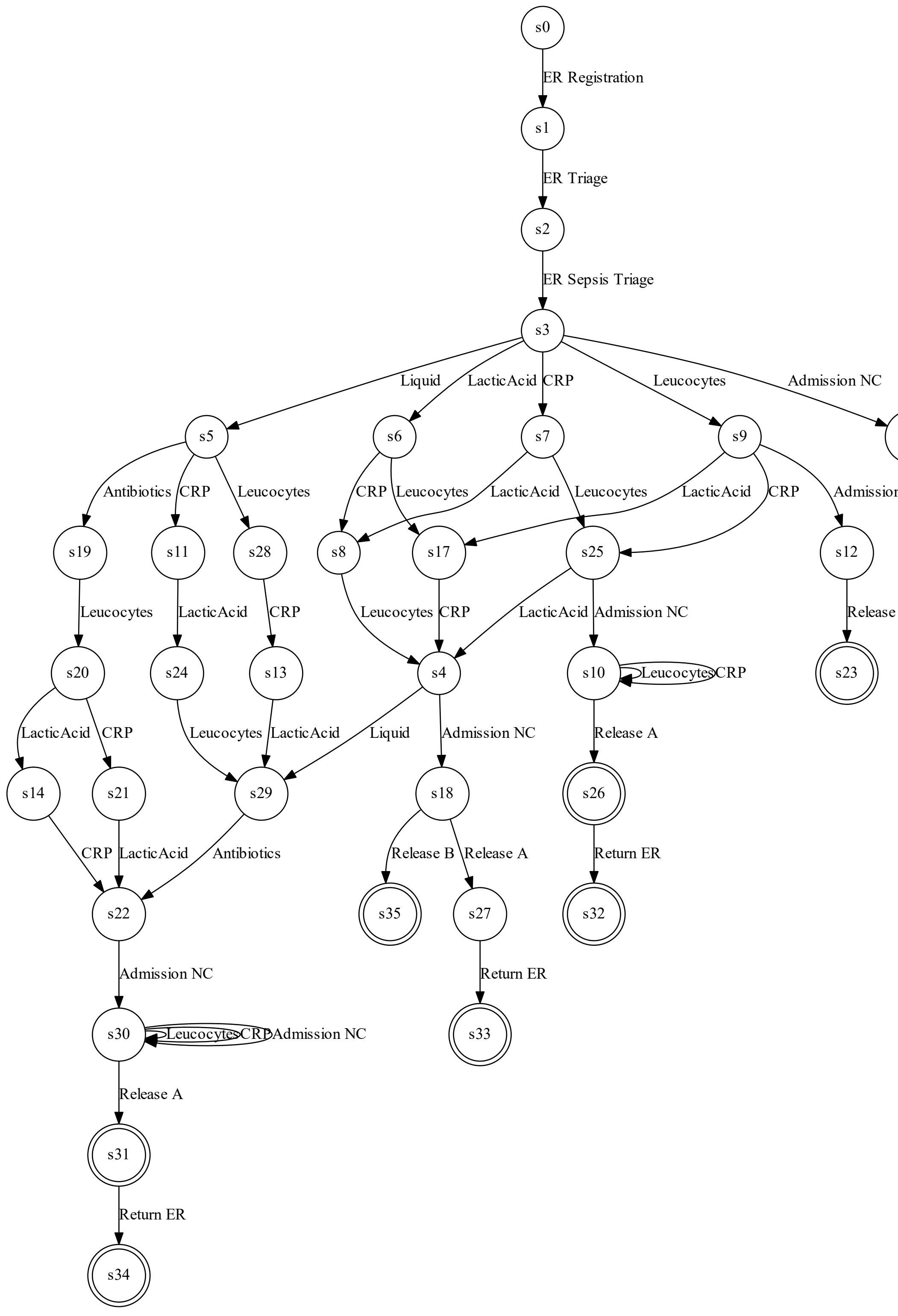


Figure 11: [**Sepsis Transition System**] Model found by DATS on the Sepsis dataset with set state abstraction. To reduce the size of the transition system, we only used sequences with at least 30% frequency compared to the most frequent sequence.

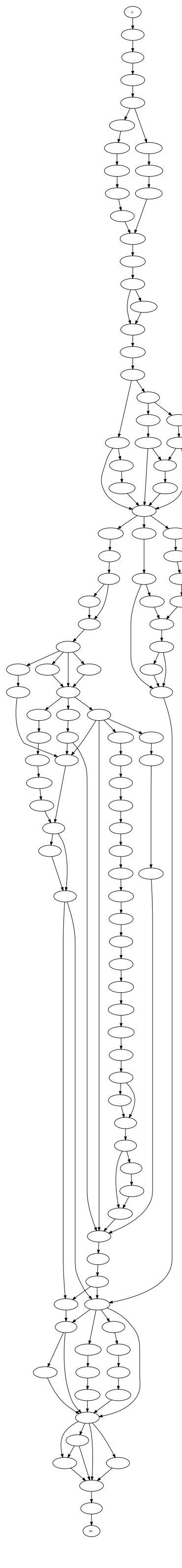


Figure 12: [**Rolling Mill Event-Flow Graph**] Complete graph found by CONSEQUENCE on the Rolling Mill dataset.