Data is Moody: Discovering Data Modification Rules from Process Event Logs

Marco Bjarne Schuster° Boris Wiegand• Jilles Vreeken*

Abstract

Although event logs are a powerful source to gain insight about the behavior of the underlying business process, existing work primarily focuses on finding patterns in the activity sequences of an event log, while ignoring event attribute data. Event attribute data has mostly been used to predict event occurrences and process outcome, but the state of the art neglects to mine succinct and interpretable rules how event attribute data changes during process execution. Subgroup discovery and rule-based classification approaches lack the ability to capture the sequential dependencies present in event logs, and thus lead to unsatisfactory results with limited insight into the process behavior.

Given an event log, we are interested in finding accurate yet succinct and interpretable if-then rules how the process modifies data. We formalize the problem in terms of the Minimum Description Length (MDL) principle, by which we choose the model with the best lossless description of the data. Additionally, we propose the greedy MOODY algorithm to efficiently search for rules. By extensive experiments on both synthetic and real-world data, we show MOODY indeed finds compact and interpretable rules, needs little data for accurate discovery, and is robust to noise.

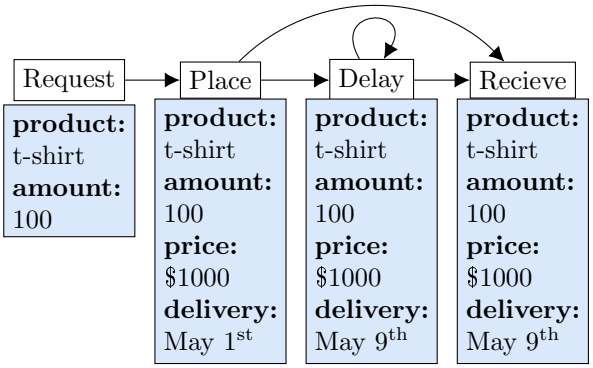

Figure 1: [**Processes change data**] Exemplary process for ordering textiles with activities *Request*, *Place*, *Delay* and *Receive*. Arrows indicate the flow of events. Further, we show how the data of an exemplary order changes throughout the process.

1 Introduction

Given a process event log, process mining [29] provides a better understanding of the underlying process and enables downstream tasks such as monitoring, anomaly detection, simulation, and optimization. Existing work, however, focuses on discovering patterns of event activities, but neglects how event attribute data changes during the process. For instance, a textile company may have an ordering process with the activities *Request*, *Place*, *Delay* and *Receive*. Process discovery algorithms [1, 27] only infer a graph of event activities, where nodes refer to activities and edges visualize the flow of execution. However, how data changes over time is crucial for understanding the process as we show in Figure 1. In our textile example, we only know the price and the delivery date of an order after placing the order. Delaying the delivery of an order, e.g. due to a shortness of supplies, changes the delivery date.

Surprisingly, data has only been used to predict the occurrences of event activities or the outcome of processes [28, 31]. To the best of our knowledge, none of the existing work mines interpretable rules for data modifications, and related work does not satisfactorily transfer to our problem. Subgroup discovery [19] and rule-based prediction methods [32] lack the ability to model the sequential dependencies present in event logs, and thus lead to unsatisfactory results with limited insight into the process behavior.

Given an event log, we are interested in finding accurate yet succinct and interpretable if-then rules how the process modifies data. To this end, we formalize the problem in terms of the Minimum Description Length (MDL) principle, by which we choose the model with the best lossless description of the data. Additionally, we propose our method MOODY, which is short for MOdification rule DiscoverY, to efficiently search for rule models in practice. Starting with an empty set, we greedily add the best compressing rule to the model, until we no longer find a rule that improves our MDL score. Through extensive experiments on both synthetic and real-world data, we show MOODY indeed finds succinct and interpretable rules, needs little data for accurate discovery, and is robust to noise.

The contributions of our paper are as follows. We (a) formulate the problem of finding data modification

°Airbus Operations GmbH, Bremen, Germany.
marco.schuster@airbus.com

•Stahl-Holding-Saar, Dillingen, Germany.
boris.wiegand@stahl-holding-saar.de

*CISPA Helmholtz Center for Information Security, Germany.
jv@cispa.de

rules from event logs,
(b) formalize the problem using the Minimum Description Length (MDL) principle,
(c) propose the MOODY algorithm to efficiently find accurate yet succinct data modification rules,
(d) run extensive experiments on both synthetic and real-world data,
(e) make code, data and appendix publicly available.[1]

Our paper is structured as usual.

2 Related Work

While the problem of finding interpretable data modification rules from process event logs has been neglected so far, related work on similar problems exists. Krismayer [12] discovers if-then rules for data modification from software execution logs. However, he only creates a large set of potentially redundant candidates, which must be manually filtered by domain experts. Other work [8, 30] infers extended finite state machines from software execution logs. Since business process event logs in contrast to software event logs are usually noisy and contain nondeterministic behavior, these methods are not applicable to our problem.

While process mining [29] focuses on business process event logs, most of the work only models the flow of process activities, and little work deals with additional data. Mannhardt et al. [14] and Mozafari et al. [17] detect at which event data changes, but do not model conditions or update values for these changes. Schönig et al. [26] find rules which cover data modifications. However, since they rely on support and confidence to filter a large set of candidate rules, their method suffers from pattern explosion, i.e., it finds many redundant rules.

Rule-based prediction is closely related to our problem. CLASSY [20] and its successor TURS [32] find classification rules by minimizing an MDL score. Both methods, however, require defining features and predicted variable beforehand, whereas we are interested in finding relationships without any initial knowledge of the data. Similarly, subgroup discovery algorithms such as SSD++ [19] find rules for differently behaving subgroups of a given dataset. None of these methods are able to model sequential relationships present in event logs.

In contrast to the above, MOODY finds compact and interpretable rules for data modifications from process event logs, needs little data for accurate discovery, and is robust to noise.

3 Preliminaries

Before formalizing the problem, we introduce preliminary concepts and notation we use in the paper.

[1] https://github.com/masc00008/moody

3.1 Notation for Data Modification Rules

As input for finding data modification rules, we consider an event log or dataset D collecting traces of a single process. Each trace refers to an instance of the process, such as a specific customer order, and consists of an event sequence. We describe data attributes of events by a set of numerical and categorical variables V.

To model how a process modifies these variables, we use different types of update rules. For a categorical variable $v \in V$, we write $v \in \{\alpha, \beta, \dots\}$, i.e., v takes one of the values in the set. For a numerical variable $v \in V$, we can set v to a specific value, $v = \alpha$, or to a range of values, $v \in [\alpha, \beta]$. We further denote relative changes by $v = v + \alpha$, $v = v + [\alpha, \beta]$, and $v = \alpha \cdot v$.

Updates typically only occur in certain circumstances. For instance, the price of an order may be dependent on the order volume, where a higher volume gives discount. Therefore, we model conditions for update rules. In the simplest case, we check for a specific value $v = \alpha$ or $v \neq \alpha$. Further, we test lower and upper bounds of numerical values by $v \leq \alpha$ and $v \geq \alpha$. Finally, we can condition on value transitions between the last and the current event with $v : \alpha \to \beta$.

We combine a condition c and an update rule u into a data modification rule **IF** c **THEN** u. To join multiple rules into a model M that covers the full complexity of the process, we use an unordered set of rules, since this allows for an independent interpretation of each rule [32]. Furthermore, since we want to avoid contradictory predictions of our model, we only allow acyclic models. For instance, if we condition on v_1 to update v_2, we are not allowed to do the reverse in the same model.

3.2 Minimum Description Length

We use the Minimum Description Length (MDL) principle [9,21] for model selection. MDL defines the best model as the one with the shortest lossless description of the given data. Formally, the best model minimizes $L(M) + L(D \mid M)$, in which $L(M)$ is the length in bits of the description of M, and $L(D \mid M)$ is the length of the data encoded with the model. This form of MDL is known as two-part or crude MDL. Although one-part or refined MDL has stronger theoretical guarantees, it is only computable in specific cases [9]. Therefore, we use two-part MDL. In MDL, we only compute code lengths, but are not concerned with actual code words. Next, we formalize our problem in terms of MDL.

4 MDL for data modifications

From an event log D, we aim to find a model M of data modification rules, which accurately describe the data, yet are as succinct as possible, such that domain experts gain insight into the process. Since real-world data is

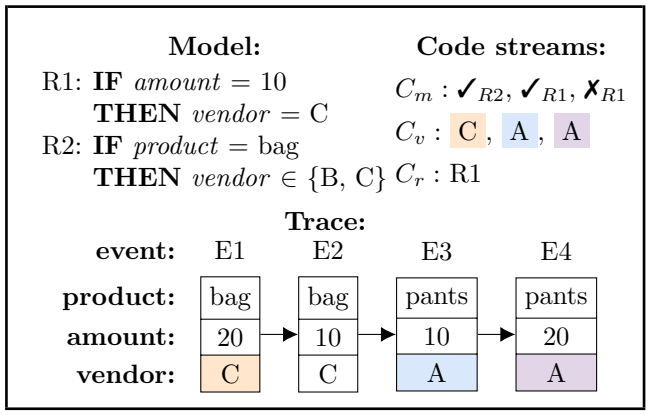

Figure 2: [**Data encoding**] We use the model (top left), the model stream C_m, value stream C_v and rule selection stream C_r (top right) to decode the trace's categorical variable *vendor* (bottom).

usually noisy, we need a robust model selection criterion. Therefore, we formalize the problem in terms of the MDL principle. To this end, we define length of the data encoding $L(D \mid M)$, length of the model encoding $L(M)$, and finally give a formal problem definition.

4.1 Data encoding To encode a given event log with a model of data modification rules, we encode the variable values at each event. For each event, we check which rules in the model fire, i.e., have satisfied conditions, and use the firing rules to encode the data values. Whenever the model makes ambiguous predictions, we encode the specific value among all possibilities. If no rule fires, we choose and encode a value from the whole domain of the target variable. To ensure a lossless encoding and to handle noisy data, we also encode any errors made by the model.

Conceptually, we split the data encoding into three code streams: In the rule selection stream C_r, we encode which of the rules with matching conditions we choose to encode the current variable value. Then, we encode in the model stream C_m if the model predicts the correct value. If not, any value of the target domain is possible. Whenever the model predicts multiple values, we choose a value by a code in the value stream C_v.

We give a toy example of a data encoding in Figure 2, which we use to describe how to decode the categorical variable *vendor*. First, at event E1, we see that only rule R2 applies. Thus, we do not need to select a rule by reading from C_r. Next, we find a checkmark as the first symbol in C_m, i.e., the model predicts correctly. However, the rule allows two values, B and C, which we disambiguate by reading C from C_v. Next, at event E2, we observe that both rules apply. Therefore, we check C_r to find that we should use rule R1 whose prediction is correct according to the second symbol in C_m. Further, its prediction is not ambiguous and we get the value C in the trace. Afterwards, at event E3, we find that only rule R1 applies but its prediction is incorrect according to the last element in C_m. We obtain the correct value by reading A from C_v. Finally, no rule applies at event E4, so we neither read from C_m nor from C_r. Instead, we read the last value from C_v, A, by which we have successfully decoded the values for *vendor*.

We compute the length of the data encoding by summing the code lengths in C_r, C_m and C_v. Whenever we have to disambiguate multiple firing rules in C_r, we assume all rules in the model are equally important. We compute the encoded length of C_r by

$$L(C_r) = \sum_{i=1}^{|C_r|} \log |R_i| \ ,$$

where $|R_i|$ denotes the set of firing rules at the i-event.

When we compute the encoded length of C_m, we do not know the probabilities of codes for checkmarks and crosses in advance. Therefore, we use a prequential plug-in code [5, 23] to compute $L(C_m)$: We initialize uniform counts for checkmarks and crosses and update counts after each event, such that we have a valid probability distribution at each step in the encoding. Asymptotically, this gives an optimal encoded length of C_m. Formally, we have

$$L(C_m) = \sum_{i=1}^{|C_m|} \frac{\mathrm{usg}_i\, C_m[i] + \epsilon}{\mathrm{usg}_i\, ✓ + \mathrm{usg}_i\, ✗ + 2\epsilon} \ ,$$

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

To compute code lengths in C_v, we use the conditional empirical probability of values. Let u_i be the update rule we selected in C_r to encode the i-th value in C_v. If no rule fires or we encoded an error in C_m, u_i falls back to all possible values in the domain of the target variable. We formally define

$$L(C_v) = -\sum_{i=1}^{|C_v|} \log \frac{\mathrm{fr}(C_v[i])}{\sum_{j \in u_i} \mathrm{fr}(j)} \ ,$$

where $\mathrm{fr}(C_v[i])$ denotes how often value $C_v[i]$ occurs in the data, and we normalize this by the frequencies of all values j possible according to the update rule u_i. To encode numerical variables, we assume a histogram-based discretization, by which we can compute all necessary probabilities and code lengths.

Altogether, this gives us a lossless data encoding.

4.2 Model encoding To define the length of the model encoding $L(M)$, we need to encode the number of rules in the model and all conditions c and update rules u in the model. Since the number of rules is unbounded, we use the universal MDL encoding for natural numbers $L_{\mathbb{N}}$ [22] that encodes any natural number $x \geq 1$ as $L_{\mathbb{N}}(x) = \log(2.865064) + \log(x) + \log(\log(x)) + \ldots$, where we sum only the positive terms and the first term ensures that Kraft's inequality holds, i.e., $L_{\mathbb{N}}$ is a lossless encoding. Denoting the number of modification rules in the model as $|M|$, which can be zero, we then obtain the encoded length of the model as

$$L(M) = L_{\mathbb{N}}(|M| + 1) + \sum_{(c,u) \in M} L(c) + L(u) \ .$$

To encode a condition c, we specify its type, which single variable $v \in V$ is tested by c, and all constants used in the condition. Formally, we define

$$L(c) = \log |\{=, \neq, \leq, \geq, \rightarrow\}| + \log |V| + \sum_{\alpha \in c} L(\alpha) \ .$$

To encode the value $\alpha \in \operatorname{dom} v$ for a categorical variable v, we have

$$L(\alpha) = \log |\operatorname{dom} v| \ ,$$

and to encode a real-valued α, we consider it in scientific notation $\alpha = k \cdot 10^s$, and have

$$L_{\mathbb{R}}(\alpha) = 2 + L_{\mathbb{N}}(|s| + 1) + L_{\mathbb{N}}(\lceil |k| \cdot 10^s \rceil + 1) \ ,$$

where we first encode the signs of k and s with 1 bit each, then the value of s, and finally the value of k up to a user-specified precision of p significant digits [15].

To encode an update rule u on a variable v, we first specify the type of u, which is one of $v \in \{\alpha, \beta, \ldots\}$, $v = \alpha$, $v \in [\alpha, \beta]$, $v = v + \alpha$, $v = v + [\alpha, \beta]$, or $v = \alpha \cdot v$, for which we need $\log 6$ bits. Then, we encode which variable $v \in V$ is updated by u, and finally we encode the constants in u the same way we do for conditions. Formally, we have

$$L(u) = \log 6 + \log |V| + \sum_{\alpha \in u} L(\alpha) \ .$$

This gives us the encoded length of the model $L(M)$.

4.3 Formal problem definition With this, we now have all the ingredients to formally define our problem.

Minimal Modification Rules Problem *Given an event log D with variables V, find an acyclic model of data modification rules M that minimizes the total encoding cost $L(D, M) = L(M) + L(D \mid M)$.*

In practice, it is infeasible to solve this problem optimally due to the potentially large number of acyclic models. To approximate this number by a lower bound, we consider the *rule dependency graph* of a model, which is a directed acyclic graph with variables as nodes and their dependencies induced by modification rules as edges. We give a simple example of a rule dependency graph in Figure 3. Since each edge requires at least one rule, there are at least as many models as rule dependency graphs. According to Rodionov [24], the number of acyclic graphs with n nodes and up to m edges is

$$A(n, m) = \sum_{i=1}^{n} \sum_{j=0}^{m} (-1)^{i-1} \binom{n}{i} \binom{i(n-i)}{m-j} A(n-i, j) \ ,$$

with $A(1, \cdot) := 1$. Because $A(n, m)$ grows exponentially in n and m [4, p. 1186], the number of acyclic models grows exponentially in the number of variables $|V|$ and the number of modification rules in the model $|M|$.

Furthermore, our search space has no trivial structure such as submodularity or monotonicity, which we could exploit to find an optimal solution in feasible time. We give counterexamples for both properties in the supplementary material. Hence, we resort to heuristics.

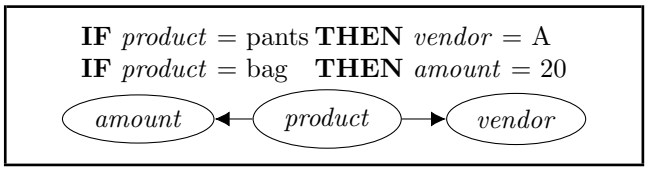

Figure 3: [**Rule dependency graph**] We represent the variable dependencies of the model (top) as a graph (bottom) where ovals represent variables and arrows show whether and in which direction a modification rule induces a dependency between variables.

5 The Moody algorithm

To efficiently discover good sets of data modification rules in practice, we prune the exponentially sized search space with a quickly computable estimate of our score that avoids repetitively passing the whole event log, and we introduce a greedy search.

5.1 Estimating the MDL score Computing the MDL score $L(D, M)$ requires a pass over all events in the event log, because we must check for each event, which of the rules in the model fire. To avoid iterating over the event log each time we evaluate adding a new rule to the model, we prune the large set of potential candidate rules by a quickly computable estimate $\widehat{L}(D, M)$. We optimistically assume that a newly generated rule is the only rule in the model which predicts its target variable, such that we do not need to update C_r or C_m.

Algorithm 1: Estimate $L(C_v)$

Input: Rule (c, u)
Output: $\widehat{L}(C_v \mid c, u)$

1 $\widehat{L}(C_v \mid c, u) \leftarrow 0$;
2 $b \leftarrow \text{supp}(c)$;
3 **forall** $j \in u$ ordered by increasing $\text{fr}(j)$ **do**
4 $\quad \Delta b \leftarrow \min\{b, \text{fr}(j)\}$;
5 $\quad \widehat{L}(C_v \mid c, u) \leftarrow \widehat{L}(C_v \mid c, u) - \Delta b \cdot \log \frac{\text{fr}(j)}{\sum_{i \in u} \text{fr}(i)}$;
6 $\quad b \leftarrow b - \Delta b$;
7 **return** $\widehat{L}(C_v \mid c, u)$;

By assuming independence between rules, we can independently estimate the contribution of a single rule with condition c and update rule u to $L(C_v)$. We give the pseudocode for estimating $L(C_v \mid c, u)$ as Algorithm 1. First, we compute the support $\text{supp}(c)$, i.e., at how many events c fires (ln. 2). For each value j predicted by u, we compute how many codes we must add to C_v, which is the minimum of the remaining events to cover, b, and the frequency $\text{fr}(j)$ of j (ln. 4). We compute the length of all these codes and add it to our estimate (ln. 5). At the end of each iteration, we update how many codes we still must add to C_v (ln. 6).

While computing the support requires a pass over the dataset, we only need a single pass when creating the candidate. By assuming independence between rules, we do not need to update the estimated code lengths of all candidate rules, every time we add another rule to the model. Using $\widehat{L}(D, M)$ we can prune out rules with high encoding costs, and thus avoid computing $L(D, M)$ for those. Next, we use $\widehat{L}(D, M)$ and $L(D, M)$ to find good modification rules for a given event log.

5.2 Finding Good Modification Rules To reduce the search space, we let the domain experts control how much time they want to invest in model search, and introduce two hyperparameters for a greedy search. Instead of generating conditions with all possible combinations of variables, signs and values, we only generate the N_c most frequently observed combinations of variables and values in the dataset for each sign of the set $\{=, \neq, \leq, \geq, \rightarrow\}$. For a given condition, instead of generating update rules with all possible combinations of variables and values, we only generate the N_u most frequent combinations of variables and values in the dataset for each type of update rule.

We provide the pseudocode of our greedy search MOODY as Algorithm 2. Our search starts with an empty model (ln. 1). We iteratively extend this model by modification rules for all target variables (ln. 3).

Algorithm 2: MOODY

Input: event log D with variables V
Output: model of data modification rules M

1 $M \leftarrow \emptyset$;
2 **do**
3 $\quad$ **forall** $v \in V$ **do**
4 $\quad\quad Q \leftarrow$ priority queue of rules r predicting v ordered by $\widehat{L}(D, M \cup \{r\})$;
5 $\quad\quad r^* \leftarrow \emptyset$;
6 $\quad\quad$ **while** $Q \neq \emptyset$ **and** $\widehat{L}(D, M \cup \{\text{top of } Q\}) < L(D, M \cup \{r^*\})$ **do**
7 $\quad\quad\quad r' \leftarrow$ pop element from Q;
8 $\quad\quad\quad r^* \leftarrow \arg\min_{r \in \{r^*, r'\}} L(D, M \cup \{r\})$
9 $\quad\quad$ **if** $L(D, M \cup \{r^*\}) < L(D, M)$ **then**
10 $\quad\quad\quad M \leftarrow M \cup \{r^*\}$;
11 **while** M was extended in the last iteration;
12 **return** M;

Since computing $L(D, M)$ needs a pass over the whole dataset, we manage the candidate rules in a priority queue sorted by an estimate of our score $\widehat{L}(D, M)$ (ln. 4), such that we check promising rules early. Next, we search the candidates from most promising to least promising and compute their actual encoded length L (ln. 6-8). To not waste computation time for computing L on inferior rules, we perform this search as long as the estimate $\widehat{L}$ is better than the best actual code length L that we have seen so far (ln. 6). After evaluating the candidates, we only add the best candidate to our model if it reduces the total encoded length (ln. 9-10). Finally, we end when no candidate for any target variable could improve our score (ln. 11) and return the resulting model (ln. 12).

In the worst case, all generated candidates improve our MDL score. Since the number of candidates grows linearly with N_c and N_u, the outer loop of MOODY grows linearly with N_c and N_u. In the worst case, our estimate does not prune any candidate, and we must compute our score for all of the $O(N_c \cdot N_u)$ candidates. Since we loop over all variables, and computing our score requires a pass over the whole dataset, the runtime complexity of MOODY is $O\left((N_c \cdot N_u)^2 \cdot |V| \cdot |D|\right)$.

6 Experiments

In this section, we empirically evaluate MOODY on both synthetic and real-world data. When we defined our MDL score, we assumed discretization of numerical variables. In our prototype implementation, we discretize numerical values into variable-width histograms. For

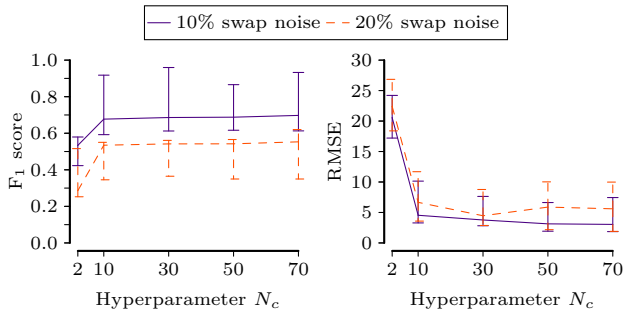

Figure 4: [**Choosing** N_c] Median F_1 score on categorical variables (left) and median RMSE on numerical variables (right) for different values of MOODY's hyperparameter N_c for 10% and 20% swap noise in the training set. Error bars show interquartile ranges.

efficiency, we determine the histogram boundaries by percentiles. We use 50 bins in all our experiments.

We run all our experiments in a Docker-based environment on a Linux server with an Intel® Xeon® Gold 6244 CPU. In all experiments, we observe 16 GB of RAM suffice. As a simple baseline, we consider the empty model $M = \emptyset$. In addition, we learn if-then-else rules using the rule-based classifier TURS [32], and using the subgroup discovery method SSD++ [19]. To ensure reproducibility of all our results, we provide code, data, and further details in the supplementary material.

6.1 Synthetic event logs To control data properties such as noise, we first experiment on synthetic event logs, such that we know the generating ground-truth rules. We randomly generate ten independent ground-truth models in our modeling language, where each model contains five rules, two categorical variables and two numerical variables. Since SSD++ cannot model sequential dependencies $v : \alpha \to \beta$, we only create models with conditions $v \leq \alpha$, $v \geq \alpha$ and $v = \alpha$. From these ground-truth models, we generate event logs with $|D| = 2000$ events. To test noise-robustness, we add different amounts of noise, where we randomly swap values of variables. 10% swap noise means that for each variable in the dataset, we randomly swap 10% of its values.

TURS in contrast to SSD++ does only discover rules for categorical target variables, and cannot predict numerical values. Therefore, we separately evaluate on models that only predict categorical variables and on models that only predict numerical variables. However, in both setups, we generate conditions containing categorical and numerical variables.

Choosing Hyperparameters Such that the user can control runtime by reducing the search space, we introduced hyperparameters N_c and N_u when we proposed MOODY. For efficiency, we only search for the most compressing update rule given a condition and set N_u to 1. To evaluate the accuracy of a set of rules, we compute its median F_1 score on predicting categorical variables, and its median root mean squared error (RMSE) in predicting numerical variables. We report the influence of N_c on prediction accuracy in Figure 4. We see that the prediction accuracy drops for only very small N_c, and is relatively constant for larger values. This means, the conditions with the highest support in the dataset tend to give the best compression and accuracy. To have a safety margin on unknown data, we set N_c to 50.

Results on categorical variables Next, we compare results on synthetic data with different amounts of swap noise for MOODY against all baselines in Figure 5. We see in the left plot that MOODY achieves by a wide margin the highest F_1 score on categorical variables for data with 0% and 10% noise. Up to 20% noise, MOODY shows good noise-robustness and still has the highest median F_1 score. For 30% noise, the F_1 score of MOODY drops down to the F_1 score of the empty model. We note that 30% noise may sound low, but swapping 30% of the values for each variable in the dataset accumulates to a much higher noise-ratio. Hence, MOODY predicts well under reasonable amounts of noise.

Results on numerical variables We see similar results on predicting values of numerical variables in the center plot of Figure 5. Since TURS cannot predict numerical variables, we compare MOODY to SSD++ and the empty model. MOODY shows by far the smallest test root mean squared error (RMSE) for training data with up to 20% noise. For 30% noise, MOODY's RMSE converges to the RMSE of the empty model. We again note swapping 30% of the values for each variable in the dataset accumulates to a much higher noise-ratio than we expect in any real-world event log. Hence, MOODY predicts well under reasonable amounts of noise.

Model complexity Not only does MOODY give the best prediction results under reasonable amounts of noise. It also discovers the rule sets with the lowest total number of rule terms as we show in the right plot of Figure 5. Both TURS and SSD++ find rule sets with a significantly higher number of rule terms. We see MOODY's models have similar complexity compared to the ground-truth models for data with low amounts of noise. If the noise level increases, MOODY converges to the empty model. This means, MOODY is robust against finding spurious rules on noisy data.

Sample complexity To empirically evaluate sample complexity, we compute F_1 scores on the test set dependent on the number of events in the training set. For a realistic setup, we add 10% swap noise to all training

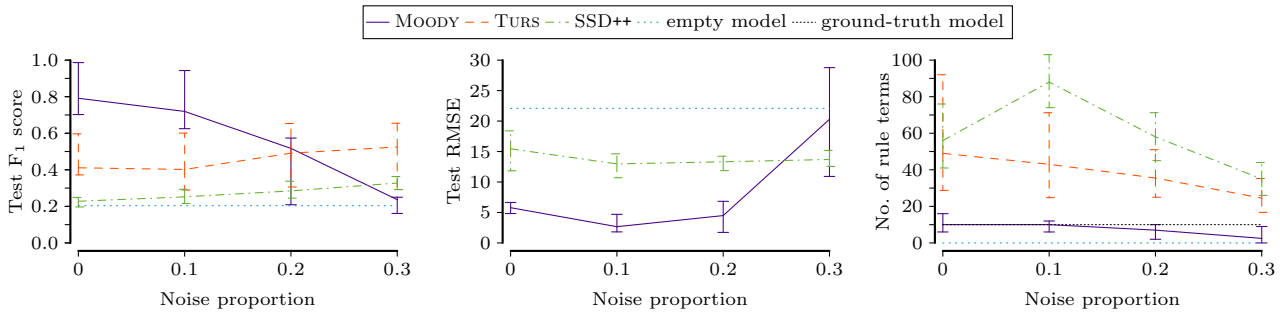

Figure 5: [**Moody predicts well under reasonable amounts of noise**] Median F_1 scores on categorical variables (left, higher is better), median root mean squared errors on numerical variables (center, lower is better) and number of rule terms in the discovered models (right, lower is less complex) at different noise levels for Moody, SSD++ and Turs. Error bars indicate interquartile ranges.

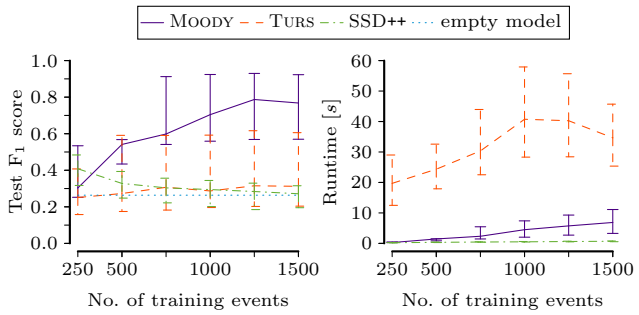

Figure 6: [**Moody shows low sample complexity and scales well**] Median test F_1 score on categorical variables (left) and median runtime (right) dependent on the number of training events for Moody, Turs and SSD++. Error bars show interquartile ranges.

sets. We report results for Moody, Turs, SSD++ and the empty model in the left plot of Figure 6. We see Moody predicts better the more training data is available, while all baselines do not improve with more training data. Already with 500 training events, Moody shows higher median F_1 score than the baselines.

Runtime We report wall-clock runtime for single-threaded execution dependent on the number of training events in the right plot of Figure 6. As we expect by our theoretical runtime analysis, we see that Moody scales well and shows a growth of runtime linear to the number of training events. While SSD++ shows a constantly fast, almost zero runtime, Moody still finishes within reasonable time and is significantly faster than Turs.

6.2 Real-world event logs Next, we evaluate on two publicly available real-world event logs, for which we give the base statistics in Table 1. The first one, *Sepsis* [13], contains event traces from treating Sepsis patients in a Dutch hospital. The second one, *Traffic Fines* [6], [14, p. 20] is an event log of handling road-traffic fines by the police of an Italian city. To reduce runtime, we randomly sample 20% of the original 150370 traces in the Traffic Fines event log. Furthermore, we parallelize candidate generation and candidate evaluation of Moody on twelve CPU cores.

First, we look at the insight we gain from some exemplary rules we find with Moody on these event logs. Then, we evaluate how well the discovered rules generalize to unseen test data.

Sepsis rules On the Sepsis dataset, Moody discovers in total 82 rules with an approximate runtime of

Data	$\|D\|$	$\|\|D\|\|$	$\|\Omega\|$	$\|V_{\text{cat}}\|$	$\|V_{\text{num}}\|$
Sepsis	782	15214	18	25	7
Traffic Fines	30074	112245	11	4	5

Table 1: [**Real-world event logs statistics**] Number of traces $|D|$, number of events $||D||$, number of different activities $|\Omega|$, number of categorical variables $|V_{\text{cat}}|$, and number of categorical variables $|V_{\text{num}}|$ for the Sepsis and Traffic Fines real-world datasets.

IF *group* = C **THEN** *activity* = ER Triage
IF *group* = E **THEN** *activity* = Release A
IF *group* = W **THEN** *activity* = Admission IC
IF *group* = P **THEN** *activity* = Admission IC
IF *group* = F **THEN** *activity* = Admission NC
IF *group* = O **THEN** *activity* = Admission NC

Figure 7: [**Responsibility rules for the Sepsis event log**] Different groups in the hospital are associated with different activities for handling sepsis patients.

IF *Leukocytes* ≤ 4.7 **THEN** *Infusion* = True
IF *Leukocytes* ≤ 2.8 **THEN** *Diagnose* = GB
IF *Leukocytes* $= 7.8$ **THEN** *Diagnose* = AA

Figure 8: [**Leukocytes rules for the Sepsis event log**] Lower values for leukocytes (white blood cells) are associated with certain diagnoses and infusions.

IF *amount* ≤ 68.77 **THEN** *points* $= 0.0$
IF $143.00 \leq$ *amount* **THEN** *points* $\in [0.0, 10.0]$
IF $131.00 \leq$ *amount* **THEN** *points* $= 0.0$
IF *amount* $= 80.00$ **THEN** *points* $\in [0.0, 2.0]$

Figure 9: [**Traffic Fines rules**] Moody finds multiple relationships between the fine *amount* and the *points* deducted from the offender's driving license.

two hours. 23 of these rules express a correlation between the *group* attribute and the activity of an event, for which we show a subset in Figure 7. The rules found by Moody indicate that certain groups in the hospital are specialized in certain activities.

Furthermore, Moody finds rules, where leukocytes measurements imply the diagnosis and the treatment of sepsis, as we show in Figure 8. These rules enable to ask targeted questions to domain experts and thus can be a valuable start to gain insight into this process.

Traffic Fines rules On the Traffic Fines dataset, Moody discovers in total 117 rules with an approximate runtime of 4.5 hours. While we would expect that a higher fine *amount* correlates with a high number of *points* deducted from the offender's driving license, the rules found by Moody contradict this intuition, as we show in Figure 9. A closer look on the dataset indeed confirms that there is no monotonic relationship between these two attributes. Finding counter-intuitive but data-supported rules like these gives valuable insight into the underlying process.

Generalization Finally, we evaluate how well the rules discovered by Moody generalize to unseen data. To this end, we split the Traffic Fines event log into a training set and a test set with a distinct 20% of traces each. Then, we compare the F_1 score respectively RMSE on the training set and the test set for each rule, which Moody discovers on the training set. We show results in Figure 10. As we see, most of the rules have a low prediction error on both sets. The gap between training and test performance is small, which means that Moody finds well-generalizing rules.

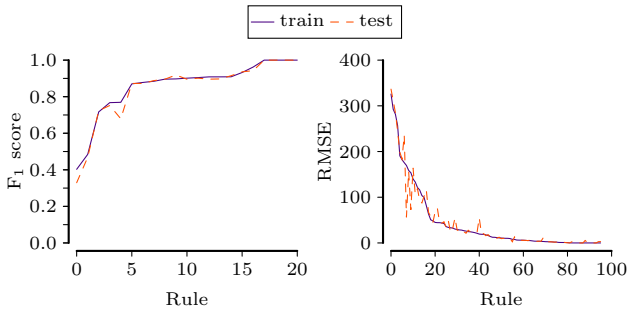

Figure 10: [**Moody generalizes well**] Train and test F_1 score on categorical variables (left) and train and test root mean squared error on numerical variables (right) for each rule found by Moody on the Traffic Fines log.

7 Discussion

In our experiments, Moody does not only discover simple and thus interpretable rules to unveil the data modifications within a process. It also finds rules that accurately predicts the data values, and is more robust to sensible amounts of noise than all baselines.

Yet, we see multiple interesting research directions to improve Moody. First, extending the modeling language of Moody would allow understanding data modifications of very complex processes. For example, this involves rules with complex conditions joined by *(and)* and *(or)*. We see adapting our MDL score to such rules is relatively easy.

A more complex modeling language implies an even larger search space, and thus we see the need to improve the runtime efficiency of Moody. As in many MDL-based methods, the bottleneck of Moody is discrete combinatorial search. Hence, we see an approach for mining data modification rules based on differentiable pattern set mining [7] as a promising future direction.

Furthermore, we would like to put a stronger focus on causality of the discovered rules. To this end, we may use well-defined measures for the causal effect of a rule [3]. Alternatively, we would like to examine the link between causality and two-part MDL codes in terms of algorithmic independence [16], and how to use this during search for causal data modification rules.

Last but not least, we also see many interesting applications for Moody. As the most compressing rules found by Moody define normal behavior, it would be interesting to use them for anomaly detection [18]. Since the behavior of real-world processes usually changes over time, we see Moody could help to identify and understand concept drift [2,25]. Finally, predicting data attributes with Moody may be used in the simulation of process behavior for process optimization [10].

8 Conclusion

We studied the hitherto largely neglected problem of discovering accurate yet concise and interpretable rules how event attribute data changes in a business process. We formalized the problem in terms of the Minimum Description Length (MDL) principle, by which we choose the model with the best lossless description of the data. To efficiently search for rule models in practice, we proposed our greedy method MOODY. Through extensive experiments on both synthetic and real-world data, we showed MOODY indeed discovers succinct and interpretable if-then rules, needs little data for accurate discovery, is robust to sensible amounts of noise, and thus gives valuable insight into data modifications.

Besides applying MOODY on downstream tasks such as anomaly detection, concept drift detection and simulation, future work involves extending the rule language of MOODY to model more complex conditions for data changes, and runtime optimizations to enable search for such complex rules in feasible time.

References

[1] A. Augusto, R. Conforti, M. Dumas, and M. La Rosa. Split Miner: Discovering accurate and simple business process models from event logs. In *ICDM*, pages 1–10, 2017.

[2] R. J. C. Bose, W. M. Van Der Aalst, I. Žliobaitė, and M. Pechenizkiy. Dealing with concept drifts in process mining. *IEEE Trans. Neural Netw. Learn. Syst.*, 25(1):154–171, 2013.

[3] K. Budhathoki, M. Boley, and J. Vreeken. Discovering reliable causal rules. In *SDM*, pages 1–9, 2021.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT Press, 3rd edition, 2009.

[5] A. P. Dawid. Present position and potential developments: Some personal views - statistical theory: The prequential approach. *J. R. Statist. Soc. A*, 147(2):278–292, 1984.

[6] M. de Leoni and F. Mannhardt. Road traffic fine management process, 2015. doi: 10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[7] J. Fischer and J. Vreeken. Differentiable pattern set mining. In *KDD*, pages 383–392, 2021.

[8] M. Foster, J. Derrick, and N. Walkinshaw. Reverse-engineering EFSMs with data dependencies. In *ICTSS*, pages 37–54, 2021.

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

[10] V. Hlupic and S. Robinson. Business process modelling and analysis using discrete-event simulation. In *WSC*, pages 1363–1369, 1998.

[11] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2012.

[12] T. Krismayer. *Automatic Mining of Constraints for Event-based Systems Monitoring*. PhD thesis, Johannes Kepler University Linz, 2020.

[13] F. Mannhardt. Sepsis cases - event log, 2016. doi: 10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

[14] F. Mannhardt, M. De Leoni, H. A. Reijers, and W. M. Van Der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98:407–437, 2016.

[15] A. Marx and J. Vreeken. Telling cause from effect by local and global regression. *Knowl Inf Syst*, 60(3):1277–1305, 2019.

[16] A. Marx and J. Vreeken. Formally justifying MDL-based inference of cause and effect. In *ITCI*, 2022.

[17] A. S. Mozafari Mehr, R. M. de Carvalho, and B. van Dongen. Detecting privacy, data and control-flow deviations in business processes. In *CAiSE*, pages 82–91, 2021.

[18] T. Nolle, A. Seeliger, and M. Mühlhäuser. Binet: multivariate business process anomaly detection using deep learning. In *BPM*, pages 271–287, 2018.

[19] H. M. Proença, P. Grünwald, T. Bäck, and M. van Leeuwen. Robust subgroup discovery. *Data Min. Knowl. Disc.*, 36(5):1885–1970, 2022.

[20] H. M. Proença and M. van Leeuwen. Interpretable multiclass classification by MDL-based rule lists. *JIS*, 512:1372–1393, 2020.

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

[22] J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals Stat.*, 11(2):416–431, 1983.

[23] J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE TIT*, 30:629–636, 1984.

[24] V. Rodionov. On the number of labeled acyclic digraphs. *Discrete Mathematics*, 105(1):319–321, 1992.

[25] D. M. V. Sato, S. C. De Freitas, J. P. Barddal, and E. E. Scalabrin. A survey on concept drift in process mining. *ACM CSUR*, 54(9):1–38, 2021.

[26] S. Schönig, C. Di Ciccio, F. M. Maggi, and J. Mendling. Discovery of multi-perspective declarative process models. In *ICSOC*, pages 87–103, 2016.

[27] D. Sommers, V. Menkovski, and D. Fahland. Process discovery using graph neural networks. In *ICPM*, pages 40–47, 2021.

[28] F. Taymouri, M. La Rosa, and S. Erfani. A deep adversarial model for suffix and remaining time prediction of event sequences. In *SDM*, pages 522–530, 2021.

[29] W. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2nd edition, 2016.

[30] N. Walkinshaw and M. Hall. Inferring computational state machine models from program executions. In *ICSME*, 2016.

[31] B. Wiegand, D. Klakow, and J. Vreeken. Discovering interpretable data-to-sequence generators. In *AAAI*, pages 4237–4244, 2022.

[32] L. Yang and M. van Leeuwen. Truly unordered probabilistic rule sets for multi-class classification. In *ECML PKDD*, pages 87–103, 2022.

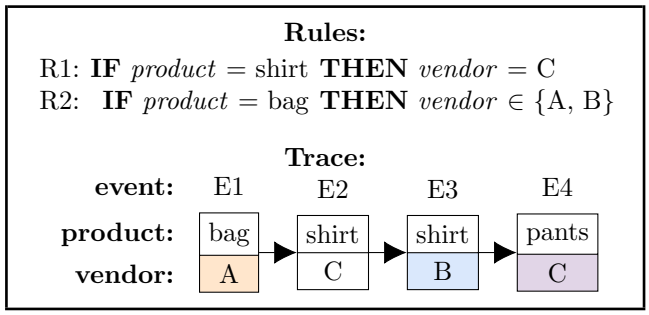

Figure 11: [**Example rules and trace**] We show two simple rules (top) and a simple trace (bottom) to disprove submodularity and monotonicity for our score.

A Appendix

In this section, we give additional details of MOODY and our empirical evaluation, which we could not include into the main paper.

A.1 Submodularity of our MDL score Submodularity [11, p. 15] requires our score to fulfill

$$L(D, M_a \cap M_b) + L(D, M_a \cup M_b) \leq L(D, M_a) + L(D, M_b)$$

for all valid models M_a, M_b. As a counterexample for this, consider the trace in Figure 11 as our event log D and the models $M_1 = \{R1\}$ and $M_2 = \{R2\}$. Here, we get $L(D, M_1 \cap M_2) + L(D, M_1 \cup M_2) \approx 59.448$ and $L(D, M_1) + L(D, M_2) \approx 59.199$ and thus

$$L(D, M_1 \cap M_2) + L(D, M_1 \cup M_2) > L(D, M_1) + L(D, M_2)\ .$$

A.2 Monotonicity of our MDL score Monotonicity [11, p. 359] requires our score to be non-increasing or non-decreasing as we add rules to our model. To disprove this for our score, assume that our event log contains the trace in Figure 11 repeated 20 times. Then, we get the scores $L(D, \emptyset) \approx 241.519$, $L(D, \{R1\}) \approx 279.595$ and $L(D, \{R2\}) \approx 239.595$. So, extending the empty model with different rules can both increase and decrease the score, and thus it is not monotone.

A.3 Supplementary material In the supplementary material[2] we make code and data publicly available. This also includes the full Master's Thesis that the main paper is based on. In this thesis, we provide further details on our encoding and the MOODY algorithm. Further, we explain the setup of the experiments on synthetic data in more detail.

[2] https://github.com/masc00008/moody