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

Marco Bjarne Schuster (✉)[1], Boris Wiegand[2], and Jilles Vreeken[3]

[1] Airbus Operations GmbH, Bremen, Germany `marco.schuster@airbus.com`
[2] Stahl-Holding-Saar, Dillingen, Germany `boris.wiegand@stahl-holding-saar.de`
[3] CISPA Helmholtz Center for Information Security, Germany `jv@cispa.de`

Abstract. Although event logs are a powerful source to gain insight into 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 describing 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 aim to find 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.

Keywords: Process mining · Rule mining · MDL.

1 Introduction

Given a process event log, process mining [39] improves our understanding of the underlying process and enables downstream tasks such as monitoring, anomaly detection, simulation, and optimization. Existing work focuses on discovering patterns of event activities, but neglects how event attribute data changes during the process. Process discovery algorithms [2, 33] only infer a graph of event activities, where nodes refer to activities and edges visualize the flow of execution. However, since event data changes, or in other words, data is moody, how these changes occur is crucial to understand the process. As a toy example, we show the ordering process of a textile company with the activities *Request*, *Place*, *Delay* and *Receive* in Figure 1. We only know the values of the attributes *price* and *delivery* of an order after the *Place* event. The *Delay* event postpones the delivery, e.g., due to a shortness of supplies.

Surprisingly, event data has only been used to predict the occurrences of activities or the outcome of processes [38,41]. 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 [25]

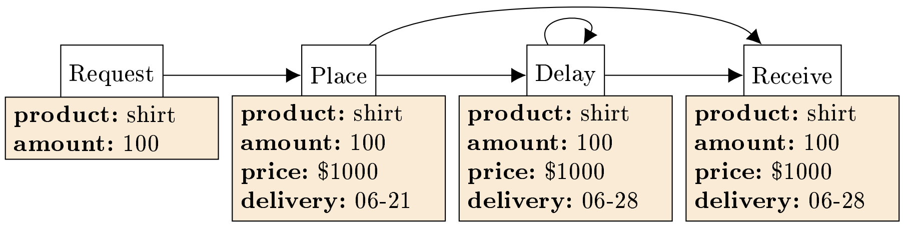


Fig. 1. [**Processes change data**] Exemplary process for ordering textiles with activities *Request*, *Place*, *Delay* and *Receive*. Arrows indicate the flow of events. Further, we show how the data of an exemplary order changes throughout the process.

and rule-based prediction methods [42] 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. In addition, to apply subgroup discovery or any prediction method, we need to define which variables are features and which are target, for which we need domain knowledge.

Given an event log, we aim to find accurate, yet succinct and interpretable if-then rules on 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. 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 rules from event logs,
(b) formalize the problem using the Minimum Description Length principle,
(c) propose the MOODY algorithm to efficiently find accurate rule models,
(d) run extensive experiments on both synthetic and real-world data,
(e) make code, data and appendix publicly available.[4]

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 [15] 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 [10, 40] infers extended finite state machines from software execution logs. Since business

[4] https://eda.rg.cispa.io/prj/moody/

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 [39] 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. [17] and Mozafari et al. [21] both define how to detect deviations between modeled and actual process behavior, where they model activities using conditions on event attributes. They, however, do not provide a method to learn these models from data. Schönig et al. [32] find correlations between activities, resources and event duration. 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 [26] and its successor TURS [42] 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++ [25] find rules for differently behaving subgroups of a given dataset. None of these methods is able to model sequential relationships present in event logs.

Finding patterns in event sequences is a classic research topic [1, 18]. Earlier proposals focused on efficient discovery of all frequent subsequences with or without gaps [23, 43], resulting in overly many and highly redundant patterns: the pattern explosion. Attention hence shifted to reducing redundancy via closures [35, 36], statistical testing [24, 34], or pattern set mining based on the MDL principle [6, 11, 37]. While all these approaches give valuable insight into event sequence data, none of them tackles the problem of explaining event data modifications in business processes.

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.

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 is an exemplary execution of the process and consists of an event sequence. We describe an event by its values for a set of numerical and categorical variables V.

To model how the events of a process modify 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 $v \neq \alpha$. We test lower and upper bounds of numerical values by $v \leq \alpha$ and $v \geq \alpha$. To model sequential dependencies, we condition on value changes between the last and the current event with $v : \alpha \rightarrow \beta$, and its negation $v : \alpha \not\rightarrow \beta$. Finally, we can join conditions using disjunctions, e.g., $(v_1 = \alpha) \vee (v_2 = \beta)$, as well as conjunctions, e.g., $(v_1 = \alpha) \wedge (v_2 = \beta)$.

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. This allows for an independent interpretation of each rule, which facilitates understanding [42]. To avoid contradictory predictions, we ban cyclic models: If we condition on variable v_1 to update variable v_2, we are not allowed to do the reverse in the same model.

3.2 Minimum Description Length

We select models by the Minimum Description Length (MDL) principle [12,27]. 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 [12]. 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 describes the data, yet is as succinct as possible, such that domain experts easily gain insight. As real-world data is usually noisy, we need a robust model selection criterion. Therefore, we formalize the problem using the MDL principle. To this end, we define the length of the data encoding $L(D \mid M)$, the length of the model encoding $L(M)$, and give a formal problem definition.

4.1 Data encoding

To encode a given event log with a model of data modification rules, we iterate over all events, using the rules in the model to encode the values of all event data variables. A well-fitting model captures the structure of the data, and thus leads to a short encoded length. For each event and variable, we check which rules in the model fire, i.e., have satisfied conditions, and use the firing rules to encode the variable. Whenever the model makes ambiguous predictions, we encode the specific value among all possibilities. If no rule fires, we encode the variable from

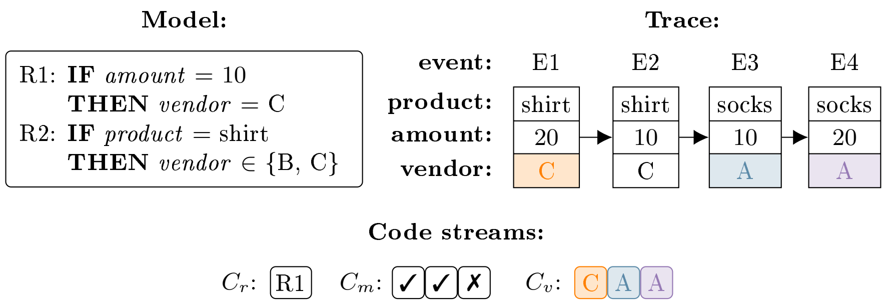

Model:
R1: IF amount = 10 THEN vendor = C
R2: IF product = shirt THEN vendor ∈ {B, C}
Trace:
event: E1 E2 E3 E4
product: shirt shirt socks socks
amount: 20 10 10 20
vendor: C C A A
Code streams:
C_r: R1
C_m: ✓ ✓ ✗
C_v: C A A

Fig. 2. **[Data encoding]** Toy example of a rule model (top left), a trace with four events where each event has the variables product, amount and vendor (top right), and a corresponding encoding of the values of the variable *vendor* (bottom).

its whole domain. 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 variable *vendor*. First, at event E1, we see 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 code 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 both rules apply. Therefore, we check C_r to find that we should use rule R1 whose prediction is correct according to the second code 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 encoded length of the data by summing the code lengths in C_r, C_m and C_v. Whenever we must disambiguate multiple firing rules in C_r, we assume all rules in the model are equally important. This means, we have

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

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 [7, 29] 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

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

where $\text{usg}_i\, C_m[i]$ 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 empirical probability of variable 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

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

where $\text{fr}(C_v[i])$ 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 encode the number of rules in the model $|M|$ and all conditions c and update rules u. As $|M|$ is unbounded, we use the universal MDL integer encoding $L_{\mathbb{N}}$ [28] that encodes any natural number $x \geq 1$ as $L_{\mathbb{N}}(x) = \log(c_0) + \log(x) + \log(\log(x)) + \dots$, where we sum only the positive terms and $c_0 = 2.865064$ ensures Kraft's inequality holds, i.e., $L_{\mathbb{N}}$ is a lossless encoding. We define 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 first encode the number of terms $|c|$. Next, we encode which of the two operators in the set $\mathcal{O} = \{\vee, \wedge\}$ we have at each of the $\max(0, |c| - 1)$ positions between terms. Then, for each term t, we encode which single variable $v \in V$ is tested by t, which comparator of the set $\mathcal{C} = \{=, \neq, \leq, \geq, \rightarrow, \not\rightarrow\}$ is used, and all constants in the term. Formally, we define

$$L(c) = L_{\mathbb{N}}(|c|+1) + \max(0, |c|-1) \cdot \log|\mathcal{O}| + \sum_{t \in c} \left(\log|V| + \log|\mathcal{C}| + \sum_{\alpha \in t} L(\alpha) \right) \ .$$

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 represent α up to a user-specified precision p by the smallest integer shift s, such that $|\alpha| \cdot 10^s \geq 10^p$, and have

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

where we first encode the signs of α and s with 1 bit each, then the value of s, and finally the value of α up to the precision of p significant digits [19].

To encode an update rule u on a variable v, we first specify the type $\mathcal{T}$ 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 $\log |\mathcal{T}| = \log 6$ 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

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

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 $L(D, M) = L(M) + L(D \mid M)$.*

Solving this problem optimally is infeasible due to the large number of acyclic models. To derive a lower bound of this number, we consider the *rule dependency graph* of a model, which is a directed acyclic graph with variables as nodes and their dependencies induced by 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 [30], 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 [5, 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.

IF *product* = pants **THEN** *vendor* = A
IF *product* = bag **THEN** *amount* = 20

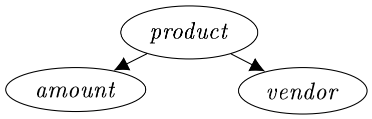


Fig. 3. **[Rule dependency graph]** Example of a model (left) and its rule dependency graph (right), in which nodes represent variables and edges show whether and in which direction a modification rule induces a dependency between variables.

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

5 The Moody algorithm

To efficiently find good data modification rules in practice, we prune the exponentially sized search space by a quickly computable estimate of our score avoiding repetitively passing the whole event log, and we present 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 at each event, which of the rules in the model fire. To avoid iterating over all events each time we evaluate adding a new rule to the model, we prune the large set of candidate rules by a quickly computable estimate $\widehat{L}(D, M)$, and only compute $L(D, M)$ for the remaining candidates. For the estimate, we optimistically assume that a new rule has no conflicts with other rules, such that we do not need to update C_r or C_m.

Using this assumption, 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 $\mathrm{supp}(c)$, 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 $\mathrm{fr}(j)$ 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 all events, 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 rules for a given event log.

5.2 Finding Good Modification Rules

We first search for conditions with a single term. Further, we let the domain experts control how much time they want to invest in model search, and introduce two hyperparameters. Instead of generating conditions with all possible combinations of variables, operators and values, we only generate the N_c most frequent values in the dataset for each variable and for each operator of the set $\{=, \neq, \rightarrow, \not\rightarrow\}$. Since condition terms with the operators $\leq$ and $\geq$ apply to a range of values, we instead generate them with the N_c values that equally cover each variable's range in terms of percentiles.

To speed up generating update rules, we use the conditions generated above to filter for the values relevant for the update rule. For a given condition, instead of generating update rules with all combinations of variables and values remaining after filtering, we only generate the N_u most frequent filtered values for α in the dataset for each variable and for each type of update rule in the set $\{(v = \alpha), (v = v + \alpha), (v = \alpha \cdot v)\}$. Similarly, to generate update rules of the type $v \in \{\alpha, \beta, \ldots\}$, we populate its set $\{\alpha, \beta, \ldots\}$ with the $1, \ldots, N_u$ most frequent filtered values and repeat this for each variable. Like before, since the update rules of type $v \in [\alpha, \beta]$ and $v = v + [\alpha, \beta]$ apply to a range of values, we cannot use frequent values here. Instead, we successively cut the interval of filtered values in half. This, way we get intervals with the central 100%, 50%, 25% etc. of values until we have generated N_u update rules for each variable.

We give the pseudocode of our greedy search Moody as Algorithm 2. We start with the empty model (ln. 1). Iteratively, we extend the model by rules for all variables (ln. 3). As computing $L(D,M)$ needs a pass over all events, we keep 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 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). Whenever two rules in the model have the same update rule, we merge them by a disjunction (ln. 11). Finally, we end when no candidate for any target variable could improve our score (ln. 12) and return the resulting model (ln. 13).

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 $O\left((N_c \cdot N_u)^2 \cdot |V| \cdot |D|\right)$.

Algorithm 2: Moody

```
Input: event log $D$ with variables $V$
Output: model of data modification rules $M$
1  $M \leftarrow \emptyset$;
2  do
3      forall $v \in V$ do
4          $Q \leftarrow$ priority queue of rules $r$ predicting $v$ ordered by $\widehat{L}(D, M \cup \{r\})$;
5          $r^* \leftarrow \emptyset$;
6          while $Q \neq \emptyset$ and $\widehat{L}(D, M \cup \{\text{top of } Q\}) < L(D, M \cup \{r^*\})$ do
7              $r' \leftarrow$ pop element from $Q$;
8              $r^* \leftarrow \arg\min_{r \in \{r^*, r'\}} L(D, M \cup \{r\})$
9          if $L(D, M \cup \{r^*\}) < L(D, M)$ then
10             $M \leftarrow M \cup \{r^*\}$;
11         replace all (IF $c_1$ THEN $u$, IF $c_2$ THEN $u$) $\in M$ by IF $c_1 \vee c_2$ THEN $u$;
12 while $M$ was extended in the last iteration;
13 return $M$;
```

6 Experiments

In this section, we evaluate Moody on both synthetic and real-world data. When defining our MDL score, we assumed discretization of numerical variables. In our prototype implementation, we use variable-width histograms. For efficiency, we compute histogram boundaries by percentiles. We use 50 bins in all experiments.

We run all 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 $M = \emptyset$. Further, for each variable, we learn a decision tree and tune its depths by 5-fold cross validation. In addition, we learn if-then-else rules using the rule-based classifier Turs [42], and subgroup discovery method SSD++ [25]. To ensure reproducibility, we provide code, data, and 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 ground-truth rules. We randomly generate 20 independent ground-truth models, where each model contains five rules, two categorical variables and two numerical variables. Since none of the baselines can model sequential dependencies $v : \alpha \to \beta$, we only create models with conditions $v \leq \alpha$, $v \geq \alpha$ and $v = \alpha$. From each model, we randomly generate five independent event logs with $|D| = 2000$ events, such that we have 100 synthetic event logs in total. 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.

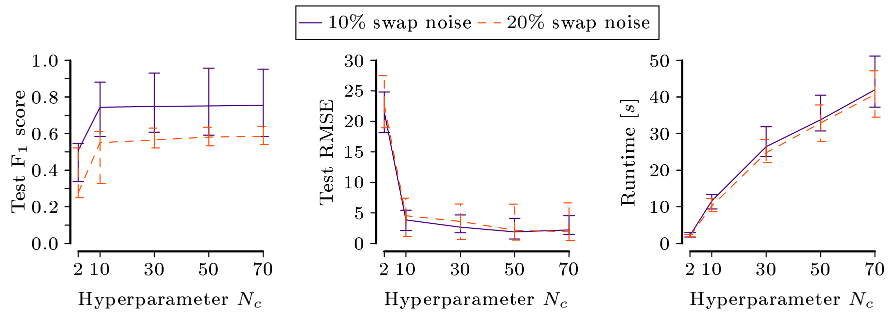


Fig. 4. **[Choosing** N_c**]** Median F_1 score on categorical variables (left, higher is better), median RMSE on numerical variables (center, lower is better) and median runtime (right, lower is better) for different values of Moody's parameter N_c for 10% and 20% swap noise in the training set. Error bars show interquartile ranges.

Turs in contrast to SSD++ only discovers rules for categorical target variables. Therefore, we separately evaluate on models predicting only categorical variables and on models predicting only numerical variables. However, in both setups, we generate conditions with both categorical and numerical variables.

Choosing Hyperparameters Such that the user can control runtime by reducing the search space, we introduced the hyperparameters N_c and N_u when proposing 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 F_1 score on predicting categorical variables, and its root mean squared error (RMSE) on predicting numerical variables. We report the influence of N_c on prediction accuracy and discovery runtime in Figure 4. We see the accuracy drops for only very small N_c, and is almost constant for larger values. This means, the conditions with the highest support in the data tend to give the best compression and accuracy. We also see that the runtime does not grow quadratically with N_c, as we would expect by the theoretical runtime analysis. This means, our score estimate successfully prunes the search space. For a safety margin on unknown data, we set N_c to 50 in the following experiments.

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 by a wide margin has the highest F_1 score 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. While 20% noise may sound low, swapping 20% 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.

Results on numerical variables We see similar results on predicting numerical variables in the center of Figure 5. As Turs cannot predict numerical

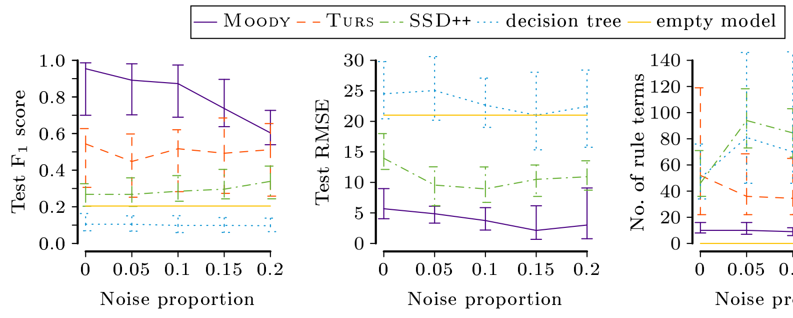


Fig. 5. **[Moody predicts well under reasonable amounts of noise]** Median F_1 scores on categorical variables (left, higher is better), median root mean squared errors on numerical variables (center, lower is better) and number of rule terms in the discovered models (right, lower is less complex) at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

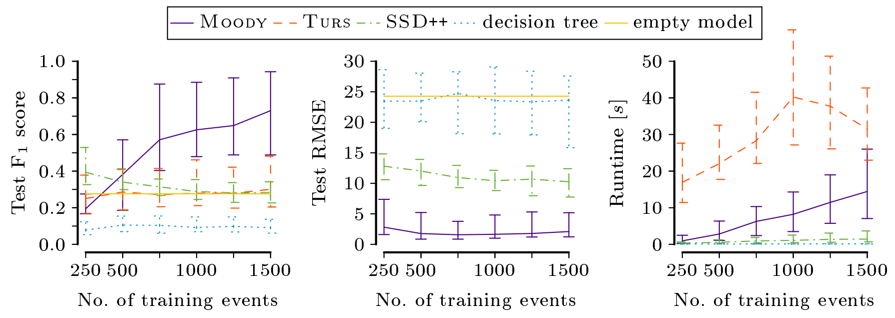


Fig. 6. **[Moody shows low sample complexity and scales well]** Median F_1 scores on categorical variables (left, higher is better), median root mean squared errors on numerical variables (center, lower is better) and median runtime (right, lower is better) dependent on the number of training events for Moody, Turs, SSD++ and decision trees. Error bars indicate interquartile ranges.

variables, we leave it out in this comparison. Moody shows by far the smallest test root mean squared error (RMSE) for training data with up to 20% noise. 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 finds the rule sets with the lowest total number of rule terms as we show on the right of Figure 5. All baselines find sets with significantly more rule terms. The size of the ground-truth models is ten rule terms. We see Moody's models have similar complexity as the ground-truth models for data with low amounts of noise. As the noise level increases, Moody does not discover more rule terms. This means, Moody is robust against finding spurious rules on noisy data.

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

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

Sample complexity To evaluate sample complexity, we compute F_1 score and RMSE 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 sets. We report results for MOODY and the baselines in Figure 6. As expected, we see MOODY predicts better the more training data is available. Already with 500 training events, MOODY shows a higher median F_1 score than the baselines. On numerical variables, it shows a lower median RMSE for all training set sizes.

Runtime We report wall-clock runtime for single-threaded execution dependent on the number of training events on the right of Figure 6. As we expect by our theoretical analysis, we see MOODY scales well and shows a runtime linear to the number of events. While SSD++ and decision trees are constantly fast, MOODY 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* [16], contains event traces from treating Sepsis patients in a Dutch hospital. The second one, *Traffic Fines* [8], [17, 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 insights we gain from rules found by MOODY on these logs. Then, we evaluate how well the rules generalize to unseen test data.

Sepsis On the Sepsis log, MOODY needs about two hours runtime and returns 49 rules in total. We observe that our greedy search strategy (namely ln. 6 and ln. 9) eliminates the vast majority of 41769 candidates that MOODY considers. Further, we find sequential dependencies in 7 rules (14%) which shows that they appear in practice. None of our baseline methods CLASSY [26], TURS [42] and SSD++ [25] are able to find these.

Next, we give examples of rules, which we can interpret without domain knowledge about the underlying process. For reference, we show the complete set of rules in the supplementary materials. Six of these rules express a correlation between the *group* and *activity* variable, e.g.,

$$\begin{aligned}&\textbf{IF}\,\mathit{group} = \mathrm{C} && \textbf{THEN}\ \mathit{activity} = \text{ER Triage}\\ &\textbf{IF}\,\mathit{group} = \mathrm{E} && \textbf{THEN}\ \mathit{activity} = \text{Release A}\\ &\textbf{IF}\,\mathit{group} \in \{\mathrm{W}, \mathrm{P}\} && \textbf{THEN}\ \mathit{activity} = \text{Admission IC}\ .\end{aligned}$$

The rules indicate groups in the hospital are specialized in certain activities. Furthermore, Moody finds the rules

$$\begin{aligned}&\textbf{IF}\,\mathit{Leukocytes} \leq 4.7\ \textbf{THEN}\ \mathit{Infusion} = \text{True}\\ &\textbf{IF}\,\mathit{Leukocytes} \leq 2.8\ \textbf{THEN}\ \mathit{Diagnose} = \text{GB}\\ &\textbf{IF}\,\mathit{Leukocytes} = 7.8\ \textbf{THEN}\ \mathit{Diagnose} = \text{AA}\ ,\end{aligned}$$

where leukocytes measurements imply the diagnosis and the treatment of sepsis. 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 On the Traffic Fines log, Moody takes 4.5 hours to find 84 rules. It eliminates the majority of the 4427 candidates by the greedy search and finds sequential dependencies in 18 rules (21%). The latter shows an even larger advantage over Classy [26], Turs [42] and SSD++ [25].

We give all discovered rules in the supplementary materials and show examples of rules, which we can interpret without domain knowledge in the following. While we expected that a higher fine *amount* correlates with a higher number of *points* to penalize the offender, the rules

$$\begin{aligned}&\textbf{IF}\,\mathit{amount} \leq 68.77 \vee \mathit{amount} \geq 131.00\ && \textbf{THEN}\ \mathit{points} = 0.0\\ &\textbf{IF}\,\mathit{amount} = 80.00 && \textbf{THEN}\ \mathit{points} \in [0.0, 2.0]\end{aligned}$$

contradict this intuition. A look into the data confirms there is no monotonic relationship between the two variables. Counter-intuitive but data-supported rules like these give valuable insight. In contrast to other methods, Moody finds rules with sequential dependencies, such as

$$\begin{aligned}&\textbf{IF}\,\mathit{activity} : \text{Insert Fine Notification} \rightarrow \text{Send Fine}\ \textbf{THEN}\ \mathit{expense} = 0\\ &\textbf{IF}\,\mathit{activity} : \text{Payment} \rightarrow \text{Payment}\ \textbf{THEN}\ \mathit{totalPayment} \mathrel{+}= 38\ .\end{aligned}$$

When sending a new fine notification, the *expense* is initialized to zero. Repetitive executions of the *Payment* activity increase the total payment amount.

Generalization Finally, we evaluate how well the rules found 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 and RMSE on both sets for each rule discovered by Moody on the training set. We show results in Figure 7. 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 Moody finds well-generalizing rules.

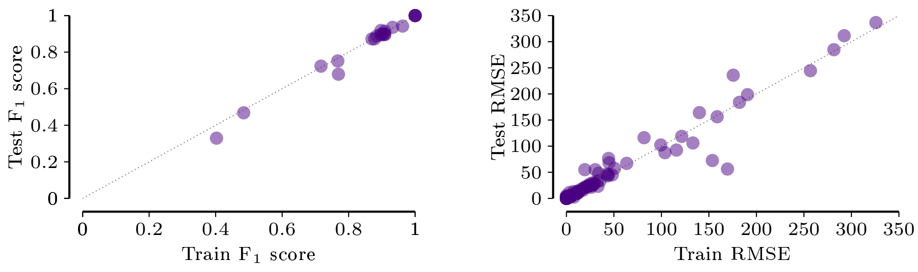


Fig. 7. **[Moody generalizes well]** Train and test F$_1$ score on predicting categorical variables (left, higher is better) and train and test root mean squared error on predicting numerical variables (right, lower is better) for each rule found by Moody on the Traffic Fines event log.

7 Discussion

In our experiments, Moody does not only discover simple and thus interpretable rules to unveil how event data changes within a process. It also finds rules accurately predicting the data values, and is more robust to sensible amounts of noise than all baselines. Yet, we see interesting directions to improve Moody.

As in many MDL-based methods, Moody's bottleneck is combinatorial search. Hence, we see an approach for mining rules based on differentiable pattern set mining [9] as promising. While both the modeling language and the MDL score of Moody allow rules with complex conditions joined by conjunctions ($\wedge$) and disjunctions ($\vee$), the algorithm does not search for conjunctions. Conjunctions imply an even larger search space, and thus we see the need to improve the runtime efficiency of Moody. Here, the implementation behind methods like Turs or SSD++ could help to efficiently search for conjunctions. Alternatively, we could first learn a graph summarizing the possible set of activity sequences of the process [41]. Learning conditions and updates for each node in this graph could result in an even easier understandable model and could help to separate traces during model search and thus might speed up the search.

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 [4]. Alternatively, we would like to examine the link between causality and two-part MDL codes in terms of algorithmic independence [20], and how to use this during search for causal data modification rules.

Last but not least, we 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 [22]. As the behavior of real-world processes usually changes over time, we see Moody could help to identify and understand concept drift [3,31]. Finally, predicting data attributes with Moody may be used in the simulation of process behavior for process optimization [13].

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 insights into data modifications.

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

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, Los Alamitos, CA, USA, 1995. IEEE Computer Society.
2. 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.
3. 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.
4. K. Budhathoki, M. Boley, and J. Vreeken. Discovering reliable causal rules. In *SDM*, pages 1–9, 2021.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT Press, 3rd edition, 2009.
6. J. Cüppers, P. Krieger, and J. Vreeken. Discovering sequential patterns with predictable inter-event delays. In *AAAI*, volume 38, pages 8346–8353, 2024.
7. 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.
8. M. de Leoni and F. Mannhardt. Road traffic fine management process, 2015. https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.
9. J. Fischer and J. Vreeken. Differentiable pattern set mining. In *KDD*, pages 383–392, 2021.
10. M. Foster, J. Derrick, and N. Walkinshaw. Reverse-engineering EFSMs with data dependencies. In *ICTSS*, pages 37–54, 2021.
11. E. Galbrun. The minimum description length principle for pattern mining: A survey. *Data Min. Knowl. Disc.*, 36(5):1679–1727, 2022.
12. P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.
13. V. Hlupic and S. Robinson. Business process modelling and analysis using discrete-event simulation. In *WSC*, pages 1363–1369, 1998.
14. B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2012.

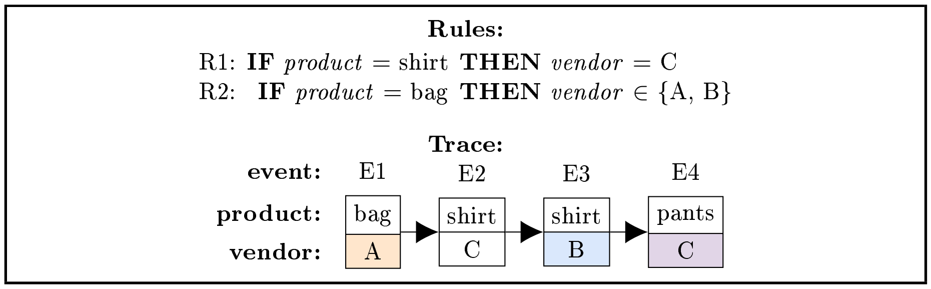
15. T. Krismayer. *Automatic Mining of Constraints for Event-based Systems Monitoring*. PhD thesis, Johannes Kepler University Linz, 2020.
16. F. Mannhardt. Sepsis cases - event log, 2016. https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.
17. F. Mannhardt, M. De Leoni, H. A. Reijers, and W. M. Van Der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98:407–437, 2016.
18. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *KDD*, pages 210–215, 1995.
19. A. Marx and J. Vreeken. Telling cause from effect by local and global regression. *Knowl Inf Syst*, 60(3):1277–1305, 2019.
20. A. Marx and J. Vreeken. Formally justifying MDL-based inference of cause and effect. In *ITCI*, 2022.
21. 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.
22. T. Nolle, A. Seeliger, and M. Mühlhäuser. Binet: multivariate business process anomaly detection using deep learning. In *BPM*, pages 271–287, 2018.
23. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE TKDE*, 16(11):1424–1440, 2004.
24. F. Petitjean, T. Li, N. Tatti, and G. Webb. Skopus: Mining top-k sequential patterns under leverage. *Data Min. Knowl. Disc.*, 30, 2016.
25. H. M. Proença, P. Grünwald, T. Bäck, and M. van Leeuwen. Robust subgroup discovery. *Data Min. Knowl. Disc.*, 36(5):1885–1970, 2022.
26. H. M. Proença and M. van Leeuwen. Interpretable multiclass classification by MDL-based rule lists. *JIS*, 512:1372–1393, 2020.
27. J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.
28. J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals Stat.*, 11(2):416–431, 1983.
29. J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE TIT*, 30:629–636, 1984.
30. V. Rodionov. On the number of labeled acyclic digraphs. *Discrete Mathematics*, 105(1):319–321, 1992.
31. 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.
32. S. Schönig, C. Di Ciccio, F. M. Maggi, and J. Mendling. Discovery of multi-perspective declarative process models. In *ICSOC*, pages 87–103, 2016.
33. D. Sommers, V. Menkovski, and D. Fahland. Process discovery using graph neural networks. In *ICPM*, pages 40–47, 2021.
34. N. Tatti. Significance of episodes based on minimal windows. In *ICDM*, pages 513–522, 2009.
35. N. Tatti and B. Cule. Mining closed episodes with simultaneous events. In *KDD*, pages 1172–1180, 2011.
36. N. Tatti and B. Cule. Mining closed strict episodes. *Data Min. Knowl. Disc.*, 2011.
37. N. Tatti and J. Vreeken. The long and the short of it: Summarizing event sequences with serial episodes. In *KDD*, pages 462–470. ACM, 2012.
38. 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.
39. W. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2nd edition, 2016.

40. N. Walkinshaw and M. Hall. Inferring computational state machine models from program executions. In *ICSME*, 2016.
41. B. Wiegand, D. Klakow, and J. Vreeken. Discovering interpretable data-to-sequence generators. In *AAAI*, pages 4237–4244, 2022.
42. L. Yang and M. van Leeuwen. Truly unordered probabilistic rule sets for multi-class classification. In *ECML PKDD*, pages 87–103, 2022.
43. M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Mach. Learn.*, 42(1-2):31–60, 2001.

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



Rules:

R1: **IF** *product* = shirt **THEN** *vendor* = C

R2: **IF** *product* = bag **THEN** *vendor* ∈ {A, B}

Fig. 8. [Example rules and trace] We show two simple rules (top) and a simple trace (bottom) to disprove submodularity and monotonicity for our score.

Submodularity [14, 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 8 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 [14, 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 8 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 Rules discovered for Sepsis

In the following we show the complete set of rules that Moody discovered on the Sepsis event log [16].

IF True	**THEN** $\mathit{Age} \mathrel{+}= 0$
IF $\mathit{activity} \neq \mathit{CRP}$	**THEN** $\mathit{CRP} \mathrel{+}= 0$
IF $\mathit{DisfuncOrg} = \text{True}$	**THEN** $\mathit{Diagnose} = \text{G}$
IF $\mathit{SIRSCriteria2OrMore} = \text{False}$	
$\vee\ \mathit{Diagnose} \in \{\text{ZD}, \text{Y}, \text{LA}, \text{GB}\}$	
$\vee\ \mathit{Diagnose} : \text{EB} \rightarrow \text{EB}$	
$\vee\ \mathit{DiagnosticOther} = \text{True}$	**THEN** $\mathit{DiagnosticArtAstrup} = \text{False}$
IF $\mathit{DiagnosticIC} = \text{False}$	
$\vee\ \mathit{Diagnose} = \text{CA}$	**THEN** $\mathit{DiagnosticBlood} = \text{False}$
IF $\mathit{DiagnosticXthorax} = \text{False}$	**THEN** $\mathit{DiagnosticECG} = \text{False}$
IF $\mathit{SIRSCriteria2OrMore} = \text{False}$	**THEN** $\mathit{DiagnosticIC} = \text{False}$
IF $\mathit{DiagnosticIC} = \text{False}$	
$\vee\ \mathit{Diagnose} = \text{S}$	**THEN** $\mathit{DiagnosticLacticAcid} = \text{False}$
IF $\mathit{DiagnosticUrinarySediment} = \text{False}$	**THEN** $\mathit{DiagnosticLiquor} = \text{False}$
IF $\mathit{DiagnosticXthorax} = \text{False}$	**THEN** $\mathit{DiagnosticSputum} = \text{False}$
IF $\mathit{DiagnosticUrinarySediment} = \text{False}$	**THEN** $\mathit{DiagnosticUrinaryCulture} = \text{False}$
IF $\mathit{DiagnosticIC} = \text{False}$	
$\vee\ \mathit{Diagnose} : \text{EB} \rightarrow \text{EB}$	**THEN** $\mathit{DiagnosticUrinarySediment} = \text{False}$
IF $\mathit{DiagnosticIC} = \text{False}$	**THEN** $\mathit{DiagnosticXthorax} = \text{False}$
IF $\mathit{Hypotensie} = \text{True}$	**THEN** $\mathit{DisfuncOrg} = \text{True}$
IF $\mathit{Oligurie} = \text{True}$	**THEN** $\mathit{Hypotensie} = \text{True}$
IF $\mathit{DiagnosticECG} = \text{False}$	**THEN** $\mathit{Hypoxie} = \text{False}$
IF $\mathit{SIRSCriteria2OrMore} = \text{False}$	**THEN** $\mathit{InfectionSuspected} = \text{False}$
IF $\mathit{SIRSCriteria2OrMore} = \text{False}$	**THEN** $\mathit{Infusion} = \text{False}$
IF $\mathit{activity} \neq \text{LacticAcid}$	**THEN** $\mathit{LacticAcid} \mathrel{+}= 0$
IF $\mathit{activity} \neq \text{Leukocytes}$	**THEN** $\mathit{Leukocytes} \mathrel{+}= 0$
IF $\mathit{SIRSCritTemperature} = \text{False}$	**THEN** $\mathit{Oligurie} = \text{False}$
IF $\mathit{SIRSCriteria2OrMore} = \text{False}$	
$\vee\ \mathit{Diagnose} = \text{LB}$	**THEN** $\mathit{SIRSCritHeartRate} = \text{False}$
IF $\mathit{DiagnosticECG} = \text{False}$	**THEN** $\mathit{SIRSCritLeucos} = \text{False}$
IF $\mathit{SIRSCriteria2OrMore} = \text{False}$	**THEN** $\mathit{SIRSCritTachypnea} = \text{False}$
IF $\mathit{InfectionSuspected} = \text{False}$	**THEN** $\mathit{SIRSCritTemperature} = \text{False}$

IF *activity* = IV Liquid $\vee$ *activity* = IV Antibiotics	**THEN** *SIRSCriteria2OrMore* = True
IF *group* = C	**THEN** *activity* = ER Triage
IF *Hypoxie* = True	**THEN** *Diagnose* = CA
IF *Diagnose* = CA	**THEN** *DiagnosticArtAstrup* = True
IF *DiagnosticLacticAcid* = True	**THEN** *DiagnosticBlood* = True
IF *DisfuncOrg* = True $\vee$ *DiagnosticSputum* = True	**THEN** *DiagnosticECG* = True
IF *Infusion* = True	**THEN** *DiagnosticIC* = True
IF *DiagnosticUrinaryCulture* = True	**THEN** *DiagnosticLacticAcid* = True
IF *DiagnosticUrinarySediment* = True	**THEN** *DiagnosticUrinaryCulture* = True
IF *Diagnose* $\in$ {ZD, F, AA}	**THEN** *DiagnosticUrinarySediment* = True
IF *Oligurie* = True	**THEN** *DiagnosticXthorax* = True
IF *SIRSCritTachypnea* = False	**THEN** *DisfuncOrg* = False
IF *DiagnosticIC* = True	**THEN** *InfectionSuspected* = True
IF *activity* = IV Liquid $\vee$ *Leukocytes* $\leq$ 4.7	**THEN** *Infusion* = True
IF *Oligurie* = True	**THEN** *SIRSCritTachypnea* = True
IF *group* = E	**THEN** *activity* = Release A
IF *Age* = 50	**THEN** *Diagnose* = EB
IF *Age* $\leq$ 60 $\vee$ *Diagnose* = ZD	**THEN** *SIRSCritHeartRate* = True
IF *group* = ?	**THEN** *activity* = Return ER
IF *Leukocytes* $\leq$ 2.8	**THEN** *Diagnose* = GB
IF *group* : A $\to$ A $\vee$ *group* : L $\to$ L	**THEN** *activity* = IV Antibiotics
IF *Leukocytes* = 7.8	**THEN** *Diagnose* = AA
IF *group* $\in$ {F, O, G, I, M, Q, R, H, D, N, T, S, V, U, K, J}	**THEN** *activity* = Admission NC
IF *group* $\in$ W, P	**THEN** *activity* = Admission IC

A.4 Rules discovered for Traffic Fines

In the following we show the complete set of rules that Moody discovered on the Traffic Fines event log [8]

IF $\mathit{article} \neq 142$	**THEN** $\mathit{amount} = 35$
IF $\mathit{resource} \in \{29, 30\}$	**THEN** $\mathit{article} = 142$
IF $\mathit{vehicleClass} \neq \text{R}$	**THEN** $\mathit{activity} = \text{Create Fine}$
IF $\mathit{activity} : \text{Add penalty} \rightarrow \text{Send Appeal to Prefecture}$	**THEN** $\mathit{dismissal} = \#$
IF $\mathit{activity} : \text{Insert Fine Notification} \not\rightarrow \text{Send Fine}$	**THEN** $\mathit{expense} = 13.50$
IF $\mathit{notificationType} = \text{C}$	**THEN** $\mathit{lastSent} = \text{C}$
IF $\mathit{activity} = \text{Appeal to Judge}$	**THEN** $\mathit{resource} = 0$
IF $\mathit{totalPaymentAmount} = 35$	**THEN** $\mathit{paymentAmount} = 35$
IF $\mathit{amount} \leq 68.77 \vee 131 \leq \mathit{amount}$ $\vee\ \mathit{amount} = 125.19 \vee \mathit{amount} = 78$	**THEN** $\mathit{points} = 0$
IF $\mathit{activity} = \text{Create Fine}$	**THEN** $\mathit{totalPaymentAmount} = 0$
IF $\mathit{activity} = \text{Add penalty}$	**THEN** $\mathit{amount} = 71.50$
IF $\mathit{vehicleClass} = \text{C}$	**THEN** $\mathit{article} = 171$
IF $\mathit{notificationType} \in \{\text{P}, \text{C}\}$	**THEN** $\mathit{activity} = \text{Insert Fine Notification}$
IF $\mathit{activity} : \text{Insert Fine Notification} \rightarrow \text{Send Fine}$	**THEN** $\mathit{expense} = 0$
IF $\mathit{totalPaymentAmount} = 36$	**THEN** $\mathit{paymentAmount} = 36$
IF $143 \leq \mathit{amount}$	**THEN** $\mathit{points} \in [0, 10]$
IF $\mathit{activity} : \mathit{Payment} \rightarrow \mathit{Payment}$	**THEN** $\mathit{totalPaymentAmount} =$ $\mathit{totalPaymentAmount} + [0, 400]$
IF $\mathit{article} = 142$	**THEN** $\mathit{amount} = 125.19$
IF $\mathit{totalPaymentAmount} = 33.60$	**THEN** $\mathit{paymentAmount} = 33.60$
IF $\mathit{activity} : \mathit{Payment} \rightarrow \mathit{Payment}$	**THEN** $\mathit{totalPaymentAmount} = 82.50$
IF $\mathit{resource} \in \{559, 560, 561, 563\}$ $\vee\ \mathit{totalPaymentAmount} : 35 \not\rightarrow 0$	**THEN** $\mathit{amount} = 36$
IF $\mathit{resource} \in \{558, 550, 541, 537, 557, 559,$ $538, 546, 561, 536, 560, 548\}$	**THEN** $\mathit{article} = 157$
IF $\mathit{totalPaymentAmount} = 38$	**THEN** $\mathit{paymentAmount} = 38$
IF $\mathit{amount} = 80$	**THEN** $\mathit{points} \in [0, 2]$
IF $\mathit{activity} : \mathit{Payment} \rightarrow \mathit{Payment}$	**THEN** $\mathit{totalPaymentAmount} \mathrel{+}= 38$
IF $\mathit{article} = 7$ $\vee\ \mathit{resource} \in \{49, 63, 852, 53\}$	**THEN** $\mathit{amount} = 38$
IF $\mathit{totalPaymentAmount} = 39$	**THEN** $\mathit{paymentAmount} = 39$

IF $\mathit{article} = 146$	**THEN** $\mathit{points} \in [0, 6]$
IF $\mathit{totalPaymentAmount} = 31.30$	**THEN** $\mathit{paymentAmount} = 31.30$
IF $\mathit{resource} \in \{538, 536, 28\}$	
$\vee\, \mathit{totalPaymentAmount} : 33.60 \to 0$	**THEN** $\mathit{amount} = 33.60$
IF $\mathit{totalPaymentAmount} = 32$	**THEN** $\mathit{paymentAmount} = 32$
IF $\mathit{article} = 172 \vee \mathit{amount} = 78$	**THEN** $\mathit{points} \in [0, 5]$
IF $\mathit{totalPaymentAmount} = 32.80$	**THEN** $\mathit{paymentAmount} = 32.80$
IF $\mathit{totalPaymentAmount} = 46$	**THEN** $\mathit{paymentAmount} = 46$
IF $\mathit{article} = 181$	**THEN** $\mathit{amount} = 23$
IF $\mathit{totalPaymentAmount} = 24$	**THEN** $\mathit{paymentAmount} = 24$
IF $\mathit{article} = 173$	**THEN** $\mathit{points} = 5$
IF $\mathit{article} = 158$	**THEN** $\mathit{amount} = 78$
IF $\mathit{totalPaymentAmount} = 42$	**THEN** $\mathit{paymentAmount} = 42$
IF $\mathit{article} = 146$	**THEN** $\mathit{points} = 6$
IF $\mathit{activity} : \text{Insert Fine Notification} \not\to \text{Add penalty}$	**THEN** $\mathit{amount} = 68.77$
IF $\mathit{totalPaymentAmount} = 51.50$	**THEN** $\mathit{paymentAmount} = 51.50$
IF $\mathit{resource} = 548$	**THEN** $\mathit{amount} = 32$
IF $\mathit{totalPaymentAmount} = 82.50$	**THEN** $\mathit{paymentAmount} = 82.50$
IF $\mathit{totalPaymentAmount} = 39.51$	**THEN** $\mathit{paymentAmount} = 39.51$
IF $\mathit{totalPaymentAmount} = 80$	**THEN** $\mathit{paymentAmount} = 80$
IF $\mathit{article} = 180$	**THEN** $\mathit{amount} = 398$
IF $\mathit{totalPaymentAmount} = 44.60$	**THEN** $\mathit{paymentAmount} = 44.60$
IF $\mathit{article} = 181$	**THEN** $\mathit{amount} \in [18.78, 25]$
IF $\mathit{totalPaymentAmount} = 87$	**THEN** $\mathit{paymentAmount} = 87$
IF $\mathit{resource} \in \{26, 31\}$	**THEN** $\mathit{amount} = 39$
IF $\mathit{totalPaymentAmount} = 23$	**THEN** $\mathit{paymentAmount} = 23$
IF $\mathit{article} = 80$	**THEN** $\mathit{amount} = 137.55$
IF $\mathit{totalPaymentAmount} = 91$	**THEN** $\mathit{paymentAmount} = 91$
IF $\mathit{totalPaymentAmount} = 37.75$	**THEN** $\mathit{paymentAmount} = 37.75$
IF $\mathit{totalPaymentAmount} = 49$	**THEN** $\mathit{paymentAmount} = 49$
IF $94.50 \leq \mathit{totalPaymentAmount}$	**THEN** $\mathit{paymentAmount} \in [3.25, 798]$
IF $\mathit{vehicleClass} = \text{C}$	
$\vee\, \mathit{dismissal} \neq \text{NIL}$	
$\vee\, \mathit{article} = 172$	
$\vee\, \mathit{resource} = 30$	**THEN** $\mathit{amount} = 31.30$
IF $\mathit{totalPaymentAmount} = 85.75$	**THEN** $\mathit{paymentAmount} = 85.75$
IF $\mathit{article} = 171$	
$\vee\, \mathit{resource} = 11$	**THEN** $\mathit{amount} = 32.80$

IF $totalPaymentAmount = 79.77$	**THEN** $paymentAmount = 79.77$
IF $totalPaymentAmount = 49.25$	**THEN** $paymentAmount = 3.25$
IF $totalPaymentAmount = 82.50$	**THEN** $paymentAmount -= 9.50$
IF $totalPaymentAmount = 91$	**THEN** $paymentAmount -= 12$
IF True	**THEN** $amount = amount + [19.68, 780.50]$
IF $totalPaymentAmount = 87$	**THEN** $paymentAmount -= 11$
IF $totalPaymentAmount \leq 23$	**THEN** $paymentAmount \in [0, 23]$
IF $94.50 \leq totalPaymentAmount$	**THEN** $paymentAmount += 0$
IF $article = 193$	**THEN** $amount \in [179, 798]$
IF $totalPaymentAmount = 79.77$	**THEN** $paymentAmount -= 9.43$
IF $article = 23$	**THEN** $amount = 389$
IF $totalPaymentAmount \leq 54$	**THEN** $paymentAmount = paymentAmount + [-42.75, 11.05]$
IF $article = 20$	**THEN** $amount = 155$
IF $94.50 \leq totalPaymentAmount$	**THEN** $paymentAmount = 94.50$
IF $totalPaymentAmount = 85.75$	**THEN** $paymentAmount -= 6.25$
IF $totalPaymentAmount = 49.25$	**THEN** $paymentAmount \in [3, 49.25]$
IF $article = 180$	**THEN** $amount \in [18.78, 419]$
IF True	**THEN** $amount += 31.29$
IF $article = 146$	**THEN** $amount = 143$
IF $article = 193$	**THEN** $amount = 779$
IF $article = 116$	**THEN** $amount = 516$
IF $article = 41$	**THEN** $amount = 65.60$
IF $article = 23$	**THEN** $amount \in [312.97, 398]$
IF $article = 173$	**THEN** $amount = 148$