Automating Process Discovery Through Meta-learning

Gabriel Marques Tavares[1(✉)], Sylvio Barbon Junior[2], and Ernesto Damiani[3]

[1] Università degli Studi di Milano (UNIMI), Milan, Italy
gabriel.tavares@unimi.it
[2] Università degli Studi di Trieste (UniTS), Trieste, Italy
sylvio.barbonjunior@units.it
[3] Khalifa University (KUST), Abu Dhabi, UAE
ernesto.damiani@kustar.ac.ae

Abstract. Analyzing event logs generated during the execution of digital processes, organizations can monitor the behavior of dysfunctional or unspecified processes. For achieving the most refined results, high-quality and up-to-date process models are required. However, the selection of the proper process discovery algorithm is often addressed by human experts that can relate quality criteria, event logs behavior, and discovery techniques. Exploiting a meta-learning approach, we created a procedure that identifies the optimal discovery technique based on a user-defined balance of quality metrics. Our experiments exploited 1091 event logs representing extensive possible business process behaviors. Given a set of available algorithms, we obtained an F-score of 0.76 for recommending the discovery algorithm that maximizes quality criteria. Moreover, our method supports a more in-depth investigation of the process discovery problem by mapping log behavior and discovery techniques.

Keywords: Process discovery · Meta-learning · Model quality · Recommendation · Process mining

1 Introduction

Extracting information from event data can enhance management capabilities, revealing process deviations and improvement opportunities using techniques referred to as Process Mining (PM) [1]. One of the most active research topics in PM is Process Discovery (PD) [26]. The result of PD is a process model, which is representative of the underlying processes executions, based on the event logs recorded in organizations' information systems. PD reduces the deviations between the documented target process and the actual executed process. These deviations lead to incorrect process analyses, which, in turn, lead to little or no effective optimization measures. Nonetheless, eliciting an appropriate and up-to-date process model may require significant effort [2]. One of the key

design choices is selecting the proper discovery algorithm among several different options [14–16,28,32]. Depending on the dimension to be optimized, the algorithms that offer the best performance, in terms of model quality, are different [3]. Besides, the event log characteristics also impact the model quality [3,5]. Experiments with several different discovery algorithms showed a substantial performance complementarity since different algorithms perform best in different scenarios [5,12]. Among the multiple quality dimensions identified, the literature has largely discussed *recall* (a.k.a. *fitness*), *precision*, *generalization* and *simplicity* [4]. PD algorithms are often suggested to balance these dimensions [3]. Although the analytical goal to be addressed can determine the most prominent one. For example, audit questions are best answered using a model with high recall, optimization is best performed on a model with high precision, implementation is simplified by models with high generalization, and human interpretation is eased by simple models [10]. Less attention has been devoted in the literature to studying the relationships between event log profiles, i.e., the features characterizing an event log, PD algorithms, and quality criteria.

To overcome these issues, we studied a method to *automate the selection of the optimal process discovery algorithm* given an event log. In particular, we are interested in two research questions. **RQ1**: *To which extent can process discovery algorithm selection be automated?* **RQ2**: *Which event log features are significant in selecting the best discovery algorithm?* To answer these questions, we investigate the process discovery task from an algorithm selection perspective using a Meta-learning (MtL) approach [13]. Leveraging the MtL potential, we developed a data-driven solution to recommend a suitable process discovery algorithm given an event log. Our MtL approach provides a novel tool for identifying a discovery technique that balances model quality metrics based on user preference and given a specific event log profile. Moreover, it provides a method to exhaustively compare discovery algorithms in terms of the quality metrics they optimize and to study the relationship between quality results and event log features. Our method does not prevent the use of domain-specific knowledge an expert can apply in handling the PD procedure. For example, filtering the event log or applying conformance checking results in light of organizational constraints. However, we believe connecting the event log profile to the suitable PD algorithms, is an essential pre-flight instrument our approach can provide to guide an expert. The number of source event logs and characterizing features exploited in the training of the MtL model offers a solution unparalleled in the literature and shines a light on the algorithm selection for PD.

The paper is organized as follows. Section 2 introduces concepts that support our work. Section 3 delves into the details of the proposed MtL framework, introduces the theoretical foundations required, and exposes the feature extraction step, and the materials used for experimentation. Section 4 reports the results of the experiments and presents a discussion of the relationship between the features of the event log, the discovery techniques, and the quality of the model. Section 5 discusses the related work, while Sect. 6 summarizes and concludes the paper.

2 Basic Notions

This section defines crucial concepts from PM and MtL that substantiate the development of our approach.

Definition 1 (Event, Attribute, Case, Event log). *Let Σ be the event universe, i.e., the set of all possible event identifiers. Σ^* denotes the set of all sequences over Σ. Events may have various attributes, such as timestamp, activity, resource, cost, and others. Let $\mathcal{AN}$ be the set of attribute names. For any event $e \in \Sigma$ and an attribute $\text{A} \in \mathcal{AN}$, then $\#_{\text{A}}(e)$ is the value of attribute A for event e. Let C be the case universe, that is, the set of all possible identifiers of a business case execution. C is the domain of an attribute $\text{CASE} \in \mathcal{AN}$. An event log L can be viewed as a set of cases $L \subseteq \Sigma^*$, where each event appears only once in the log, i.e., for any two different cases, the intersection of their events is empty.*

A model can be discovered given an event log as input. Therefore, considering the relationship between events, a process discovery technique produces a model representing the event log behavior.

Definition 2 (Process discovery [11]). *An event log L can be viewed as the multiset of traces induced by the cases in L. Formally, $\overline{L} := \{t | \exists c_i \in L, c_i(i \rightarrow n) = t(i \rightarrow n)\}$. The behavior of L can be viewed as the set of the distinct elements of $\overline{L}$, formally $\mathcal{B}(L) = support(\overline{L})$. Given a process model M, we refer to its behavior $\mathcal{B}_{\mathcal{M}}$ as the multiset of traces that can be generated by its execution. A process discovery algorithm constructs a process model from an event log and can thus be seen as a function $\delta : L \rightarrow M | \mathcal{B}(L) \cong \mathcal{B}_{\mathcal{M}}$.*

Therefore, the quality of the models produced can be associated with characteristics of the event log, which can potentially be mined.

Definition 3 (Event log features[22]). *Let $\mathcal{SF}$ be a set of statistical functions (e.g., mean length) and $\mathcal{RL}$ be the set of process representational levels of L (e.g., event, case, and event log). $\mathcal{PF} = \mathcal{SF} \times \mathcal{RL}$ is the set of process features, i.e., the cartesian product of functions and representational levels.*

Considering that different event logs demonstrate different behaviors, the features extracted from the logs should depict their distinctive nature. Following, discovery algorithms may produce models with varying quality depending on data characteristics. In this way, some algorithms might be favored depending on the underlying process behavior.

Definition 4 (Algorithm Selection Problem [25]). *Let $x \in \mathcal{P}$ be a problem in a problem space, let $f(x) \in \mathcal{F}$ be a function that extracts features from the problem x, let $S(f(x))$ be a function that selects the mapping between the problem space to the algorithm space $A \in \mathcal{A}$, and let $p(A, x)$ be a function that maps the performance of an algorithm to the performance measure space, i.e., $p \in \mathcal{R}^n$ where $\mathcal{R}^n$ is a n-dimensional real vector space. Then, a norm mapping is applied*

on p to reduce $\mathcal{R}^n$ to a one-dimensional value that indicates the final performance of the algorithm. The goal is to determine $S(f(x))$ (the mapping of problems to algorithms) that maximizes the performance of the algorithm.

Figure 1 shows functions and their respective domains within the Algorithm Selection Problem (ASP). Given the ASP, MtL is a strategy to solve such a problem.

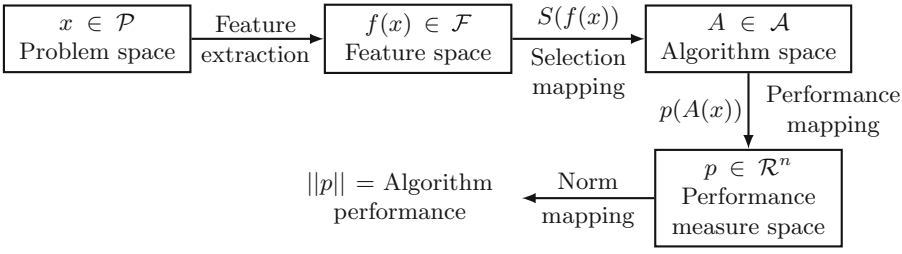


Fig. 1. Model for the Algorithm Selection Problem with features (extracted from [25]).

Definition 5 (Meta-learning [29]). *In traditional learning, the hypothesis space $\mathcal{H}_\mathcal{A}$ of a learning algorithm A is fixed. Applying A to a data set described by $\mathcal{F}$ produces a hypothesis that depends on the fixed bias embedded by the learner. Let $\mathcal{S}$ be the space of all possible learning tasks, algorithm A can learn efficiently over a limited region R_A in $\mathcal{S}$ that favors the bias embedded in A. The meta-learning strategy is to learn what causes A to dominate over R_A. Therefore, meta-learning aims at mapping the relationship between the problem features and the algorithm performance.*

Moreover, the MtL problem is further divided into two parts [29]: (i) discover the properties of the task in R_A that make A suitable for such region, and (ii) discover the properties of A that contribute to dominate R_A. Hence, a solution to the MtL problem can suggest how to match algorithms and task properties, producing a guided approach to the dynamic selection of algorithms. Therefore, by applying meta-learning to the process discovery problem, we shine light on ASP for process discovery in PM.

3 Methodology

In this section, we present the procedure executed to study the applicability of MtL to the selection of process discovery algorithms. The material used in the experiments, along with the event logs and implementation, is publicly available.[1] We note that detailed instructions are given to ensure scientific reproducibility.

[1] https://github.com/gbrltv/process_discovery_meta_learning.

3.1 The Proposed MtL Approach

We created an MtL procedure grounded on rich PM-specific event log features (that is, mapping $f(x) \in \mathcal{F}$) for suggesting the best discovery algorithm (that is, the algorithm that maximizes $p(A(x))$). The workflow implementing our MtL approach is made up of five steps. Figure 2 presents them as follows. *Meta-Feature Extraction* is a step devoted to gathering the event log features $\mathcal{PF}$, a.k.a. meta-features according to the MtL terminology. The *Meta-Target Definition* step identifies each discovery algorithm A and ranks their dominance over R_A using quality metrics. Note that given an event log, the chosen meta-target is the technique that maximizes $p(A(x))$.

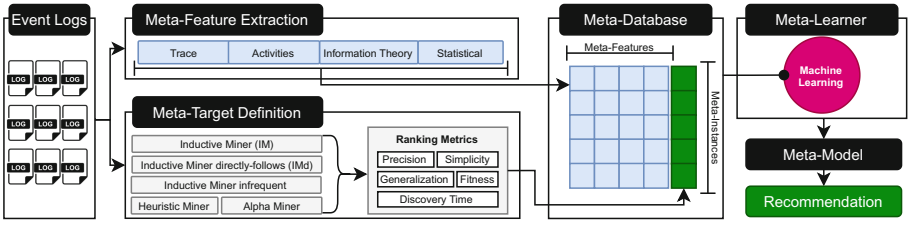


Fig. 2. Overview of the proposed MtL approach.

The *Meta-Database* step combines meta-features and meta-targets, forming the meta-instances required to train the meta-model. In the next step, a *Meta-Learner* uses machine learning to induce the meta-model based on the meta-instances. The *Meta-Model* is the outcome model able to recommend process discovery algorithms. When recommending a process discovery method for a new event log, meta-features of this new resource are extracted and forwarded to the created meta-model. The recommended discovery algorithm can be executed on the event log, generating a process model, as in Fig. 3.

3.2 Event Logs as the Problem Space

Event logs contain information about the start and completion of activities, their ordering, the resources that executed them, and the process execution to which they belong. Event logs play a significant role in our approach because they are the search space representation for creating our meta-database. Thus, we aim to build a highly heterogeneous set of logs capable of representing a wide range of possible business process behaviors. Hence, the relationship between business process characteristics and quality metrics can be better represented.

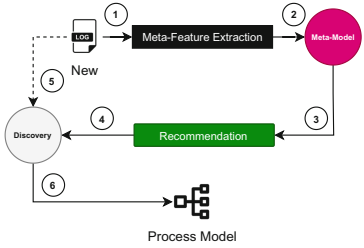


Fig. 3. Application of our MtL approach: (1) raw event log has its features extracted, (2) features are processed using the meta-model, (3) the recommendation is outputted, (4) using the suggested algorithm, the event log is processed (5) to discover a process model (6).

The data selected contains both real and synthetic event logs. Regarding synthetic event logs, the first set comes from the Process Discovery Contest (PDC) 2020.[2] The PDC 2020 dataset explores a wide range of characteristics distributed in 192 logs. The main behaviors are dependent tasks, loops, invisible and duplicate tasks, and noise. Moreover, we extracted 750 event streams from [11]. These logs were built in the context of online PM with the goal of depicting drifting scenarios, i.e., where a change in the behavior occurs while the process is acting. The last group of synthetic logs was presented in the context of evaluating encoding capabilities in event logs [6]. The set of real-life event logs contains six logs from the Business Process Intelligence Challenges (BPIC), the environmental permit, helpdesk, and sepsis logs. The final dataset contains 1091 event logs. Their main characteristics are exposed in Table 1. As demonstrated in the table, the event logs contain a wide range of behaviors, with a diverse number of cases, events, and activities that cover a variety of patterns.

Table 1. Statistics describing our data set of event logs.

Name	#Logs	#Cases	#Events	#Activities	Trace length	#Variants
PDC 2020	192	1k	6.7k–66k	14–36	3–300	503–1k
Streams	750	100–1k	900–13.5k	15–16	2–83	22–204
Encoding	140	1k	10k–44k	22–406	1–50	383–1k
BPIC12	4	5k–13k	31k–262k	7–24	2–175	17–4.3k
BPIC13	2	1k–7k	6k-65k	4	1–123	183–1.5k
Env. permit	1	1.4k	8.5k	27	1–25	116
Helpdesk	1	4.5k	21k	14	2–15	226
Sepsis	1	1k	15k	16	3–185	841

[2] https://www.tf-pm.org/competitions-awards/discovery-contest.

3.3 Meta-feature Extraction

According to the Definition 4, this step maps the problem space to the feature space by extracting features from each instance of the problem, i.e., applying the function $f(x) \in \mathcal{F}$. A challenge in our approach is to correctly capture the process behavior using a representative set of descriptors. For that, we propose a large set of features that capture complementary aspects of the event log behavior. The proposed features have an extensive range of complexity, meaning that some are very simple (e.g., number of traces) while others are more complex (e.g., entropy measures). The idea of a broad set of features is to provide the meta-learner with the most information possible. The task of selecting proper meta-features to infer a model is then the following step performed by the meta-learner.

For trace level descriptors, trace lengths and variants were used as indicators of process complexity. Regarding the trace lengths, we extracted 29 descriptors: minimum, maximum, mean, median, mode, standard deviation, variance, the 25th and 75th percentile of data, interquartile range, geometric mean and standard variation, harmonic mean, coefficient of variation, entropy, and a histogram of 10 bins along with its skewness and kurtosis coefficients. Based on trace variants, we extracted additional 11 features: mean number of traces per variant, standard variation, skewness coefficient, kurtosis coefficient, the ratio of the most common variant to the number of traces, and ratios of the top 1%, 5%, 10%, 20%, 50% and 75% variants to the total number of traces. To capture the features at the activity level, we selected three groups: all activities, start activities, and end activities. For each group, we extracted a set of 12 descriptors: number of activities, minimum, maximum, mean, median, standard deviation, variance, the 25th and 75th percentile of data, interquartile range, skewness, and kurtosis coefficients.

For log level descriptors, we obtained the number of traces, unique traces, and their ratio, along with the number of events. Recently, entropy measures were proposed in the context of business processes to capture log variability and to identify whether the log is better suited to declarative or imperative mining [5]. Consequently, these metrics measure log structuredness, a very relevant descriptor of log complexity. This way, we extracted 14 entropy measures: trace, prefix, k-block difference and ratio ($k \in \{1, 3, 5\}$), global block, k-nearest neighbor ($k \in \{3, 5, 7\}$), Lempel-Ziv, and Kozachenko-Leonenko. In total, the 93 extracted features cover several complementary perspectives, such as central tendency, statistical dispersion, probability distribution shape, log structuredness, and variability, among others.

3.4 Meta-targets to be Recommended

To define the possible meta-targets, we first need to select a set of algorithms to represent the algorithm space. Considering their wide historical use in PM and the availability of reliable source code, we selected five algorithms as meta-target candidates: α-Miner (AM), Heuristic Miner (HM), Inductive Miner (IM),

Inductive Miner–infrequent (IMf), and Inductive Miner–directly-follows (IMd). All selected algorithms are suitable meta-targets in our experiment as they share the same input and output objects. They require an event log as input and express the discovered process model using the Petri net notation [20].

AM is one of the first approaches in PM that deals with concurrency. It is considered a baseline process discovery algorithm [28] as many subsequent ideas have been developed from it. Albeit AM's historical relevance, there are many known limitations such as the lack of support for loops, weakness against noise, and no consideration of frequencies [1].

To include frequencies into account, HM was introduced, applying the idea that infrequent transitions should not be represented in the model [32]. HM uses causal nets, a modeling notation that can incorporate activities and causal dependencies. A dependence measure is calculated using frequencies, and it is further used to create a dependency graph. Arcs not conforming to a threshold frequency are removed from the dependency graph. In the final step, splits and joins are introduced to represent concurrency.

The IM family of algorithms approaches the discovery problem from a divide-and-conquer perspective, recursively splitting the event log into sub-logs [14]. A crucial point of IM is that it produces a sound process model capable of replaying the entire event log. That is, perfect recall of the traces in the event log is guaranteed by the discovered model. However, due to the lack of support for duplicate or silent activities when building the process tree, IM may produce low-precision models [1]. Finally, IM is incapable of handling fixed-length repetitions and cannot handle infrequent behavior.

The basic IM algorithm is highly extendable. Thus, many variants have been proposed [15,16]. IMf incorporates the eventually-follows relation to better deal with incompleteness in event logs [15]. Moreover, IMf introduces activity and arc filters to remove infrequent behavior and produces a more precise model. As a consequence, perfect recall is not a guarantee. A drawback of both IM and IMf is the scalability due to the recursive split, which requires multi-pass analysis. IMd was proposed to overcome this problem [16]. For that, IMd performs the recursive step only on the directly-follows graph, without partitioning sub-logs. However, the non-partitioning adaptation hurts the rediscoverability property. Hence, perfect recall is not preserved. Similar to IM, IMd also cannot handle duplicate and silent activities.

3.5 Ranking Quality Criteria

The meta-database regards a suitable matching of meta-instances and meta-targets. The definition of the best choice of meta-target for a given event log is based on a ranking strategy considering several complementary perspectives, i.e., there is no unique dimension that captures the overall model quality. In this work, together with *discovery time*, we adopt the four traditional model quality metrics used in PM: *fitness*, *precision*, *generalization* and *simplicity*.

The *fitness* metric aims to measure how much behavior in the log is allowed by the model, that is, to which extent traces in the log compare to valid execution paths derived from the model [9]. It is measured by applying replay techniques that aim to compare log and model. Therefore, perfect fitness is achieved when the model can replay all the traces in the log. In this work, we adopt the fitness proposed in [7] due to its improvements in scalability and avoidance of known problems such as token flooding. Another quality dimension, *precision*, measures the extent of the behavior allowed by the model that is not observed in the log [9]. A poor precision indicates an underfitted model, i.e., it allows too many patterns not present in the event log. As the precision measures negative examples, i.e., the behavior allowed by the model but not seen in the log, and a model can potentially generate infinite behavior, the calculation is complex and often depends on approximation. In our experiment, we adopt the precision proposed in [19] because it is more efficient and granular than previous measures.

Generalization goes in the opposite direction by measuring model overfitting. Event logs present a sample of possible behavior allowed by the system, implying that there may be valid execution traces not present in the log because they were not executed yet. Process models should then describe the log behavior but also generalize it to some extent [1,9]. We adopt the computation proposed in [9] as it covers the generalization based on the usefulness of the model. *Simplicity* is the last quality dimension used in this work. The idea behind simplicity is to indicate how complex a model is [1,9]. That is, the simpler the model structure that reflects the log behavior, the better its intelligibility. We adopt the simplicity measure proposed in [30], which is based on the weighted average degree of a place/transition in the Petri net, ultimately defined by the sum of input and output arcs.

Recognizing the importance of balancing perspectives to achieve an adequate model, we propose a ranking mechanism that aggregates all dimensions. Table 2 provides an example of the ranking to identify the best discovery technique for a single event log. The log L is submitted to three discovery algorithms (A_1, A_2, A_3), then three quality metrics (Q_1, Q_2, Q_3) are extracted using the discovered model. Following, a positional rank is built for each metric considering the performance of the algorithms (R_{Q_1}, R_{Q_2}, R_{Q_3}), that is, algorithms are assigned to a position based on the comparison within them. A final rank (R) is produced by averaging the metrics ranks, i.e., the mean of $\{R_{Q_1}, R_{Q_2}, R_{Q_3}\}$. The lowest R is selected as the best discovery algorithm because it minimizes the average rank position, thus maximizing the quality criteria. In our example, $R(A_1)$ is 1.75, $R(A_2)$ is 1.5, and $R(A_3)$ is 2.75. Thus, we can conclude that A_1 is the discovery algorithm that produces the best model for log L. Therefore, when building the meta-database, A_1 is the meta-target associated with log L. Depending on the analytical goal, alternative quality metrics could be included or alternative weights could be assigned to the obtained ranks. To exemplify this possibility, in our work we consider discovery time as a quality dimension.

Table 2. Example of ranking algorithms. The final rank (R) is generated from the average rank of each quality dimension. In this example, A_1 is the recommended discovery technique for log L as it maximizes the quality metrics, i.e., produces the lowest R value.

Log	Algorithm	Q_1	Q_2	Q_3	R_{Q_1}	R_{Q_2}	R_{Q_3}	R
L	A_1	1	0.27	0.93	1	2	1	1.33
L	A_2	0.98	0.38	0.91	3	1	2	2
L	A_3	0.99	0.2	0.9	2	3	3	2.67

3.6 Meta-model

In the final step of our experiment, a machine learning algorithm, the meta-learner, is employed to induce a meta-model using the meta-database. Once the meta-model has been created, any event log can be linked to the recommended PD algorithm by extracting its meta-features and consulting the meta-model. For this experiment, we used a Random Forest [8] classifier due to its well-known stability. A holdout strategy was applied to divide the meta-database into train and test sets, using a 75%/25% split. In generating the meta-model, we performed a 30-fold cross-validation to reduce the influence of outliers. We note that the meta-database, as tabular data, represents a classification problem modeled using traditional machine learning techniques. The use of deep learning is not feasible as (i) it requires a huge amount of instances (hundreds of thousands) and (ii) the data are not sequential.

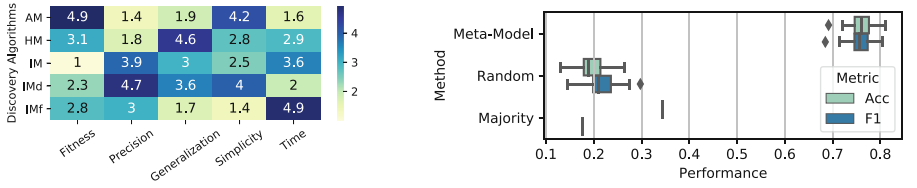
4 Results and Discussion

Figure 4a reports the discovery algorithm's average position across all event logs considering the five quality criteria. These results already confirm theoretical findings. For instance, IM's *fitness* takes the three first positions. Both IMd and IMf do not guarantee perfect fitness; still, they perform better than AM and HM. Considering that IMd does not partition sub-logs, such an important characteristic of its predecessor, the fitness result is good. Although IMf incorporates the eventually-follows relation, it performs worse than IMd from a fitness perspective. HM and AM follow with 3.1 and 4.9 average positions. AM is by far the worst-performing algorithm for fitness purposes, a problem recognized since its inception. The inability to deal with incompleteness and weakness in handling noise has a high toll on fitness performance. Another explanation for the IM family overcoming HM and AM is that IM algorithms produce a sound model, while HM and AM do not have this guarantee.

Positions change considerably when evaluating *precision*. AM becomes the algorithm with the best performance (1.4), followed by HM (1.8). IMf, IM, and IMd come next, averaging 3, 3.9, and 4.7, respectively. The IM family performs

poorly in precision due to its extensive use of hidden transitions, which contributes to creating underfitting models. On the contrary, AM and HM hold superior control of the escaping edges, strengthening the precision values.

From the *generalization* perspective, IMf is the best algorithm (1.7), closely followed by AM (1.9). These algorithms excel in generating balanced models where different regions are similarly visited during trace replaying. Particularly, IMf's ability to remove infrequent behavior reflects in good generalization values. Indeed, including low-frequency traces in the model hurts the generalization capabilities. IM, IMd, and HM averaging 3, 3.6, and 4.6 are the worst-performing discovery algorithms in this dimension. *Simplicity* is an index that uniquely depends on the model and it rewards conciseness. Again, IMf is the best performing method (1.4) thanks to its cleaning procedures that remove infrequent behavior. IM and HM follow averaging 2.5 and 2.8, and IMd and AM produce the most complex models, averaging 4 and 4.2, respectively.



(a) Metrics ranking. (b) Recommendation performance.

Fig. 4. a) Discovery algorithms ranked across all event logs considering each dimension. b) Our approach obtained an average accuracy of 0.76 and an F-score of 0.76. Given event log features, the method indicate the discovery algorithm that maximizes model quality. This experiment compares five discovery algorithms using five quality dimensions (fitness, precision, generalization, simplicity and time).

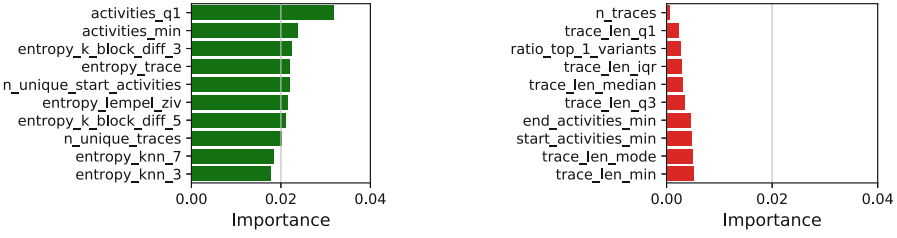
Time analysis tends to benefit simpler algorithms as they require fewer steps to generate a model. This explains why AM has the best time performance, averaging 1.6. IMd comes after (2) since its primary design purpose is focused on time improvement. HM, a more robust algorithm, comes third with 2.9 as the average. Lastly, IM and IMf average at 3.6 and 4.9. These two algorithms are the slowest as they demand a multi-pass recursive analysis of the log. Especially, IMf spends more time due to additional steps to remove infrequent behavior.

Following, we evaluate the meta-model's efficiency to recommend the best discovery method for a given event log. Not having other literature references, we used majority voting and random selection as baseline approaches, employed for comparison reasons. Majority voting works by indicating the class that appears most frequently in the meta-database, i.e., the algorithm that appears most frequently in the first position of the rank function (R) considering the complete meta-database. Random selection is a method that randomly chooses one of the five process discovery techniques. That is, given a meta-instance, the random selection approach arbitrarily associates it to one of the five meta-targets.

Both baselines are useful comparisons. From the machine learning perspective, majority voting is valid as it sets the minimum performance threshold. Complimentarily, the random selection approach simulates a practitioner that makes a non-guided choice of a PD algorithm.

Performance metrics are computed by comparing the ground-truth label, i.e., the best discovery algorithm and the recommended algorithm. Figure 4b presents the results for all methods, which were measured using the accuracy and F-score metrics. Our approach demonstrates a viable solution for recommending discovery algorithms with an average accuracy of 0.76 and an F-score of 0.76. The majority method has the next best accuracy (0.34), followed by the random method (0.2). Regarding F-score, random selection reaches 0.22 while majority voting stays at 0.18. The advantage of applying MtL in comparison to unintelligent approaches is clear. More importantly, the experiment confirms the relationship between event log characteristics and model quality depending on the discovery technique. Instead of simply applying the algorithm that ranks better according to the majority criterion, we can provide a guided decision based on the process behavior.

After meta-model learning, the Random Forest provides an *importance* measure to quantify the contribution of each meta-feature provided to the classification output. Figures 5a and 5b show that among the most influential features (Fig. 5a), there is a high predominance of the entropy family, namely, k-block difference, Lempel-Ziv, trace, and k-nearest neighbor. The entropy features were designed to capture log complexity and, more specifically, structuredness. Activities appear as the second most important group, with the 25th percentile ranking as the most informative feature. The activity and entropy groups are correlated as both rely on activity information. Trace variants are the last group among the top 10 most influential features with the number of unique traces. These results highlight that high-level descriptors for event logs can aid in determining the best discovery algorithm. Regarding the least influential features (Fig. 5), trace length-based features have the most appearances. These results indicate that both the number of traces and trace lengths are the worst features to describe business process behavior for the discovery problem.



(a) Top 10 features

(b) Bottom 10 features

Fig. 5. Features relevance to induce a meta-model recommending a discovery algorithm.

4.1 Relating Problem, Feature and Algorithm Spaces

In the next experiment, we employed a leave-one-out cross-validation (LOOCV) strategy for testing the meta-model. The LOOCV approach consists of inferring a model using all instances except one, then testing the prediction in the instance excluded from training. The process is repeated for all instances. This way, a model is learned and tested $\#MI$ times, with $\#MI$ being the number of meta-instances. This analysis produced 0.78 in both accuracy and F-score. These results go in line with the previous experiment (see Fig. 4b), again confirming the method's robustness. Here we also evaluated more closely the performances related to specific meta-targets. For example, IMd was the most challenging class to identify, reaching an F-score of 0.09. This extreme performance is due to the low number of appearances as a meta-target (only 13 meta-instances were matched to IMd). Obviously, the meta-model was unable to learn the event log profiles associated with IMd, given the small set of examples to generalize. Performance changes considerably for HM and IM with 0.71 and 0.73 as F-score, respectively. HM is not an easy meta-target to recognize since it usually produces models that balance well the analyzed quality criteria. However, very rarely it maximizes the performance in one of the criteria. IM can potentially be misidentified due to its proximity to other methods from the same family. The best individual performances were reached by IMf and AM with F-scores of 0.79 and 0.86. The meta-model can more easily associate meta-instances with these methods, meaning that event log behaviors that match such algorithms were better captured by the framework.

Furthermore, we applied a dimensionality reduction technique to better visualize the meta-instances in the feature space. For that, we employed the Principal Component Analysis (PCA) algorithm [27]. Figure 6 shows the resulting feature space reduced to two dimensions. First, we note that one Principal Component (PC) explains 55.11% of data variance while the second PC explains 30.53% variance. These results indicate that most data correlations were preserved after the dimensions were reduced. Regarding class separation, we can observe that AM, IM, and IMf are the most identifiable meta-targets. This explains why in the LOOCV experiment, these three meta-targets obtained the best F-scores. AM is spread across PC1 with an evident distance from other classes. When combining PC1 and PC2, we observe that IM and IMf are also recognizable. In a particular region, several instances of different classes overlap. This behavior is due to (i) information that supports the mapping of this region was lost during the dimensionality reduction and (ii) lack of meta-features that capture additional behavior in the problem space. Figure 6 also depicts the correct and incorrect predictions in the LOOCV experiment. As the feature space shows, recommendation errors occur across the whole space, and no particular pattern was identified. Overall, the PCA analysis shows that meta-features were able to differentiate the processes' behaviors. Considering the limited dimension of the problem space, increasing the number of meta-instances and meta-features will probably lead to better class separation, hence, better recommendation performances.

Finally, we also applied a complexity analysis to complement the evaluation of class separation based on the work of Lorena et al. [17]. The Directional-vector Maximum Fisher's Discriminant Ratio (F1v) metric searches for a vector that can separate classes after instances are projected in the hyperplane. F1v is bounded by $(0, 1]$ interval with lower values indicating simpler classification problems. When applied to our meta-database, we obtain an F1v of 0.12, indicating that the classification problem (modeled using the meta-learner) is simple. The classification problem can only be simple because the $f(x) \in \mathcal{F}$ function preserves most of the behavior in the problem space when mapping instances into the feature space. In other words, our set of meta-features has a comprehensible quality. We also assessed the Ratio of Intra/Extra Class Nearest Neighbor Distance (N2) metric, which aims at capturing the shape of the decision boundary and class overlap. N2 relates the intra- and extra-class distances by comparing each instance to its closest sample from the same class and closest sample from another class. Ideally, instances from the same class should be closer between them than from instances from other classes. The N2 for our meta-database is 0.35 (optimal value is 0), indicating a simpler problem where the overall distance for different classes is higher than the overall distance to the same class. Although pointing to a simpler problem, N2 indicates that there is indeed some overlap between classes, which is corroborated by the PCA analysis (Fig. 6).

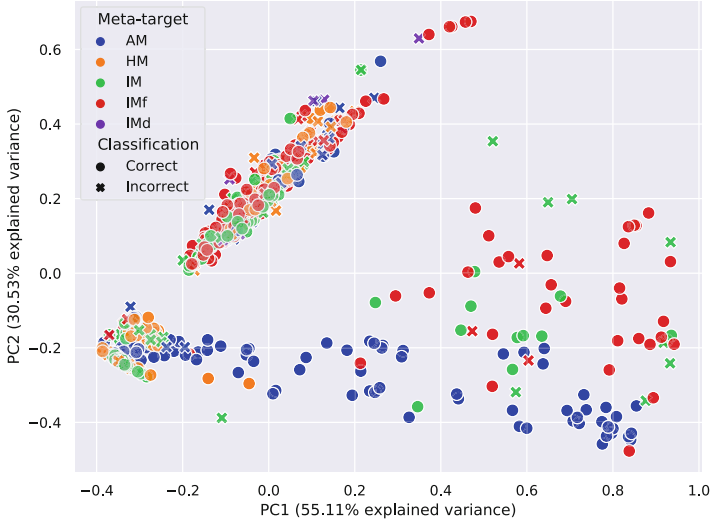


Fig. 6. Reduced feature space after applying PCA. Meta-targets have different coloring, while shapes indicate if the model correctly predicted the meta-target.

4.2 Threats to Validity

A crucial aspect of an experimental design is internal validity. Mendling et al. [18] stated that a research design is internally valid when its manipulation is

causally responsible for an observed effect. Therefore, randomization is often adopted to avoid the injection of confounding factors through data selection. In our experiment, one could argue that the training dataset, although wide, may contain unidentified bias due to the unbalanced representation of some log profiles. To confute this conjecture we compared the results obtained in our first experiment, illustrated in Fig. 4b, to the results that can be obtained by resampling the dataset. In practical terms, we divided the meta-database into 10 bins considering the *activities_q1* meta-feature since it showed a considerable representational capability as the best-ranked descriptor (see Fig. 5a). Then, by randomly selecting a fixed number of samples per bin, we created a resampled meta-database, which was submitted to the same testing pipeline. Table 3 shows the accuracy and F-score performances considering the increasing size of sampled instances from each bin. As the number of selected meta-instances increases, accuracy and F-score also increase, converging to the performance obtained using the complete meta-database. This confirms the validity of our approach that, using cross-validation, was able to generalize well.

Table 3. Resampling experiment performance.

#Samples per bin	5	10	20	30	40	50
Meta-database size	42	82	148	198	246	286
Accuracy	0.54	0.6	0.67	0.7	0.72	0.72
F-score	0.49	0.56	0.64	0.68	0.71	0.71

5 Related Work

Most literature regarding the evaluation of process discovery algorithms aims to compare these techniques rather than selecting the most suitable for a given event log. In general, these frameworks create a knowledge base to support the comparison of different discovery techniques using real and synthetic event logs. An approach to recommend process discovery techniques for event logs is presented in Ribeiro et al. [24]. The authors propose a solution using a portfolio-based algorithm selection strategy to feed a recommender system. The portfolio is built using 12 log features and tested in 13 event logs. A subsequent problem was studied by Ribeiro et al. in [23] where the authors proposed a methodology based on a sensitivity analysis technique to evaluate the impact of parameter setting on process discovery algorithms. With the goal of computing model similarity among different techniques, Wang et al. [31] presented an approach based on behavioral and structural measures of process models. The information generated in the evaluation step supported the recommendation of PM algorithms. However, the need for high-quality reference models poses an important constraint to this technique. Alfonso et al. [21] constructed a knowledge base using standard model features such as control-flow patterns, invisible tasks, and infrequent behavior. The goal is to propose a group of classifiers that could associate

features with predefined quality metrics the extracted models have to satisfy. However, the recommendation approach is based on model features, which are not available in scenarios where there is no gold model. On the other hand, our approach is suitable in more general scenarios since it is unsupervised, meaning it does not rely on model features (our meta-features are extracted directly from event logs).

Research developed in [24] is the most closely related to this article. The authors framed the problem from a portfolio-based selection perspective while we use MtL. We build upon this original contribution by scaling the set of log features and instances. Moreover, we rely on a single model for recommendation while the reference work builds several models, with a negative impact on computing resource consumption. Finally, we leverage the problem formalization, enabling the application of a different set of experiments to study the discovery algorithms and the discovery problem. This construction paves the way for analyzing the relationship between process behavior, discovery algorithms, and quality criteria (as seen in Sect. 4.1).

6 Conclusion

Discovering a model for an event log is a difficult task due to the many available algorithms and characteristics of business processes. This work proposes an MtL approach to map event log profiles, discovery algorithms, and quality criteria. We propose a set of 93 meta-features extracted from the event logs to capture the behavior of the process at different levels, such as activity, trace, and log. Using MtL, we show that it is possible to take advantage of the quality of the process model by automatically recommending the appropriate discovery algorithms, answering *RQ1*. The best-fitting discovery method is one that maximizes output quality based on a set of metrics. Our approach correctly assigns the best technique with 76% of accuracy in scenarios of five discovery algorithms. Overall, the method confirms the results previously discussed in the literature, but our research design provides a more rigorous evaluation of them and reveals more details on feature importance. Due to space limitations, we did not demonstrate the ability of our framework to provide information for each specific event log. However, the framework can identify the recommended algorithm for each event log and which features triggered the suggestion. Furthermore, the proposed MtL approach is highly extensible, allowing the possibility of adding more features, discovery algorithms, and quality metrics. This way, our approach paves the way for a set of experimental analyses that can further verify how generalizable is the relationship between PM tasks and event log features. For instance, we showed that entropy and activity-related features are the most important in describing log behavior for the process discovery problem, answering *RQ2*. As future work, we plan to increase *problem*, *feature*, *algorithm* and *performance spaces* to comprehend how more complex scenarios can impact MtL performance. Further, we also aim at investigating optimization techniques to increase the quality of the recommendations.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Van der Aalst, W.M., Rubin, V., Verbeek, H., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**(1), 87 (2010). https://doi.org/10.1007/s10270-008-0106-z
3. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)
4. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2018). https://doi.org/10.1007/s10115-018-1214-x
5. Back, C.O., Debois, S., Slaats, T.: Entropy as a measure of log variability. J. Data Semant. **8**(2), 129–156 (2019)
6. Barbon Junior, S., Ceravolo, P., Damiani, E., Marques Tavares, G.: Evaluating trace encoding methods in process mining. In: Bowles, J., Broccia, G., Nanni, M. (eds.) DataMod 2020. LNCS, vol. 12611, pp. 174–189. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-70650-0_11
7. Berti, A., van der Aalst, W.M.P.: Reviving token-based replay: increasing speed while improving diagnostics. In: van der Aalst, W.M.P., Bergenthum, R., Carmona, J. (eds.) Proceedings of the International Workshop on Algorithms Theories for the Analysis of Event Data 2019 ATAED@Petri Nets/ACSD 2019. CEUR Workshop Proceedings, vol. 2371, pp. 87–103. CEUR-WS.org (2019)
8. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
9. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: the importance of fitness, precision, generalization and simplicity. Int. J. Coop. Inf. Syst. **23**(01), 1440001 (2014)
10. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman, R., et al. (eds.) OTM 2012. LNCS, vol. 7565, pp. 305–322. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33606-5_19
11. Ceravolo, P., Tavares, G.M., Junior, S.B., Damiani, E.: Evaluation goals for online process mining: a concept drift perspective. IEEE Trans. Serv. Comput. **15**, 2473–2489 (2020)
12. De Weerdt, J., De Backer, M., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Inf. Syst. **37**(7), 654–676 (2012)
13. He, X., Zhao, K., Chu, X.: AutoML: a survey of the state-of-the-art. Knowl.-Based Syst. **212**, 106622 (2021)
14. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_6

16. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery with guarantees. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (eds.) CAISE 2015. LNBIP, vol. 214, pp. 85–101. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19237-6_6
17. Lorena, A.C., Garcia, L.P.F., Lehmann, J., Souto, M.C.P., Ho, T.K.: How complex is your classification problem? A survey on measuring classification complexity. ACM Comput. Surv. **52**(5), 1–34 (2019)
18. Mendling, J., Depaire, B., Leopold, H.: Theory and practice of algorithm engineering (2021)
19. Muñoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 211–226. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15618-2_16
20. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
21. Pérez-Alfonso, D., Yzquierdo-Herrera, R., Lazo-Cortés, M.: Recommendation of process discovery algorithms: a classification problem. Res. Comput. Sci **61**, 33–42 (2013)
22. Pourbafrani, M., van der Aalst, W.M.P.: Extracting process features from event logs to learn coarse-grained simulation models. In: La Rosa, M., Sadiq, S., Teniente, E. (eds.) CAiSE 2021. LNCS, vol. 12751, pp. 125–140. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79382-1_8
23. Ribeiro, J., Carmona, J.: A method for assessing parameter impact on control-flow discovery algorithms, pp. 83–96. CEUR-WS.org (2015)
24. Ribeiro, J., Carmona, J., Mısır, M., Sebag, M.: A recommender system for process discovery. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 67–83. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_5
25. Rice, J.R.: The algorithm selection problem. In: Advances in Computers, vol. 15, pp. 65–118. Elsevier (1976)
26. dos Santos Garcia, C., et al.: Process mining techniques and applications a systematic mapping study. Expert Syst. Appl. **133**, 260–295 (2019)
27. Tipping, M.E., Bishop, C.M.: Mixtures of probabilistic principal component analyzers. Neural Comput. **11**(2), 443–482 (1999)
28. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
29. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artif. Intell. Rev. **18**(2), 77–95 (2002)
30. Vázquez-Barreiros, B., Mucientes, M., Lama, M.: ProDiGen: mining complete, precise and minimal structure process models with a genetic algorithm. Inf. Sci. **294**, 315–333 (2015)
31. Wang, J., Wong, R.K., Ding, J., Guo, Q., Wen, L.: Efficient selection of process mining algorithms. IEEE Trans. Serv. Comput. **6**(4), 484–496 (2013)
32. Weijters, A., Aalst, W., Medeiros, A.: Process mining with the heuristics miner-algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven (2006)