Discovering Declarative Process Model Behavior from Event Logs via Model Learning

Simone Agostinelli*, Giacomo Bergami†, Alessio Fiorenza*, Fabrizio M. Maggi†, Andrea Marrella*, Fabio Patrizi*
*Sapienza Universitá di Roma, Rome, Italy
Email: {agostinelli,marrella,patrizi}@diag.uniroma1.it, fiorenza.1661504@studenti.uniroma1.it
†Free University of Bozen-Bolzano, Bozen, Italy
Email: gibergami@unibz.it, maggi@inf.unibz.it

***Abstract*—Declarative business process (BP) models define the behavior of BPs as a set of temporal constraints, which can be summarized as a deterministic finite state automaton (DFA). Declarative BP discovery aims at inferring such constraints from event logs. To this aim, it requires as additional input the set of candidate constraints to be verified with respect to the event log. Intuitively, this restricts the discovery task to a conformance checking activity between a predefined set of constraint templates and an event log, preventing to learn any observed behavior that is not captured by those templates. In this paper, we investigate how to leverage Model Learning (ML) for the automated discovery of the DFA underlying the behavior of a declarative BP model, without using any further a-priori information in addition to the event log. To assess the quality of the discovered DFA, we introduce a novel definition of the standard process mining quality metrics, i.e., precision, generalization and simplicity, tailored to DFAs. Finally, a preliminary evaluation performed with real-life logs shows that ML enables to generate extremely simpler DFAs than state-of-the-art BP declarative discovery techniques, keeping similar values of precision and generalization.**

***Index Terms*—Model Learning, Declarative Process Models, Finite State Automata, Process Mining Quality Metrics**

I. Introduction

Traditional business process (BP) modeling notations, including the standard Business Process Model and Notation (BPMN), rely on a *procedural* paradigm wherein the BP model captures all allowed activity flows. In the past decade, several research works have exposed the limitations of this paradigm in the context of unpredictable, variable BPs working in turbulent environments [1]. As an alternative, in [2], the authors introduced a *declarative* BP modeling language called Declare, which allows the business analyst to focus on the formalization of few relevant temporal constraints, which must hold true during the execution of the BP, avoiding the burden to model many BP control-flow details that are fated to change through time. A strength of Declare is that its semantics can be characterized with Linear Temporal Logic over finite traces (LTL_f) [3]. Consequently, a collection of Declare constraints can be summarized as a deterministic finite state automaton (DFA) [4], which can be employed to perform formal reasoning over the original Declare constraints. Using Declare, the BPs are kept under-specified so that few constraints allow for multiple execution paths. In this way, the BP representation is more robust to changeable behaviors remaining compact.

In the field of Process Mining, many procedural BP discovery techniques exist that enable to generate a procedural BP model (e.g., a Petri net, a BPMN model, etc.) taking as input only the event log that records the past executions of the BP itself [5]. On the other hand, declarative BP discovery techniques require as further input the set of candidate constraints to be verified with respect to the event log. Intuitively, this restricts the discovery task to a conformance checking activity between a predefined set of constraint templates and an event log, thus preventing to learn any observed behavior that is not captured by those templates. While such an approach is suitable to verify that the behavior of a BP complies with certain (known) regulations, it becomes cumbersome when this knowledge is only partially known or even missing.

In this paper, we aim at mitigating this issue by investigating how to leverage a well-known Model Learning (ML) algorithm, called MDL [6], for the automated discovery of the DFA underlying the behavior of a declarative BP model, without using any further a-priori information except the event log. ML is a group of algorithms conceived for constructing state machine models of software and hardware components relying on observed input-output data of their runs [7]. Nonetheless, over the years, much progress has been made in the design of novel algorithms to make ML applicable in many different fields, such as telecommunication, network protocols, and control software. In this paper, for the first time (if compared with the previous literature solutions on this matter), we show how ML can successfully be employed for the discovery of the DFA underlying the behavior of a declarative BP model in an *unsupervised* way.

To assess the quality of the discovered DFAs, we introduce a novel definition of the standard process mining quality metrics, namely precision, generalization and simplicity, tailored to DFAs. In particular, we adapt the definition of these metrics existing in the literature for Petri nets to automata [8] [9]. This allows us to measure precision, generalization and simplicity of DFAs also and, transitively, of any (declarative) BP model whose semantics can be expressed using DFAs. We use these

Pre-print copy of the manuscript published by IEEE (available at ieeexplore.ieee.org)
identified by doi: 10.1109/ICPM53251.2021.9576870

metrics to perform an evaluation with 5 real-life event logs to compare the "DFAs' construction performance" of MDL against the Declare Miner [10], which is nowadays recognized among the best BP declarative discovery techniques available in the literature [5]. Note that, starting from a collection of Declare constraints, the Declare Miner employs the algorithm in [11] to generate a DFA that aims at formally representing their collective behavior. The results show that Declare Miner and MDL are both able to construct DFAs having similar values of generalization and precision. However, MDL does not require any other input than the event log, and generates DFAs that are extremely simpler (in terms of the amount of nodes/arcs discovered) than the ones discovered by the Declare Miner, thus being potentially more efficient and effective when used in the context of verification tasks.

The rest of the paper is organized as follows. In Section II, we provide the relevant background on Declare and declarative BP discovery in process mining. In Section III, we present an overview of the ML algorithms available in the literature, motivating why we choose MDL for the enactment of the DFA generation task and introducing its main features. In Section IV, we define our working assumptions and present the definitions of the standard process mining quality metrics for DFAs. Then, we benchmark Declare Miner and MDL against real-life datasets and discuss their performance to generate DFAs with respect to the previously defined quality metrics (Section V). Last, we draw our final conclusions and propose future work (Section VI).

II. Background

A. Declarative Process Modeling with Declare

In contrast with procedural techniques, which are suitable to represent the entire behavior of BPs in stable environments where BPs are repetitive and can be controlled, declarative BP modeling languages enable to focus on the formalization of few relevant and stable constraints over time that the BP should satisfy, thus avoiding the burden to model the details of the BP control flow that are bound to change over time. For this reason, declarative process mining techniques have been widely applied in the context of healthcare BPs [12]–[14], since they are characterized by a high degree of variability.

In [2], the authors introduce a declarative BP modeling language called Declare. Technically, a Declare model $\mathcal{M} = (A, \pi_{\mathcal{M}})$ consists of a set of activities A involved in a BP and a collection of constraints $\pi_{\mathcal{M}}$ defined over such activities. Declare constraints are instantiations of templates, i.e., patterns that define parameterized classes of properties. Such templates are grouped into four families: (i) *existence* templates, used to constrain the number of times an activity must/can be executed; (ii) *choice* templates, requiring the execution of some activities selecting them among a set of available alternatives; (iii) *relation* templates, expecting the execution of some activity when some other activity has been executed; (iv) *negation* templates, forbidding the execution of some activity when some other activity has been executed. The reader can refer to [15] for a full description of the language.

One of the key advantages of Declare is that its semantics can be characterized in different logic-based approaches, including Linear Temporal Logic over finite traces (LTL_f) [3]. Thus, any Declare constraint $c_i \in \pi_{\mathcal{M}}$ can be represented as an LTL_f formula φ_{c_i}. With φ, we indicate the LTL_f formula resulting from the conjunction of the single formulas φ_{c_i} (with i varying from 1 to the amount of Declare constraints in $\pi_{\mathcal{M}}$). φ reflects the behavior allowed by the entire declarative BP described by $\pi_{\mathcal{M}}$. Notably, it is well known in the literature that every LTL_f formula φ can be associated with a deterministic finite state automaton (DFA) that accepts exactly all the traces satisfying φ [4]. Consequently, the behavior of a declarative BP modeled with Declare can be represented as a DFA, making it formally verifiable and executable. For this reason, most of the declarative process mining approaches proposed in recent years are based on DFAs [16].

B. Declarative Process Discovery

In the area of process discovery, Lamma et al. [17] and Chesani et al. [18] describe the use of inductive logic programming techniques to extract models expressed as a SCIFF first-order logic theory [19]. Bellodi et al. [20] extend this technique by weighting the constraints with probabilistic estimation in a second step. Specifically, the learned constraints are translated into logical Markov formulas that allow probabilistic classification of traces. Both techniques in [17] and [20] rely on the availability of positive and negative execution traces.

Maggi et al. [10] proposed a two-step approach for discovering patterns of declarative processes expressed using Declare from positive execution traces only. The first phase is based on identifying frequent sets of related activities using an apriori algorithm. In the second phase, candidate constraints are verified by reproducing the log on specific automata, each accepting only those traces that conform to a constraint. The candidate constraints that are satisfied by a percentage of traces above a user-defined threshold are discovered. Other variants of the same approach are presented in [21], [22] and [23].

In [24], the authors introduce MINERful, a two-step algorithm for the discovery of Declare constraints. The first step of the approach is the building of a knowledge base, with information about temporal statistics about the (co-)occurrence of tasks within the log. Then, the validity of candidate constraints is computed by querying that knowledge base. Di Ciccio et al. [25], [26] design an extension of MINERful to discover target-branched Declare constraints, i.e., constraints in which the target parameter is replaced by a disjunction of actual tasks.

Another approach for the discovery of Declare models is described by Schönig et al. in [27]. Their technique is based on the translation of Declare templates into SQL queries on a

relational database instance, where the event log has previously been stored. The query answer assigns the free variables with those tasks that lead to the satisfaction of the constraint in the event log. Recently, an approach for discovering declarative BP models in the form of DCR-Graphs [28] has been proposed in [29].

All the above approaches are based on the construction of a set of candidate constraints that is then pruned using conformance checking. The candidate constraints to be checked are constructed using specific constraint templates that need to be specified in input. Differently from these approaches, in the technique presented in this paper, we construct from scratch the DFA representing the behavior of a declarative process model as observed in an event log, without any preliminary knowledge about the constraint templates to be discovered.

III. Model Learning

Model Learning (ML) refers to a group of test-based and counterexample-driven algorithms conceived for learning the models of black-box hardware (HW) and software (SW) systems. Examples of learned models are DFAs, state charts and Mealy machines, among others [7]. In ML, two classes of algorithms exist, namely *active* and *passive* algorithms.

Active learning (also called on-line learning) is based on the so-called MAT framework developed by Dana Angluin [30] in 1987, where the construction of a model involves a "learner" and a "teacher". The learner, who only knows the input/output alphabet of the system under learning (SUL), asks the teacher whether a specific trace belongs to the SUL (*Membership Query*). The teacher can answer "Yes" or "No". Based on the observed response, the learner tries to iteratively construct a model whose behavior aims at matching the model of the SUL. Once a hypothesized model is ready, then the learner asks the teacher whether the model is correct or not via an *Equivalence Query*. In the case of a correct model, the learning process ends. Otherwise, the teacher returns a counterexample that can be used by the learner to build new potentially valid traces to be verified through the Membership Query. By applying this procedure repeatedly, a model is approximated that represents the complete behavior of a SUL [31]. The basic active learning algorithm from the literature is called L^*, where a DFA is used to describe the behavior of the SUL [30]. A number of polynomial time algorithms utilizing a MAT model that generate DFAs have been designed and developed over time to enhance the performance of L^*, e.g., [31], [32].

There is also an extensive body of work on *passive learning* (also called off-line learning), where models are constructed from runs (i.e., available prerecorded traces) of the SW/HW systems. In passive learning, there is no interaction between the learner and SUL. The passive learning algorithms learn the models of the SUL from the available set of positive and negative traces (training data) stored in a log file [6], [33], [34]. Positive traces are those that belong to the target language, and negative traces do not belong to the target language. Some well-known passive learning algorithms include RPNI, (RPNI-)MDL, (RPNI-)EDSM, DeLeTe2, and OSTIA_4MM [31].

The main advantage of active learning algorithms relies in the generation of models capturing the full behavior of a SW/HW system, and not just of the specific runs that have occurred during an actual operation. Nonetheless, such algorithms are not particularly suitable for being applied to tackle the generation task of a DFA describing the behavior of a declarative BPM model, which is intrinsically an off-line activity and relies on the availability of a pre-recorded event log. For this reason, we decided to focus our attention on passive learning algorithms, and in particular on (RPNI-)MDL [6].

RPNI is a heuristic for DFA inference that merges states in an automaton representation of observations until a local minimum is reached [35]. Specifically, it performs a breadth-first search by trying to merge a newly encountered state with states already explored. However, whereas the basic RPNI approach merges the very first pair of nodes that resemble a valid merge, the MDL variant computes an additional score and only commits to a merge if the resulting hypothesis will yield a better score. This passive approach to state-merging works better in scenarios where only positive training data is available. Hence, in contrast to the majority of passive learning algorithms that require as input a set of negative training data, MDL only expects positive traces, thus representing the best candidate to be employed for the DFA generation task [36] from event logs.

IV. Quality metrics for DFAs

To evaluate the quality of the DFAs discovered via ML and compare them with the ones produced by state-of-the-art tools for declarative process discovery, we redefine here the standard quality metrics defined for procedural process discovery [37], namely precision, generalization and simplicity, within the context of DFAs. The quality metrics are based on the concept of alignment of a trace and a declarative process model. As already mentioned in Section II, any declarative process model $\mathcal{M}$ can be expressed as a DFA $G_{\mathcal{M}} = (V, E)$ over an alphabet Σ, where V is a set of vertices and E is a set of directed and labeled edges connecting those vertices. A trace is a temporally ordered sequence of activities in Σ. A trace is a log trace if it represents a BP execution stored in an event log. The set of model traces associated to a model $\mathcal{M}$ corresponds, instead, to the (possibly infinitely enumerable) set $\mathcal{L}[G_{\mathcal{M}}]$ of the traces that are accepted by the corresponding automaton $G_{\mathcal{M}}$ (i.e., generated from $G_{\mathcal{M}}$ by unfolding).

We can quantify the deviation of a trace from a DFA representing the behavior of a declarative BP model via alignments [38]–[42]. An alignment $\gamma(\sigma, \sigma')$ is a transformation between a log trace σ and a model trace σ'; such a transformation is a sequence of editing operations corresponding to one of the following situations: *(a)* an activity correspondence, *(b)* a deletion of an activity from σ, *(c)* an insertion of an activity in

σ. Given two traces σ and σ' of length l, the number of all the possible alignments is exponential over the trace length, i.e., $\approx \sqrt{l}\cdot(1+\sqrt{2})^{2l+1}$ [43]. However, we are not interested in all the possible alignments, but only in the alignments γ_O minimizing the quantity of deviation expressed via a cost function $\mathcal{K}$. We first define a cost function $\mathcal{K}$ quantifying the severity of a single deviation within the alignment: the simplest "correct" cost function $\mathcal{K}^\star$ returns 0 if the activities match and, otherwise, returns an unitary cost (e.g., 1). The total cost of an alignment γ, $\mathcal{K}(\gamma)$, is defined as the sum of the costs of the individual editing operations in the alignment. Given two traces σ and σ', we are interested in the alignments $\gamma_O(\sigma,\sigma')$ minimizing $\mathcal{K}(\gamma_O(\sigma,\sigma'))$ for σ and σ'. Last, given two traces σ and σ', we obtain the maximal possible alignment cost $\gamma^{\text{MAX}}(\sigma,\sigma')$ by deleting all the activities from σ and by inserting all the activities from σ'.

Based on the foregoing, we can now define the following set of metrics that can be used to evaluate the quality of a DFA $G_{\mathcal{M}}$ underlying the behavior of a declarative BP model $\mathcal{M}$:

- **Fitness**[1]: it quantifies the extent to which the discovered model can accurately reproduce the traces recorded in the log. A model has a perfect fitness if all traces in the log can be replayed from the beginning to the end on it.
- **Precision**: it assesses the coverage of the log traces by an inferred model. More precise models exhibit fewer under-fitting problems.
- **Generalization**: it estimates how an inferred model from a given log will reproduce future behaviors not seen in the log. More general models exhibit fewer over-fitting problems.
- **Simplicity**: it determines the size of a model, i.e., the number of elements required to represent it. Given two models describing the same language, the simplest model is the one having the smallest size.

We define fitness over trace alignments, thus enabling the identification of deviations within a log trace from a model trace. The desired fitness function should return 1 if the log trace can be replayed on the model from the beginning to the end with no conformance costs, and returns 0 for the lowest possible fitness level. Fitness is then defined as follows:

Definition 1 (Fitness)**.** *Given a log ℓ and a model $\mathcal{M}$, the fitness of a log trace $\sigma_L \in \ell$ and $\mathcal{M}$ is defined as:*

$$F_T(\sigma_L,\mathcal{M}) = 1 - \frac{\mathcal{K}(\gamma_O(\sigma_L,\sigma_{\mathcal{M}}))}{\mathcal{K}(\gamma^{\text{MAX}}(\sigma_L,\sigma_{\mathcal{M}}))},$$

where $\sigma_{\mathcal{M}}$ is a solution to the optimal alignment problem $\mathcal{A}(\sigma_L,\mathcal{M})$. The fitness of a log ℓ and a model $\mathcal{M}$ is defined by finding a function ϕ associating to each log trace σ_L a model trace $\phi(\sigma_L)$ such that the overall cost is minimized; we then divide the result by $\text{card}\,(\ell)$ *for comparing alignment costs over differently sized logs:*

$$F_L(\ell,\mathcal{M}) = 1 - \frac{1}{\text{card}\,(\ell)} \sum_{\sigma_L\in\ell} \frac{\mathcal{K}(\gamma_O(\sigma_L,\phi(\sigma_L)))}{\mathcal{K}(\gamma^{\text{MAX}}(\sigma_L,\phi(\sigma_L)))}$$

□

Given that the optimal alignment cost γ_O between two traces can be computed via the Levenshtein distance [45] and by assuming to represent each model $\mathcal{M}$ as a set of traces always containing the best alignment for each trace of ℓ, and given that we can express the pairwise distance between each trace of such a set as a matrix [46], we can easily compute $F_L(\ell,\mathcal{M})$ by reducing it to an assignment problem, by summing all the minimal alignment costs and computing the average. Similar considerations can be also carried out for the incoming metrics.

A model $\mathcal{M}$ is correctly inferred from a log ℓ if $F_L(\ell,\mathcal{M}) = 1$, and incorrect otherwise; therefore, we can consider fitness as a distance function between logs and models. When such model is correct, we want to assess the precision of the inferred model: in this situation, we will have that the model traces $\mathcal{L}[G_{\mathcal{M}}]$ associated to a model $\mathcal{M}$ will always be a subset of all the log traces. As $\mathcal{L}[G_{\mathcal{M}}]$ is potentially an infinitely enumerable set of traces due to loops within the automata, we are interested in providing such a comparison only among traces of the same length k as follows:

Definition 2 (Precision@k)**.** *Given a model $\mathcal{M}$ representing the behavior of the log ℓ having a perfect fitness, $F_L(\ell,\mathcal{M}) = 1$, we define the precision over the traces of length k as:*

$$P_k(\ell,\mathcal{M}) = \frac{\text{card}\left(\ell\big|_k\right)}{\text{card}\left(\mathcal{L}[G_{\mathcal{M}}]\big|_k\right)}$$

The perfect fitness guarantees that the set of the model traces of length k is always a subset of the set of log traces of the same length, thus certifying that such definition matches with the one usually intended in machine learning[2]*.* □

The notion of generalization is based on the one presented in [8]. This concept, however, needed to be adapted to the context of DFAs. The idea behind it is to simulate the missing behavior of a process model in a log by removing some of the traces of the original log and by testing the capability of the discovery algorithm to rediscover the same model also without the hidden behavior. The generalization power of a model inference algorithm $\mathcal{I}$ with respect to a log ℓ can be assessed as a cross-validation test by firstly subdividing the log ℓ into h sublogs $\ell_1,\ldots,\ell_h$. Then, a new automaton $G_{\mathcal{M}_i}$ is inferred from each sublog ℓ_i via the algorithm of choice $\mathcal{I}$; finally, we can assess the distance between the original log

[1]Note that we use the notion of fitness taken as is from [44].

[2]More formally, such a condition guarantees that $\mathcal{L}[G_{\mathcal{M}}]\big|_k \subseteq \ell\big|_k$ always holds, that can be rewritten as $\mathcal{L}[G_{\mathcal{M}}]\big|_k \cap \ell\big|_k = \mathcal{L}[G_{\mathcal{M}}]\big|_k$. Therefore, this definition is completely equivalent to $\text{card}\left(\ell\big|_k \cap \mathcal{L}[G_{\mathcal{M}}]\big|_k\right)/\text{card}\left(\ell\big|_k\right)$.

from each inferred automaton via the previously defined fitness function, thus obtaining the following definition:

Definition 3 (Generalization)**.** *Given a model inference algorithm $\mathcal{I}$, a log ℓ, and a log subdivision $\ell_1, \ldots, \ell_h$ into h sub-logs such that their union corresponds to ℓ, the generalization of such algorithm is defined as follows:*

$$G_{\{\ell_1,\ldots,\ell_h\}}(\mathcal{I}) = \frac{1}{h}\sum_{i=1}^{h} F_L(\ell,\, \mathcal{I}(\ell_i)) \quad \text{s.t.} \quad \ell = \bigcup_{1\le i\le h} \ell_i$$

Generalization values near to 1 imply that the associated inference algorithms over a given dataset generate models admitting all the traces seen in the training data.

Last, simplicity for a DFA representing the behavior of a declarative model $\mathcal{M}$ is defined as follows:

Definition 4 (Simplicity)**.** *Given a model $\mathcal{M}$ and its associated (minimized) automaton $G_{\mathcal{M}} = (V, E)$, the simplicity of the former via the latter is defined as $S(\mathcal{M}) = (\mathrm{card}\,(V), \mathrm{card}\,(E))$, where $\mathrm{card}\,(V)$ (and $\mathrm{card}\,(E)$) is the number of nodes (and edges) required to encode $\mathcal{M}$ as a minimal automaton.* □

V. Experiments

To compare the effectiveness of Declare Miner and MDL in generating DFAs from event logs, we have developed an interactive tool[3] as a standard Python application that implements some well-known ML algorithms discussed in Section III, including L*, RPNI, MDL and EDSM. The tool can be run interactively using a command-line interface, and allows the user to load existing event logs formatted with the XES (eXtensible Event Stream) standard. After choosing the ML learning algorithm to run, the tool generates as output the DFA observed from an input event log, together with the computation of the values for precision, generalization and simplicity, as defined in Section IV. In addition, the tool is able to digest a DFA represented in DOT (graph description language) format. This enables to quantify the values of the quality metrics related to DFAs previously discovered by other tools, such as the Declare Miner. We performed the experiments with a machine consisting of an Intel Core i5 Quad-Core CPU 2GHz and 16GB RAM.

To guarantee the reproducibility of our experiments, we employed 5 real-life logs:

- a log of a loan application process (LOAN) [47];
- a log of a road traffic fines management process (ROAD) log [48];
- a log that keeps track of incoming patients with sepsis in a hospital (SEPSIS) [49];
- a log of a reimbursement process for international declarations (REIMB) [50];
- a log that keeps track of travel permits (TRAVEL) [51];

TABLE I: Descriptive statistics of real-life logs.

Log Name	Total traces	Distinct traces (%)	Total events	Event classes	Trace length		
					min	avg	max
LOAN	13,087	33.4	262,200	36	3	20	175
ROAD	150,370	0.2	561,470	11	2	4	20
SEPSIS	1,050	80.6	15,214	16	3	14	185
REIMB	6,449	11.7	72,151	34	3	11	27
TRAVEL	7,065	20.9	86,581	51	3	12	90

TABLE II: Precision@k of the DFAs generated with Declare Miner and MDL

(a) Declare Miner

Log (ℓ)	k = 2	k = 3	k = 4
LOAN	0.304	0.042	0.005
ROAD	0.624	0.150	0.026
SEPSIS	0.467	0.110	0.018
REIMB	0.376	0.039	0.007
TRAVEL	0.977	0.114	0.023

(b) MDL

Log (ℓ)	k = 2	k = 3	k = 4
LOAN	0.579	0.237	0.085
ROAD	0.583	0.135	0.022
SEPSIS	0.701	0.234	0.052
REIMB	0.305	0.039	0.004
TRAVEL	0.238	0.023	0.002

Table I reports the characteristics of the five logs used. These logs are widely heterogeneous ranging from simple to very complex, with a log size ranging from 1050 traces (for the SEPSIS log) to 150,370 traces (for the ROAD log). A similar variety can be observed in the percentage of distinct traces, ranging from 0,2% to 80,6%, and the number of event classes (i.e., activities executed within the BP), ranging from 11 to 51. Finally, the trace length also varies from very short traces (containing only two events), to very long traces (containing 185 events).

To perform a fair comparison between Declare Miner and MDL, we acted as follows:

- for the discovery of the declarative BP models with the Declare Miner, we made use of RuM[4] [52], a desktop application that provides a comprehensive set of declarative process mining tools in a single unified package, including an implementation of the Declare Miner. Specifically, to perform the process discovery task, we instructed RuM to exploit all the constraint patterns implemented in the tool and we set the minimum constraint support to 100% for including in the model all those constraints satisfied by all the log traces. Then, we set the activity support filter to 0 to include constraint satisfaction for infrequent activities. Finally, we exploited a dedicated functionality provided by the Declare Miner to translate the discovered Declare model $\mathcal{M}$ into a minimized DFA $G_{\mathcal{M}}$.
- for the discovery of DFAs with MDL, we disabled the preprocessing step provided by the algorithm to remove the duplicate log traces.

The results of the experiments are shown in Tables II, III and IV, respectively. We notice that all the generated DFAs have a perfect fitness value with respect to the event logs used for

[3]https://github.com/bpm-diag/DECMOL

[4]https://rulemining.org/

TABLE III: Generalization of the DFAs generated with Declare Miner and MDL

(a) Declare Miner

Log (ℓ)	$G(\mathcal{I})$
LOAN	1
ROAD	1
SEPSIS	0.997
REIMB	—
TRAVEL	—

(b) MDL

Log (ℓ)	$G(\mathcal{I})$
LOAN	0.997
ROAD	0.883
SEPSIS	0.959
REIMB	0.761
TRAVEL	0.905

TABLE IV: Simplicity of the DFAs generated with Declare Miner and MDL

(a) Declare Miner

Log (ℓ)	$S(\mathcal{M})$
LOAN	$(1026, 13087)$
ROAD	$(513, 2817)$
SEPSIS	$(2048, 22528)$
REIMB	$(512, 9472)$
TRAVEL	$(1024, 19456)$

(b) MDL

Log (ℓ)	$S(\mathcal{M})$
LOAN	$(48, 130)$
ROAD	$(8, 41)$
SEPSIS	$(5, 21)$
REIMB	$(8, 59)$
TRAVEL	$(5, 120)$

their generation. As an example, in Figure 1, we show the DFA discovered by using MDL over the ROAD event log.

In Table II, we show the results obtained by assessing Precision@k for $k \in \{2, 3, 4\}$. We notice that for $k = 2$ both algorithms exhibit reasonable precision values close to 0.5, which can be considered as a good trade off value for precision. The only exception is the TRAVEL log, where the DFA generated by the Declare Miner has a precision close to 1 for k=2.

Then, we evaluated Generalization by splitting each log into sub-logs of around 50 traces each. Table III shows the obtained results. Given the significantly large size of the DFAs discovered by the Declare Miner (see also the next discussion about simplicity), we were able to compute all the values of the generalization only for the DFAs discovered with MDL, while the computation of the score for the Declare Miner exceeded the 5 hours timeout for the REIMB and TRAVEL datasets. In summary, all the results show that both Declare Miner and MDL generate DFAs that tend to have generalization values very close to 1. This can be explained by the fact that the DFAs underlying declarative models are not prescriptive like procedural BP models, thus enabling many possible behaviors not explicitly visible in the event logs.

Finally, in Table IV, we show the computation of Simplicity. Analyzing the results in the table, it is evident that the DFAs discovered by MDL are much simpler than the ones generated by the Declare Miner. This can be explained by the fact that the DFAs generated by the Declare Miner are computed as the product of the DFAs representing the single Declare constraints defined at the outset. As a consequence, the Declare Miner produces "spaghetti" DFAs that are not only impossible to be analyzed manually, but also very complex to be verified using formal techniques, given their size.

VI. Concluding Remarks

In this paper, we have investigated how to leverage ML for the automated discovery of DFAs representing the behavior of declarative BP models from event logs, without using any further a-priori information, such as the knowledge about the constraint templates to discover. Specifically, we have adopted a passive learning algorithm to conduct our experiments, called MDL. To assess the quality of the generated DFAs, we have introduced a novel definition of the standard process mining quality metrics, i.e., precision, generalization and simplicity, tailored to DFAs. Finally, we have performed a preliminary evaluation with real-life logs, showing that ML enables to generate much simpler DFAs than the Declare Miner, keeping similar values of precision and generalization.

In summary, the major strength of employing ML algorithms to discover DFAs relies in its unsupervised fashion, which enables to push the boundaries of the literature on declarative process discovery techniques far from their traditional supervised assumptions. In addition, the generation of simple DFAs representing the BP models of interest can improve the performance for the application of formal reasoning and verification techniques over them. Last but not least, starting from a discovered DFA, it is possible to check whether all the traces accepted by a DFA satisfy a given LTL_f formula and, therefore, to understand if any LTL_f formula is inferred by an event log.

The main issue with passive learning is that the quality of the learned DFA depends upon the quality of the recorded log. More recorded traces means the availability of more behavioral information to build an accurate BP model. In a nutshell, if the available datasets consist of few log traces, the risk exists that the generated DFAs are too precise, i.e., unable to capture any unobserved yet potentially reasonable behavior. This limitation can be mitigated by employing active learning algorithms, which are better capable to abstract the behavior of specific runs that have occurred during an actual operation to a more general behavior. However, such algorithms require a continuous interaction between two entities, the learner and the teacher, and need the definition of a new type of discovery algorithms that can work interactively.

As future work, first, we aim at investigating the boundaries of Declare Miner and MDL by testing them against more real-life logs and synthetic event logs of increasing complexity. Moreover, we also plan to investigate if ML algorithms are suitable for the generation of concise DFAs representing the behavior of procedural BP models, in particular, in cases where procedural discovery approaches introduce too many model constructs [5]. In fact, by nature, an event log is neither declarative nor procedural although the process generating

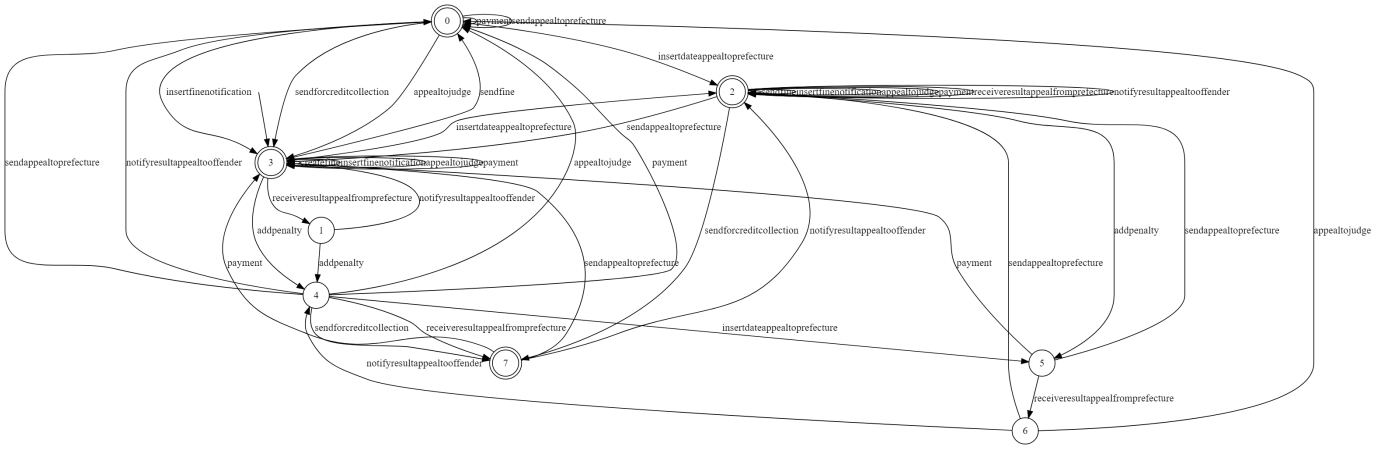

Fig. 1: The DFA discovered from the ROAD event log through MDL

it might have been conceived as such. Therefore, even if, in this paper, we have focused on the discovery of DFAs as representative of declarative BP models, it can be claimed that a DFA is potentially representative of a procedural model as well.

Finally, we aim at testing further passive learning algorithms, such as RPNI and EDSM, employing also negative traces to verify their impact on the improvement of the quality of the generated DFAs. Negative traces can contribute to filter out all those behaviors that should not be allowed by a DFA. Negative traces can be synthesized starting from rules that a process designer knows should never be satisfied in the process under analysis.

Acknowledgements: The work of S. Agostinelli and A. Marrella has been supported by the H2020 project DataCloud (Grant number 101016835) and the Sapienza grant BPbots. The work of G. Bergami has been supported by the project IDEE (FESR1133) funded by the Eur. Reg. Dev. Fund (ERDF) Investment for Growth and Jobs Prog. 2014-2020. The work of F. Patrizi has been supported by the project "DataawaRe Automatic Process Execution" (DRAPE), by the ERC Advanced Grant WhiteMech (No. 834228) and by the EU ICT-48 2020 project TAILOR (No. 952215).

REFERENCES

[1] D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, and S. Zugal, "Declarative versus Imperative Process Modeling Languages: The Issue of Understandability," in *10th International Working Conference on Business Process Modeling, Development and Support (BPMDS 2009)*. Springer, 2009, pp. 353–366.

[2] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "Declare: Full support for loosely-structured processes," in *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*. IEEE, 2007, pp. 287–287.

[3] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1977, pp. 46–57.

[4] G. De Giacomo and M. Y. Vardi, "Synthesis for LTL and LDL on Finite Traces," in *24th Int. Conf. on AI (IJCAI'15)*, 2015.

[5] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated Discovery of Process Models from Event Logs: Review and Benchmark," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 686–705, 2019.

[6] W. Daelemans, "Colin de la Higuera: Grammatical inference: learning automata and grammars - Cambridge University Press," *Mach. Transl.*, vol. 24, no. 3-4, pp. 291–293, 2010.

[7] F. Vaandrager, "Model Learning," *Communications of the ACM*, vol. 60, no. 2, pp. 86–95, 2017.

[8] A. F. Syring, N. Tax, and W. M. P. van der Aalst, "Evaluating conformance measures in process mining using conformance propositions," in *Transactions on Petri Nets and Other Models of Concurrency XIV*. Springer, 2019, pp. 192–221.

[9] A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, M. La Rosa, and D. Reißner, "Abstract-and-Compare: A Family of Scalable Precision Measures for Automated Process Discovery," in *16th International Conference on Business Process Management (BPM 2018)*. Springer, 2018, pp. 158–175.

[10] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Information Systems*, vol. 74, pp. 136–152, 2018.

[11] M. Westergaard, "Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL," in *9th International Conference on Business Process Management (BPM 2011)*, 2011, pp. 83–98.

[12] M. A. Grando, W. M. P. van der Aalst, and R. S. Mans, "Reusing a declarative specification to check the conformance of different CIGs," in *BPM Workshops*, 2012, pp. 188–199.

[13] M. A. Grando, M. H. Schonenberg, and W. M. P. van der Aalst, "Semantic-based conformance checking of computer interpretable medical guidelines," in *BIOSTEC*, vol. 273, 2013, pp. 285–300.

[14] M. Rovani, F. M. Maggi, M. de Leoni, and W. M. P. van der Aalst, "Declarative process mining in healthcare," *Expert Syst. Appl.*, vol. 42, no. 23, pp. 9236–9251, 2015.

[15] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science-Research and Development*, vol. 23, no. 2, pp. 99–113, 2009.

[16] F. M. Maggi, *Declarative Process Mining. In: Encyclopedia of Big Data Technologies*. Springer International Publishing, 2019, pp. 625–632.

[17] E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari, "Inducing Declarative Logic-Based Models from Labeled Traces," in *5th International Conference on Business Process Management (BPM 2007)*. Springer, 2007, pp. 344–359.

[18] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari, "Exploiting inductive logic programming techniques for declarative process mining," in *Transactions on Petri Nets and Other Models of Concurrency II*. Springer, 2009, pp. 278–295.

[19] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, "Verifiable agent interaction in abductive logic programming: the SCIFF framework," *ACM Transactions on Computational Logic (TOCL)*, vol. 9, no. 4, pp. 1–43, 2008.

[20] E. Bellodi, F. Riguzzi, and E. Lamma, "Probabilistic declarative process mining," in *International Conference on Knowledge Science, Engineering and Management*. Springer, 2010, pp. 292–303.

[21] M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Discovering cross-organizational business rules from the cloud," in *2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. IEEE, 2014, pp. 389–396.

[22] F. M. Maggi, "Discovering metric temporal business constraints from event logs," in *International Conference on Business Informatics Research*. Springer, 2014, pp. 261–275.

[23] T. Kala, F. M. Maggi, C. Di Ciccio, and C. Di Francescomarino, "Apriori and sequence analysis for discovering declarative process models," in *2016 IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2016, pp. 1–9.

[24] C. Di Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *ACM Transactions on Management Information Systems (TMIS)*, vol. 5, no. 4, pp. 1–37, 2015.

[25] C. Di Ciccio, F. M. Maggi, and J. Mendling, "Discovering target-branched declare constraints," in *International Conference on Business Process Management*. Springer, 2014, pp. 34–50.

[26] ——, "Efficient discovery of target-branched declare constraints," *Information Systems*, vol. 56, pp. 258–283, 2016.

[27] S. Schönig, A. Rogge-Solti, C. Cabanillas, S. Jablonski, and J. Mendling, "Efficient and customisable declarative process mining with SQL," in *International Conference on Advanced Information Systems Engineering*. Springer, 2016, pp. 290–305.

[28] T. Hildebrandt, R. Mukkamala, and T. Slaats, "Nested dynamic condition response graphs," in *4th IPM International Conference on Fundamentals of Software Engineering (FSEN 2011)*. Springer, 2011, pp. 343–350.

[29] V. Nekrasaite, A. T. Parli, C. O. Back, and T. Slaats, "Discovering Responsibilities with Dynamic Condition Response Graphs," in *31st International Conference on Advanced Information Systems Engineering (CAiSE 2019)*. Springer, 2019, pp. 595–610.

[30] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[31] S. Ali, H. Sun, and Y. Zhao, "Model Learning: A Survey on Foundation, Tools and Applications," *arXiv 1901.01910*, 2018.

[32] J. L. Balcázar, J. Díaz, R. Gavalda, and O. Watanabe, "Algorithms for learning finite automata from queries: A unified view," in *Advances in Algorithms, Languages, and Complexity*. Springer, 1997, pp. 53–72.

[33] A. W. Biermann and R. Krishnaswamy, "Constructing programs from example computations," *IEEE Transactions on Software Engineering*, no. 3, pp. 141–153, 1976.

[34] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic generation of software behavioral models," in *30th International Conference on Software Engineering (ICSE'08)*, 2008, pp. 501–510.

[35] J. Oncina and P. Garcia, "Inferring regular languages in polynomial updated time," in *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*. World Scientific, 1992, pp. 49–61.

[36] Z. Xu, C. Wen, S. Qin, and M. He, "Extracting automata from neural networks using active learning," *PeerJ Computer Science*, vol. 7, 2021.

[37] W. M. P. van der Aalst, *Process Mining: Data Science in Action*. Springer, 2016.

[38] M. de Leoni, F. M. Maggi, and W. M. P. van der Aalst, "An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data," *Information Systems*, vol. 47, pp. 258–277, Jan. 2014.

[39] G. De Giacomo, F. M. Maggi, A. Marrella, and S. Sardiña, "Computing Trace Alignment against Declarative Process Models through Planning," in *Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 2016, pp. 367–375.

[40] M. de Leoni and A. Marrella, "Aligning real process executions and prescriptive process models through automated planning," *Expert Syst. Appl.*, vol. 82, pp. 162–183, 2017.

[41] G. De Giacomo, F. M. Maggi, A. Marrella, and F. Patrizi, "On the disruptive effectiveness of automated planning for LTL$_f$-based trace alignment," in *Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17)*, 2017, pp. 3555–3561.

[42] M. de Leoni, G. Lanciano, and A. Marrella, "Aligning Partially-Ordered Process-Execution Traces and Models Using Automated Planning," in *Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 2018, pp. 321–329.

[43] M. S. Waterman, *Introduction to computational biology - maps, sequences, and genomes: interdisciplinary statistics*. CRC Press, 1995.

[44] M. de Leoni, F. M. Maggi, and W. M. P. van der Aalst, "Aligning Event Logs and Declarative Process Models for Conformance Checking," in *10th International Conference on Business Process Management (BPM 2012)*. Springer, pp. 82–97.

[45] G. Bergami, F. M. Maggi, M. Montali, and R. Peñaloza, "A Tool for Computing Probabilistic Trace Alignments," in *CAiSE Forum 2021*. Springer, 2021, pp. 118–126.

[46] T. Sagi and A. Gal, "Non-binary evaluation measures for big data integration," *VLDB J.*, vol. 27, no. 1, pp. 105–126, 2018.

[47] B. van Dongen, "BPI Challenge 2012," Apr 2012. [Online]. Available: https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204/1

[48] M. de Leoni and F. Mannhardt, "Road Traffic Fine Management Process," Feb 2015. [Online]. Available: https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249/1

[49] F. Mannhardt, "Sepsis Cases - Event Log," Dec 2016. [Online]. Available: https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639/1

[50] B. van Dongen, "International Declarations Log. BPI Challenge 2020," Mar 2020. [Online]. Available: http://icpmconference.org/2020/wp-content/uploads/sites/4/2020/03/InternationalDeclarations.xes_.gz

[51] ——, "Travel Permits Log. BPI Challenge 2020," Mar 2020. [Online]. Available: http://icpmconference.org/2020/wp-content/uploads/sites/4/2020/03/PermitLog.xes_.gz

[52] A. Alman, C. Di Ciccio, D. Haas, F. M. Maggi, and A. Nolte, "Rule mining with rum," in *2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020*, 2020, pp. 121–128.