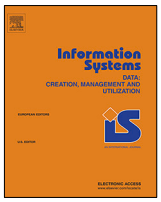
Information Systems 114 (2023) 102180

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis

Simone Agostinelli [a], Francesco Chiariello [a], Fabrizio Maria Maggi [b], Andrea Marrella [a,*], Fabio Patrizi [a]

[a] Sapienza Università di Roma, via Ariosto 25, 00185 Roma, Italy
[b] Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy

ARTICLE INFO

Article history:
Received 30 March 2022
Received in revised form 30 September 2022
Accepted 22 January 2023
Available online 1 February 2023
Recommended by Gottfried Vossen

Keywords:
Model learning
Deterministic finite state automata
Process mining quality metrics

ABSTRACT

Within the process mining field, Deterministic Finite State Automata (DFAs) are largely employed as foundation mechanisms to perform formal reasoning tasks over the information contained in the event logs, such as conformance checking, compliance monitoring and cross-organization process analysis, just to name a few. To support the above use cases, in this paper, we investigate how to leverage Model Learning (ML) algorithms for the automated discovery of DFAs from event logs. DFAs can be used as a fundamental building block to support not only the development of process analysis techniques, but also the implementation of instruments to support other phases of the Business Process Management (BPM) lifecycle such as business process design and enactment. The quality of the discovered DFAs is assessed wrt customized definitions of fitness, precision, generalization, and a standard notion of DFA simplicity. Finally, we use these metrics to benchmark ML algorithms against real-life and synthetically generated datasets, with the aim of studying their performance and investigate their suitability to be used for the development of BPM tools.

1. Introduction

Process mining [1] is a research area of Business Process Management (BPM) concerned with analyzing process data recorded in the so-called event logs, to gain insights into business processes. Within the process mining field, deterministic finite state automata (DFAs) have been largely employed to perform formal reasoning over the information contained in the logs [2].

For instance, DFAs have been used as an instrument to select, starting from a list of candidate temporal process constraints, the ones that are the most frequently satisfied in an event log, thus allowing for the selection, among those candidates, of the constraints that are the most suitable to represent the process behavior [3]. DFAs are also the foundation mechanism underlying the development of well-known techniques for conformance checking and compliance monitoring. They allow, indeed, for the verification of temporal properties over completed process executions in an event log for conformance checking [4,5], but they can also be used to formalize run-time verification semantics for compliance monitoring, i.e., for verifying, at run-time, the compliance of ongoing process executions with respect to some expected *de-jure* behavior [6,7].

Using the aforementioned approaches, it is possible, using DFAs, to carry on process analysis within a single organization, but also for cross-organizational analysis [8,9], e.g., to compare how processes are executed in different organizations that record their process executions into process data. To this aim, the behavior of a process within an organization can be discovered and represented as a DFA that can be then used to check if the process behavior recorded in a different log by another organization deviates from the behavior of the first one. The process behavior discovered from an event log in the form of a DFA (and recognized to be a *gold standard* for a given process) can also be used at run-time to recommend or even enforce the process participants to execute the process following that behavior. More in general, DFAs have been used in the enactment phase of the BPM lifecycle to support the execution of business processes according to some predefined constraints [10,11].

To support all the above BPM use cases, techniques for discovering DFAs from event logs would be very valuable. A first attempt in this direction was performed in our previous work [12], where we investigated how the use of a well-known Model Learning (ML) algorithm, namely (RPNI-)MDL, enables the generation of DFAs representing the behavior of declarative business process models from logs. ML is a group of algorithms conceived

* Corresponding author.
E-mail addresses: agostinelli@diag.uniroma1.it (S. Agostinelli), chiariello@diag.uniroma1.it (F. Chiariello), maggi@inf.unibz.it (F.M. Maggi), marrella@diag.uniroma1.it (A. Marrella), patrizi@diag.uniroma1.it (F. Patrizi).

https://doi.org/10.1016/j.is.2023.102180
0306-4379/

for constructing black-box finite-state diagram models of software and hardware systems relying on observed input–output data of their runs [13]. Differently from Model Checking, which is widely used for analyzing finite-state models [14], ML is a complementary technique for building such models from the available observations of the system behavior.

Recently, much progress has been made in the design of novel ML algorithms relying on advanced abstraction techniques that make them applicable to complex systems in areas such as telecommunication, network protocols, and control software [15]. Over the years, two main categories of ML algorithms emerged, namely *active* and *passive* algorithms. Active learning algorithms work by posing queries to the System Under Learning (SUL); then, based on the received responses, they infer the run-time behavioral model of the target system in the form of a state diagram [16]. On the other hand, passive learning algorithms learn the behavioral model of the SUL from a pre-defined set of training data, consisting of positive and negative traces, which are known to belong (or to not belong, respectively) to the language of the SUL [17].

In this paper, we extend the work presented in [12] by investigating the effectiveness of the best-known active and passive ML algorithms for the discovery of DFAs underlying the process behavior as observed in an event log.[1] In addition to using positive traces as it is common in process mining, we assume available also negative traces, collected in negative logs. The use of negative logs is, in principle, greatly beneficial in improving the quality of the learnt process, as they prevent including not only behaviors that are explicitly undesired (i.e., occurring in the negative log), but also those similar to them. The learning process, indeed, generalizes over negative traces in the same way as it does over the positive ones.

To assess the quality of the discovered DFAs, we use customized definitions of standard process mining quality metrics, namely *fitness*, *precision*, *generalization*, and a standard notion of DFA *simplicity*. In particular, for precision and fitness, we adopt the definition introduced in [18], which is based on the notion of Markovian Abstraction of logs and processes. On the other hand, the notion of generalization is based on the one presented in [19], where the main idea is to simulate the missing behavior of a process model by: *(i)* removing some traces from the log, *(ii)* re-discovering a process model using the log without such behaviors, and *(iii)* testing the capabilities of the discovery algorithm to come up with a model that fits also the removed traces. We use these metrics to perform an evaluation with 5 real-life event logs to assess the "DFAs' construction performance" of any tested ML algorithms against the Declare Miner tool [20], which is nowadays recognized among the best declarative process discovery techniques available in the literature [21]. In addition, we employed 17 synthetic event logs of increasing complexity to evaluate the scalability of ML algorithms.

The results show that active learning algorithms are not suitable to generate DFAs from real-life event logs available in process mining literature, because of their computational issues in the presence of inputs including a large amount of different behaviors. Conversely, Declare Miner and passive learning algorithms are able to construct DFAs having similar values of generalization and precision, even if passive learning algorithms tend to generate DFAs that are extremely simpler (in terms of the amount of nodes/arcs discovered) than the ones discovered by the Declare Miner tool, thus being not only more understandable, but also potentially more efficient when used in the context of automatic verification tasks.

[1] For readability purposes, the details of the additional contributions with respect to our previous work [12] are explained at the end of Section 6.

Since different passive ML algorithms provide different learning mechanisms, they can be used for different types of process analysis. In particular, MDL can be used for *standard process discovery*, since it requires as input an unlabeled event log. On the other hand, passive learning algorithms that use positive and negative traces to build a DFA (such as RPNI and EDSM, presented in Section 2.3) are more suitable to be used for *deviance analysis*, namely for the implementation of techniques for extracting relevant patterns discriminating between positive and negative (i.e., deviant and non-deviant) traces [22,23].

The rest of the paper is organized as follows. In Section 2, after providing the relevant background on event logs and DFAs, we present an overview of the ML algorithms available in the literature, focusing on the main features of the active and passive learning algorithms investigated in this paper. In Section 3, we discuss the state-of-the-art BPM techniques that could exploit our ML-based approach to build DFAs. In Section 4, we define our working assumptions and present the novel definitions of the standard process mining quality metrics for DFAs. Then, in Section 5, we use these metrics to benchmark ML algorithms against real-life and synthetically generated datasets. Last, in Section 6, we draw our final conclusions and spell out directions for future work.

2. Background

In this section, we introduce some preliminary notions on DFAs (Section 2.1), event logs (Section 2.2) and active and passive ML algorithms (Section 2.3), which are needed to understand the rest of the paper.

2.1. Deterministic Finite State Automata

We start by recalling the notion of Deterministic Finite State Automaton (DFA), a well-known structure used in language theory to model and recognize regular languages.

Definition 1 (*Deterministic Finite State Automaton (DFA)*)**.** A DFA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, where: *(i)* Σ is the finite *input alphabet*; *(ii)* Q is the finite set of *automaton states*; *(iii)* $q_0 \in Q$ is the *initial state*; *(iv)* $\delta : Q \times \Sigma \to Q$ is the *transition function*; and *(v)* $F \subseteq Q$ is the set of *final states*.

Let $t = e_1 \cdots e_n$ be a sequence of input symbols, called *word*, such that $e_i \in \Sigma$ $(i = 1 \ldots n)$, and $\mathcal{A}$ a DFA. A *path* of $\mathcal{A}$ on a word σ is a sequence $\pi = q_0 \xrightarrow{e_1} q_1 \cdots q_{n-1} \xrightarrow{e_n} q_n$ such that, for $i = 0, \ldots, n-1$, it is the case that $\delta(q_i, e_{i+1}) = q_{i+1}$, written $q_i \xrightarrow{e_{i+1}} q_{i+1}$. Notice that, $\mathcal{A}$ being deterministic, for every word σ, the corresponding path π is unique. We say that $\mathcal{A}$ *accepts* a word σ if the last state of the path π on σ is final, i.e., belongs to F.

2.2. Events, traces and event logs

A business process is a set of activities executed in a given setting to achieve predefined business objectives [24]. An *activity (signature)* is an expression of the form $A(a_1, \ldots, a_{n_A})$, where A is the *activity name* and each a_i is an *attribute name*. We call n_A the *arity* of A. The attribute names of an activity are all distinct, but different activities may contain attributes with matching names. We assume a finite set *Act* of activities, all with distinct names; thus, activities can be identified by their name, instead of by the whole tuple. Every attribute (name) a of an activity A is associated with a *type* $D_A(a)$, i.e., the set of values that can be assigned to a when the activity is executed.

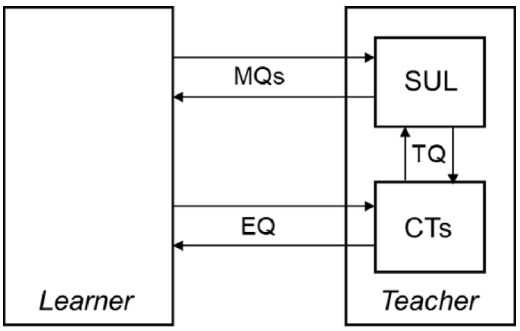


Fig. 1. Active Learning within the MAT framework.

An *event* is the execution of an activity (at some time) and is formally captured by an expression of the form $e = A(v_1, \ldots, v_{n_A})$, where $A \in Act$ is an activity name and $v_i \in D_A(a_i)$. The properties of interest in this work concern *(log) traces*, formally defined as finite sequences of events $\tau = e_1 \cdots e_n$, with $e_i = A_i(v_1, \ldots, v_{n_{A_i}})$. Traces model process executions, i.e., the sequences of activities performed by a process instance. A finite collection of executions into a set L of traces is called an *event log*. In this paper, a word (introduced in the previous section) and a log trace represent the same concept that, in the rest of the paper, we refer to as a *trace*.

2.3. Model Learning

Model Learning (ML) refers to a group of test-based and counterexample-driven algorithms conceived for learning the models (in the form of state diagrams) of black-box hardware (HW) and software (SW) systems from observed input–output data. Examples of learned models are DFAs, state charts and Mealy machines, among others [13]. The challenge of black-box learning of state diagrams was originally investigated by Moore in 1956 [25], who proved that the problem is inherently exponential. Over the years, the same challenge was also studied under different names, namely *active automata learning* [26], *regular extrapolation* [27], *regular inference* [28] and *grammatical inference* [29]. In ML, two classes of algorithms exist, namely *active* and *passive* algorithms.

2.3.1. Active learning

Active learning (also called on-line learning) is based on the so-called *Minimally Adequate Teacher* (MAT) framework, developed by Dana Angluin [30] in 1987, where the construction of a state diagram involves a "learner" and a "teacher". In the MAT framework, learning is considered as a game in which the learner asks queries to the teacher to infer the behavior of the System Under Learning (SUL), whose anatomy (represented in the form of a state diagram) is known by the teacher but not by the learner. The general approach underlying active learning within the MAT framework is shown in Fig. 1.

The learner, which only knows the input/output alphabet of the SUL, asks the teacher whether a specific trace belongs to the SUL through a *Membership Query* (MQ). 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 model hypothesis is ready, the learner asks the teacher whether the model is correct or not via an *Equivalence Query* (EQ). EQ is implemented using a conformance checking (CT) tool via a finite number of *Test Queries* (TQs) [31] targeted at the SUL. If one of the TQs exhibits a counterexample, then the answer to the TQ is "No", otherwise the answer is "Yes". The counterexample is returned to the learner, which can use it to build new traces to be verified through MQs. By iterative applications of this procedure, a model can be eventually obtained which correctly represents the complete behavior of the SUL [15].

The basic active learning algorithm from the literature is called L*, where a Mealy machine is used to describe the behavior of the SUL [30]. One inefficiency of the original L* algorithm is that it incrementally constructs an observation table that keeps track of the counterexamples obtained via TQs. Then, the observation table is used to build a new MQ for each new recorded entry. However, this is often redundant, because all prefixes of a counterexample are added as rows to the table. Counterexamples obtained through CT may be extremely long and are rarely minimal, which results in numerous redundant MQs. According to [13], while the required number of MQs posed by L* grows linearly with the number of inputs and quadratically with the number of states (meaning that it scales rather well when the number of inputs grows), CT quickly becomes a bottleneck for larger numbers of inputs, making it hard to apply the algorithm for more than 100 inputs [31]. This issue is mitigated by the recent TTT variant of L* [32], which replaced the observation table with discrimination trees enabling the filtering of redundant long counterexamples for determining state equivalences, thus improving the performance of L*. The TTT variant, which is currently the most efficient one for active learning, is the one used in Section 5 to perform our experiments where the inputs are (possibly large) event logs.

2.3.2. Passive learning

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 pre-recorded traces) of the SW/HW systems. In passive learning, there is no interaction between the learner and the SUL. 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 [17,33,34]. Positive traces are those that belong to the language of the SUL, while negative traces represent behaviors not belonging to the language. If compared with active learning algorithms, which aim at inferring the full behavior of the SUL, the weakness of passive learning relies in the "completeness assumption" made on the runs used for the generation of the model. In a nutshell, the focus is exclusively on the specific runs that have occurred during the actual operations of the SW/HW system under analysis, meaning that potentially relevant behaviors of the SUL are not captured by the model (because they are not recorded in the runs).

In this paper, we have analyzed the best-performing passive learning algorithm available in the literature [15], namely RPNI, and two optimizations developed over it: (RPNI-)EDSM and (RPNI-)MDL.

- RPNI (Regular Positive and Negative Inference) is an algorithm for DFA inference that merges states into an automaton representation of observations until a local minimum is reached [35]. Specifically, RPNI starts with a prefix tree acceptor, namely a tree-like DFA built from the learning examples by taking all the prefixes of the examples as states, and then performs a breadth-first search trying to merge a newly encountered state with states already explored. This enables to greedily create clusters of states to come up with a DFA that is always consistent with the examples.
- EDSM (Evidence Driven State Merging) is an algorithm for RPNI that can be employed to optimize the state merging step performed by RPNI. Specifically, EDSM selects the pair of states to merge based on a function that evaluates their evidence of equivalence. Only states having the greatest evidence of equivalence are selected for merging. The algorithm ends when all mergeable states have been considered [36].

- While the basic RPNI approach merges the very first pair of nodes that resemble a valid merge, the MDL (Minimum Description Length) algorithm 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 [33].

3. Related work

In this section, we introduce the techniques existing in the literature to support different phases of the BPM lifecycle and based on DFAs. Several of these works have been developed to support the analysis phase in the context of the process mining field [1] and, in particular, for process discovery and conformance checking. Other approaches provide instruments for log generation, for checking process models with respect to temporal rules, and for building workflow execution engines.

Process discovery aims at producing a process model based on example executions in an event log and without using any apriori information. [3] first proposed a two-phase approach for the discovery of declarative process models expressed using DECLARE [37], whose semantics can be formally defined using DFAs. The first phase is based on an apriori algorithm used to identify frequent sets of correlated activities. A list of candidate constraints is built on the basis of the correlated activity sets. In the second phase, the constraints are checked by replaying the log on specific DFAs, each accepting only those traces that are compliant to one constraint. Those constraints satisfied by a percentage of traces higher than a user-defined threshold, are discovered. Other studies have been conducted to remove inconsistencies and redundancies from discovered declarative models [38,39]. The proposed solutions resort on the language-inclusion and cross-product of the DFAs underlying the constraints. Approaches for ensuring the relevance of a set of LTL_f (Linear Temporal Logic on finite traces) [40] constraints discovered from an event log in terms of non-vacuity to the log have been presented in [41,42]. These approaches fully resort on DFAs underlying LTL_f semantics.

Since DFAs are focused on ruling out forbidden behavior, they are very suitable for defining compliance models that are used for checking that the process behavior as recorded in an event log complies certain regulations. The compliance model defines the rules related to a single instance of a process, and the expectation is that all the instances follow the model. Processes can be diagnosed for compliance violations in an a posteriori or offline manner, i.e., after process instance execution has been finished (conformance checking). However, the progress of a potentially large number of process instances can be monitored at runtime to detect or even predict compliance violations. For this, typically, terms such as compliance monitoring or online auditing are used [6,7].

The work described in [4,5] consists of converting a DECLARE model into a DFA and perform conformance checking of a log with respect to the generated DFA. The conformance checking approach is based on the concept of alignment and as a result of the analysis each trace is converted into the most similar trace that the model accepts. Other approaches for trace alignment with respect to DFA specifications have been presented in [43–45]. In [43,44], automated planning is used to improve the performance of the alignment task. In [45], a DFA-based approach for trace alignment with respect to data-aware constraints (i.e., constraints involving activity attributes) is presented. In addition to conformance checking approaches based on DECLARE, DFAs have also been used for process conformance wrt procedural specifications, like in [46,47].

Several DFA-based approaches have been presented for compliance monitoring of business processes with respect to a set of compliance rules. For example, in [48,49], the authors present a compliance monitoring tool implemented as a provider of the operational support in ProM [50–52]. The tool takes as input a reference model expressed in the form of LTL_f rules. At runtime, a stream of events can be sent by an external workflow management system to ProM through the operational support. The provider returns rich diagnostics about the status of each rule in the reference model with respect to the traces included in the stream. This approach has been extended in [53] to monitor data-aware constraints and in [54] to cover a larger set of rules expressed using LDL [55]. Recently, DFAs have also been used for the "multi-model" monitoring of business processes [56], in which process executions are monitored wrt hybrid specifications including both procedural and declarative models. All these approaches use run-time verification semantics based on DFAs.

Some approaches based on DFAs provide instruments to simulate process models and generate event logs from them. For example, in [57], the authors present a DFA-based approach to generate logs starting from a DECLARE model. In [58], the DFA semantics is encoded using Answer Set Programming (ASP) thus providing an approach for generating event logs from data-aware constraint specifications. Other DFA-based approaches, like the one presented in [59], can be used to identify which process paths in a process model may lead to violations with respect to predefined regulations expressed as temporal rules. The approach presented in [59] allows the user to understand how the process model should be changed to solve temporal rule violations. In addition, DFAs are also used for noise filtering from event logs [60] and for encoding background knowledge used to guide predictive models in predicting future developments of a business process [61].

Finally, some works have been developed for business process enactment based on DFAs. In [10], the authors propose a DFA generation algorithm from LTL_f tailored for the BPM context, where one single event at a time can be executed (differently from standard LTL_f allowing for the execution of multiple events simultaneously). The generated DFAs are used as a basis for business process enactment. Similarly, in [11], the authors present a tool to model workflows with DCR Graphs [62] and supporting their enactment. The semantics of DCR graphs can be expressed using DFAs as described in [62].

4. Quality metrics for DFAs

To evaluate the quality of the DFAs discovered via ML, we adapt the standard quality metrics available in process mining, namely *fitness*, *precision*, *generalization* and *simplicity* [1], traditionally defined for Petri nets, to DFAs. Before providing their formal definitions, we briefly describe their intuitive meaning:

- **Precision** is used to assess the degree to which the behaviors allowed by the process model are observed in the event log;
- **Fitness** quantifies the extent to which the discovered model accurately reproduces the traces recorded in the log;
- **Generalization** estimates how well a model inferred from a given log will reproduce future behaviors not seen in the log;
- **Simplicity** corresponds to the size of the model.

Different formalizations of these concepts can be found in the literature. For precision and fitness, we follow [18], which defines them based on the notion of *Markovian Abstraction*. In particular, the Markovian Abstractions of the log and of the process model (expressed using BPMN, or Petri nets) under examination are

computed. To build the Markovian Abstraction of the process model, the model is first transformed into its respective *Behavioral Automaton*, which is a particular graph modeling all the process executions allowed by the model. Since we use a DFA as a process specification, we can directly take advantage of its underlying graph-like structure to compute the Markovian Abstraction without producing the Behavioral Automaton first. Once the Markovian Abstractions have been computed, we refer to the very same definitions provided in [18] to compute precision and fitness.

Based on these considerations, we now first describe the metrics for positive event logs, i.e., logs containing positive traces representing the process behavior, and then we adapt them to the case of negative logs, i.e., logs containing negative traces representing the process behaviors that must not be covered by the discovered model. We start by recalling the definition of Markovian Abstraction (referring to [18] for full details) and then describe its computation for the case of DFAs.

A *k-th* order Markovian abstraction is a structure built starting from a set of traces β. The purpose of such a structure is to characterize all the strings σ indistinguishable from those in β when a memory of size k is used to compare σ against the traces in β. In other words, σ is considered as a possible trace of β if it is obtained by combining only subtraces of length k from β, in such a way that when a k-length sliding window W is used to scan σ, only subtraces from β are observed through W. Notice that this is different from requiring σ to be obtained by simple concatenation of subtraces.

Formally, a kth order Markovian Abstraction (*M^k-abstraction*, for short) over a set β of traces is a finite graph $M^k = (S, E)$, with nodes S and edges $E \subseteq S \times S$, such that:

- $S = \{-\} \cup S_1 \cup S_2 \cup S_3 \cup S_4$, where:
  - "-" is a special state;
  - $S_1 = \{\sigma \in \beta : |\sigma| \leq k\}$ is the set of traces of β having length up to k;
  - $S_2 = \{\sigma[i,k] : \sigma \in \beta, i = 1, |\sigma| > k\}$, with $\sigma[i,k]$ denoting the k-length subtrace of σ starting at position i, is the set of k-length prefixes of some trace in β (with length greater than k);
  - $S_3 = \{\sigma[i,k] : \sigma \in \beta, |\sigma| > k, i = |\sigma| - k + 1\}$ is the set of k-length suffixes of some trace in β (with length greater than k);
  - $S_4 = \{\sigma[i,k] : \sigma \in \beta, |\sigma| > k, 1 < i < |\sigma| - k + 1\}$ is the set of k-length subtraces of some trace in β (with length greater than k), excluding prefixes and suffixes;
- $E = \{(-,\sigma) : \sigma \in S_1 \cup S_2\} \cup \{(\sigma,-) : \sigma \in S_1 \cup S_3\} \cup \{(\sigma,\sigma') : \sigma, \sigma' \in S_2 \cup S_3 \cup S_4, \exists \hat{\sigma}, i \text{ s.t. } \hat{\sigma} \in \beta, |\hat{\sigma}| > k, 1 \leq i \leq |\hat{\sigma}| - k, \sigma = \hat{\sigma}[i,k], \sigma' = \hat{\sigma}[i+1,k]\}$.

We define the *kth order Markovian Abstraction* over a DFA $G_{\mathcal{M}}$ associated to a model $\mathcal{M}$ as the *kth order Markovian Abstraction* over the language $\mathcal{L}[G_{\mathcal{M}}]$ of the traces accepted by $G_{\mathcal{M}}$. Observe that, due to possible loops, such a language is in general infinite, thus the computation of M^k is not as straightforward as in the case of a finite set of traces β, where one can exhaustively consider all the traces. However, we are only interested in traces of length up to k and in subtraces of length $k+1$ which are obviously finitely many for a finite alphabet, and can be obtained by simply visiting the paths and the subpaths of $G_{\mathcal{M}}$ up to length $k+1$ (recall that the length of a path in a DFA is the number of edges in the path). In details, to compute $M^k = (S, E)$ for a DFA $G_{\mathcal{M}} = \langle \Sigma, Q, q_0, \delta, F \rangle$, we proceed as follows:

- we *trim* $G_{\mathcal{M}}$, i.e., we remove all its unreachable states and all the states that do not lead to an accepting state, in such a way that every subpath (and corresponding subtrace) in the automaton is part of an accepting path (and corresponding accepting trace);
- we compute all the (finitely many) paths $\pi = q_0 \xrightarrow{e_1} q_2 \cdots q_{m-1} \xrightarrow{e_{m-1}} q_m$ of length $m \leq k$, and check whether $q_m \in F$, in order to obtain all the accepted traces $\sigma = e_1 \cdots e_m$ of length $\leq k$, for each of which we then add $(-,\sigma)$ and $(\sigma,-)$ to E;
- we compute all the (finitely many) subpaths $\pi = q_1 \xrightarrow{e_1} q_2 \cdots q_k \xrightarrow{e_{k+1}} q_{k+1}$ of length $k+1$, extract their corresponding subtrace $\sigma = e_1 \cdots e_{k+1}$, split them into two subtraces $\sigma_p = e_1 \cdots e_k$ and $\sigma_s = e_2 \cdots e_{k+1}$ of length k and, for each so-obtained pair:
  - if $q_1 = q_0$ (i.e., σ_p is prefix of some accepted trace), we add the edge $(-, \sigma_p)$ to E;
  - if $q_{k+1} \in F$ (i.e., σ_s is a suffix of some accepted trace), we add the edge $(\sigma_s, -)$ to E;
  - we add the edge (σ_p, σ_s) to E;
- $S = \{\sigma : \exists \sigma' \text{ s.t. } (\sigma,\sigma') \in E \text{ or } (\sigma',\sigma) \in E\}$.

We thus have a method to compute the M^k-abstraction for both a log (set of traces) ℓ and a DFA $G_{\mathcal{M}}$ for a model $\mathcal{M}$. We can then proceed as in [18] and compare the two abstractions. To define precision, a matrix C is first constructed with rows labeled by edges of the M^k-abstraction of $G_{\mathcal{M}}$, columns labeled by edges of the log abstraction, and entries containing the Levenshtein distance (as a similarity measure) between the labeling subtraces, normalized to [0, 1]. Then, a mapping μ_C is defined from row labels (i.e., edges of the M^k-abstraction of $G_{\mathcal{M}}$) to column labels (i.e., edges of the M^k-abstraction of ℓ), which minimizes the overall sum of the entries $C(e, \mu_C(e))$, regarded as costs. Such assignment problem can be solved using, e.g., the Hungarian algorithm [63,64]. In case there are more rows than columns, the maximum cost 1 is given to unmapped rows.

Definition 2 (*(Markovian-Abstraction-Based) Precision [18]*)**.** Given a log ℓ and a DFA $G_{\mathcal{M}}$, let $M^k_\ell = (S_\ell, E_\ell)$ and $M^k_{G_{\mathcal{M}}} = (S_{G_{\mathcal{M}}}, E_{G_{\mathcal{M}}})$ be their respective M^k-abstractions, C the Levenshtein-distance-based cost matrix, and let $\mu_C : E_{G_{\mathcal{M}}} \to E_\ell$ be a partial function, solution of the assignment problem represented by C. The (Markovian-abstraction-based) kth order precision of $G_{\mathcal{M}}$ wrt ℓ is defined as:

$$MAP^k(\ell, G_{\mathcal{M}}) = 1 - \frac{\Sigma_{e \in E_{G_{\mathcal{M}}}} C(e, \mu_C(e))}{|E_{G_{\mathcal{M}}}|},$$

by taking $C(e, \mu_C(e)) = 1$, if $\mu_C(e)$ is undefined.

Intuitively, the precision of a DFA $G_{\mathcal{M}}$ wrt a (positive) log ℓ is a measure of how probable it is that a subtrace of a generic trace accepted by $G_{\mathcal{M}}$ is also a subtrace of some trace from ℓ. If precision equals 0, no subtrace from $\mathcal{L}[G_{\mathcal{M}}]$ comes from the log, while if precision is 1, all the (k-length) subtraces of $\mathcal{L}[G_{\mathcal{M}}]$ are also subtraces of some trace from ℓ. Thus, an increased precision value corresponds to an increased probability that a trace from $\mathcal{L}[G_{\mathcal{M}}]$ is found in ℓ.

The definition of fitness is similar, except that now: log subtraces (row labels) are mapped into process subtraces (column labels), a Boolean function checking whether the labels are equal or not is used as a cost function, and the cost of each assignment is weighted by the frequency of the edge in the log.

Definition 3 (*(Markovian-Abstraction-Based) Fitness [18]*)**.** Given a log ℓ and a DFA $G_{\mathcal{M}}$, let $M^k_\ell = (S_\ell, E_\ell)$ and $M^k_{G_{\mathcal{M}}} = (S_{G_{\mathcal{M}}}, E_{G_{\mathcal{M}}})$ be their respective M^k-abstractions, C the Boolean cost matrix,

and let $\mu_C : E_\ell \to E_{G_\mathcal{M}}$ be a partial function, solution of the assignment problem represented by C. The (Markovian-abstraction-based) kth order fitness of $G_\mathcal{M}$ wrt ℓ is defined as:

$$MAF^k(\ell, G_\mathcal{M}) = 1 - \frac{\Sigma_{e \in E_\ell} C(e, \mu_C(e)) F_e}{\sum_{e \in E_\ell} F_e},$$

where F_e stands for the frequency of edge e in E_ℓ and taking $C(e, \mu_C(e)) = 1$, if $\mu_C(e)$ is undefined.

Intuitively, the fitness of a DFA $G_\mathcal{M}$ wrt a (positive) log ℓ is a measure of how probable it is that a subtrace of a generic trace from ℓ is also a subtrace of some trace accepted by $G_\mathcal{M}$. If fitness equals 0 then no subtrace from the log is a subtrace from the language $\mathcal{L}[G_\mathcal{M}]$ of $G_\mathcal{M}$, whereas if fitness is 1, all the (k-length) subtraces from ℓ are also subtraces from $\mathcal{L}[G_\mathcal{M}]$. Thus, a higher fitness value yields a higher probability that a log trace is accepted by $G_\mathcal{M}$.

The notion of generalization is based on that presented in [19]. Generalization is meant to measure the ability of the DFA to correctly accept a behavior not seen before, i.e., not present in the log used for discovering the process. The idea is to simulate new behaviors of a process model by: removing some traces from the log, discovering a process model using the log without such behaviors, and then testing the capabilities of the discovery algorithm to come up with a model that fits also the removed traces. This is detailed next.

The generalization power of a model inference algorithm $\mathcal{I}$ with respect to a log ℓ can be assessed with a cross-validation approach. To this end, we partition the log ℓ as $\ell = \{\ell_1, \ldots, \ell_h\}$, in such a way that, for all ℓ_i's, $|\ell_i| \approx \frac{|\ell|}{h}$. Then, for every $i = 1, \ldots, h$, we infer a DFA $G_{\mathcal{M}_i}$ using sublog $\ell - \ell_i$, via algorithm $\mathcal{I}$. Finally, for every such DFA, we compute the corresponding fitness $MAF^k(\ell_i, G_{\mathcal{M}i})$, and average these fitness values over the number of sublogs. Observe that fitness is computed wrt the traces *not used* to discover the model; as a result, increasing values of this measure imply an increasing ability to correctly accept new behaviors.

Definition 4 (*Generalization*)**.** Given a model inference algorithm $\mathcal{I}$, a log ℓ, and a partition $\ell = \{\ell_1, \ldots, \ell_h\}$, the kth order *generalization* of $\mathcal{I}$ is defined as follows:

$$G_{\{\ell_1,\ldots,\ell_h\}}(\mathcal{I}) = \frac{1}{h} \sum_{i=1}^{h} MAF^k(\ell_i,\ \mathcal{I}(\ell - \ell_i)).$$

Generalization values close to 1 imply that the inference algorithm has high fitness over logs including unseen traces, thus indicating that the discovered DFA is a reliable model of the (unknown) process that generated the logs.

So far, we have considered the definition of precision, fitness, and generalization of a DFA wrt to a log of traces representing the desired behaviors (positive log). We now define also the *negative* versions of these metrics. In particular, given a (negative) log and a DFA $G_\mathcal{M}$, the *negative precision* of $G_\mathcal{M}$ is defined as the precision of the *complement* DFA of $G_\mathcal{M}$ wrt the log, i.e., the DFA accepting all and only the log traces not accepted by $G_\mathcal{M}$. Analogously, the *negative fitness* $G_\mathcal{M}$ is defined as the fitness of the complement DFA of $G_\mathcal{M}$ wrt the log. *Negative generalization* is analogous to generalization, once fitness is replaced by negative fitness.

The negative versions of each metric account for features that are dual wrt their positive counterpart. More intuitively: the negative variant of precision increases with the probability that a generic trace not accepted by the DFA occurs in the negative log, while negative fitness increases with the probability that a trace from the negative log is rejected by the DFA. Thus, the negative versions of the quality metrics provide additional valuable information to evaluate the performance of the discovered DFA wrt discarding behaviors not supposed to be covered by the process under analysis.

Finally, we introduce the definition of DFA simplicity, a measure used to quantify the complexity of a DFA:

Definition 5 (*Simplicity*)**.** Given a DFA $G_\mathcal{M} = \langle \Sigma, Q, q_0, \delta, F \rangle$, the *simplicity* of $G_\mathcal{M}$ is defined as $S(G_\mathcal{M}) = (|\tilde{Q}|, |\tilde{\delta}|)$, where $G_{\tilde{\mathcal{M}}} = \langle \tilde{\Sigma}, \tilde{Q}, \tilde{q}_0, \tilde{\delta}, \tilde{F} \rangle$ is the DFA obtained by trimming $G_\mathcal{M}$ and $|\tilde{\delta}| = |\{(s, c) \in Q \times \tilde{\Sigma} \text{ s.t. } \delta(s, c) \text{ is defined}\}|$.

Intuitively, simplicity accounts for the number of states and transitions in the trimmed DFA. Trimming is needed in order to avoid measuring redundant features of the DFA (such as bisimilar states). DFAs with larger number of states and transitions are considered more complex. This captures the intuition that a larger number of states and transitions provides the DFA with an increased power of discerning among different traces.

5. Experiments

To compare the effectiveness of the ML algorithms in generating DFAs from event logs, we have developed an interactive tool[2] as a standard Python application that employs four well-known ML algorithms, namely L* (in its TTT variant, cf. Section 2.3), RPNI, MDL and EDSM. The tool extends LearnLib [26], an open-source library implemented in Java and developed at the Dortmund University of Technology. LearnLib consists of three main modules: *(i)* the *automata learning* module, *(ii)* the *infrastructure* module, and *(iii)* the *equivalence queries* module. The automata learning module includes different learning algorithms and their supported modeling structures. It also provides algorithms for handling data structures efficiently that enable learning techniques to learn large-scale systems. The infrastructure module is targeted for query optimization providing utilities for statistical analysis (e.g., number of MQs, number of EQs, memory consumption, running time, etc.). One of the goals of this module is the reduction of MQs so that ML algorithms can be effectively applied to real-world complex systems. Finally, the equivalence queries module implements search-based techniques, conformance testing, and equivalence tests [26,65,66].

Our tool can be run interactively using a command-line interface, and allows the user to load existing (positive and negative) event logs formatted with the XES (eXtensible Event Stream) standard [67]. After choosing the ML algorithm to run, the tool generates as output the DFA discovered from the logs, together with the precision, fitness, generalization, and simplicity values, computed based on the definitions introduced in Section 4.

To guarantee the reproducibility of our experiments, we employed publicly available real-life [68–72] and synthetic[3] logs. To properly run (and fairly compare) the investigated algorithms, we split each tested log into a positive and a negative counterpart using the following procedure: *(i)* we computed the average trace length (corresponding to the number of events in a trace) within the log; *(ii)* we created a *positive* sublog including the traces with a length below the average; *(iii)* we created a *negative* sublog including the traces with a length greater than or equal to the average. The discovered DFAs should allow the behaviors observed in the positive sublog and reject the ones observed in the negative sublog.

To compare the effectiveness of the ML algorithms in generating DFAs from the event logs against traditional process discovery techniques, we also ran the experiments with the real-life logs

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

[3] The synthetic logs are available for download at: https://tinyurl.com/synth-logs.

Table 1
Descriptive statistics of real-life logs.

Log name	Total traces	Total$^+$ traces	Total$^-$ traces	Distinct traces (%)	Total events	Activity types	Trace length min	avg	max
LOAN	13 087	8164	4923	33.4	262 200	36	3	20	175
ROAD	150 370	82737	67633	0.2	561 470	11	2	4	20
SEPSIS	1050	838	212	80.6	15 214	16	3	14	185
REIMB	6449	4248	2201	11.7	72 151	34	3	11	27
TRAVEL	7065	4249	2816	20.9	86 581	51	3	12	90

using the Declare Miner tool [20]. For MDL and the Declare Miner, we provided as input only the positive sublogs. On the other hand, to simulate the interactive behavior of L*, we relied on the implementation presented in [32] employing the positive and negative sublogs to build the knowledge of the teacher about the DFA and to properly answer the MQs and EQs. Specifically, when a MQ related to a candidate trace is asked, the teacher's answer will be "Yes" if the trace belongs to the positive sublog (and, consequently, does not belong to the negative sublog). When a candidate DFA is built and the EQ is asked, a CT is performed checking that the DFA accepts all the traces in the positive sublog and none in the negative sublog. If the candidate DFA does not accept a trace included the positive sublog or accepts one from the negative sublog, that trace is returned as a counterexample to refine the DFA under construction.

We performed the experiments on a machine with an Intel Core i7 Quad-Core CPU 1.90 GHz and 16 GB RAM.

5.1. Experiments with real-life logs

We employed 5 real-life logs: *(i)* a log pertaining to a loan application process (LOAN) [68]; *(ii)* a log pertaining to a road traffic fines management process (ROAD) [69]; *(iii)* a log keeping track of incoming patients with sepsis in a hospital (SEPSIS) [70]; *(iv)* a log pertaining to a reimbursement process (REIMB) [71]; *(v)* a log keeping track of travel permit applications (TRAVEL) [72].

Table 1 reports the characteristics of the real-life logs. These logs are widely heterogeneous ranging from simple to very complex, with a log size 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 activity types (i.e., the size of the activity alphabet), ranging from 11 to 51. Finally, the trace length also varies from very short traces (containing only two events), to very long traces (with 185 events). Note that the columns Total$^+$ traces and Total$^-$ traces indicate the amount of traces selected from a log and moved into the positive/negative sublog according to the splitting policy defined in the previous section.

For the discovery of DFAs with the Declare Miner, we made use of RuM[4] [73], 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 use all the DECLARE constraint patterns implemented in the tool and we set the minimum constraint support to 100% for including in the output model all those constraints satisfied in all the log traces. Finally, we used a dedicated functionality provided by RuM to translate the discovered DECLARE model into a minimized DFA.

The results of the experiments involving the computation of precision and generalization on the DFAs discovered from the real-life logs are reported in Tables 2–7. First of all, we notice that L* was not able to compute any DFA starting from the real-life logs thus preventing the computation of all the quality metrics for this algorithm. This can be explained with the fact that active learning algorithms are very sensitive to inputs consisting of many different behaviors (cf. Section 2.3). This allows us to state that L* is not suitable for the discovery of DFAs from complex event logs.

Table 2
Precision$_+$@k of the DFAs generated with MDL and Declare Miner.

(a) Precision$_+$@k (MDL)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	0.395	8.94 s	0.088	16.98 s	0.014	105.86 s
ROAD	0.593	3.58 s	0.185	3.71 s	0.038	14.76 s
SEPSIS	0.672	4.06 s	0.216	6.80 s	0.046	1 m 42 s
REIMB	0.284	6.48 s	0.035	27.11 s	0.003	17 m 56 s
TRAVEL	0.214	11.91 s	0.013	10 m 54 s	/	Timeout

(b) Precision$_+$@k (Declare Miner)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	0.191	9.91 s	0.020	12.99 s	0.001	17.82 s
ROAD	0.412	3.02 s	0.077	3.17 s	0.010	4.45 s
SEPSIS	0.358	9.97 s	0.063	12.90 s	0.008	49.24 s
REIMB	0.139	43.98 s	0.009	56.70 s	0.0004	1 m 39 s
TRAVEL	/	Timeout	/	Timeout	/	Timeout

On the other hand, for passive ML algorithms, all the generated DFAs have a perfect fitness value with respect to the event logs used for their generation. This is a consequence of the fact that the considered ML algorithms, at least in the variants used here, cannot cope with noisy data and require positive and negative logs to be consistent, i.e., no trace may exist which occurs in both. While this might appear as a limitation, we observe that log consistency can be easily checked (in quadratic time, in general, in linear time if logs are sorted) and that noise-filtering approaches, possibly based on statistical criteria, can be adopted to clean the datasets before starting the learning phase. This can be applied, in general, to rule out outlier traces or to smooth their effect. While interesting, however, this issue is not in the scope of the present work.

As an example of learnt DFA, in Fig. 2, we show the DFA discovered from the ROAD log using MDL. As reported in Table 8, this DFA has been computed in 249 ms and includes 8 states and 33 transitions. Before analyzing and discussing the entire collection of experimental results, we immediately notice that MDL can generate simpler DFAs than the ones generated by the other passive ML algorithms and by the Declare Miner. This is witnessed by all the tested real-life logs.

In Tables 2, 3, and 4, we show the results obtained by assessing both positive and negative *k-th* order precision (indicated as Precision$_+$@k and Precision$_-$@k, respectively) for $k \in \{1, 2, 3\}$ (except for MDL and Declare Miner that only work with positive sublogs and for which we only provide Precision$_+$@k). Analyzing the results, we notice that, for $k = 1$, all the algorithms exhibit reasonable precision values between 0.139 and 0.672, which can be considered as a good trade-off range of values for precision, since the output DFA does not underfit nor overfit the log. The EDSM algorithm failed to compute the DFA for the LOAN log

[4] https://rulemining.org/

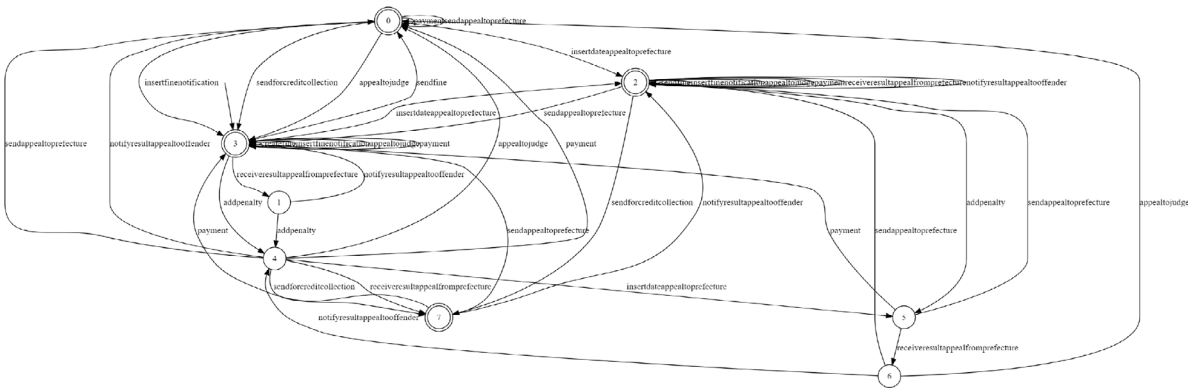


Fig. 2. The DFA discovered from the ROAD event log through MDL.

Table 3
Precision$_+$@k and Precision$_-$@k of the DFAs generated with RPNI.

(a) Precision$_+$@k (RPNI)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	0.222	17.37 s	0.022	47.37 s	0.002	16 m 40 s
ROAD	0.492	3.82 s	0.115	4.84 s	0.017	20.25 s
SEPSIS	0.457	5.44 s	0.101	13.95 s	0.015	4 m 4 s
REIMB	0.156	3.75 s	0.010	88.38 s	0.0005	62 m 46 s
TRAVEL	0.185	12.37 s	0.009	9 m 24 s	/	Timeout

(b) Precision$_-$@k (RPNI)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	0.191	16.97 s	0.020	1 m 3 s	0.001	40 m 5 s
ROAD	0.412	6.56 s	0.077	4.99 s	0.010	92 m 30 s
SEPSIS	0.358	3.07 s	0.063	6.89 s	0.008	169.18 s
REIMB	0.139	3.73 s	0.009	1 m 1 s	0.0004	62 m 13 s
TRAVEL	0.156	11.08 s	0.007	9 m 19 s	/	Timeout

Table 4
Precision$_+$@k and Precision$_-$@k of the DFAs generated with EDSM.

(a) Precision$_+$@k (EDSM)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	N/A DFA		N/A DFA		N/A DFA	
ROAD	0.489	2.47 s	0.117	2.73 s	0.018	13.35 s
SEPSIS	0.448	3.29 s	0.096	9.22 s	0.014	4 m 20 s
REIMB	0.160	3.66 s	0.011	59.62 s	0.001	47 m 7 s
TRAVEL	0.187	10.14 s	0.010	8 m 7 s	/	Timeout

(b) Precision$_-$@k (EDSM)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	N/A DFA		N/A DFA		N/A DFA	
ROAD	0.413	2.14 s	0.077	2.58 s	0.010	13.63 s
SEPSIS	0.358	2.87 s	0.063	7.26 s	0.008	3 m 7 s
REIMB	0.139	3.92 s	0.009	1 m 40 s	0.0004	87 m 27 s
TRAVEL	0.157	16.68 s	0.007	10 m 9 s	/	Timeout

(within a timeout of 24 h, indicated as "N/A DFA"), thus preventing the computation of all the quality metrics for this log. In addition, the Declare Miner exceeded the 24 h timeout threshold for the computation of the precision of the DFA (for any value of k, indicated as "timeout") discovered from the TRAVEL dataset. Similarly, all the passive algorithms exceeded the same timeout for the computation of the precision with $k = 3$. We can notice how the values for precision decrease while increasing the order k of the Markovian abstraction. This behavior is, however, expected and is due to the fact that the number of the considered substrings increases significantly faster for the automaton language than for the log.

Table 5
Generalization$_+$@k of the DFAs generated with MDL and Declare Miner.

(a) Generalization$_+$@k (MDL)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	0.999	2 m 50 s	0.999	3 m 9 s	0.999	3 m 35 s
ROAD	0.997	19 s	0.992	18 s	0.988	23 s
SEPSIS	0.999	21 s	0.999	22 s	0.998	55 s
REIMB	0.998	22 s	0.997	30 s	0.994	6 m 57 s
TRAVEL	0.998	33 s	0.997	3 m 42 s	/	Timeout

(b) Generalization$_+$@k (Declare Miner)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	0.999	53 s	0.999	57 s	0.999	1 m 20 s
ROAD	0.994	20 s	0.987	21 s	0.978	21 s
SEPSIS	0,998	58 s	0.997	1 m 12 s	0.996	2 m 48 s
REIMB	0.992	6 m 36 s	0.984	3 m 56 s	0.975	6 m 8 s
TRAVEL	/	Timeout	/	Timeout	/	Timeout

Tables 5, 6, and 7 show the generalization values obtained with the passive ML algorithms by splitting each log into sublogs of around 50 traces each. We assessed both positive and negative *k-th* order generalization (indicated as Generalization$_+$@k and Generalization$_-$@k, respectively) for $k \in \{1, 2, 3\}$ (except for MDL and Declare Miner for which we only provide Generalization$_+$@k). Also in this case, with the Declare Miner (for any value of k) and the passive ML algorithms (for $k = 3$), we were not able to compute the generalization values for the TRAVEL dataset. The results show that all algorithms generate DFAs that tend to have generalization values very close to 1, making them potentially more suitable to represent less prescriptive behaviors, such as the ones characterizing declarative processes (where many behaviors not explicitly visible in the event logs may be allowed). The slightly different values between positive and negative *k-th* order generalization are related to the fact that the amount of traces in the positive sublogs is always greater than the amount of traces in the negative sublogs.

Finally, in Table 8, we provide the simplicity values of the discovered DFAs and the time required for their generation. From the results, it is evident that the DFAs discovered with MDL are much simpler than the ones generated with the Declare Miner. This can be explained by the fact that the DFAs generated with the Declare Miner are computed as the product of the DFAs representing the individual DECLARE constraints composing the output DECLARE model. As a consequence, the Declare Miner produces "spaghetti-like" DFAs that are not only impossible to be analyzed manually, but also very complex to be verified using formal techniques, given their size.

From a time perspective, the results in Table 8 show that the performance of the passive ML algorithms and of the Declare Miner decreases exponentially when the logs include a high

Table 6
Generalization$_+$@k and Generalization$_-$@k of the DFAs generated with RPNI.

(a) Generalization$_+$@k (RPNI)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	1.0	55 s	1.0	1 m 52 s	0.999	21 m 2 s
ROAD	0.999	18 s	0.998	21.58 s	0.968	27 s
SEPSIS	1.0	24 s	1.0	31 s	1.0	3 m 25 s
REIMB	0.999	19 s	0.999	1 m 9 s	0.999	50 m 57 s
TRAVEL	0.999	49 s	0.999	7 m 7 s	/	Timeout

(b) Generalization$_-$@k (RPNI)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	1.0	1 m 29 s	1.0	4 m 27 s	1.0	1 h 38 m
ROAD	1.0	23 s	1.0	24 s	1.0	32 s
SEPSIS	1.0	22 s	1.0	34 s	1.0	3 m 28 s
REIMB	1.0	29 s	1.0	1 m 32 s	1.0	1 h 20 s
TRAVEL	1.0	40 s	1.0	7 m 4 s	/	Timeout

Table 7
Generalization$_+$@k and Generalization$_-$@k of the DFAs generated with EDSM.

(a) Generalization$_+$@k (EDSM)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	N/A DFA		N/A DFA		N/A DFA	
ROAD	0.999	31 s	0.998	32 s	0.989	39 s
SEPSIS	1.0	11 m 17 s	0.999	16 m 18 s	0.999	17 m 39 s
REIMB	0.999	50 m 20 s	0.999	57 m 39 s	0.998	1 h 24 m
TRAVEL	0.999	7 h 23 m	0.998	15 h 50 m	/	Timeout

(b) Generalization$_-$@k (EDSM)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	N/A DFA		N/A DFA		N/A DFA	
ROAD	1.0	29 s	1.0	35 s	1.0	45 s
SEPSIS	1.0	23 m 31 s	1.0	31 m 50 s	1.0	37 m 3 s
REIMB	1.0	1 h 11 m	1.0	48 m 35 s	1.0	2 h 1 m
TRAVEL	1.0	10 h 8 m	1.0	17 h 28 m	/	Timeout

Table 8
Simplicity of the DFAs generated with MDL, RPNI, EDSM, and Declare Miner.

(a) MDL

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(42, 123)	23344 ms
ROAD	(8, 33)	249 ms
SEPSIS	(4, 19)	360 ms
REIMB	(8, 53)	393 ms
TRAVEL	(8, 107)	2352 ms

(b) RPNI

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(185, 2442)	723 ms
ROAD	(23, 99)	462 ms
SEPSIS	(39, 313)	274 ms
REIMB	(39, 568)	261 ms
TRAVEL	(51, 1057)	1380 ms

(c) EDSM

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	/	/
ROAD	(21, 97)	2164 ms
SEPSIS	(55, 398)	947039 ms
REIMB	(38, 521)	3489343 ms
TRAVEL	(59, 1088)	21105718 ms

(d) DeclareMiner

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(435, 1673)	21732 ms
ROAD	(121, 433)	710 ms
SEPSIS	(631, 3259)	2409 ms
REIMB	(3565, 13168)	4182 ms
TRAVEL	(182763, 1424349)	38752548 ms

number of activity types. For example, by looking at the time required to compute the DFAs for the TRAVEL log, which includes 51 activity types, it is clear the worsening of the performance. A similar consideration can be done by analyzing the time required for computing precision and generalization for this log. This confirms the observation that large DFAs are not only difficult to understand for humans, but also inconvenient to be verified using automatic tools.

Our experiments also show that MDL produces simpler DFAs than the other tested ML passive algorithms. However, as discussed before, the learning algorithms experimented in this paper can be used orthogonally to support different process mining use cases, or even different phases of the BPM lifecycle. From this point of view, since RPNI and EDSM learn the DFAs based on explicit negative behaviors, they guarantee that the output DFAs are able to better discriminate between allowed and disallowed process behaviors and are more suitable to be used to support use cases that aim at characterizing different classes of process behaviors (like deviance analysis).

5.2. Experiments with synthetic logs

In order to have an idea of the performance of L* that we could not evaluate on real-life logs and, also, to test the passive algorithms in a controlled setting, we applied the algorithms under evaluation to 17 synthetic logs of increasing complexity.

For the generation of the synthetic event logs, we relied on the log generator presented in [57], which enables to synthesize logs whose behavior is compliant with respect to an input declarative process model. The declarative models were created using the Declare language introduced in [37]. Specifically, a Declare model consists of a set of activities and a collection of constraints defined over such activities. Declare constraints have a formal semantics based on LTL$_f$, and are instantiations of templates, i.e., patterns that define parameterized classes of temporal properties. We defined 4 different Declare models having the same alphabet of activities and containing 3, 5, 7 and 10 Declare constraints, respectively. Each model was used to generate from 3 to 5 different logs (on the basis of the amount of constraints employed) containing traces (1000 traces per log) of different lengths, i.e., of 10, 15, 20, 25 and 30 events.

The Tables describing the values of precision and generalization computed for the DFAs discovered using both active (in this case, L* was able to compute the DFAs) and passive algorithms are shown in an online appendix available at http://dx.doi.org/10.5281/zenodo.7591758, and confirm the results obtained for the real-life logs, i.e., the synthesized DFAs tend to have generalization values close to 1, independently of the characteristics of the log, thus confirming their suitability to represent the behavior of declarative processes. From the results obtained by computing the simplicity (see Table 9) of the discovered DFAs, we can see that MDL still produces the simplest automata and that L*, instead, produces significantly larger DFAs with respect to the ones produced by the passive algorithms.

Finally, from a time perspective, the results in Table 9 suggest that the passive ML algorithms scale very well when the length and the number of the distinct log traces increase.

6. Concluding remarks

In this paper, we have investigated how to leverage active and passive ML algorithms for the discovery of a DFA, representing the

Table 9
Simplicity of the DFAs generated with MDL, RPNI, EDSM and L*.

(a) MDL

Log (ℓ)	$S(\mathcal{M})$	t
log_3_10	(3, 11)	728 ms
log_3_15	(2, 16)	435 ms
log_3_20	(2, 17)	343 ms
log_3_25	(2, 15)	466 ms
log_3_30	(2, 16)	457 ms
log_5_10	(3, 19)	387 ms
log_5_15	(5, 27)	472 ms
log_5_20	(4, 28)	432 ms
log_5_25	(2, 18)	434 ms
log_5_30	(3, 20)	507 ms
log_7_15	(3, 17)	388 ms
log_7_20	(2, 17)	428 ms
log_7_25	(2, 18)	452 ms
log_7_30	(2, 17)	489 ms
log_10_20	(3, 19)	373 ms
log_10_25	(2, 18)	455 ms
log_10_30	(2, 18)	421 ms

(b) RPNI

Log (ℓ)	$S(\mathcal{M})$	t
log_3_10	(7, 27)	202 ms
log_3_15	(6, 45)	278 ms
log_3_20	(7, 53)	222 ms
log_3_25	(11, 73)	206 ms
log_3_30	(11, 66)	211 ms
log_5_10	(5, 37)	206 ms
log_5_15	(6, 53)	238 ms
log_5_20	(6, 58)	218 ms
log_5_25	(11, 70)	209 ms
log_5_30	(9, 92)	215 ms
log_7_15	(5, 40)	241 ms
log_7_20	(7, 65)	210 ms
log_7_25	(8, 83)	213 ms
log_7_30	(8, 71)	224 ms
log_10_20	(6, 56)	268 ms
log_10_25	(7, 73)	219 ms
log_10_30	(7, 73)	255 ms

(c) EDSM

Log (ℓ)	$S(\mathcal{M})$	t
log_3_10	(6, 22)	218 ms
log_3_15	(5, 42)	303 ms
log_3_20	(5, 52)	277 ms
log_3_25	(10, 88)	406 ms
log_3_30	(5, 52)	354 ms
log_5_10	(5, 36)	326 ms
log_5_15	(7, 53)	346 ms
log_5_20	(8, 73)	363 ms
log_5_25	(8, 85)	430 ms
log_5_30	(11, 123)	714 ms
log_7_15	(6, 47)	370 ms
log_7_20	(8, 77)	478 ms
log_7_25	(7, 70)	460 ms
log_7_30	(9, 94)	427 ms
log_10_20	(6, 58)	303 ms
log_10_25	(7, 79)	455 ms
log_10_30	(7, 71)	457 ms

(*continued on next page*)

Table 9 (*continued*).

(d) L*

Log (ℓ)	$S(\mathcal{M})$	t
log_3_10	(24, 277)	373 ms
log_3_15	(77, 1217)	462 ms
log_3_20	(124, 2092)	709 ms
log_3_25	(262, 3916)	1780 ms
log_3_30	(256, 4081)	2215 ms
log_5_10	(46, 631)	436 ms
log_5_15	(101, 1601)	670 ms
log_5_20	(174, 2942)	890 ms
log_5_25	(300, 5084)	2584 ms
log_5_30	(407, 6903)	3768 ms
log_7_15	(99, 1569)	483 ms
log_7_20	(180, 3044)	846 ms
log_7_25	(382, 6478)	3541 ms
log_7_30	(361, 6121)	3050 ms
log_10_20	(152, 2568)	867 ms
log_10_25	(347, 5883)	3199 ms
log_10_30	(429, 7277)	4703 ms

behavior of a business process, from a log. To assess the quality of the generated DFAs, we have introduced novel definitions of the standard process mining quality metrics, i.e., precision, fitness, generalization and simplicity, tailored to DFAs.

We have performed an evaluation with real-life and synthetic logs. The results of the evaluation showed that, among the tested ML algorithms, the active ML algorithm L* is not suitable to be applied to complex real-life logs since, for all the considered real-life logs, the algorithm was not able to generate a DFA in a reasonable amount of time. In addition, when applied to simpler synthetic logs, L* generates DFAs that are significantly more complex than the ones generated with the tested passive ML algorithms. However, we have to notice that, in our experiments, we adapted L* to be used for the automated learning of DFAs, while the algorithm was, instead, originally developed to involve humans in the loop. In the future, we plan to investigate the use of this type of algorithms in their original version (i.e., with users) for process analysis. Moreover, we will perform further experiments to compute precise threshold values (in terms of number of total traces/distinct traces / activity types in the input event log) enabling L* to properly compute the DFAs

Our evaluation also pointed out that MDL generates much simpler (and, therefore, more understandable) DFAs than the other passive algorithms, keeping similar values of precision and generalization. However, since RPNI and EDSM learn the DFAs from explicit negative behaviors, they produce DFAs that are able to better discriminate between positive and negative behaviors. This can represent an advantage wrt state-of-the-art process discovery algorithms that work with event logs including (non-labeled) positive and negative behaviors. For these algorithms, the decision to include (or exclude) a certain behavior underlying an execution trace into (from) the process being discovered depends on the noise filtering mechanisms embedded in the algorithms themselves, which cannot be easily customized.

From a time perspective, we can conclude that the performance of the passive ML algorithms decreases exponentially for logs including a large activity alphabet. Nonetheless, passive ML algorithms seem to scale very well for logs including a large number of distinct traces and/or traces including many events.

The main limitation of the ML algorithms relies in their limited robustness to noise, in particular, when positive and negative logs are inconsistent (i.e., they are not disjoint). Traditional process discovery techniques, instead, implement noise-filtering mechanisms that make them more robust to noise. One possible approach to mitigate this problem could be to make the logs consistent by applying suitable noise-filtering mechanisms before starting the learning phase with the ML algorithms.

Since, from our experimentation, it was clear that ML algorithms, being able to generalize well the behavior recorded in an event log, are suitable to represent processes that better fit in an open-world assumption representation such as declarative models, in the future, we would like to go a step forward

towards the development of a new approach for declarative process discovery and use alternate automata [74] to extract LTL_f-based temporal rules out of the DFAs generated with ML algorithms. In addition, since DFAs can formalize data-aware semantics via the propositionalization of data-aware constraints, we would like to investigate how ML algorithms can be employed in the context of "multi-perspective" process mining. Finally, we would like to introduce and evaluate, for the ML algorithms leveraging both positive and negative information, alternative quality metrics inspired by Machine Learning accuracy metrics (such as precision and recall) based on the evaluation of a confusion matrix.

This paper extends previous work in [12] in several directions and includes many new elements that were previously neglected:

- A revised introduction that makes immediately clear for the reader the research problem to be tackled and its significance in the process mining field;
- A new section that makes the background and the relevant preliminary concepts more explicit;
- A completely new related work section, targeted to describe the relevant works in process mining that employ DFAs to perform reasoning tasks over event logs;
- A revised section on ML algorithms that has been edited and refined to present the material more thoroughly;
- A novel section presenting a revised and thoroughly formalized version of the quality metrics. If compared with the ones proposed in [12], the novel metrics introduced in this paper are obtained adapting the state-of-the-art metrics presented in [18] (based on the notion of Markovian Abstraction of logs and processes) to DFAs.
- A novel evaluation section to investigate the extent to which the analyzed ML algorithms are suitable to be used in the context of process mining. Suitability is measured employing the novel quality metrics. Moreover, in addition to the real-life logs adopted in [12], we have generated 17 synthetic logs of increasing complexity to test the ML algorithms in a controlled setting and provide more robust findings.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The work of S. Agostinelli and A. Marrella has been supported by the H2020 project DataCloud, Italy (Grant number 101016835) and the Sapienza grant DISPIPE, Italy. The work of F. M. Maggi has been supported by the UNIBZ, Italy project CAT. The work of F. Patrizi has been supported by the project "Data-awaRe Automatic Process Execution" (DRAPE), by the ERC Advanced Grant WhiteMech, Italy (No. 834228) and by the EU ICT-48 2020 project TAILOR, Italy (No. 952215).

References

[1] W.M.P. van der Aalst, Process Mining - Data Science in Action, second ed., Springer, 2016.
[2] G. De Giacomo, M.Y. Vardi, Synthesis for LTL and LDL on finite traces, in: Q. Yang, M.J. Wooldridge (Eds.), Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, AAAI Press, 2015, pp. 1558–1564.
[3] F.M. Maggi, R.P.J.C. Bose, W.M.P. van der Aalst, Efficient discovery of understandable declarative process models from event logs, in: 24th International Conference on Advanced Information Systems Engineering, CAiSE 2012, 2012, pp. 270–285.
[4] M. de Leoni, F.M. Maggi, 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, 2012, pp. 82–97.
[5] M. de Leoni, F.M. Maggi, W.M.P. van der Aalst, An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Inf. Syst. 47 (2014) 258–277.
[6] L.T. Ly, F.M. Maggi, M. Montali, S. Rinderle-Ma, W.M.P. van der Aalst, A framework for the systematic comparison and evaluation of compliance monitoring approaches, in: 17th International Conference on Enterprise Distributed Object Computing, EDOC 2013, IEEE, 2013, pp. 7–16.
[7] L.T. Ly, F.M. Maggi, M. Montali, S. Rinderle-Ma, W.M.P. van der Aalst, Compliance monitoring in business processes: Functionalities, application, and tool-support, Inf. Syst. 54 (2015) 209–234.
[8] J.C.A.M. Buijs, B.F. van Dongen, W.M.P. van der Aalst, Towards cross-organizational process mining in collections of process models and their executions, in: Business Process Management International Workshops, Springer, 2011, pp. 2–13.
[9] M.L. Bernardi, M. Cimitile, F.M. Maggi, Discovering cross-organizational business rules from the cloud, in: 2014 IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2014, 2014, pp. 389–396.
[10] M. Pesic, D. Bosnacki, W.M.P. van der Aalst, Enacting declarative languages using LTL: avoiding errors and improving performance, in: 17th International SPIN Workshop on Model Checking Software, Springer, 2010, pp. 146–161.
[11] T. Slaats, R.R. Mukkamala, T.T. Hildebrandt, M. Marquard, Exformatics declarative case management workflows as DCR graphs, in: 11th International Conference on Business Process Management, BPM 2013.
[12] S. Agostinelli, G. Bergami, A. Fiorenza, F.M. Maggi, A. Marrella, F. Patrizi, Discovering declarative process model behavior from event logs via model learning, in: 3rd International Conference on Process Mining, ICPM 2021, IEEE, 2021, pp. 48–55.
[13] F. Vaandrager, Model learning, Commun. ACM 60 (2) (2017) 86–95.
[14] E.M. Clarke, Model checking, in: International Conference on Foundations of Software Technology and Theoretical Computer Science, Springer, 1997, pp. 54–56.
[15] S. Ali, H. Sun, Y. Zhao, Model learning: A survey on foundation, tools and applications, 2018, arXiv:1901.01910.
[16] H. Raffelt, B. Steffen, T. Berg, T. Margaria, LearnLib: A framework for extrapolating behavioral models, Int. J. Doftw. Tools Technol. Transfer 11 (5) (2009) 393–407.
[17] A.W. Biermann, R. Krishnaswamy, Constructing programs from example computations, IEEE Trans. Softw. Eng. 2 (3) (1976) 141–153.
[18] A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, M. La Rosa, Measuring fitness and precision of automatically discovered process models: A principled and scalable approach, IEEE Trans. Knowl. Data Eng. 34 (4) (2022) 1870–1888.
[19] A.F. Syring, N. Tax, 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.
[20] F.M. Maggi, C. Di Ciccio, C. Di Francescomarino, T. Kala, Parallel algorithms for the automated discovery of declarative process models, Inf. Syst. (2018) http://dx.doi.org/10.1016/j.is.2017.12.002.
[21] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F.M. Maggi, A. Marrella, M. Mecella, A. Soo, Automated discovery of process models from event logs: Review and benchmark, IEEE Trans. Knowl. Data Eng. 31 (4) (2019) 686–705.
[22] F. Taymouri, M. La Rosa, M. Dumas, F.M. Maggi, Business process variant analysis: Survey and classification, Knowl.-Based Syst. 211 (2021).
[23] G. Bergami, C. Di Francescomarino, C. Ghidini, F.M. Maggi, J. Puura, Exploring business process deviance with sequential and declarative patterns, 2021, CoRR, arXiv:2111.12454.
[24] M. Dumas, M. La Rosa, J. Mendling, H.A. Reijers, Fundamentals of Business Process Management, second ed., Springer, 2018.
[25] E.F. Moore, et al., Gedanken-experiments on sequential machines, Automata Stud. 34 (1956) 129–153.
[26] M. Isberner, F. Howar, B. Steffen, The open-source LearnLib, in: International Conference on Computer Aided Verification, Springer, 2015, pp. 487–495.
[27] A. Hagerer, H. Hungar, O. Niese, B. Steffen, Model generation by moderated regular extrapolation, in: International Conference on Fundamental Approaches to Software Engineering, Springer, 2002, pp. 80–95.
[28] T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, B. Steffen, On the correspondence between conformance testing and regular inference, in: International Conference on Fundamental Approaches to Software Engineering, Springer, 2005, pp. 175–189.

[29] H. Hungar, O. Niese, B. Steffen, Domain-specific optimization in automata learning, in: International Conference on Computer Aided Verification, Springer, 2003, pp. 315–327.

[30] D. Angluin, Learning regular sets from queries and counterexamples, Inform. and Comput. 75 (2) (1987) 87–106.

[31] D. Lee, M. Yannakakis, Principles and methods of testing finite state machines-a survey, Proc. IEEE 84 (8) (1996) 1090–1123.

[32] M. Isberner, F. Howar, B. Steffen, The TTT algorithm: a redundancy-free approach to active automata learning, in: International Conference on Runtime Verification, Springer, 2014, pp. 307–322.

[33] W. Daelemans, Colin de la Higuera: Grammatical inference: learning automata and grammars - Cambridge University Press, Mach. Transl. 24 (3–4) (2010) 291–293.

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

[35] J. Oncina, 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] O. Cicchello, S.C. Kremer, Beyond edsm, in: International Colloquium on Grammatical Inference, Springer, 2002, pp. 37–48.

[37] M. Pesic, H. Schonenberg, 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, p. 287.

[38] C. Di Ciccio, F.M. Maggi, M. Montali, J. Mendling, Ensuring model consistency in declarative process discovery, in: BPM, Springer, 2015, pp. 144–159, http://dx.doi.org/10.1007/978-3-319-23063-4_9.

[39] C. Di Ciccio, F.M. Maggi, M. Montali, J. Mendling, Resolving inconsistencies and redundancies in declarative process models, Inf. Syst. 64 (2017) 425–446.

[40] A. Pnueli, The temporal logic of programs, in: Foundations of Computer Science, Annual IEEE Symposium on, 1977, pp. 46–57.

[41] F.M. Maggi, M. Montali, C. Di Ciccio, J. Mendling, Semantical vacuity detection in declarative process mining, in: BPM, 2016, pp. 158–175.

[42] C. Di Ciccio, F.M. Maggi, M. Montali, J. Mendling, On the relevance of a business constraint to an event log, Inf. Syst. (2018).

[43] G. De Giacomo, F.M. Maggi, A. Marrella, 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.

[44] G. De Giacomo, F.M. Maggi, A. Marrella, 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.

[45] G. Bergami, F.M. Maggi, A. Marrella, M. Montali, Aligning data-aware declarative process models and event logs, in: Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings, 2021, pp. 235–251.

[46] J. Munoz-Gama, J. Carmona, Enhancing precision in process conformance: Stability, confidence and severity, in: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, Part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France, IEEE, 2011, pp. 184–191.

[47] S.J.J. Leemans, W.M.P. van der Aalst, T. Brockhoff, A. Polyvyanyy, Stochastic process mining: Earth movers' stochastic conformance, Inf. Syst. 102 (2021) 101724, URL https://doi.org/10.1016/j.is.2021.101724.

[48] F.M. Maggi, M. Montali, M. Westergaard, W.M.P. van der Aalst, Monitoring business constraints with linear temporal logic: An approach based on colored automata, in: BPM, 2011, pp. 132–147.

[49] F.M. Maggi, M. Westergaard, M. Montali, W.M.P. van der Aalst, Runtime verification of LTL-based declarative process models, in: Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers, 2011, pp. 131–146.

[50] M. Westergaard, F.M. Maggi, Modeling and verification of a protocol for operational support using coloured Petri nets, in: PETRI NETS, 2011, pp. 169–188.

[51] F.M. Maggi, M. Westergaard, Designing software for operational decision support through coloured Petri nets, Enterp. IS 11 (5) (2017) 576–596.

[52] F.M. Maggi, M. Montali, W.M.P. van der Aalst, An operational decision support framework for monitoring business constraints, in: Fundamental Approaches To Software Engineering - 15th International Conference, FASE 2012, Held As Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings, 2012, pp. 146–162.

[53] R. De Masellis, F.M. Maggi, M. Montali, Monitoring data-aware business constraints with finite state automata, in: ICSSP, 2014, pp. 134–143.

[54] G. De Giacomo, R. De Masellis, M. Grasso, F.M. Maggi, M. Montali, Monitoring business metaconstraints based on LTL and LDL for finite traces, in: BPM, 2014, pp. 1–17.

[55] G. De Giacomo, M.Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: International Joint Conference on Artificial Intelligence, 2013, pp. 854–860.

[56] A. Alman, F.M. Maggi, M. Montali, F. Patrizi, A. Rivkin, Multi-model monitoring framework for hybrid process specifications, in: Advanced Information Systems Engineering - 34th International Conference, CAiSE 2022, Leuven, Belgium, June 6-10, 2022, Proceedings, 2022, pp. 319–335.

[57] C. Di Ciccio, M.L. Bernardi, M. Cimitile, F.M. Maggi, Generating event logs through the simulation of declare models, in: Enterprise and Organizational Modeling and Simulation - 11th International Workshop, EOMAS 2015, Held At CAiSE 2015, Stockholm, Sweden, June 8-9, 2015, Selected Papers, 2015, pp. 20–36.

[58] F. Chiariello, F.M. Maggi, F. Patrizi, ASP-based declarative process mining, in: Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI'22), 2022.

[59] F.M. Maggi, A. Marrella, G. Capezzuto, A. Armas-Cervantes, Explaining non-compliance of business process models through automated planning, in: Service-Oriented Computing - 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings, 2018, pp. 181–197.

[60] R. Conforti, M. La Rosa, A.H.M. ter Hofstede, Filtering out infrequent behavior from business process event logs, IEEE Trans. Knowl. Data Eng. 29 (2) (2017) 300–314.

[61] C. Di Francescomarino, C. Ghidini, F.M. Maggi, G. Petrucci, A. Yeshchenko, An eye into the future: Leveraging A-priori knowledge in predictive business process monitoring, in: Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings, 2017, pp. 252–268.

[62] T.T. Hildebrandt, R.R. Mukkamala, T. Slaats, Nested dynamic condition response graphs, in: F. Arbab, M. Sirjani (Eds.), Fundamentals of Software Engineering - 4th IPM International Conference, FSEN 2011, Tehran, Iran, April 20-22, 2011, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 7141, 2011, pp. 343–350.

[63] H.W. Kuhn, The hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1–2) (1955) 83–97.

[64] F. Bourgeois, J. Lassalle, An extension of the munkres algorithm for the assignment problem to rectangular matrices, Commun. ACM 14 (12) (1971) 802–804, http://dx.doi.org/10.1145/362919.362945, URL https://doi.org/10.1145/362919.362945.

[65] H. Raffelt, B. Steffen, T. Berg, Learnlib: A library for automata learning and experimentation, in: Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems, 2005, pp. 62–71.

[66] M. Isberner, F. Howar, B. Steffen, Inferring automata with state-local alphabet abstractions, in: NASA Formal Methods Symposium, Springer, 2013, pp. 124–138.

[67] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, W.M.P. van der Aalst, XES, XESame, and ProM 6, in: P. Soffer, E. Proper (Eds.), Information Systems Evolution - CAiSE Forum 2010, Hammamet, Tunisia, June 7-9, 2010, Selected Extended Papers, in: Lecture Notes in Business Information Processing, vol. 72, Springer, 2010, pp. 60–75.

[68] B. van Dongen, BPI Challenge 2012, 4TU.ResearchData, 2012, http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f, URL https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204/1.

[69] M. de Leoni, F. Mannhardt, Road Traffic Fine Management Process, 4TU.ResearchData, 2015, http://dx.doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5, URL https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249/1.

[70] F. Mannhardt, Sepsis Cases - Event Log, 4TU.ResearchData, 2016, http://dx.doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460, URL https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639/1.

[71] B. van Dongen, International Declarations Log. BPI Challenge 2020, 4TU.ResearchData, 2020, http://dx.doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51, URL http://icpmconference.org/2020/wp-content/uploads/sites/4/2020/03/InternationalDeclarations.xes_.gz.

[72] B. van Dongen, Travel Permits Log. BPI Challenge 2020, 4TU.ResearchData, 2020, http://dx.doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51, URL http://icpmconference.org/2020/wp-content/uploads/sites/4/2020/03/PermitLog.xes_.gz.

[73] A. Alman, C. Di Ciccio, D. Haas, F.M. Maggi, 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.

[74] A. Camacho, E. Triantafillou, C. Muise, J.A. Baier, S.A. McIlraith, Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces, in: Thirty-First AAAI Conference on Artificial Intelligence, AAAI 2017, 2017.