ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Process Discovery on Deviant Traces and Other Stranger Things

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Chesani, F., Di Francescomarino, C., Ghidini, C., Loreti, D., Maggi, F.M., Mello, P., et al. (2023). Process Discovery on Deviant Traces and Other Stranger Things. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 35(11), 11784-11800 [10.1109/TKDE.2022.3232207].

Availability:
This version is available at: https://hdl.handle.net/11585/913012 since: 2023-01-30

Published:
DOI: http://doi.org/10.1109/TKDE.2022.3232207

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

12 October 2024

This is the final peer-reviewed accepted manuscript of:

F. Chesani *et al.*, "Process Discovery on Deviant Traces and Other Stranger Things," in *IEEE Transactions on Knowledge and Data Engineering*.

The final published version is available online at: *https://dx.doi.org/10.1109/TKDE.2022.3232207*

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/)

When citing, please refer to the published version.

Process discovery on deviant traces and other stranger things

Federico Chesani, Chiara Di Francescomarino, Chiara Ghidini, Daniela Loreti, Fabrizio Maria Maggi, Paola Mello, Marco Montali, Sergio Tessaris

Abstract—As the need to understand and formalise business processes into a model has grown over the last years, the process discovery research field has gained more and more importance, developing two different classes of approaches to model representation: procedural and declarative. Orthogonally to this classification, the vast majority of works envisage the discovery task as a one-class supervised learning process guided by the traces that are recorded into an input log.
In this work instead, we focus on declarative processes and embrace the less-popular view of process discovery as a binary supervised learning task, where the input log reports both examples of the normal system execution, and traces representing a "stranger" behaviour according to the domain semantics. We therefore deepen how the valuable information brought by both these two sets can be extracted and formalised into a model that is "optimal" according to user-defined goals. Our approach, namely `NegDis`, is evaluated w.r.t. other relevant works in this field, and shows promising results regarding both the performance and the quality of the obtained solution.

Index Terms—Process discovery, declarative process models, binary classification task.

1 Introduction

The modelling of business processes is an important task to support decision-making in complex industrial and corporate domains. Recent years have seen the birth of the Business Process Management (BPM) research area, focused on the analysis and control of process execution quality, and in particular, the rise in popularity of *process mining* [1], which encompasses a set of techniques to extract valuable information from event logs. *Process discovery* is one of the most investigated process mining techniques. It deals with the automatic learning of a process model from a given set of logged traces, each one representing the digital footprint of the execution of a case. Process discovery algorithms are usually classified into two categories according to the language they employ to represent the output model: procedural and declarative. Procedural techniques envisage the process model as a synthetic description of all possible sequences of actions that the process accepts from an initial to an ending state. Declarative discovery algorithms—which represent the context of this work—return the model as a set of constraints equipped with a declarative, logic-based semantics. Both approaches have their strengths and weaknesses depending on the characteristics of the considered process. For example, procedural techniques often produce intuitive models, but may sometimes lead to "spaghetti"-like outputs [2], [3]: in these cases declarative-based approaches might be preferable.

Declarative techniques rely on shared metrics to establish the quality of the extracted model, for example in terms of *fitness*, *precision*, *generality*, and *simplicity* [4], [5], [6]. In particular, fitness and precision focus on the quality of the model w.r.t. the log, i.e., its ability to accept desired traces and reject unlikely ones, respectively; generality measures the model's capability to abstract from the input by reproducing the desired behaviours, which are not assumed to be part of the log in the first place; finally, simplicity is connected to the clarity and understandability of the result for the final user.

Besides the declarative-procedural classification, process discovery approaches can be also divided into two categories according to their vision on the model-extraction task. As also pointed out by Ponce-de-Leòn et al. [6], the vast majority of works in the process discovery spectrum (e.g. [7], [8], [9], [10]) can be seen as a one-class supervised learning technique, while fewer works (e.g. [11], [12], [13], [14]) intend model extraction as a two-class supervised task—which is driven by the possibility of partitioning the log traces into two sets according to some business or domain-related criterion. In a sense, two-class process discovery can be related to sequence classification, which deals with the task of discriminating between classes of sequences by deriving sequential patterns from a temporal database. The main enhancement of process discovery is that it focuses on extracting behavioural constraints, which are in general much more informative (and concise), than a set of sequences [15]. Another significant difference is in the semantics of the classes, which for process discovery are usually referred to as *positive* and *negative* examples [6], [14], intending them as disjuncts sets of desirable and unwanted behaviours, respectively. In sequence classification instead, classes can have any meaning (and in general be more than just two, not necessarily disjunct). This also entails the slightly different goal of binary process discovery, that is to learn a model characterising the positive set while taking

- *F. Chesani, D. Loreti, and P. Mello are with DISI - University of Bologna, Italy. E-mail: daniela.loreti@unibo.it*
- *C. Di Francescomarino and C. Ghidini with Fondazione Bruno Kessler, Italy.*
- *F. M. Maggi, M. Montali, and S. Tessaris are with Free University of Bozen/Bolzano, Italy.*

Manuscript received April 19, 2005; revised August 26, 2015.

into account also the negative one.

A further consideration stems from the *completeness* of the log. Generally, a log contains and represents only a subset of the possible process executions. Other executions might be accepted or rejected from the viewpoint of the process, but this can be known only when a process model is learned or made available. This territory of *unknown traces* will be "shaped" by the learned model, and more precisely by the choice of the discovery technique, and possibly by its configuration parameters. Approaches that consider positive examples only provide a number of heuristics, thus allowing the user to decide to which extent the yet-to-be-seen traces will be accepted or rejected by the discovered model—ranging from the extremity of accepting them all, to the opposite of rejecting them all. The use of a second class of examples, identified on the basis of some domain-related criterion, allows to introduce some business-related criterion besides the heuristics.

In this work, we focus on declarative process models expressed in the Declare language [16], and embrace the view of process discovery as a binary supervised learning task. Hence, our starting point is a classification of business traces into two sets, which can be driven by various motivations. For example, in order to avoid underfitting and overfitting [10], many authors, as well as the majority of tools, suggest to ignore less-frequent traces (sometimes referred to as *deviances* from the usual behaviour [17]), thus implicitly splitting the log according to a frequency criterion. Another motivation for log partitioning could be related to the domain-specific need to identify "stranger" execution traces, e.g., traces asking for more (or less) time than expected to terminate, or undesirable traces that the user might want to avoid in future executions.

Independently of the chosen criterion for splitting the log, we adopt the terms negative and positive example sets to identify the resulting partitioning, keeping in mind that the "negative" adjective is not only connected to unwanted traces, but also to a sort of "upside-down world" of "stranger" behaviours. The information carried by this world diverges from that of the positive examples but—coupled with it—can be used to understand the reasons why differences occur, ultimately providing a more accurate insight of the business process. For this reason, we hereby focus on learning a model that is able to reconstruct which traces belong to which set (by accepting all the positive and rejecting, if possible, all the negative), while reflecting the user expectations on the quality of the extracted model according to predefined metrics. In particular our approach, namely `NegDis`, aims to discover a minimal set of constraints that allows to distinguish between the two classes of examples: in this sense, it can enrich existing approaches that provide descriptions of the positive example set only. Indeed, our exploitation of the "upside-down world" is useful not only to better clarify what should be deemed compliant with the model and what should not, but also to better control the degree of generalisation of the resulting model, as well as to improve its simplicity.

1.1 Contributions

In the framework described so far, our work advances the state of the art by proposing the following contributions.

- A novel discovery approach, `NegDis`, based on the underlying logic semantics of Declare, which makes use of the information brought by the positive and negative example sets to produce declarative models. The resulting technique can be used to either mine Declare process models from scratch, or refine an existing model with additional knowledge.
- The adoption of a satisfiability-based technique to identify the models.
- A heuristic to select the models according to the user preferences of generalisation or simplicity, where the language bias and the subsumption rules between Declare templates are not hard-coded in the discovery algorithm, but provided as an input for a finer-grain configurability.
- An evaluation of the performance of `NegDis` w.r.t. other relevant works in the same field, highlighting strengths and weaknesses of our technique.

To explain our approach, we start by providing an overview of the background and relevant concepts in Section 2, then we describe the approach in detail in Section 3, and we evaluate it in Section 4. Related work and conclusion follow.

2 Background

Our technique relies on the key concept of *event log*, intending it as a *set* of observed process executions, logged into a file in terms of all the occurred events. In this work, we adopt the eXtensible Event Stream (XES) storing standard [18] for the input log. According to this standard, each *event* is related to a specific process *instance*, and describes the occurrence of a well-defined step in the process, namely an *activity*, at a specific timestamp. The logged set of events composing a process instance is addressed as *trace* or *case*. From the analysis of the *event log*, we want to extract a Declare [16], [19] *model* of the process. Declare is one of the most used languages for declarative process modeling. Thanks to its declarative nature, it does not represent the process as a sequence of activities from a start to an end, but through a set of constraints, which can be mapped into Linear Temporal Logic (LTL) formulae over finite traces (LTL_f) [20], [21]. These constraints must all hold true when a trace completes. Declare specifies a set of templates that can be used to model the process. A constraint is a concrete instantiation of a template involving one ore more process activities. For example, the constraint $\mathsf{EXISTENCE}(\mathsf{a})$ is an instantiation of the template $\mathsf{EXISTENCE}(\mathsf{X})$, and is used to specify that activity a must occur in every trace; $\mathsf{INIT}(\mathsf{a})$ specifies that all traces must start with a. $\mathsf{RESPONSE}(\mathsf{a}, \mathsf{b})$ imposes that if the activity a occurs, then b must follow, possibly with other activities in between. For a description of the most common Declare templates see [16].

We assume that the log contains both *positive* traces—i.e., satisfying all the constraints in the business model—and *negative* traces—i.e., diverging from the expected behaviour by violating at least one constraint in the (intended) model.

Language bias. Given a set of Declare templates D and a set of activities A, we identify with $D[A]$ the set of all possible grounding of templates in D w.r.t. A, i.e. all the constraints that can be built using the given activities.

Traces and Logs. We assume that a *Trace* t is a *finite* word over the set of activities (i.e., $t \in A^*$, where A^* is the set of all the words that can be build on the alphabet defined by A). Usually a log is defined as a multi-set of traces, thus allowing multiple occurrences of the same trace: the frequency of a certain trace is then considered as an indicator, for example, of the importance of that trace within the process. Since our goal is to learn a (possibly *minimal*) set of constraints able to discriminate between two example classes, we rather opt to consider a *log* as a *finite set* of traces. As a consequence, multiple occurrences of the same trace will not affect our discovery process.

Declare constraints satisfaction and violation. Recalling the LTL$_f$ semantics [20], [21], and referring to the standard Declare semantics as in [16], we say that a constraint c *accepts* a trace t, or equivalently that t *satisfies* c, if $t \models c$. Similarly, a constraint c rejects a trace t, or equivalently t *violates* c, if $t \not\models c$. Given that a Declare model M is a conjunction of constraints, it follows that M *accepts* a trace t (t *satisfies* M) if $\forall c \in M, t \models c$. Analogously, a model M *rejects* a trace t (t *violates* M) if $\exists c \in M, t \not\models c$. In the following, we will write $t \models M$ meaning that M accepts t.

Positive and negative examples. Finally, we respectively denote with L^+ and L^- the sets of positive and negative examples (traces), reported in the input event log. We assume that: (*i*) $L^+ \cap L^- = \varnothing$, and (*ii*) for each trace $t \in L^-$ there exists at least one grounded Declare constraint $c \in D[A]$ that accepts all the positive traces and excludes t. In other words, we assume that the problem is feasible[1].

3 THE APPROACH

`NegDis` aims to extract a model which correctly classifies the log traces by accepting all cases in L^+ and rejecting those in L^-[2]. Besides that, it is required to perform an abstraction step in order to be able to classify also unknown traces, which are not in the input log.

3.1 Definitions

Before introducing our approach, we hereby provide some preliminary definitions that are relevant for the following explanation.

Model generality. The notion of a model accepting a trace, often referred to as the compliance of a trace w.r.t. the model, naturally introduces the relation of *generality* (or the converse *specificity*) between models. Intuitively, a model M is more general than another M' if M accepts a superset of the traces accepted by M'. That is, denoting with T_M the set of all traces compliant with a model M, M is more general than another model M'—and symmetrically M' is more specific than M—if and only if $T_{M'} \subseteq T_M$. More precisely, we say that

Definition 3.1. a model $M \subseteq D[A]$ is *more general* than $M' \subseteq D[A]$ (written as $M \succeq M'$) when for any $t \in A^*$, $t \models M' \Rightarrow t \models M$, and *strictly more general* (written as $M \succ M'$) if M is more general than M' and there exists a $t' \in A^*$ s.t. $t' \not\models M'$ and $t' \models M$.

Note that this definition is consistent with that of subsumption between Declare templates provided in Di Ciccio et al. [22]. Indeed, Declare templates can be organised into a subsumption hierarchy according to the logical implications that can be derived from their semantics.

Example 3.1. *The model* $M' = \{\mathsf{INIT(a)}\}$ *accepts only traces that start with* a. *Hence,* a *exists in each one of those accepted traces. In other words, all those traces also satisfy the model* $M = \{\mathsf{EXISTENCE(a)}\}$. *However, the latter model also accepts traces that contain* a *even if they do not start with* a *(i.e.,* $M \succ M'$*). This relation is valid irrespectively of the involved activity. In a sense, we could say that the template* EXISTENCE(X) *is* more general *than* INIT(X).

This idea is frequently expressed through the subsumption operator $\sqsupseteq$. Given two templates $d, d' \in D$, we say that d *subsumes* d', i.e. d *is more general than* d' (written $d \sqsupseteq d'$), if for any grounding of the involved parameters w.r.t. the activities in A, whenever a trace $t \in A^*$ is compliant with d', it is also compliant with d [22] .

Remark 3.1. For any pair of models $M, M' \subseteq D[A]$, $M \subseteq M'$ implies that M is more general than M' ($M \succeq M'$). This stems from the Declare semantics [16] on LTL$_f$ [21].

Unfortunately, the opposite implication does not hold, i.e. if we have $M, M' \subseteq D[A]$ such that $M \succeq M'$, we cannot guarantee that $M \subseteq M'$. A clear example is $M = \{\mathsf{EXISTENCE(a)}\}$ and $M' = \{\mathsf{INIT(a)}\}$.

Initial model. A wide body of research has been devoted to techniques to mine declarative process models that characterise a given event log (our positive traces). Our approach can leverage these techniques and refine their results by taking into account the negative examples as well. To this end, we consider a—possibly empty—*initial model* P, i.e. a set of Declare constraints that are known to characterise the positive traces. For example, such set can be the expression of domain knowledge or the result of a state-of-the-art discovery algorithm previously run on L^+. To apply our technique we only require that the constraints in P are non-conflicting and all the positive traces are compliant with all the constraints in P[3].

Candidate solution. As the goal of our technique is to refine the initial model P taking into account both positive and negative traces, we can define which are the necessary conditions for a set of constraints S to be a candidate solution for our discovery task.

Definition 3.2. Given the initial model P, a *candidate solution* for the discovery task is any $S \subseteq D[A]$ s.t.

(i) $P \subseteq S$;
(ii) $\forall t \in L^+$ we have $t \models S$;
(iii) $\forall t \in L^-$ we have $t \not\models S$.

1. Notice that sometimes real cases might not fulfill these assumptions. We will discuss this issue in section 3.3

2. The conditions on accepting all the positives and none of the negatives can be relaxed by requiring only a percentage of them.

3. We are aware that often state-of-the-art approaches do not emit a model compliant with all the traces in the input log. In these cases, we assume that the positive traces in L^+ are a subset of the original input including only the traces compliant with P.

Note that condition *(ii)* in the definition above ensures that any candidate solution is consistent (i.e., there are no conflicting constraints) when L^+ is non-empty.

Optimality criterion. Clearly, there can be several sets satisfying *(i), (ii), (iii)*. They differ from the way they classify the unknown traces, which are not in L^+, nor in L^-. Therefore, we need to introduce some way to compare the multiple output models in order to identify the preferable ones. In some context, *generality* can be a measure of the quality of the solution, i.e. we want to identify the set that is less committing in terms of restricting the admitted traces. In some other context on the contrary, we might be interested in the identification of a more specific model. So besides allowing all traces in L^+ and forbidding all traces in L^-, the choice between a general or specific model, obviously affects the classification of the unknown traces. Alternatively, *simplicity* is another criterion: one can be interested in the most *simple* solution, i.e. the one that is presumed to be easier to understand irrespectively from the degree of generality/specificity it accomplishes.

Let us focus on *generality* for the moment. In this case, we are interested in the candidate solution S (i.e., satisfying the properties of Definition 3.2) such that there is no other candidate solution $S' \subseteq D[A]$ strictly more general than S (i.e., $\nexists\ S'$ s.t. $S \prec S'$). Although testing for strict generality between two set of constraints is a decidable problem, its worst case complexity makes an exact algorithm unfeasible because, recalling definition 3.1, it would require to asses the compliance of any trace $t \in A^*$ with the two models which are going to be compared. For this reason, we propose an alternative method based on comparing the logical consequences that can be deducted from the models. The method makes use of a set of deduction rules which account for the *subsumption* between Declare templates. Our work integrates the rules introduced in [22], into a function, namely the *deductive closure operator*, which satisfies the properties of extensivity, monotonicity, and idempotence.

Definition 3.3. Given a set R of subsumption rules, a *deductive closure operator* is a function $cl_R : \mathcal{P}(D[A]) \rightarrow \mathcal{P}(D[A])$ that associates any set $M \in D[A]$ with all the constraints that can be logically derived from M by applying one or more deduction rules in R.

Example 3.2. *Let be:*

- $R = \{\ \mathsf{EXISTENCE(X)} \sqsupseteq \mathsf{INIT(X)}\ \}$
- $M' = \{\mathsf{INIT(a)}\ \}$

If we apply the deductive closure operator cl_R to M', we get:

$$cl_R(M') = \{\ \mathsf{INIT(a)}, \mathsf{EXISTENCE(a)}\ \}$$

Obviously, we are interested in sets R of *correct* subsumption rules, that is for any $M \subseteq D[A]$ and $t \in A^*$, $t \models M$ iff $t \models cl_R(M)$. In the rest of the paper, for the ease of understanding, we will omit the set R and we will simply write $cl(M)$. The complete set of employed rules is available in [23].

As the closure of a model is again a subset of $D[A]$, Remark 3.1 is also applicable, i.e. for any $M, M' \subseteq D[A]$, $cl(M) \subseteq cl(M')$ implies that M is more general than M' ($M \succeq M'$). Thanks to this property, the deductive closure operator can be used to compare Declare models w.r.t. generality. To provide an intuition, let us consider the following example:

Example 3.3. *Consider:*

- $R = \{\ \mathsf{EXISTENCE(X)} \sqsupseteq \mathsf{INIT(X)}\ \}$
- $M = \{\ \mathsf{EXISTENCE(a)}\ \}$
- $M' = \{\ \mathsf{INIT(a)}\ \}$

We cannot express any subset relation between M and M', thus making Remark 3.1 inapplicable. Nonetheless, if we take into account their closure, we have:

$$cl(M) = \{\mathsf{EXISTENCE(a)}\}$$
$$cl(M') = \{\mathsf{INIT(a)}, \mathsf{EXISTENCE(a)}\}$$

As $cl(M) \subseteq cl(M')$, we conclude that $M \succeq M'$.

In other words, the closure operator (being based on the subsumption rules) captures the logical consequences deriving from the Declare semantics. Due to the nature of the Declare language we cannot provide a complete calculus for the language of conjunctions of Declare constraints. For this reason, we cannot guarantee the strictness, nor the opposite implication (i.e. $M' \preceq M$ does not implies $cl(M) \subseteq cl(M')$). Anyway, the closure operator provide us a powerful tool for confronting candidate solutions w.r.t. the generality criterion. Table 1 summarises all the symbols used in this paper.

Symbol	Definition
D	set of Declare templates (language bias)
A	set of activities
$D[A]$	set of all grounding of templates in D w.r.t. A
L^+	set log traces labelled as positive. $L^+ \subseteq A^*$
L^-	set log traces labelled as negative. $L^- \subseteq A^*$
P	initial model. $P \subseteq \{c \in D[A] \mid \forall t \in L^+,\ t \models c\}$
$\mathcal{P}(D[A])$	power set of $D[A]$
cl_R or simply cl	closure operator equipped with a set R of subsumption rules
$compatibles$	set of constraints that accepts all traces in L^+
$sheriffs(t)$	constraints accepting all traces in L^+ and rejecting $t \in L^-$
$\mathcal{C}$	set of all constraints in $D[A]$ accepting all traces in L^+ and rejecting at least one $t \in L^-$
$\mathcal{Z}$	set of models accepting all traces in L^+ and rejecting all traces in L^-

TABLE 1: Symbols

3.2 Two-step procedure

In this section we introduce the theoretical basis of our `NegDis` approach. For the sake of modularity and easiness of experimenting with different hypotheses and parameters, we divide our approach into two clearly separate stages: the first which identifies the candidate constraints, and a second optimisation stage which selects the solutions. However, these two steps are merged into a single monolithic search-based approach.

Starting from the set of all constraints $D[A]$, the first stage aims at identifying all the constraints of $D[A]$ that accept all positive traces and reject at least a negative one. To this end, it first computes the set of constraints that accepts all traces in L^+, namely the set *compatibles*.

$$compatibles(D[A], L^+) = \{c \in D[A] \mid \forall t \in L^+,\ t \models c\} \quad (1)$$

By construction, any subset of $compatibles(D[A], L^+)$ is non-conflicting, because any trace in L^+ is accepted by all its constraints. This simplifies the optimisation step, since there is no need to verify whether any of the selected models are conflicting (see point *(ii)* of Definition 3.2).

For simplicity of the notation, since the set of constraints $D[A]$ and the log L^+ are given, we will omit them in the following. The *compatibles* set is then used to build a *sheriffs* function that associates to any trace t in L^- the constraints of *compatibles* that rejects t. The result is therefore a function with domain L^- and co-domain $\mathcal{P}(compatibles)$ s.t.:

$$\textit{sheriffs}(t) = \{c \in compatibles \mid t \not\models c\} \tag{2}$$

The second stage aims at finding the optimal solution according to some criterion. Therefore, it starts by computing two sets $\mathcal{C}$ and $\mathcal{Z}$. Let $\mathcal{C}$ be the set of those constraints in $D[A]$ that accept all positive traces and reject at least one negative trace. Such set can be derived from the *sheriffs* function as: $\mathcal{C} = \bigcup_{t \in L^-} \textit{sheriffs}(t)$. Let $\mathcal{Z}$ be all the subsets of $\mathcal{C}$ excluding all negative traces[4], i.e.,

$$\mathcal{Z} = \{M \in \mathcal{P}(\mathcal{C}) \mid \forall t \in L^- \; t \not\models M\} \tag{3}$$

Example 3.4. *Let* $L^+ = \{\mathsf{ab}\}$, $L^- = \{\mathsf{a}, \mathsf{b}, \mathsf{ba}\}$, *and* $D = \{\mathsf{EXISTENCE(X)}, \mathsf{RESPONSE(Y,Z)}\}$. *The set of activities is* $A = \{\mathsf{a}, \mathsf{b}\}$. *The grounded set of constraints is then* $D[A] = \{\mathsf{EXISTENCE(a)}, \mathsf{EXISTENCE(b)}, \mathsf{RESPONSE(a,b)}, \mathsf{RESPONSE(b,a)}\}$. *The compatibles set would be:*

$$compatibles(D[A], L^+) = \{\, \mathsf{EXISTENCE(a)}, \mathsf{EXISTENCE(b)}, \mathsf{RESPONSE(a,b)} \,\}$$

and the computation of the sheriffs function finds:

$$\begin{aligned}
&\textit{sheriffs}(\mathsf{a}) = \{\mathsf{EXISTENCE(b)},\ \mathsf{RESPONSE(a,b)}\}\\
&\textit{sheriffs}(\mathsf{b}) = \{\mathsf{EXISTENCE(a)}\}\\
&\textit{sheriffs}(\mathsf{ba}) = \{\mathsf{RESPONSE(a,b)}\}
\end{aligned}$$

In this case, there are two subsets of $\mathcal{C}$ *excluding all traces in* L^-, *i.e.,* $\mathcal{Z} = \{M_1, M_2\}$, *where*

$$\begin{aligned}
M_1 &= \{\mathsf{EXISTENCE(b)},\ \mathsf{EXISTENCE(a)},\ \mathsf{RESPONSE(a,b)}\}\\
M_2 &= \{\, \mathsf{RESPONSE(a,b)},\ \mathsf{EXISTENCE(a)} \,\}
\end{aligned}$$

Once $\mathcal{C}$ and $\mathcal{Z}$ are computed, the goal of the optimisation step is to select the "best" model in $\mathcal{Z}$ which can be either devoted to *generality/specificity*, or *simplicity*. When the most general model is desired, the procedure selects as solution the model $S \in \mathcal{Z}$ such that

$$\text{there is no } S' \in \mathcal{Z} \text{ s.t. } cl(S' \cup P) \subset cl(S \cup P) \tag{6a}$$

$$\text{there is no } S' \subset S \text{ s.t. } cl(S' \cup P) = cl(S \cup P) \tag{6b}$$

The first condition, Eq. (6a), ensures generality by selecting the model S for which the logical consequences of $S \cup P$ are the less restricting. In this way, the initial model P (containing a set of Declare constraints that are known to characterise L^+) is enriched taking into account the information derived by L^-. Furthermore, since from the point of view of generality we are not interested in the content of the selected model, but rather in its logical consequences, the closure operator cl ensures that no other model in $\mathcal{Z}$ is more general than the chosen S. The second condition, Eq. (6b), allows to exclude redundancy inside the selected model S by ensuring that it does not contain constraints that are logical consequence of others in S. Considering the previous example, this optimisation step allows to chose model M_2 as solution because $\mathsf{EXISTENCE(b)}$ is a logical consequence of $\mathsf{EXISTENCE(a)} \wedge \mathsf{RESPONSE(a,b)}$.

If we were interested in the less general model, condition (6a) would be

$$\text{there is no } S' \in \mathcal{Z} \text{ s.t. } cl(S' \cup P) \supset cl(S \cup P) \tag{7}$$

whereas the redundancy constraint would be ensured through the same Eq. (6b) because, even when we look for the most specific model, redundancy compromises its readability, without adding any value.

Generality/specificity is not the only desirable optimality criterion. If we are interested in the *simplest* model instead, a solution composed of a limited number of constraints is certainly preferable. So, we also experimented with an alternative optimisation formulation based on the set cardinality. The procedure selects the $S \in \mathcal{Z}$ such that:

$$\text{there is no } S' \in \mathcal{Z} \text{ s.t. } |cl(S' \cup P)| < |cl(S \cup P)| \tag{8a}$$

$$\text{there is no } S' \in \mathcal{Z} \text{ s.t. } |cl(S' \cup P)| = |cl(S \cup P)| \text{ and } |S'| < |S| \tag{8b}$$

where the first equation selects the set with the smaller closure, whereas the second allows to choose the solution with less constraints among those with closure of equal cardinality.

Theorem 3.1. The models that are solutions according to the simplicity criterion are also solutions for the generality criterion.

Proof. Suppose ad absurdum that there is a model $S \in \mathcal{Z}$ that is optimal according to the simplicity criterion of Eq. (8a) and (8b) but it is not the most general, i.e. either Eq. (6a) or Eq. (6b) are violated for S. If S violated Eq. (6a), it would exist an $S' \in \mathcal{Z}$ s.t. $cl(S' \cup P) \subset cl(S \cup P)$. But clearly, this implies that $|cl(S' \cup P)| < |cl(S \cup P)|$, which contradicts Eq. (8a). On the other hand, if S violated Eq. (6b), it would exist an $S' \subset S$ s.t. $cl(S' \cup P) = cl(S \cup P)$. Obviously we would also have $|S'| < |S|$ and $|cl(S' \cup P)| = |cl(S \cup P)|$, which contradict Eq. (8b). □

Conversely, the opposite implication in Theorem 3.1 does not necessarily hold. Indeed, let assume that P is empty and cl is the identity function[5]; consider two negative traces t_1, t_2 and a *sheriffs* function producing three constraints $\{c_1, c_2, c_3\}$. In particular, $\textit{sheriffs}(t_1) = \{c_1, c_2\}$ and $\textit{sheriffs}(t_2) = \{c_1, c_3\}$. The only simplicity-oriented solution would be $\{c_1\}$, whereas as with regard to the generality-oriented solutions we would have both $\{c_1\}, \{c_2, c_3\}$. We must remark that the simplicity principle is based on the intuition that "smaller" Declare models should be easier to understand for humans. However, we might notice that,

4. As we will discuss in section 3.3, the implementation must take into account that it might not be always possible to find models fulfilling Eq.3

5. This may also be the case when the constraints selected in the first stage are logically independent.

since the two models $\{c_1\}$ and $\{c_2, c_3\}$ are not directly comparable according to their semantics, deciding which is the "best" might depend on the constraints themselves, as well as the specific domain.

3.3 Implementation

The first stage is implemented via Algorithm 1, which starts by collecting the set $compatibles$ of the constraints that accept all the positive traces (Line 2). Subsequently, each negative trace is associated (by means of the function *sheriffs*) with those constraints in $compatibles$ that reject a trace in L^- (Line 4).

Algorithm 1 Identification of the constraints accepting all traces in L^+ and rejecting at least one trace in L^-.

Input: $D[A], L^+, L^-$
Output: *sheriffs* : $L^- \rightarrow \mathcal{P}(D[A])$

1: **procedure** SHERIFFSGENERATION($D[A]$, L^+, L^-)
2: $compatibles$=$\{c \in D[A] | \forall t \in L^+, \text{COMPLIANT}(t, c) = \texttt{True}\}$
3: **for** $t \in L^-$ **do**
4: *sheriffs*(t)=$\{c \in compatibles | \text{COMPLIANT}(t, c) = \texttt{False}\}$
5: **end for**
6: **return** *sheriffs*
7: **end procedure**

The implementation of the compliance verification COMPLIANT (i.e. $t \models c$) leverages the semantics of Declare patterns defined by means of regular expressions [22] to verify the compliance of the traces. It is implemented in Go language employing a regexp implementation that is guaranteed to run in time linear in the size of the input[6].

The second optimisation stage has been implemented using the Answer Set Programming (ASP) system CLINGO [24]. The main reason for selecting an ASP system for finite domain optimisation is that rules provide an effective and intuitive framework to implement a large class of closure operators. Indeed, all the deductive systems for Declare that we analysed in the literature (see e.g. [22], [25]) are expressed in the form of logic rules $c_1 \wedge \ldots \wedge c_n \implies c_{n+1}$, where c_i are Declare constraints or their negation; therefore they can be equivalently expressed as Normal Logic Programs (NLPs) [26] by exploiting the assumption that the set of activities is finite and known in advance. The declarative semantics of NLPs ensures that the (unique) deductive closure of the rules is taken into account for the optimisation stage. For example the valid formula $\text{INIT}(a) \wedge b \neq a \implies \text{PRECEDENCE}(a, b)$ that holds for any pair of activities a, b can be written as the rule

```
precedence(A,B):-init(A),activity(B), A!=B.
```

using a specific predicate (activity/1) representing the set of all activities.[7]

The optimisation stage is described in Algorithm 2. The required input parameters, properly encoded as an ASP program, are the initial model P, the *sheriffs* function computed by Algorithm 1 and a custom function is_better : $\mathcal{P}(D[A]) \times \mathcal{P}(D[A]) \rightarrow \{\texttt{True}, \texttt{False}\}$. The purpose of the latter is to implement the chosen optimality criterion by taking as input two constraint sets and providing as output a boolean value representing whether the first set is better than the second. If the two sets are not comparable according to the criterion, is_better returns `False`. Indeed, such a function is the expression of the global or partial ordering that the optimality criterion induces on the solutions.

Algorithm 2 Selection of the best solutions according to a custom criterion.

Input: P, *sheriffs* : $L^- \rightarrow \mathcal{P}(D[A]), is_better : \mathcal{P}(D[A]) \times \mathcal{P}(D[A]) \rightarrow \{\texttt{T}, \texttt{F}\}$
Output: $\mathcal{Z}$, i.e. the set of the best solutions

1: **procedure** SELECTION(*sheriffs*, P)
2: $\mathcal{Z} = \varnothing$
3: $L'^- = \{t \in L^- \mid sheriffs(t) \neq \varnothing \wedge t \models P\}$
4: $\mathcal{C} = \bigcup_{t \in L'^-} sheriffs(t)$
5: **for each** $S \subseteq \mathcal{C}$ **s.t.** $\forall t \in L'^-,\ S \cap sheriffs(t) \neq \varnothing$ **do**
6: **if** $(\forall S' \in \mathcal{Z}\ !is_better(S', S))$ **then**
7: $\mathcal{Z} \leftarrow \{S\} \cup \{S' \in \mathcal{Z} \mid !is_better(S, S')\}$
8: **end if**
9: **return** $\mathcal{Z}$
10: **end procedure**

The algorithm starts by computing a set L'^- of all those negative traces that can be excluded by at least a constraint ($sheriffs(t) \neq \varnothing$) and are still instead accepted by the initial model P ($t \models P$). Indeed, albeit from the theoretical point of view we assumed that for each trace $t \in L^-$ there exists at least one Declare constraint in $D[A]$ accepting all positive traces and discarding t, real cases might not fulfil this assumption because the chosen language might not be expressive enough to find a constraint able to reject t while admitting all traces of L^+.

Example 3.5. *Consider for example the case of a chosen language bias D including* EXISTENCE1(X), EXISTENCE2(X), ABSENCE(X), ABSENCE2(X) *and all binary Declare templates. Let $L^+ = \{\texttt{bbb}\}$, $L^- = \{\texttt{bbbb}\}$. In this case, there is no way to distinguish the input sets because existence1/2 and absence1/2 are not enough, and all binary constraints must be used with two distinct activities.*

When it is not possible to exclude a negative trace t, Algorithm 1 returns $sheriffs(t) = \varnothing$, and Algorithm 2 overlooks this case by computing L'^-. The set $\mathcal{C}$ is then built (Line 4) by considering the constraints allowing all traces in L^+ and disallowing at least one trace in L'^- (Line 4). From that, the algorithm selects any subset S fulfilling the condition $\forall t \in L'^-\ S \cap sheriffs(t) \neq \varnothing$ (Line 5), i.e., any S accepting all positive traces and rejecting all the negatives that can be actually excluded.

Any such S is then included in the solution set $\mathcal{Z}$ if the latter does not contain another solution S' that is better than S according to the custom optimality criterion expressed by the is_better operator (Line 6). Solutions S' previously founded are kept into $\mathcal{Z}$ only if the newly found solution S is not better than them (Line 7). Notice that both Line 6 and 7 make use of the function is_better in a negated form: this is due to the fact that, according to the chosen optimality

6. For more details, see the Go package regexp documentation at https://golang.org/pkg/regexp/

7. By leveraging CLINGO function symbols, Declare constraints are encoded as function terms; for example, the constraints derived are represented using the unary predicate holds/1: holds(precedence(a_1,a_2)).

criterion, sometimes it would not be possible to compare two solutions.

If the chosen optimality criterion is *generality*, is_better employs the criteria of Eq. (6a) (for generality) and (6b) (to avoid redundancy). Conversely, if we are interested in the most specific solution, is_better must implement Eq. (7) and (6b). Finally, when the optimality criterion is *simplicity*, Eq. (8a) and (8b) must be used. In our experiments we implemented the optimisation criteria using ASP *weak constraints* [27]. Describing the underlying technique and precise formulation of the problem is outside the scope of this article; so the reader is referred to our code, available in [23], and [24] for details on the usage of weak rules in CLINGO. For example, conditions of Equations (8a) and (8b) are implemented using two predicates selected/1, holds/1; representing, respectively, the selected constraints among the candidates and their deduction, with the optimisation statements:

```
#minimize{1@2,C: holds(C)}.
#minimize{1@1,C: selected(C)}.
```

The selection from the set of candidates is performed by the common *generate and test* ASP paradigm:

```
{ selected(C) : choice(_,C) }.
```

where choice/2 is the input to the ASP program, encoding the above *sheriffs* function. The *test* part of the paradigm is encoded by constraints enforcing the fact that, for each negative trace, at least one of the constraints in *sheriffs* belongs to the deduction; i.e.

```
rejected(T) :- choice(T,C), holds(C).
:- choice(T,_), not rejected(T).
```

In real cases, the returned set $\mathcal{Z}$ might contain several solutions. If the number of solutions provided by the procedure is too high for human intelligibility, the optimality condition could be further refined by inducing a preference order in the returned solution. For example, among the most general solutions one can be interested in being reported first those models with the lower number of constraints, or with certain Declare templates. The advantage of our approach is precisely in the possibility to implement off-the-shelves optimisation strategies, where—adapting the is_better function or even the definition of the closure operator—the developer can easily experiment with different criteria.

Example 3.6. *Consider the sets of positive and negative examples composed by only one trace each:* $L^+ = \{\mathsf{bac}\}$ *and* $L^- = \{\mathsf{ab}\}$. *Suppose also that:*

- $P = \varnothing$;
- $D = \{\mathsf{EXISTENCE(X)}, \mathsf{INIT(X)}\}$;
- *the alphabet of activities is just* $A = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$.

Then, the set of ground constraints can be easily elicited: $D[A] = \{\mathsf{EXISTENCE(a)}, \mathsf{EXISTENCE(b)}, \mathsf{EXISTENCE(c)}, \mathsf{INIT(a)}, \mathsf{INIT(b)}, \mathsf{INIT(c)}\}$. *If we want to learn the most general model, Algorithm 1 elects the following compatible constraints and emits sheriffs*(ab)*:*

$$\begin{aligned} compatibles &= \{\mathsf{EXISTENCE(a)}, \mathsf{EXISTENCE(b)}, \\ &\qquad \mathsf{EXISTENCE(c)}, \mathsf{INIT(b)}\} \\ sheriffs(\mathsf{ab}) &= \{\mathsf{EXISTENCE(c)}, \mathsf{INIT(b)}\}. \end{aligned}$$

In this simple case, the subsets satisfying the condition of Line 5 in Algorithm 2 would be:

$$\begin{aligned} S_1 &= \{\mathsf{EXISTENCE(c)}\} \\ S_2 &= \{\mathsf{INIT(b)}\} \\ S_3 &= \{\mathsf{EXISTENCE(c)}, \mathsf{INIT(b)}\} \\ S_4 &= \{\mathsf{EXISTENCE(c)}, \mathsf{EXISTENCE(a)}\} \\ &\ldots \\ S_n &= \{\mathsf{EXISTENCE(c)}, \mathsf{INIT(b)}, \mathsf{EXISTENCE(a)}\} \\ &\ldots \end{aligned}$$

As we are interested in the most general models, both $S_1 = \{\ \mathsf{EXISTENCE(c)}\ \}$ *and* $S_2 = \{\ \mathsf{INIT(b)}\ \}$ *are optimal solutions. Note that these two solutions cannot be compared according to the definitions of generality because there exist traces (such as the unknown trace* b*) compliant with* S_2 *and non-compliant with* S_1 *(i.e., there is no subset relation between* S_1 *and* S_2*). Obviously, the choice of one model over another influences the classification of all those unknown traces that—being not part of the input log—are not labelled as positive of negative.*

If we were interested in the most simple solution instead, $S_1 = \{\ \mathsf{EXISTENCE(c)}\ \}$ *would have been our choice, because its closure is the smaller in cardinality.*

Finally, if we were interested in the most specific set of constraints, the application of a convenient is_better *function would have determined the choice of* $\{\mathsf{EXISTENCE(a)}, \mathsf{EXISTENCE(c)}, \mathsf{INIT(b)}\}$*—where the redundancy check of Eq.* (6b) *operated by discarding* EXISTENCE(b).

3.4 Complexity considerations

To understand the complexity of the above algorithms we focus on the size of the input logs; i.e., we do not consider the Declare language bias as part of the input. This assumption fits most of the use cases, where the set of Declare templates is fixed. The choice of templates affects the maximum arity of the templates (commonly used templates restrict to 1 or 2 arguments), regular expressions used in Algorithm 1, and the deductive rules; therefore, also the rule part of ASP program can be considered fixed.

The size of the input (n) is therefore the sum of positive and negative log traces. We can also assume that all actions appear in the logs, so the input size bounds also the number of actions. Note that the set $D[A]$ of all grounded templates is linearly bound by the number of actions; in particular $|D[A]| \leq |A|^\delta$, where δ is the maximum arity of the selected Declare templates. This bounds linearly both the complexity of Algorithm 1 and the size of *sheriffs*; the latter because each negative trace is associated to a subset of *compatibles*. In general, we expect that the *sheriffs* function is going to be much smaller than all the traces; note that a larger number of positive traces further restricts the size of *compatibles*. To understand the linear bound of Algorithm 1 we consider that each grounded template must be checked at most once for each trace (both positive and negative), and we use an optimal implementation of regexp that guarantees liner time w.r.t. the input to be checked.

The size of the input to the optimisation stage is the size of *sheriffs* and, since we do not consider the ASP rules as part of the input, the size of the *grounded* ASP program is

linearly bound w.r.t. the size of *sheriffs*. Since we do not use Disjunctive ASP, the complexity of ASP optimisation using weak constraints is Δ_2^P [28].

Characterising the theoretical complexity analysis of the general problem of optimisation in the context of binary process discovery [14] is outside the objectives of this work. However, the fact that verifying that a given trace satisfies a set of Declare constraints can be performed in linear time, suggests that the problem of optimising the choice of a Declare model falls under the general umbrella of NP-complete optimisation problems analysed in [29]. Therefore its complexity should be Δ_2^P as well.

4 Experimental evaluation

One of the difficulties of evaluating process mining algorithms is that given a log, the underlying model might not be known before. As a consequence, it might be difficult to establish an *ideal model* (a golden standard) to refer and confront with. In this regard, a number of metrics and evaluation indexes have been proposed in the past to evaluate how a discovered model fits a given log [4], [5], [6]. However, those metrics might provide only a partial answer to the question of "how good" the discovered model is. In the case of `NegDis`, a further issue influences the evaluation process: the difficulty of performing a "fair" comparison with existing techniques because the majority of the methods we could access have been designed to use "positive" traces only.

We pursued two different evaluation strategies. On one side, we defined a model, and from that model we generated a synthetic, artificial log, taking care that it exhibits a number of desired properties: in a sense, this part of the evaluation can be referred as being about a "controlled setting". A first aim is to understand if `NegDis` succeeds to discover a *minimum* set of constraints for distinguishing positive from negative examples; a second aim is to qualitatively evaluate the discovered model, having the possibility to confront it with the original one. Experiments conducted on that synthetic log are discussed in Section 4.1.

On the other side, we applied `NegDis` to some existing logs, thus evaluating it on some real data set. Again, this experiment has two aims: to understand weakness and strengths of `NegDis` w.r.t. to some relevant literature; and to confront the proposed approach with real-world data—and difficulties that real-world data bring along. Section 4.2 is devoted to present the selected logs and discuss the obtained results. The source code and the experiments are available in [23].

4.1 Experiments on a synthetic dataset

The synthetic log has been generated starting from a Declare model, using a tool [30] based on Abductive Logic Programming. The model has been inspired by the Loan Application process reported in [31]. In our model, the process starts when the *loan application* is received. Before *assessing the eligibility*, the bank proceeds to *appraise the property* of the customer, and to *assess the loan risk*. Then, the bank can either *reject the application* or *send the acceptance pack* and, optionally, *notify the approval* (if not rejected). During the process execution the bank can also *receive positive* or *negative feedback* (but

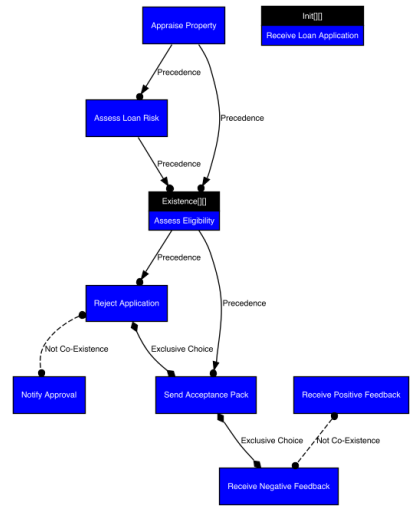

Fig. 1: Loan approval declare process model.

not both), according to the experience of the loan requester. It is not expected, however, that the bank receives a *negative feedback* if the *acceptance pack* has been sent. Moreover, due to temporal optimisation, the bank requires that the *appraisal of the property* is done before *assessing the loan risk*. To ease the understanding of the loan application process, a Declare model of the process is reported in Fig. 1. Moreover, all the activities have been constrained to either not be executed at all, or to be executed at most once: in Declare terminology, all the activities have been constrained to $\mathsf{absence2}(\mathsf{X})$.

To test `NegDis`, besides positive traces, we generated also negative traces. In particular, we generated traces that violate two different constraints:

(a) the $\mathsf{precedence}(\mathsf{assess_loan_risk}, \mathsf{assess_eligibility})$, that is violated when either the latter activity is executed and the former is absent, or if both the activities appear in the log, but in the wrong temporal order;

(b) the $\mathsf{exclusive_choice}(\mathsf{send_acceptance_pack}, \mathsf{receive_negative_feedback})$, that is violated when a trace either contains both the activities, or does not contain any of them.

The resulting log consists of 64,000 positives traces, 25,600 traces that violate the constraint as in (a), and 10,240 traces that violate the constraint as specified in (b). When fed with the positives traces and traces violating the constraint in (a), `NegDis` successfully manages to identify constraints that allow to clearly distinguish positive/negative traces. Moreover, the discovered constraint coincides with the one we originally decided to violate during the generation phase. When confronted with the scenario (b), `NegDis` again successfully managed to identify a minimum model able to discriminate between positive and negative traces, and the identified constraint is indeed logically consistent with the constraint originally selected for the violation. Table 2 summarize the obtained results and reports the best selected model for each scenario. The Time column highlights how the most expensive task is the generation of the $compatibles$ set. This is indeed expected, as it requires to check all constraints in $D[A]$ against all traces in L^+.

For the sake of completeness, we decided to experiment also with the Process Discovery Tool of the RuM

Scenario	Positive Trace #	Negative Trace #	Time	Originally Violated Constraint	Best Discovered Model
(a)	64 000	25 600	**Total: 22.93s** Compatibles: 13.96s *sheriffs*: 7.85s Optimisation: 1.12s	*precedence(assess_loan_risk, assess_eligibility)*	*precedence(assess_loan_risk, assess_eligibility)*
(b)	64 000	10 240	**Total: 17.68s** Compatibles: 13.96s *sheriffs*: 3.4s Optimisation: 0.18s	*exclusive_choice(send_acceptance_pack, receive_negative_ feedback)*	*coExistence(reject_application, receive_negative_feedback)*

TABLE 2: Models discovered when dealing with the synthetic data set.

Framework[8], that is based on the Declare Miner algorithm [32]. Based on the exploitation of positive traces only, Declare Miner discovers a rich model that describes as "most exactly" as possible the given traces. When fed with the positive traces of our artificial log, and with the *coverage* parameter set to 100% (i.e., prefer constraints that are valid for all the traces in the logs), the RuM Framework discovers a model made of 514 constraints. If the coverage is relaxed to 80% (prefer constraints that are satisfied by at least the 80% of the traces), the model cardinality grows up to 1031 constraints.

In both cases the discovered model is able to distinguish between the positive and the negative traces. This is not surprising, since Declare Miner aims to identify all the constraints that hold for a given log: hence, it will discover also those constraints that allow to discern positive from negative traces. Rather, this result is a clear indication that indeed our artificial log has been constructed "correctly", since negative traces differ from the positive ones for some specific constraints, and the positive traces exhaustively elicit the cases that can occur. This is typical of artificial logs, while real-life logs might not enjoy these properties.

Another consideration is about the cardinality of the discovered model: the Declare Miner approach provides a far richer description of the positive traces, at the cost perhaps of bigger models. Our approach instead has the goal of identifying the *smallest* set of constraints that allow to discriminate between positive and negatives. In this sense, approaches like the one presented in this paper and Declare Miner are complementary.

4.2 Evaluation on case studies from real data

For the experimentation with real datasets, we used six real-life event logs: BPIC12, CERV, DREYERS, PROD, SEPSIS and TRAFFIC_FINES. Starting from these event logs we generated 18 different datasets, each composed of a set of positive and a set of negative traces, by applying different criteria to distinguish between positive and negative traces, i.e., by labelling the event log with different labelling functions. All logs have been labeled according to three criteria: the first one is domain dependent and varies for each dataset, while the remaining two datasets (XXX_{mean} and XXX_{median}) for each log are generated based on the execution cycle time. In the XXX_{mean} dataset (resp. XXX_{median}), traces with a cycle time lower than the mean (resp. median) duration of the traces in the event log are labeled as positive; as negative otherwise.[9]

BPIC12 is a real-life logs from the Business Process Intelligence Challenge 2012 [33]. The log pertains to the application process for personal loans or overdrafts in a Dutch financial institute. It merges three intertwined sub-processes. Besides the cycle-time datasets, in the BPIC12_{CANC} dataset, traces have been labeled based on the occurrence of the activity O_CANCELLED [34].

CERV is an event log related to the process of cervical cancer screening carried out in an Italian cervical cancer screening centre [35]. Cervical cancer is a disease in which malignant (cancer) cells form in the tissues of the cervix of the uterus. The screening program proposes several tests in order to early detect and treat cervical cancer. It is usually composed by five phases: Screening planning; Invitation management; First level test with pap-test; Second level test with colposcopy, and eventually biopsy. The traces contained in the event log (CERV_{compl}) have been analysed by a domain expert and labeled as compliant (positive traces) or non-compliant (negative traces) with respect to the cervical cancer screening protocol adopted by the screening centre.

The DREYERS log [36] is an event log documenting the application grant process of the Dreyers Foundation, a Danish foundation supporting budding lawyers and architects. The application requests are traced through an information system and the DREYERS event log also collects the early stages of deployment and the testing phase of the system. The domain-specific labelling for the DREYERS_{reset} dataset is the one used in [14]: it classifies the executions based on whether they were reset due to a system failure (negative traces) or not (positive traces).

The production event log (PROD) [34] contains data from a manufacturing process. Each trace records information about activities, workers and machines involved in the production of items. The labelling for the PROD_{rej} dataset distinguishes between executions with at least a rejected work order (negative traces) and executions with no rejected orders (positive traces) [34].

SEPSIS [37] is an event log that records trajectories of patients with symptoms of the life-threatening sepsis condition in a Dutch hospital. Each case logs events since the patient's registration in the emergency room until her discharge from the hospital. Among others, laboratory tests together with their results are recorded as events. The traces contained in the event log have been labeled based on the formula $\phi_1 =$

8. https://rulemining.org/

9. Since the DREYERS log does not contain timestamps, we synthetically added them for the generation of the DREYERS_{mean} and DREYERS_{median} dataset.

RESPONSE(IVAntibiotics, Leucocytes) ∧ RESPONSE(LacticAcid, IVAntibiotics) ∧ RESPONSE(ERTriage, CRP). The positive traces satisfies ϕ_1, while the negative ones violate the constraint.

Finally, the TRAFFIC fines event log [38], which comes from an Italian local police force, contains events on fine notifications, as well as (partial) repayments. This event log has also been labeled based on whether the fine is finally repaid in full (positive traces) or sent for credit collection (negative traces) [34] (TRAFFIC$_{paid}$ dataset).

Table 3 summarizes the data related to the 18 resulting datasets.

Dataset	Log	Trace #	Activity #	Label	Positive Trace #	Negative Trace #
BPIC12$_{CANC}$	BPIC12	13087	36	O_CANCELLED occurs	2660	10427
BPIC12$_{mean}$				mean duration	8160	4927
BPIC12$_{median}$				median duration	6544	6543
CERV$_{compl}$	CERV	157	16	compliant	55	102
CERV$_{mean}$				mean duration	93	64
CERV$_{median}$				median duration	92	65
DREYERS$_{reset}$	DREYERS	700	33	reset executions	492	208
DREYERS$_{mean}$				mean duration	206	494
DREYERS$_{median}$				median duration	406	294
PROD$_{rej}$	PROD	220	26	reset executions	103	117
PROD$_{mean}$				mean duration	148	72
PROD$_{median}$				median duration	110	110
SEPSIS$_{\phi_1}$	SEPSIS	1050	16	ϕ_1	685	365
SEPSIS$_{mean}$				mean duration	838	212
SEPSIS$_{median}$				median duration	525	525
TRAFFIC$_{paid}$	TRAFFIC	129 615	10	fine paid	70 602	59 013
TRAFFIC$_{mean}$				mean duration	70 585	59 030
TRAFFIC$_{median}$				median duration	65 003	64 612

TABLE 3: Dataset description

The results obtained by applying the NegDis algorithm are summarised in Table 4. The table reports for each dataset, the results related to the SUBSET (connected to the *generality* criterion of Eq. 6a and 6b) and CARDINALITY (*simplicity* criterion of Eq. 8a and 8b) optimizations in terms of number of returned models[10], minimum size of the returned models, as well as percentage of negative traces violated by the returned model. Moreover, the table reports the time required for computing the set of compatibles, the set of sheriffs, as well as the SUBSET and CARDINALITY optimizations.

The table shows that for the CERV$_{compl}$ dataset, NegDis is able to return models that satisfy the whole set of positive traces and violate the whole set of negative traces (the percentage of violated traces in L^- is equal to 100%) with a very low number of constraints (4). A similar result is obtained also with the BPIC12$_{CANC}$ dataset (with a minimum model of a single constraint) and with the DREYERS$_{mean}$ dataset (with a minimum model composed of 3 constraints). For the other datasets, the returned models are always able to satisfy all traces in L^+, however not all the negative traces are violated by the returned models. For some of the datasets, as for instance, the DREYERS$_{reset}$ and the DREYERS$_{median}$, the percentage of violated traces is quite close to 100% (98.56% and 96.94%, respectively), with relatively small models (8 and 14 constraints, respectively). For other datasets, as the BPIC12$_{mean}$, SEPSIS$_{\phi 1}$, SEPSIS$_{mean}$, TRAFFIC$_{paid}$, TRAFFIC$_{mean}$ and TRAFFIC$_{median}$, the percentage of violated traces is relatively small (4.25% for SEPSIS$_{mean}$ and less than 1% for all the other datasets), as the number of constraints of the returned models (ranging from 1 for TRAFFIC$_{paid}$ to 13 for BPIC12$_{mean}$, TRAFFIC$_{mean}$ and TRAFFIC$_{median}$. However, while for SEPSIS$_{mean}$ and BPIC12$_{mean}$ the number of L^- traces violated by the model returned by NegDis is relatively low (around 49 and 4, respectively), in case of the TRAFFIC log, NegDis returns a model able to violate around 600 traces in L^-. Moreover, NegDis is able to obtain reasonable results also with the other real datasets. For the datasets related to the PROD event log it returns slightly larger minimum models (including about 21-30 constraints) able to to accept all traces in L^+ and to violate from around 40% to 80% of L^- traces. In the case of BPIC12$_{median}$ and SEPSIS$_{median}$, the models discovered with NegDis are instead able to violate around 30% of the traces in L^-. Overall, the inability to identify a set of constraints that is able to fulfil all traces in L^+ and to violate all negative ones is due to a bias of the considered language (Declare without data) that does not allow to explain the positive traces without the negative ones. A language bias issue could also be the cause of the huge difference in terms of percentage of L^- violated traces obtained with the mean and the median cycle time for some of the datasets (e.g., BPIC12 and SEPSIS). Indeed, when the positive and negative trace sets are quite balanced (i.e., for BPIC12$_{median}$ and SEPSIS$_{median}$) NegDis is able to identify a set of constraints (related to the control flow) describing the traces with a low-medium cycle time and excluding the ones with a medium-high cycle time; when, instead, the sets of the positive and the negative traces are quite imbalanced (i.e., for BPIC12$_{mean}$ and SEPSIS$_{mean}$) characterizing the high number of traces with a low or medium cycle time while excluding the ones with a very high cycle time can become hard.

The table also shows that NegDis is overall very fast for small datasets with SUBSET (e.g., less than 30 seconds for CERV, PROD and SEPSIS), while it requires some more time for the large ones (e.g., BPIC12 and TRAFFIC) and the ones with a large alphabet (e.g., DREYERS). While the time required for computing *compatibles* and *sheriffs* seems to be related to the size and complexity of the dataset, the time required for computing the optimizations especially the CARDINALITY seems to depend also on other characteristics of the datasets.

Compared to state-of-the-art techniques for the discovery of declarative models starting from the only positive traces, NegDis is able to return a small number of constraints satisfying all traces in L^+ without decreasing the percentage of violated traces in L^-. Among the classical declarative discovery approach, we selected the state-of-the-art DeclareMiner algorithm [32] implemented in the RuM toolkit [39]. We discovered the models using the only positive traces and setting the support parameter, which measures the percentage of (positive) traces satisfied by the Declare model, to 100%[11].

10. We stop generating models after 20 models, i.e., max in Table 4 indicates that more than 20 models have been returned within the timeout (one hour).

11. We run the DeclareMiner algorithm with vacuity detection disabled, activity support filter set to 0%, using both transitive closure and hierarchy-based reduction of the discovered constraints, as well as with the whole set of Declare templates.

Dataset	SUBSET			CARDINALITY			Required Time (s)			
	Number of models	Min model size	Violated L^- trace %	Number of models	Min model size	Violated L^- trace %	Comp.	*sheriffs*	SUBSET	CARDINALITY
BPIC12$_{CANC}$	max	1	100%	2	1	100%	4.92	12.39	22.15	4
BPIC12$_{mean}$	max	13	0.91%	max	13	0.91%	2.83	5.87	19.99	0.08
BPIC12$_{median}$	max	23	32.71%	max	23	32.71%	2.13	7.24	6.59	54.24
CERV$_{compl}$	max	4	100%	max	4	100%	0.4	0.06	19.24	0.05
CERV$_{mean}$	max	2	100%	max	2	100%	0.01	0.91	10.86	7.79
CERV$_{median}$	max	1	100%	1	1	100%	0.01	0.84	11.8	0.5
DREYERS$_{reset}$	max	8	98.56%	4	8	98.56%	1.62	0.1	78.03	0.12
DREYERS$_{mean}$	max	4	100%	max	4	100%	0.6	2.56	23.77	462.09
DREYERS$_{median}$	max	14	96.94%	1	14	96.94%	1.25	1.07	81.16	timeout
PROD$_{rej}$	max	21	40.17%	1	21	40.17%	0.8	0.41	7.03	timeout
PROD$_{mean}$	max	21	81.94%	1	21	81.94%	0.71	0.66	8	timeout
PROD$_{median}$	max	30	78.18%	1	30	78.18%	0.56	0.93	12.31	timeout
SEPSIS$_{\phi_1}$	3	3	0.82%	2	3	0.82%	0.67	0.32	0.37	0.05
SEPSIS$_{mean}$	2	8	4.25%	2	8	4.25%	0.72	0.26	0.4	0.06
SEPSIS$_{median}$	max	14	26.86%	max	14	26.86%	0.46	0.62	2.99	0.12
TRAFFIC$_{paid}$	1	1	0.01%	1	1	0.01%	20.51	20.82	0.26	0.07
TRAFFIC$_{mean}$	max	13	0.76%	2	13	0.76%	18.98	23.4	5.63	0.05
TRAFFIC$_{median}$	max	13	0.75%	4	13	0.75%	15.35	28.03	2.02	0.04

TABLE 4: `NegDis` results on the real-life logs

Table 5 summarises the obtained results[12]. The table reports for each dataset, the size of the model in terms of number of constraints, as well as the percentage of negative traces violated by the model. For lower values of the support parameter, i.e., for a lower percentage of positive traces satisfied by the model, the model returned by the `DeclareMiner` violates a higher percentage of negative traces. In this way, the support parameter allows for balancing the percentage of positive traces satisfied and negative traces violated.

As hypothesised, the optimisation mechanism in `NegDis` is able to identify a small set of constraints, that guarantees the satisfaction of all traces in L^+ and the same or higher percentage of negative trace violations obtained with `DeclareMiner` (with support to 100%).

Dataset	Model size	Violated L$^-$ trace %
BPIC12$_{CANC}$	795	100%
BPIC12$_{mean}$	515	0.91%
BPIC12$_{median}$	532	32.55%
CERV$_{compl}$	323	100%
CERV$_{mean}$	6	0%
CERV$_{median}$	2	0%
DREYERS$_{compl}$	909	98.56%
DREYERS$_{mean}$	425	100%
DREYERS$_{median}$	798	96.94%
PROD$_{rej}$	377	36.75%
PROD$_{mean}$	269	79.17%
PROD$_{median}$	241	77.27%
SEPSIS$_{\phi_1}$	211	0.82%
SEPSIS$_{mean}$	213	4.25%
SEPSIS$_{median}$	203	22.29%
TRAFFIC$_{paid}$	67	0.01%
TRAFFIC$_{mean}$	84	0.76%
TRAFFIC$_{median}$	86	0.75%

TABLE 5: `DeclareMiner` results

Finally, we evaluated the results obtained with `NegDis` relying on the same procedure and dataset (CERV$_{compl}$) used in [35] to assess the results of `DecMiner`, a state-of-the-art declarative discovery approach based on Inductive Logic Programming (ILP) that is able to use both positive and negative execution traces. Five fold-cross validation is used, i.e., the CERV$_{compl}$ dataset is divided into 5 folds and, in each experiment, 4 folds are used for training and the remaining one for validation purposes. The average *accuracy* of the five executions is collected, where the accuracy is defined as the sum of the number of positive (compliant) traces that are (correctly) satisfied by the learned model and the number of negative (non-compliant) traces that are (correctly) violated by the learned model divided by the total number of traces.

Table 6 reports the obtained accuracy values for the `DecMiner`, the `DeclareMiner` (with the support parameter set to 100%) and the `NegDis` (both for the CARDINALITY and SUBSET optimization) approach. The table shows that on this specific dataset, `NegDis` and `DecMiner` have very close performance (with `NegDis` CARDINALITY performing slightly better). `DeclareMiner` presents instead on average a slightly lower accuracy mainly due to the highly constrained discovered model that, on one hand, allows for violating all negative traces in the validation set, and, on the other hand, leads to the violation of some of the positive traces in the validation set.

5 RELATED WORK

Process discovery is generally considered a challenging task of process mining [40]. The majority of works in this field are focused on discovering a process model from a set of input traces that are supposed compliant with it. In this

Approach	Accuracy
`DecMiner`	97.44%
`DeclareMiner`	96.79%
`NegDis` (SUBSET)	97.38%
`NegDis` (CARDINALITY)	97.57%

TABLE 6: Accuracy results obtained with `DeclareMiner`, `DecMiner` and `NegDis`

12. Note that the results obtained with DeclareMiner and NegDis are not fully comparable. Indeed, the discovery mechanisms behind the two tools are different, and different is even the information that the tools take as input. DeclareMiner takes as input only the positive log and bases the discovery on the positive log alphabet. Instead, NegDis uses information from both positive and negative logs, including the alphabet of the discovered model that, in general, is not the same as the one retrieved from the positive log only.

sense, process discovery can be seen as the application of a machine learning technique to extract a grammar from a set of positive sample data. Angluin et al. [41] provide an interesting overview on this wide field. Differently from grammar learning, where the model is often expressed with automata, regular expressions or production rules, process discovery usually adopts formalisms that can express concurrency and synchronization in a more understandable way [12]. The language to express the model is a crucial point, which inevitably influences the learning task itself. Indeed, the two macro-categories of business process discovery approaches—procedural and declarative—differ precisely by the type of language to express the model. Well known examples of procedural process discoverers are the ones presented in the works [7], [8], [9], [10], [42], [43], [44]. See [45], [46] for systematic literature reviews of this field. Like most procedural approaches, all these works contemplate the presence of non-informative noise in the log, which should be separated from the rest of the log and disregarded.

Traditional declarative approaches to process discovery stem from the necessity of a more friendly language to express loosely-structured processes. Indeed—as also pointed out by [40]—process models are sometimes less structured than one could expect, so procedural discovery could produce spaghetti-models. In that case, a declarative approach is more suitable to briefly list all the required or prohibited behaviours in a business process. Similarly to our technique, the one exposed by Maggi et al. in [47] starts by considering the set of all activities in the log and building a set of all possible candidate Declare constraints. The performance of this technique is improved in [40], while [48] proposes a model refinement to efficiently exclude vacuously satisfied constraints.

The MINERFul approach described in [49] proposes to employ four metrics to guide the declarative discovery approach: support, confidence and interest factor for each constraint w.r.t. the log, and the possibility to include in the search space constraints on prohibited behaviours. Particularly relevant for our purposes is the work by Di Ciccio et al. [22], who focus on refining Declare models to remove the frequent redundancies and inconsistencies. The algorithms and the hierarchy of constraints described in that work were particularly inspiring to define our discovery procedure. Similarly to the procedural approaches, all the declarative ones described so far do not deal with negative examples, although the vast majority of them envisage the possibility to discard a portion of the log by setting thresholds on the value of specific metrics that the discovered model should satisfy. In the work [50] the authors propose a declarative miner for learning Dynamic Condition Response (DCR) graphs from event logs. Similarly to our approach, they focus on control-flow-related aspects only, and disregard timing, data and resource perspectives. Negative examples are not exploited in the learning process, but taken into account when measuring underfitting and overfitting of the resulting model.

In the wider field of grammar learning, the foundational work by Gold [51] showed how negative examples are crucial to distinguish the right hypothesis among an infinite number of grammars that fit the positive examples. Both positive and negative examples are required to discover a grammar with perfect accuracy. Since process discovery does not usually seek perfection, but only a good performance according to defined metrics, it is not surprising that many procedural and declarative discoverers disregard the negative examples. Nonetheless, in this work we instead claim that negative traces are extremely important when learning declarative process models.

The information contained in the negative examples is actively used in a subset of the declarative process discovery approaches [13], [14], [52], [53], [54]. All these works can be connected to the basic principles of the Inductive Constraint Logic (ICL) algorithm [55], whose functioning principle is intrinsically related to the availability of both negative and positive examples. The Declarative Process Model Learner (DPML) described in [35], [52] by Lamma et al. focuses on learning integrity constraints expressed as logical formulas. The constraints are later translated into an equivalent construct of the declarative graphical language DecSerFlow [56]. Similarly to this approach, the DecMiner tool described in [13], learns a set of Social Constrained if-and-only-if (SCIFF) rules [57] which correctly classify an input set of labelled examples. Such rules are then translated into ConDec constraints [58]. Differently from [13], the present work directly learns Declare constraints without any intermediate language. DPML has been later used in [53] to extract integrity constraints, then converted into Markov Logic formulas. Taking advantage of the negative examples, the approach of [53] is improved in [54]. Another recent related work is the Rejection Miner by Slaats et al. [14], which—analogously to our vision—considers process discovery as a binary classification task and provides an algorithm where some parts can be customised to include more constraints and different model minimisation strategies. Likewise, our approach envisages the possibility to employ a different language bias or closure operator. Since the aim of [14] is to provide a general miner (not limited to Declare notation), they do not focus on notation-specific improvements like our use of the closure to avoid redundancies and provide more compact models. Interestingly, [14] also reports a demonstration of the reasons why Declare is not able to perfectly separate any pair of positive/negative traces. Consistently with this finding, our approach aims at discovering a model that accepts all traces in $L+$ and rejects as many traces in L^- as possible. Despite its limits, we believe that Declare's diffusion in both industry and academia motivates the need to further investigate process discovery with this language.

Since for all these works the availability of negative examples is crucial, recent years have seen the development of synthetical log generators able to produce not only positive but also negative process cases [12], [30], [59], [60], [61], [62]. In the experimental evaluation of this work, we employ the abductive-logic generator by Loreti et al. [30] to synthesise part of the input event logs.

Two-class declarative process discovery is sometimes related to sequence mining and classification [15], [63], [64], [65], [66]. In [63] the authors propose a probabilistic machine learning approach that infers subsequences which best compress a sequence database. A similar Bayesian technique is used by Egho et al. [64] to mine Standard Classification

Rule Models, where the antecedent is the relevant subsequence, and the consequent expresses the support of that subsequence in each class. The above-mentioned work by Maggi et al. [47] originated from the idea that Apriori-like approaches can discover local patterns in a log, but not rules representing prohibited behaviours and choices. The same motivation is behind the work by De Smedt et al. [15], which significantly enriches the expressiveness of the discovered models employing Declare behavioural constraints in sequence classification. Differently from [15], our work does not focus on characterising the traces of multiple, possibly intersected, input classes, but rather on refining the relevant feature of the traces belonging to a single class (the positive). In order to do so, we assume two disjunct sets of examples and suggest extracting from one class (the negative) the information relevant to characterise the other (the positive). In other words, the information that is relevant for our purpose is the fact that a trace *does not belong* to a certain class. Another important difference regards the mining algorithm: while [15] explicitly considers each relevant Declare constraint, we envisage the language bias, as well as the subsumption hierarchy as input and we exploit the power of a satisfiability-based solver to determine the best model in terms of generality or simplicity.

Particularly related to our approach are the works by Neider et al. [67], Camacho et al. [68], and Reiner [69] where a satisfiability-based solver is employed to learn a simple set of LTL formulas consistent with an input data set of positive and negative examples. In particular, Neider et al. [67] employ decision tree to improve the performance and manage large example sets; Camacho et al. [68] exploit the correspondence of LTL formulae with Alternating Finite Automata (AFA); whereas Reiner [69] uses partial Directed Acyclic Graphs (DAGs) to decompose the search space into smaller subproblems. Furthermore, the concept of negative example used in this work could be related to both the definitions of syntactical and semantic noise of [70].

It is important to underline that also a limited number of procedural approaches envisage the need for taking into account the information contained into the negative examples. In particular, the work [71] showed how negative examples can be employed to alleviate problems like log incompleteness and noise. The Artificially Generated Negative Events (AGNEs) tool described in [12] increases the dimension of an event log with artificially generated negative examples, then uses ILP multi-relational classification to discover a Petri net model. Negative examples are generated in a rather syntactical way, by adding a unique negative event at the end of a positive trace. This concept is extended in the work of Ponce de Leon et al. [6] which envisage the concatenations of a sequence of negative events. Differently from these approaches, our technique does not assume syntactical restrictions on the input negative examples. ILP is also used in [72], where the authors suppose a set of negative examples provided by domain experts. The approach uses partial-order planning to discover a structured model.

Deviant cases—intended as traces whose sequence of activities deviates from the expected behaviour—are the subject of deviance mining approaches reviewed and evaluated by Nguyen et al. in [17]. Some applications of deviance mining tend to highlight the differences between models discovered from deviant and non-deviant traces [73], [74]. Other works intend deviance mining as a classification task, where the miner is required to identify normal and deviant traces given a set of examples. The classification inherently causes the discovery of patterns which distinguish different types of traces. In this sense, deviance mining is particularly similar to sequence classification. The discovered patterns can be based on the simple frequency of individual activities as in [75], [76], their co-occurrence as in [77], or the occurrence of specific subsequences [25], [78], [79]. Table 7 briefly illustrate the classification of the cited related works.

Procedural	One-class discovery	[7], [8], [9], [10], [42], [43], [44], [46]
	Two-class discovery	[6], [71], [72]
	Log generation	[12], [61], [62]
Declarative	One-class discovery	[22], [40], [47], [48], [49], [50]
	Two-class discovery	[13], [14], [35], [52], [53], [54], [67], [68], [69]
	Log generation	[30], [59], [60]
Deviance mining		[17], [25], [73], [74], [75], [76], [77], [78], [79]
Sequence mining/classification		[15], [63], [64], [65], [66]

TABLE 7: Classification of related works.

6 Conclusion

While the vast majority of works see process discovery as a one-class supervised learning task, we embrace the less popular view of process discovery as a binary supervised learning job, where traces representing a "stranger" behaviour can be considered as heralds of valuable information about the process itself. Devoted to this vision, we developed a technique which considers both positive and negative traces and performs process discovery as a satisfiability problem, where different heuristics can be adopted to assess the optimal model according to different goals.

Being able to extract valuable knowledge from negative examples, our proposal `NegDis` can be employed to enrich the process description extracted by a state-of-the-art declarative process discovery algorithm. However, it is worth to underline that the resulting declarative process discoverer taking advantage of explicitly defined positive and negative examples would not be necessarily an alternative to procedural discovery techniques. Indeed in some cases, when correct thresholds and language biases are adopted, procedural discoverers have the great advantage to provide the user with a rather easy-to-understand definition of the process model. Nonetheless, the informative content provided by those process cases that are discarded by procedural discoverer can still be extremely important. For the future, as `NegDis` extracts valuable information from L^- without excluding any trace of L^+, it could also be applied as a post processing technique to enrich the output of procedural discoverers with declarative constraints. The

resulting output would be an hybrid procedural/declarative process model, showing a simple and handy structured representation of the main business process together with a set of declarative constraints. The goal of such constraints would be to account for less frequent deviances or prohibited behaviours in a much more synthetic and easy-to-understand way with respect to an equivalent spaghetti-like procedural formulation.

Such a hybrid solution could also greatly simplify the elicitation of long-term dependencies between activities that occur at the beginning of the process and those carried out towards the end. Indeed, the structured nature of procedural approaches makes them not properly suitable to express such dependencies. One current way to tackle such issue is through the employment of global variables and "if" statements to control the execution flow of each instance. For example, Kalenkova et al. [80] propose a process discovery technique devoted to repair free-choice procedural workflows with additional modelling constructs, which can more easily capture non-local dependencies. Nonetheless, since such additional constraints are intended to preserve the procedural nature of the model, the result may increase its complexity and ultimately affect its readability. An hybrid procedural/declarative model formulation would maintain a structured form to express the model while integrating it with handy declarative long-term constraints involving activities occurring far from each other in the workflow. This idea of a hybrid procedural/declarative model formulation has been explored by various works and proved to be particularly effective in the field of medical clinical guidelines [81], [82], [83]. A wider landscape of applications is considered by Maggi et al. in the work [3].

Finally, the performance of `NegDis` presented in this paper could be boosted through a parallel approach. Analogously to previous works [32], [84], [85] we can envisage two possible directions to decompose our task: by spitting the model (i.e. in this case the set of constraints to be learned), or the input data (i.e. the business log). The algorithm presented here could easily adopt the first kind of partitioning, whereas the second might be more challenging.

ACKNOWLEDGMENTS

This work was partially supported by the European Commission funded projects "Humane AI: Toward AI Systems That Augment and Empower Humans by Understanding Us, our Society and the World Around Us" (grant # 820437) and "AI4EU: A European AI On Demand Platform and Ecosystem" (grant # 825619). The support is gratefully acknowledged.

REFERENCES

[1] van der Aalst, et al., "Process mining manifesto," in *Business Process Management Workshops*, F. Daniel, K. Barkaoui, and S. Dustdar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 169–194.

[2] D. Fahland, D. Lübke, J. Mendling, H. A. Reijers, B. Weber, M. Weidlich, and S. Zugal, "Declarative versus imperative process modeling languages: The issue of understandability," in *BMMDS/EMMSAD*, ser. LNBIP, vol. 29. Springer, 2009, pp. 353–366.

[3] F. M. Maggi, A. Marrella, G. Capezzuto, and A. Armas-Cervantes, "Explaining non-compliance of business process models through automated planning," in *ICSOC*, ser. Lecture Notes in Computer Science, vol. 11236. Springer, 2018, pp. 181–197.

[4] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst, "Measuring precision of modeled behavior," *Inf. Syst. E Bus. Manag.*, vol. 13, no. 1, pp. 37–67, 2015.

[5] S. K. L. M. vanden Broucke, J. D. Weerdt, J. Vanthienen, and B. Baesens, "Determining process model precision and generalization with weighted artificial negative events," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1877–1889, 2014.

[6] H. P. de León, L. Nardelli, J. Carmona, and S. K. L. M. vanden Broucke, "Incorporating negative information to process discovery of complex systems," *Inf. Sci.*, vol. 422, pp. 480–496, 2018.

[7] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.

[8] A. J. M. M. Weijters and W. M. P. van der Aalst, "Rediscovering workflow models from event-based data using little thumb," *Integr. Comput. Aided Eng.*, vol. 10, no. 2, pp. 151–162, 2003.

[9] C. W. Günther and W. M. P. van der Aalst, "Fuzzy mining - adaptive process simplification based on multi-perspective metrics," in *BPM*, ser. Lecture Notes in Computer Science, vol. 4714. Springer, 2007, pp. 328–343.

[10] W. M. P. van der Aalst, V. A. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software and Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2010.

[11] L. Maruster, A. J. M. M. Weijters, W. M. P. van der Aalst, and A. van den Bosch, "A rule-based approach for process discovery: Dealing with noise and imbalance in process logs," *Data Min. Knowl. Discov.*, vol. 13, no. 1, pp. 67–87, 2006.

[12] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, "Robust process discovery with artificial negative events," *J. Mach. Learn. Res.*, vol. 10, pp. 1305–1340, 2009.

[13] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari, "Exploiting inductive logic programming techniques for declarative process mining," *Trans. Petri Nets Other Model. Concurr.*, vol. 2, pp. 278–295, 2009.

[14] T. Slaats, S. Debois, and C. O. Back, "Weighing the Pros and Cons: Process Discovery with Negative Examples," in *Business Process Management*, A. Polyvyanyy, M. T. Wynn, A. Van Looy, and M. Reichert, Eds. Cham: Springer International Publishing, 2021, pp. 47–64.

[15] J. D. Smedt, G. Deeva, and J. D. Weerdt, "Mining behavioral sequence constraints for classification," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 6, pp. 1130–1142, 2020.

[16] M. Pesic, "Constraint-based workflow management systems : shifting control to users," Ph.D. dissertation, Industrial Engineering & Innovation Sciences, 2008.

[17] H. Nguyen, M. Dumas, M. L. Rosa, F. M. Maggi, and S. Suriadi, "Business process deviance mining: Review and evaluation," *CoRR*, vol. abs/1608.08252, 2016.

[18] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, "Xes, xesame, and prom 6," in *CAiSE Forum (Selected Papers)*, ser. Lecture Notes in Business Information Processing, vol. 72. Springer, 2010, pp. 60–75.

[19] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Comput. Sci. Res. Dev.*, vol. 23, no. 2, pp. 99–113, 2009.

[20] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari, "Declarative specification and verification of service choreographiess," *ACM Trans. Web*, vol. 4, no. 1, pp. 3:1–3:62, 2010.

[21] G. D. Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, F. Rossi, Ed. IJCAI/AAAI, 2013, pp. 854–860.

[22] C. D. Ciccio, F. M. Maggi, M. Montali, and J. Mendling, "Resolving inconsistencies and redundancies in declarative process models," *Inf. Syst.*, vol. 64, pp. 425–446, 2017.

[23] S. Tessaris, C. Di Francescomarino, and F. Chesani, "Code for Negdis experiments," Aug. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5158528

[24] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot ASP solving with clingo," *Theory and Practice of Logic Programming*, vol. 19, no. 1, pp. 27–82, Jan. 2019, publisher: Cambridge University Press. [Online]. Available: https://www.cambridge.

org/core/journals/theory-and-practice-of-logic-programming/article/abs/multishot-asp-solving-with-clingo/FAED3429900D84CDD5155326A36548F2

[25] M. L. Bernardi, M. Cimitile, C. Di Francescomarino, and F. M. Maggi, "Do activity lifecycles affect the validity of a business rule in a business process?" *Inf. Syst.*, vol. 62, pp. 42–59, 2016.

[26] V. Lifschitz, "What is answer set programming?" in *AAAI*. AAAI Press, 2008, pp. 1594–1597.

[27] F. Buccafurri, N. Leone, and P. Rullo, "Enhancing Disjunctive Datalog by constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 5, pp. 845–860, Sep. 2000.

[28] ——, "Enhancing Disjunctive Datalog by constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 5, pp. 845–860, Sep. 2000.

[29] M. W. Krentel, "The complexity of optimization problems," *Journal of Computer and System Sciences*, vol. 36, no. 3, pp. 490–509, Jun. 1988. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022000088900396

[30] D. Loreti, F. Chesani, A. Ciampolini, and P. Mello, "Generating synthetic positive and negative business process traces through abduction," *Knowl. Inf. Syst.*, vol. 62, no. 2, pp. 813–839, 2020.

[31] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[32] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Inf. Syst.*, vol. 74, no. Part, pp. 136–152, 2018.

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

[34] I. Teinemaa, M. Dumas, M. L. Rosa, and F. M. Maggi, "Outcome-oriented predictive process monitoring: Review and benchmark," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 2, pp. 17:1–17:57, 2019. [Online]. Available: https://doi.org/10.1145/3301300

[35] E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari, "Inducing declarative logic-based models from labeled traces," in *Business Process Management*, G. Alonso, P. Dadam, and M. Rosemann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 344–359.

[36] S. Debois and T. Slaats, "The analysis of a real life declarative process," in *IEEE Symposium Series on Computational Intelligence, SSCI 2015, Cape Town, South Africa, December 7-10, 2015*. IEEE, 2015, pp. 1374–1382.

[37] F. Mannhardt, "Sepsis cases - event log," Dec 2016. [Online]. Available: https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639/1

[38] M. M. de Leoni and F. Mannhardt, "Road Traffic Fine Management Process," 2 2015. [Online]. Available: https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249

[39] 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*, B. F. van Dongen, M. Montali, and M. T. Wynn, Eds. IEEE, 2020, pp. 121–128.

[40] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst, "Efficient discovery of understandable declarative process models from event logs," in *CAiSE*, ser. Lecture Notes in Computer Science, vol. 7328. Springer, 2012, pp. 270–285.

[41] D. Angluin and C. H. Smith, "Inductive inference: Theory and methods," *ACM Comput. Surv.*, vol. 15, no. 3, pp. 237–269, 1983.

[42] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs - A constructive approach," in *Petri Nets*, ser. Lecture Notes in Computer Science, vol. 7927. Springer, 2013, pp. 311–329.

[43] Q. Guo, L. Wen, J. Wang, Z. Yan, and P. S. Yu, "Mining invisible tasks in non-free-choice constructs," in *BPM*, ser. Lecture Notes in Computer Science, vol. 9253. Springer, 2015, pp. 109–125.

[44] A. Augusto, R. Conforti, M. Dumas, and M. L. Rosa, "Split miner: Discovering accurate and simple business process models from event logs," in *ICDM*. IEEE Computer Society, 2017, pp. 1–10.

[45] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 4, pp. 686–705, 2019.

[46] I. Yürek, D. Birant, and K. Birant, "Interactive process miner: a new approach for process mining," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 26, no. 3, pp. 1314–1328, 2018. [Online]. Available: https://journals.tubitak.gov.tr/elektrik/vol26/iss3/16

[47] F. M. Maggi, A. J. Mooij, and W. M. P. van der Aalst, "User-guided discovery of declarative process models," in *CIDM*. IEEE, 2011, pp. 192–199.

[48] D. M. M. Schunselaar, F. M. Maggi, and N. Sidorova, "Patterns for a log-based strengthening of declarative compliance models," in *IFM*, ser. Lecture Notes in Computer Science, vol. 7321. Springer, 2012, pp. 327–342.

[49] C. D. Ciccio, M. H. M. Schouten, M. de Leoni, and J. Mendling, "Declarative process discovery with minerful in prom," in *BPM (Demos)*, ser. CEUR Workshop Proceedings, vol. 1418. CEUR-WS.org, 2015, pp. 60–64.

[50] C. O. Back, T. Slaats, T. T. Hildebrandt, and M. Marquard, "DisCoveR: accurate and efficient discovery of declarative process models," *Int. J. Softw. Tools Technol. Transfer*, 2021.

[51] E. M. Gold, "Language identification in the limit," *Inf. Control.*, vol. 10, no. 5, pp. 447–474, 1967.

[52] E. Lamma, P. Mello, F. Riguzzi, and S. Storari, "Applying inductive logic programming to process mining," in *ILP*, ser. Lecture Notes in Computer Science, vol. 4894. Springer, 2007, pp. 132–146.

[53] E. Bellodi, F. Riguzzi, and E. Lamma, "Probabilistic logic-based process mining," in *CILC*, ser. CEUR Workshop Proceedings, vol. 598. CEUR-WS.org, 2010.

[54] ——, "Statistical relational learning for workflow mining," *Intell. Data Anal.*, vol. 20, no. 3, pp. 515–541, 2016.

[55] L. D. Raedt and W. V. Laer, "Inductive constraint logic," in *ALT*, ser. Lecture Notes in Computer Science, vol. 997. Springer, 1995, pp. 80–94.

[56] W. M. P. van der Aalst and M. Pesic, "Decserflow: Towards a truly declarative service flow language," in *WS-FM*, ser. Lecture Notes in Computer Science, vol. 4184. Springer, 2006, pp. 1–23.

[57] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, "Verifiable agent interaction in abductive logic programming: The SCIFF framework," *ACM Trans. Comput. Log.*, vol. 9, no. 4, pp. 29:1–29:43, 2008.

[58] M. Pesic and W. M. P. van der Aalst, "A declarative approach for flexible business processes management," in *Business Process Management Workshops*, ser. Lecture Notes in Computer Science, vol. 4103. Springer, 2006, pp. 169–180.

[59] F. Chesani, C. Di Francescomarino, C. Ghidini, D. Loreti, F. M. Maggi, P. Mello, M. Montali, V. Skydanienko, and S. Tessaris, "Towards the generation of the "perfect" log using abductive logic programming," in *CILC*, ser. CEUR Workshop Proceedings, vol. 2396. CEUR-WS.org, 2019, pp. 179–192.

[60] F. Chesani, A. Ciampolini, D. Loreti, and P. Mello, "Abduction for generating synthetic traces," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, vol. 308. Springer, 2017, pp. 151–159.

[61] T. Stocker and R. Accorsi, "Secsy: A security-oriented tool for synthesizing process event logs," in *BPM (Demos)*, ser. CEUR Workshop Proceedings, vol. 1295. CEUR-WS.org, 2014, p. 71.

[62] K. M. van Hee and Z. Liu, "Generating benchmarks by random stepwise refinement of petri nets," in *ACSD/Petri Nets Workshops*, ser. CEUR Workshop Proceedings, vol. 827. CEUR-WS.org, 2010, pp. 403–417.

[63] J. M. Fowkes and C. Sutton, "A subsequence interleaving model for sequential pattern mining," in *KDD*. ACM, 2016, pp. 835–844.

[64] E. Egho, D. Gay, M. Boullé, N. Voisine, and F. Clérot, "A user parameter-free approach for mining robust sequential classification rules," *Knowl. Inf. Syst.*, vol. 52, no. 1, pp. 53–81, 2017.

[65] M. J. Zaki, "Sequence mining in categorical domains: Incorporating constraints," in *CIKM*. ACM, 2000, pp. 422–429.

[66] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders, "Mining compressing sequential patterns," *Stat. Anal. Data Min.*, vol. 7, no. 1, pp. 34–52, 2014.

[67] D. Neider and I. Gavran, "Learning linear temporal properties," in *FMCAD*. IEEE, 2018, pp. 1–10.

[68] A. Camacho and S. A. McIlraith, "Learning interpretable models expressed in linear temporal logic," in *ICAPS*. AAAI Press, 2019, pp. 621–630.

[69] H. Riener, "Exact synthesis of LTL properties from traces," in *FDL*. IEEE, 2019, pp. 1–6.

[70] C. W. Günther, "Process mining in flexible environments," Ph.D. dissertation, Technische Universiteit Eindhoven, 2009.

[71] H. P. de León, C. Rodríguez, J. Carmona, K. Heljanko, and S. Haar, "Unfolding-based process discovery," in *ATVA*, ser. Lecture Notes in Computer Science, vol. 9364. Springer, 2015, pp. 31–47.

[72] H. M. Ferreira and D. R. Ferreira, "An integrated life cycle for workflow management based on learning and planning," *Int. J. Cooperative Inf. Syst.*, vol. 15, no. 4, pp. 485–505, 2006.
[73] S. Suriadi, R. Mans, M. T. Wynn, A. Partington, and J. Karnon, "Measuring patient flow variations: A cross-organisational process mining approach," in *AP-BPM*, ser. Lecture Notes in Business Information Processing, vol. 181. Springer, 2014, pp. 43–58.
[74] A. Armas-Cervantes, P. Baldan, M. Dumas, and L. García-Bañuelos, "Behavioral comparison of process models based on canonically reduced event structures," in *BPM*, ser. Lecture Notes in Computer Science, vol. 8659. Springer, 2014, pp. 267–282.
[75] S. Suriadi, M. T. Wynn, C. Ouyang, A. H. M. ter Hofstede, and N. J. van Dijk, "Understanding process behaviours in a large insurance company in australia: A case study," in *CAiSE*, ser. Lecture Notes in Computer Science, vol. 7908. Springer, 2013, pp. 449–464.
[76] A. Partington, M. T. Wynn, S. Suriadi, C. Ouyang, and J. Karnon, "Process mining for clinical processes: A comparative analysis of four australian hospitals," *ACM Trans. Management Inf. Syst.*, vol. 5, no. 4, pp. 19:1–19:18, 2015.
[77] J. Swinnen, B. Depaire, M. J. Jans, and K. Vanhoof, "A process deviation analysis - A case study," in *Business Process Management Workshops (1)*, ser. Lecture Notes in Business Information Processing, vol. 99. Springer, 2011, pp. 87–98.
[78] R. P. J. C. Bose and W. M. P. van der Aalst, "Discovering signature patterns from event logs," in *CIDM*. IEEE, 2013, pp. 111–118.
[79] D. Lo, S. Khoo, and C. Liu, "Efficient mining of iterative patterns for software specification discovery," in *KDD*. ACM, 2007, pp. 460–469.
[80] A. A. Kalenkova, J. Carmona, A. Polyvyanyy, and M. L. Rosa, "Automated repair of process models using non-local constraints," in *Petri Nets*, ser. Lecture Notes in Computer Science, vol. 12152. Springer, 2020, pp. 280–300.
[81] A. Bottrighi, F. Chesani, P. Mello, M. Montali, S. Montani, S. Storari, and P. Terenziani, "Analysis of the GLARE and GPROVE approaches to clinical guidelines," in *KR4HC*, ser. Lecture Notes in Computer Science, vol. 5943. Springer, 2009, pp. 76–87.
[82] A. Bottrighi, F. Chesani, P. Mello, G. Molino, M. Montali, S. Montani, S. Storari, P. Terenziani, and M. Torchio, "A hybrid approach to clinical guideline and to basic medical knowledge conformance," in *AIME*, ser. Lecture Notes in Computer Science, vol. 5651, 2009, pp. 91–95.
[83] A. Bottrighi, F. Chesani, P. Mello, M. Montali, S. Montani, and P. Terenziani, "Conformance checking of executed clinical guidelines in presence of basic medical knowledge," in *Business Process Management Workshops (2)*, ser. Lecture Notes in Business Information Processing, vol. 100. Springer, 2011, pp. 200–211.
[84] D. Loreti, F. Chesani, A. Ciampolini, and P. Mello, "A distributed approach to compliance monitoring of business process event streams," *Future Gener. Comput. Syst.*, vol. 82, pp. 104–118, 2018.
[85] D. Loreti, M. Lippi, and P. Torroni, "Parallelizing machine learning as a service for the end-user," *Future Gener. Comput. Syst.*, vol. 105, pp. 275–286, 2020.

Federico Chesani , Ph.D. in Computer Science, is associate professor of Computer Science at DISI—University of Bologna. His research activities focus on the area of Logic Programming, Business Processes Modelling, Distributed Verification and Monitoring, and Rule-based Decision Support Systems.

Chiara Di Francescomarino is a researcher at Fondazione Bruno Kessler in the Process and Data Intelligence Unit. She received her PhD in Information and Communication Technologies from the University of Trento, working on business process modeling and reverse engineering from execution logs. She is currently working in the field of process mining, investigating problems related to process monitoring, process discovery, as well as predictive process monitoring. She has published papers in the top business process conferences and journals and she has worked in local and international research projects.

Chiara Ghidini is a Senior Research Scientist at Fondazione Bruno Kessler where she leads the Process & Data Intelligence Research Unit. Her scientific work in the areas of Semantic Web, Knowledge Engineering and Representation, Multi-Agent Systems and Process Mining is internationally well known and recognised, and she has actively been involved in the organisation of several reference conferences in these areas. She is a board member of the Italian Association for Artificial Intelligence and is involved in the scientific coordination of the Centre for Digital Health and Well Being of Head of Fondazione Bruno Kessler.

Daniela Loreti is junior assistant professor of Operating Systems at Department of Computer Science and Engineering, University of Bologna. She received her Ph.D. in Computer Science in 2016. Her research focuses on distributed systems for big data management and stream processing as well as parallel paradigms for high performance computing. She is also interested in the parallelization of artificial intelligence techniques in the fields of machine learning, process mining and expert systems.

Fabrizio Maria Maggi is Associate Professor at the KRDB Research Centre for Knowledge and Data. He devises techniques grounded in business process management, process mining, predictive analytics and his work is specifically focused on the application of artificial intelligence and machine learning in the context of business process analytics. On these topics, he authored more than 130 papers, appeared in top-tier, international journals and conferences. He is recipient of 4 best paper awards, two of which in the International Conference on Business Process Management.

Paola Mello is Full Professor at the University of Bologna, where she conducts research, both practical and theoretical in Artificial Intelligence. In particular, her research activity is about knowledge representation, computational logic, multi-agent and decision support systems, with applications in medicine, web services, business processes, monitoring and verification. She has been President of the Italian Association for Artificial Intelligence and Head of the Department of Computer Science and Engineering of the University of Bologna. She is fellow of the European Association for Artificial Intelligence.

Marco Montali is Full Professor and Vice-Dean for Studies at the Faculty of Computer Science, Free University of Bozen-Bolzano, Italy, where he also coordinates the MSc Program in Computational Data Science. He investigates foundational and applied techniques grounded in artificial intelligence and formal methods for the model- and data-driven analysis of business processes and multiagent systems. He is member of the Steering Committee of the IEEE Task Force on Process Mining. He is co-author of more than 200 papers, many of which in top-tier conferences and journals, and recipient of 8 best paper awards.

Sergio Tessaris received the PhD degree in computer science from the University of Manchester (UK). He is Assistant Professor at the Faculty of Computer Science of the Free University of Bozen-Bolzano (Italy). His research interests include the use of Semantic Technologies and Knowledge Representation formalisms to support the management, access, and exploitation of data and processes.