# Shape Your Process: Discovering Declarative Business Processes from Positive and Negative Traces Taking into Account User Preferences

Federico Chesani[1], Chiara Di Francescomarino[2], Chiara Ghidini[2], Giulia Grundler[1], Daniela Loreti[1], Fabrizio Maria Maggi[3], Paola Mello[1], Marco Montali[3], Sergio Tessaris[3]

[1] DISI - University of Bologna, Italy
[2] Fondazione Bruno Kessler, Trento, Italy
[3] Free University of Bozen/Bolzano, Italy

**Abstract.** Process discovery techniques focus on learning a process model starting from a given set of logged traces. The majority of the discovery approaches, however, only consider one set of examples to learn from, i.e., the log itself. Some recent works on declarative process discovery, instead, advocated the usefulness of taking into account two different sets of traces (a.k.a. positive and negative examples), with the goal of learning a set of constraints that is able to discriminate which trace belongs to which set. Sometimes, however, too many possible sets of constraints might be available, thus nullifying the discovery effort. Therefore, some preference criteria would be helpful to guide the discovery process towards a set of constraints among the many. In this work, we present an approach for the discovery of declarative models providing the possibility, from the user viewpoint, of specifying preferences on activities and constraint templates to be used to build the final set of constraints. Such preferences are used to guide the discovery process, so that the output set will include, if possible, the preferred constraints, thus exploiting some expert knowledge about the desired outcome. The approach is grounded in a logic-based framework that provides a sound and formal meaning to the notion of expert preferences.

## 1 Introduction

*Process discovery* is one of the most investigated process mining techniques [43]. 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.

If we focus on the way process discovery techniques see the model-extraction task, we can divide them into two broad categories. The first category is constituted by works that tackle the problem of process discovery with one-class supervised learning techniques (see, e.g., [3,4,23,44,1]). These works are driven by the assumption that all available log traces are instances of the process to be discovered and constitute the wast majority of works in the process discovery spectrum. The second category comprises works that 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 criteria. Usually, these sets are referred to as *positive* and *negative*

examples, and the goal is to learn a model that characterizes one set w.r.t. the other. These works are traditionally less represented (see [16,21,31]). Nonetheless, few recent works [15,26,39] have highlighted the importance of performing model-extraction as a two-class supervised task with different motivations: first, the actual existence of *positive* and *negative* examples in real use cases [26,39]; second, the need to balance *accuracy* and *recall* [39]; and third, the need to discover a particular process variant (e.g., the process characterizing "fast" traces) against the one that characterizes other variants, thus using the labels *positive* and *negative* to distinguish between two classes of examples [15]. Hereafter, we refer to miners of the first and second category as *unary* and *binary* miners, respectively.

A problem that remains unsolved in process discovery, in general, and in binary miners, in particular, is the need to select, among all possible discovered models, the ones that fit better the expectations of expert users, that is, users who are knowledgeable about the specific domain and because of this have specific desiderata and expectations. This is true for the traditional discovery of procedural and declarative models, where the discovered model that accepts all the positive examples is usually too complex (e.g., too spaghetti like), and mechanisms are introduced to "select" specific behaviors. Examples of criteria for this selection can be the frequency of a certain element (e.g., an activity or a path), or the presence of certain modeling patterns (e.g., a specific declarative pattern). The problem becomes even more compelling when we approach process discovery as a two-class supervised task. In fact, as recently shown in [39], perfect binary miners, able to discover models that accept all positive examples and none of the negative examples, do not necessarily exist. In such cases, many sub-optimal models can be returned, leading to the issue of identifying criteria for preferring one model or the other.

In this paper, we address the problem of inserting expert user preferences, (hereafter expert preferences) in the discovery of declarative process models as a two-class supervised task. We start from a recent work [15] that introduces the `NegDis` binary miner for the `Declare` modeling language [36] (introduced in Section 2). Being `NegDis` based on the logic-based framework ASP [12], it provides a formal framework with a clear semantics that allows the users to "prioritize" the discovery results. In this work, we extend `NegDis` by introducing the *ASPrin* tool [11], so as to support the notion of expert preferences while remaining within the context of a formal, logic-based semantics. The following contributions are provided:

(i) we introduce and motivate two types of expert preferences: the first one on the `Declare` patterns to be used in the discovery task, and the second one on the activities appearing in the output model. Moreover, we discuss also a third type of preference coming from the combination of the first two (Section 3).
(ii) we extend the original mechanism of `NegDis` (Section 4), by incorporating the *ASPrin* tool [11] into it. This allows us to integrate within a single framework both the expert preferences, as well as the original `NegDis` mechanism based on *model subsumption* (that is treated as a preference as well). In this way, we retain the original ability of obtaining models that vary in generality/specificity, or simplicity.
(iii) we provide some hints about the implementation (Section 5);
(iv) we report on exploratory experiments applying an instantiation of `NegDis` to the data sets used in [24,39] (Section 6).

Related works (Sect. 7) and final considerations (Sect. 8) conclude this work.

## 2 The modeling language

The discovery approach we introduce in this paper is based on `Declare`, a language for describing declarative process models first introduced in [36]. A `Declare` model consists of a set of constraints applied to (atomic) activities. Constraints are, in turn, based on templates. Templates are abstract parameterized patterns and constraints are their concrete instantiations on real activities. Templates have a graphical representation and their semantics can be formalized using different logics, the main one being LTL for finite traces, making them verifiable and executable. Each constraint inherits the graphical representation and semantics from its template. The major benefit of using templates is that analysts do not have to be aware of the underlying logic-based formalization to understand the models. They work with abstract representations of templates, while the underlying formulas remain hidden. Table 1 summarizes the main `Declare` constructs used in this paper. The reader can refer to [36] for a full description of the language.

| **Template** | **Explanation** |
|---|---|
| existence(n, A) | A occurs at least $n$ times |
| absence(m + 1, A) | A occurs at most $m$ times |
| responded_existence(A, B) | If A occurs, then B occurs |
| response(A, B) | If A occurs, then B occurs after A |
| alternate_response(A, B) | Each time A occurs, then B occurs afterwards, before A recurs |
| precedence(A, B) | B occurs only if preceded by A |
| chain_precedence(A, B) | Each time B occurs, then A occurs immediately before |
| co_existence(A, B) | If B occurs, then A occurs, and vice versa |
| not_succession(A, B) | A never occurs before B |
| not_chain_succession(A, B) | A and B occur if and only if the latter does not immediately follow the former |

Table 1: `Declare` templates

## 3 Why preferences on the discovered models?

Users look for discovering models for a variety of reasons. A common one is related to the need of having a description/explanation of a process. Other reasons might be, for example, the need for detecting process deviations or process drifts. Or, as in the case of `NegDis`, expert users might be interested in understanding, from a model viewpoint, what distinguishes one set of traces from another.

Depending on the discovery technique and the target language, many alternative models might describe the same process. For example, both BPMN and `Declare` allow us to describe the same process using different constructs or templates. However, not all the discovered models are equivalent[4], and even when they are equivalent, there could be too many models to choose from.

[4] Roughly speaking, two models are *equivalent* if they accept and reject the same traces. Such a notion of equivalence hints to the possibility that given two models $M_1$ and $M_2$, opting for the former or the latter will not change which traces will be accepted or rejected.

Since the availability of many models might, in turn, hinder the usefulness of the discovery approach, the expert user would need a criterion for selecting few models among the many discovered. Preferences on the discovered models represent then a way for prioritizing the discovered models based on the expert user's needs. In particular, we envisage three different types of preferences: preferences over activities, preferences over templates, and a combination of both.

### 3.1 Preferences over process activities

A first type of preferences on the discovered models is strictly related to the application domain. Indeed, depending on the expert user's goals, models that focus more on certain activities might be preferable.

*Example 1.* Let us consider a "loan scenario", where a bank receives a request for a loan, evaluates it, and provides an answer. Let us assume that process instances have been classified into two sets, for example including successful and unsuccessful applications. The bank employee will look then for a model that helps her to understand the differences. Of course, the employee will not directly look into the logs, which, for simplicity, we can suppose to be as follows:

$$L^+ = \{\ \langle \mathsf{loanRequest}, \mathsf{requestEval}, \mathsf{notifyOutcome} \rangle\ \}$$
$$L^- = \{\ \langle \mathsf{requestEval}, \mathsf{loanRequest} \rangle\ \}$$

where the positive example set $L^+$ contains only one trace (composed of three activities), and the negative example set $L^-$ contains a single trace as well.
If the employee is an employee working in the marketing department, she could have in mind the bank slogan "we always answer our customers". Hence, she would be surely interested in the notifyOutcome activity. By specifying such preference, the discovery algorithm would return two models both involving the preferred activity[5]:

$$M_1 = \{\mathsf{response}(\mathsf{requestEval}, \mathsf{notifyOutcome})\}$$
$$M_2 = \{\mathsf{existence}(\mathsf{notifyOutcome})\} \quad \square$$

Generally speaking, being able to specify a preference for models that refer to specific activities allows expert users to answer the question *"Is it possible to discriminate between two sets of traces by looking at certain activities?"*. The discovery process becomes, in this way, domain-driven: many models describe the process, but those ones that focus on certain domain aspects should be returned before others.

### 3.2 Preferences over Declare templates

Process description languages like, e.g., BPMN and `Declare`, are quite rich in their expressiveness, and allow us to describe a process using different constructs or templates. This leads to the availability of alternative models that could be equivalent or not. Unfortunately, even when restricting our attention to equivalent models only, it is easy to see that they might not convey the information in exactly the same way to users.

[5] Other models exist, of course, but, for the sake of clarity, we only mention two of them.

*Case 1: Equivalent models.* Let us consider first the case where a discovery algorithm provides as output two equivalent models. If from a "conformance viewpoint" nothing changes, from a high-level viewpoint different models might bear subtle meaning distinctions, as shown in the following example.

*Example 2.* Let us assume to have the following log, whose traces have been classified into two sets:

$$L^+ = \{ \langle \mathsf{a}, \mathsf{b} \rangle, \langle \mathsf{b}, \mathsf{a} \rangle \} \qquad L^- = \{ \langle \mathsf{a} \rangle, \langle \mathsf{b} \rangle \}$$

Alternative models allowing us to represent the traces that belong to $L^+$ and exclude the ones that belong to $L^-$ are:

$$\begin{aligned} M_1 &= \{\mathsf{existence}(\mathsf{a}), \mathsf{existence}(\mathsf{b})\} \\ M_2 &= \{\mathsf{existence}(\mathsf{a}), \mathsf{responded\_existence}(\mathsf{a}, \mathsf{b})\} \\ M_3 &= \{\mathsf{existence}(\mathsf{b}), \mathsf{responded\_existence}(\mathsf{b}, \mathsf{a})\} \\ M_4 &= \{\mathsf{existence}(\mathsf{a}), \mathsf{co\_existence}(\mathsf{a}, \mathsf{b})\} \\ M_5 &= \{\mathsf{existence}(\mathsf{b}), \mathsf{co\_existence}(\mathsf{a}, \mathsf{b})\} \end{aligned}$$

From a logical viewpoint, models $M_1$–$M_5$ are equivalent. However, models $M_2$–$M_5$ suggest that what distinguishes the traces in $L^+$ from the traces in $L^-$ is a relation between activities a and b: indeed, these models contain a binary constraint, whose purpose is to highlight a relation between these two activities. Model $M_1$, instead, does not tell us anything about possible links between activities a and b, and a user might conclude that no relation exists between them. □

`Declare` binary templates, by their nature, suggest a link between activities. Hence, a discovery algorithm that would return models with relation constraints would emphasize such links. The user would be left with the burden of understanding if such links are mere coincidences or artifacts of the discovery technique, or if rather some new knowledge has been discovered about the process.

We can imagine scenarios where expert users prefer models containing the minimum number of binary templates, so as not to incur into the risk of perceiving in-existent relations. On the other hand, we can easily think about situations where an expert user is actively looking for relations. In both cases, preferences on which `Declare` templates should be preferably included into a model would allow the expert user to tailor the discovery process to her needs.

Notice also that Example 2 might mislead the reader to think that preferences over templates is a matter of unary vs. binary constraints only. This is not the case, since equivalence is a logic property that stems from the interplay between all the constraints within each single model. Models with many binary constraints might be proved to be equivalent, as shown in the following example.

*Example 3.* Let us consider the following log:

$$L^+ = \{ \langle \mathsf{a}, \mathsf{b} \rangle, \langle \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle, \langle \mathsf{a}, \mathsf{c}, \mathsf{b} \rangle \} \qquad L^- = \{ \langle \mathsf{a} \rangle, \langle \mathsf{a}, \mathsf{c} \rangle \}$$

Two alternative models that accept the positive traces and reject the negative ones are:

$$M_1 = \{\mathsf{absence2(a)}, \mathsf{response(a,b)}\} \quad M_2 = \{\mathsf{absence2(a)}, \mathsf{alternate\_response(a,b)}\}$$

Models $M_1$ and $M_2$ are equivalent due to the interplay of the constraint absence2 with the the response and the alternate_response constraints: roughly speaking, being activity a forbidden to appear more than once, the effects of the stricter constraint alternate_reponse are nullified. □

*Case 2: Non-equivalent models.* Let us consider now the case where alternative non-equivalent models are discovered. This might happen because a log is usually a partial view of all the possible execution traces. Not-yet-seen traces are *unknown* w.r.t. the classification, but different models could classify them in different manners. Different models would *shape the unknown* differently.

*Example 4.* Let us consider the following log:

$$L^+ = \{\ \langle \mathsf{a,b}\rangle,\ \langle \mathsf{b,a}\rangle\ \} \qquad\qquad L^- = \{\ \langle \mathsf{a}\rangle\ \}$$

Alternative models that accept the traces in $L^+$ and discard the ones in $L^-$ are:

$$M_1 = \{\mathsf{existence(a)}, \mathsf{existence(b)}\} \qquad M_2 = \{\mathsf{responded\_existence(a,b)}\}$$

Let us consider then the trace $\langle \mathsf{b}\rangle$, that was not recorded in the log. Model $M_1$ would reject it, whereas model $M_2$ would accept it. □

Example 4 shows how traces not appearing in the log used for the discovery might be classified differently by the discovered models. A preference elicitation mechanism would allow the expert user to decide how the not-yet-seen traces would be classified, in a restricting or in a broader way. Another example is given below.

*Example 5.* Let us consider the following log:

$$L^+ = \{\ \langle \mathsf{a,b}\rangle,\ \langle \mathsf{a,b,c}\rangle, \langle \mathsf{a,c,b}\rangle\ \} \qquad\qquad L^- = \{\ \langle \mathsf{a}\rangle, \langle \mathsf{a,c}\rangle\ \}$$

Two non-equivalent models that accept the positive examples and reject the negative ones are:

$$M_1 = \{\mathsf{response(a,b)}\} \qquad\qquad M_2 = \{\mathsf{alternate\_response(a,b)}\} \qquad \square$$

Both models in Example 5 suffice to classify a trace into one or the other class. However, model $M_2$ is *stricter*, since it accepts less traces and rejects more traces than $M_1$. A expert user might express her preference for stricter or more general models.

### 3.3 Preferences over both activities and templates

The third type of preferences on the discovered models is a straightforward combination of the preference types introduced in Subsections 3.1 and 3.2. Domain-related knowledge would drive the attention to certain activities, and preferences over templates would allow focusing on certain relation types.

*Example 6.* Let us consider again the "loan scenario" and the log:

$$L^+ = \{\ \langle \mathsf{loanRequest}, \mathsf{requestEval}, \mathsf{notifyOutcome} \rangle\ \}$$
$$L^- = \{\ \langle \mathsf{requestEval}, \mathsf{loanRequest} \rangle\ \}$$

Let us consider now the viewpoint of an employee working in the internal auditing department. Given that the wrong execution order of certain activities might be a symptom of some fraud, the employee would like to focus the attention over templates of type response and/or precedence, and, in particular, over those constraints involving the requestEval activity. The discovery algorithm would exploit such preference by looking for models with the elicited features, and would provide in output:

$$M = \{\mathsf{precedence}(\mathsf{requestEval}, \mathsf{loanRequest})\}$$ □

Notice that Example 6 shares the exact same log as Example 1. However, the output is completely different: the preferences are used, indeed, to guide the search for a model, which is of interest for the expert user.

## 4 Discovering Business Processes from Positive & Negative Traces

Our approach is based on the `NegDis` binary miner [15], which, given two input sets of positive and negative examples, aims at extracting a model accepting all positive traces and rejecting all negative ones. In this work, we enrich `NegDis` with the possibility to express domain-dependent preferences on the discovered models. Therefore, we report some definitions and explanations from [15] that are useful to understand our approach.

`NegDis` starts from a certain *language bias*: given a set of `Declare` templates $D$ and a set of activities $A$, we indicate with $D[A]$ the set of all possible groundings of templates in $D$ w.r.t. $A$, i.e., all the constraints that can be built using activities in $A$.

We respectively denote with $L^+$ and $L^-$ the sets of positive and negative examples in the input event log. `NegDis` starts by considering a, possibly empty, initial model $P$, that is a set of `Declare` constraints known to characterize the examples in $L^+$. The goal of `NegDis` is to refine $P$ taking into account both the positive and the negative examples.

**Definition 1.** *Given the initial model $P$, a candidate solution for the discovery task is any set of constraints $S \subseteq D[A]$ s.t. (i) $P \subseteq S$; (ii) $\forall t \in L^+$ we have $t \models S$; (iii) $S$ maximizes the set $\{t \in L^- \mid t \not\models S\}$.*

`Declare` templates can be organized into a hierarchy of *subsumption* [19] according to the logical implications derivable from their semantics. Consistently with this concept, we introduce the following definition of *generality* relation between models.

**Definition 2.** *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 $t' \in A^*$ s.t. $t' \not\models M'$ and $t' \models M$.*

`NegDis` integrates the *subsumption* rules introduced in [19], into the *deductive closure operator*.

**Definition 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 $R$.*

For brevity, in the rest of the paper, we will omit the set $R$ and we will simply write $cl(M)$ to indicate the deductive closure of $M$. The complete set of employed deduction rules is available in the source code [42].[6]

Conceptually, the `NegDis` approach can be seen as a two-step procedure: in the first step, a set of candidate constraints is built, and then solutions are selected among subsets of candidates via an optimization algorithm. The set of candidate constraints is composed of those in $D[A]$ that accept all positive examples and reject at least a negative one. To build this set, `NegDis` constructs a *compatibles* set, i.e., the set of constraints that accept all traces in $L^+$:

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

Then, it defines the *sheriffs* function to associate to any trace $t$ in $L^-$ the constraints of *compatibles* that reject $t$:

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

The *sheriffs* function is used to construct the set of all candidate constraints from which a discovered model is derived, i.e., the set $\mathcal{C} = \bigcup_{t \in L^-} \textit{sheriffs}(t)$ of all the constraints in $D[A]$ accepting all positive traces and rejecting at least one negative trace. The solution space is therefore:

$$\mathcal{Z} = \{M \in \mathcal{P}(\mathcal{C}) \mid \forall t \in L^-\ t \not\models M \cup P \text{ or } \textit{sheriffs}(t) = \emptyset\} \tag{3}$$

Due to the fact that not all the pairs of negative and positive sets of traces can be perfectly separated using `Declare` [39], there can be traces in $L^-$ for which the *sheriffs* is empty, meaning that those traces cannot be excluded by any model that guarantees the acceptance of all the positive ones.

The second step of `NegDis` uses an optimization strategy to identify the solutions; in [15], two different criteria were taken into account: *generality* (or conversely, *specificity*), and *simplicity*. If the user is interested in the most general model, then `NegDis` employs the closure operator $cl$ to select the models $S \in \mathcal{Z}$ with the less restrictive behavior. If the user wants the simplest model, `NegDis` looks for the solutions with minimal closure size. In case of ties, the solution with the minimal size is preferred.

## 5 Adding preferences to process discovery: an implementation through *ASPrin*

### 5.1 Specifying the preferences

As discussed in Section 3, in this work we support three different types of preferences. Preferences over domain activities are simply expressed through ASP Prolog facts of the type:

[6] The file `declare_rules.txt` can be found in the`data` directory.

```
good_action(X).
```

where `X` is a placeholder for an activity name. The intended meaning is that models containing `Declare` constraints about the the action `X` should be preferred to models that do not contain it.

Analogously, to specify a preference for a `Declare` template, we simply add a sentence of the type:

```
good_constraint(X).
```

where `X` now is a placeholder for a template name. Again, the intended meaning is about preferring models containing the specified templates, to other models.

Finally, to express a preference that is a combiantion of the previous types, we allow the user to write facts like:

```
good_constraint_action(decl(Template, Action1 [, Action2])).
```

where `Template` is a placeholder for the `Declare` template name, `Action1` is the placeholder for the activity name and, in case the preferred template is a binary one, `Action2` is the placeholder for the second activity. It is worthy to mention that it is possible to express several preferences at the same time: the intended meaning is that models satisfying more preferences are preferable to models that satisfy less preferences.

### 5.2 Exploiting *ASPrin* for searching preferred models

At a first glance, one could think that the *sheriffs* function in Eq. 2 includes all that we need to generate "preferred" models. Indeed, a naive idea would be to select exactly one constraint from each $\mathit{sheriffs}(t) \neq \emptyset$ for $t \in L^-$ according to some preferences. However, this solution does not take into account the interplay among constraints. In particular, some constraints might be more general than others, or even there might be cases in which two constraints imply the validity of a third one. This would clearly interfere with the validity of the specified preferences.

For this reason, we cannot use any combinatorial optimizer to enforce the preferences, but we need a system enabling some form of constraint propagation. In [15], we use Answer Set Programming (ASP) by leveraging the underlying rule based formalism enabling propagation, and *weak constraints* for optimization [12,20]. The encoding of the optimization problem follows the *Generate and Test* ASP paradigm where part of the rules select a candidate ASP model (e.g., a subset of $\mathcal{C}$) and a set of constraints filters only the relevant models (e.g., those "rejecting" all the negative examples). Weak constraints are used to assign a preference value to any ASP model, i.e., a violated weak constraint does not reject the model but assigns a penalty to it. In [15], simple weak constraints were used to implement subsumption preferences; however, specifying more complex preferences between ASP models (like the ones presented so far) using weak constraints would become unmanageable and error-prone.

To tackle this issue, in this work, we exploit the *ASPrin* tool [11], which layers upon the *clingo* ASP solver [20], enabling the specification of complex preference relations through user-defined types and their arguments. *ASPrin* provides a general framework for optimizing qualitative and quantitative preferences in ASP. While *ASPrin* comes

with a library of predefined preference types (subset, pareto, lexicographic, etc.), it is readily extensible by new customized preference types. Preferences can be defined and aggregated by means of higher level types, making *ASPrin* the perfect tool to support the preferences introduced in Section 3. Moreover, *ASPrin* provides a simple way to implement the "generality" and "simplicity" criteria discussed in Section 4.

Describing the *ASPrin* language and the precise encoding of the optimization problem is outside the scope of this paper (the full code is available in [42] while a detailed description of how we encoded the process discovery problem using the *ASPrin* framework is available in [14]). However, in abstract wording, we use two different predicates, which are explicitly represented by means of their template and activities as predicate arguments. This enables the characterization of the ASP models, e.g., by prioritizing those in which specific templates and/or activities are selected or excluded. These preferences can be combined with domain-independent preferences, e.g., on the size of the discovered models to provide a fine-grained ordering among them.

## 6 Evaluating the discovery

In Section 3, we introduced the preference types through simple toy-like examples. The interested reader, however, might wonder about the usability and efficacy of our approach when applied to real-life cases. We explored the applicability of our approach using two real-life event logs, namely DREYERS (492 positive traces and 208 negative ones) and CERV (55 positive traces and 102 negatives traces).[7] In both cases, we were able to find ten models satisfying the given preferences in a computation time between 1 and 3 seconds, using a normally-equipped laptop.

### 6.1 The DREYERS log

The DREYERS log describes the Dreyer Foundation's processes pertaining to their support to legal and architectural projects, and it has been used in [17,39]. Each application to request the Foundation's support goes through a pre-screen that can lead to an initial rejection. The remaining applications undergo a review, in which at least one of the reviewers must be a lawyer or an architect, depending on the application type. The review phase is followed by a board meeting, where applications to be supported are selected and eventually funded. Two sets of log traces are available in the dataset: a positive one collecting executions that did not fail and a negative one representing executions that were reset due to a system failure.

Using this dataset, we played a sort of "investigation game", and explored the hypothesis that the type of application (architect- or lawyer- type) might affect the process outcome. To this end, we initially specified a preference `good_action(Lawyer Review)`, and later on a preference `good_action(Architect Review)`. In both cases, more than one model satisfying the preferences were found. However, the two sets of models are identical (except for the architect/lawyer activity), showing that the process

[7] For reproducibility purposes the source code is available in [42], the DREYERS event log can be found in [17,39], while the CERV event log is a proprietary dataset.

| Constraints | Traces # | Variants # |
|---|---|---|
| alternateresponse(Undo payment, First payout) | 2 | 2 |
| chainprecedence(Fill out application, Initial Rejection) | 3 | 2 |
| choice(Round ends, Change phase to Abort) | 195 | 17 |
| notchainsuccession(Receive final report, First payout) | 1 | 1 |
| notchainsuccession(Change phase to Preparation, Approve application) | 1 | 1 |
| notchainsuccession(Change phase to Preparation, Execute Pre decision) | 2 | 2 |
| notchainsuccession(Set to Pre approved, Round Ends) | 2 | 2 |
| notsuccession(Architect Review, Approval on to the board) | 1 | 1 |
| Traces not ruled out by the model | 3 | 2 |
| Total | 208 | 30 |

Table 2: Traces ruled out by each constraint of model $M_1$.

is independent of the application domain. We report an example of a model obtained when specifying the preference for models containing activity Architect Review:

$$
\begin{aligned}
M_1 = \{\ & \mathsf{alternateresponse}(\mathsf{Undo\ payment}, \mathsf{First\ payout}) \\
& \mathsf{chainprecedence}(\mathsf{Fill\ out\ application}, \mathsf{Initial\ Rejection}) \\
& \mathsf{choice}(\mathsf{Round\ ends}, \mathsf{Change\ phase\ to\ Abort}) \\
& \mathsf{notchainsuccession}(\mathsf{Receive\ final\ report}, \mathsf{First\ payout}) \\
& \mathsf{notchainsuccession}(\mathsf{Change\ phase\ to\ Preparation}, \mathsf{Approve\ application}) \\
& \mathsf{notchainsuccession}(\mathsf{Change\ phase\ to\ Preparation}, \mathsf{Execute\ Pre\ decision}) \\
& \mathsf{notchainsuccession}(\mathsf{Set\ to\ Pre\ approved}, \mathsf{Round\ Ends}) \\
& \mathsf{notsuccession}(\mathsf{Architect\ Review}, \mathsf{Approval\ on\ to\ the\ board})\ \}
\end{aligned}
$$

Notably, as shown in Table 1, this model is able to discriminate between positive and negative examples except for three negative traces (two variants), that cannot be ruled out without discarding also some positive examples.

We continued our investigation by focusing on activity Initial Rejection. We report here one of the returned models:

$$
\begin{aligned}
M_2 = \{\ & \mathsf{absence2}(\mathsf{Initial\ rejection}) \\
& \mathsf{choice}(\mathsf{Round\ Ends}, \mathsf{Applicant\ informed}) \\
& \mathsf{notchainsuccession}(\mathsf{Set\ to\ Pre\ approved}, \mathsf{Round\ Ends}) \\
& \mathsf{notchainsuccession}(\mathsf{Receive\ final\ report}, \mathsf{First\ payout}) \\
& \mathsf{notchainsuccession}(\mathsf{Change\ phase\ to\ Preparation}, \mathsf{Approve\ application}) \\
& \mathsf{notchainsuccession}(\mathsf{Change\ phase\ to\ Preparation}, \mathsf{Execute\ Pre\ decision}) \\
& \mathsf{notsuccession}(\mathsf{Lawyer\ Review}, \mathsf{Change\ phase\ to\ review}) \\
& \mathsf{response}(\mathsf{Undo\ payment}, \mathsf{First\ payout})\ \}
\end{aligned}
$$

Model $M_2$ highlights the fact that some negative traces can be distinguishable from the positive ones because of the repetition of Initial Rejection: some traces, indeed, reported the execution of the activity twice, thus indicating an attention point for the process analyst.

Finally, we did compare the effect of discovering models with or without the two preferred activities. For this we asked `NegDis` to extract 10 optimal models with no preferences, 10 with the Architect Review preference and 10 with the Initial Rejection preference, and we pairwise compared the models with no preference and the ones with a preferred activity. When imposing no preferences, activity Architect Review shows up in only 4 of the 10 discovered models. Imposing the usage of Initial Rejection is instead "unnecessary" (a posteriori), as this activity is also present in all 10 models discovered without specifying any preference.

### 6.2 Evaluation on the CERV log

CERV is an event log that describes a process pertaining to the cervical cancer screening in an Italian screening center, and it has been used in previous works [24,15]. The screening program is composed of five phases, organized sequentially: screening planning, invitation management, first level test with pap-test, second level test with colposcopy (only if the first test is positive), and eventually biopsy (if the second test gives a positive response). Several subjects do not respect the planned protocol: e.g., subjects might not show up at the first test, even if they have chosen a time slot. Moreover, a number of subjects prefer to consult physicians they trust more, in case of a positive response. As it commonly happens in socio-technical systems, a large variety of process instances appear in the log, not all them being compliant with the protocol. Hence, the traces have been labeled by a domain expert as belonging either to the positive or the negative set, depending on their compliance with the adopted protocol.

We investigated the log by eliciting two preferences over the precedence and succession templates:

```
good_constraint(precedence).
good_constraint(succession).
```

The first two returned models are:

$$
\begin{aligned}
M_1 = \{\ & \mathsf{alternateresponse}(\mathsf{send\ positive\ pap\ test\ result}, \mathsf{take\ a\ colposcopy\ examination}) \\
& \mathsf{chainprecedence}(\mathsf{invite}, \mathsf{take\ a\ pap\ test\ examination}) \\
& \mathsf{exclusivechoice}(\mathsf{send\ pap\ test\ sample}, \mathsf{reject}) \\
& \mathsf{precedence}(\mathsf{send\ colposcopy\ uncertain\ result}, \mathsf{send\ biopsy\ sample})\ \}
\end{aligned}
$$

$$
\begin{aligned}
M_2 = \{\ & \mathsf{alternateresponse}(\mathsf{send\ positive\ pap\ test\ result}, \mathsf{take\ a\ colposcopy\ examination}) \\
& \mathsf{chainprecedence}(\mathsf{invite}, \mathsf{take\ a\ pap\ test\ examination}) \\
& \mathsf{exclusivechoice}(\mathsf{send\ pap\ test\ sample}, \mathsf{reject}) \\
& \mathsf{succession}(\mathsf{send\ colposcopy\ uncertain\ result}, \mathsf{send\ biopsy\ sample})\ \}
\end{aligned}
$$

In model $M_1$, the precedence constraint implies that if a biopsy is executed, then the colposcopy examination has provided an uncertain result before. The second model is identical to the first one, except for the constraint related to our preference. Interestingly,

the succession relates the same activities involved in the precedence constraint in the first model. The difference between the two models lies in the logical relation between precedence and succession: a trace that violates the former will always violate the latter (but not vice versa). It is then up to the domain expert to prefer a stricter or a more general behavior.

Finally, we did compare the effect of discovering models with or without the two preferred templates, by extracting 10 optimal models with no template preferences, and 10 each with the Precedence and Succession preference respectively. Interestingly enough, imposing the Precedence preference results in being extremely useful in this scenario. In fact, none of the 10 models discovered with no preferred template did contain a Precedence pattern. Similarly with Succession, which appears in only 1 of the 10 models discovered without specifying any preference.

## 7 Related Work

When processes are loosely-structured, procedural discovery could produce spaghetti-like models [28,15]. In that case, declarative approaches are more suitable for the purpose since they briefly list all the required or prohibited behaviors without explicitly specifying all possible process paths.

Over the last decade, several works focused on declarative process discovery [30,18,38,19]. In [30,18], the authors propose to build the set of all possible candidate `Declare` constraints considering all the activities that appear in the log, and check them against the whole log until certain levels of recall and specificity are reached. Techniques to refine the business model excluding vacuously satisfied constraints are the focus of the subsequent works by Schunselaar et al. [38], whereas Di Ciccio et al. [19] propose an approach to filter out frequent redundancies and inconsistencies. All the cited declarative approaches do not deal with negative examples. Nonetheless, interestingly from our point of view, in [29], the authors present an approach to specify "crisp" preferences that filter out constraints (discovered from positive examples only) that are not in line with some user knowledge. Differently from this approach, our approach allows the user to use preferences to "prioritize" the discovered models without necessarily filter out the ones that do not agree with the specified preferences.

Negative examples are instead actively employed in the declarative discovery approaches [25,24,7,8,16,15,39]. The technique by Lamma et al. [25,24] learns integrity constraints expressed as logical formulas, and translates them into the equivalent DecSerFlow constructs [2]. Bellodi et al. [7,8] employ the same approach and automatically convert the results into Markov Logic formulas—statistical relational learning is used to determine the weight of each formula. Analogously, Chesani et al. [16] propose to learn a set of SCIFF rules [5] and translate them into ConDec constraints [35]. The approach that we adopt in this work instead, is the one presented in [15], which directly learns `Declare` constraints without any intermediate language. This approach is grounded on a SAT-based solver analogously to the works in [32,13,37], where simple Linear Temporal Logic (LTL) formulas are generated to analyze sets of positive and negative examples. Particularly relevant for our work is the contribution by Slaats et al. [39], which proposes a binary classification procedure for process discovery evaluated on a set of real-life logs with negative examples from industry.

Our notion of negative example is similar to the definitions of syntactical and semantic noise of [22] since our approach is able to extract both the syntactic information that characterizes the positive examples w.r.t. negative ones, and the relevant semantic difference between traces that have been partially or totally modified at a certain point in time. In this sense, our work is also closely related to deviance mining approaches [33], i.e., techniques to extract the relevant details characterizing those traces that deviate from the expected behavior. Whereas some deviance mining approaches [40,6] focus on the differences between models discovered from deviant and non-deviant traces, others [41,34,10,27,9] intend deviance mining as a sort of sequence classification for the discovery of activity patterns discriminating between different sets of traces.

## 8 Conclusions

In this paper, we address the problem of inserting expert preferences in the discovery of declarative process models as a two-class supervised task. In particular, we extend the `NegDis` binary miner for the `Declare` modeling language with preferences over `Declare` templates and activities appearing in the model (plus a combination of the two). The computation of the preferred models, which take into account the preferences posed by the expert user, is performed using *ASPrin*, a general framework for computing optimal ASP models with preferences. The provided approach is described by means of motivating examples and an application to the real-life event logs DREYERS and CERV shows how to describe - in a discriminative manner - execution traces on the basis of preferred activities and `Declare` patterns. Future works will include a wider evaluation, which will also involve end-users. This will enable the assessment of the potential benefits of involving users (through their preferences) in the loop of process discovery.

**Acknowledgments.** This work has been partially supported by the European Union's H2020 projects HumaneAI-Net (g.a. 952026), StairwAI (g.a. 101017142), and TAILOR (g.a. 952215).

## References

1. van der Aalst, W.M.P., De Masellis, R., Di Francescomarino, C., Ghidini, C.: Learning hybrid process models from events - process discovery without faking confidence. In: Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10445, pp. 59–76. Springer (2017)
2. van der Aalst, W.M.P., Pesic, M.: Decserflow: Towards a truly declarative service flow language. In: WS-FM. Lecture Notes in Computer Science, vol. 4184, pp. 1–23. Springer (2006)
3. van der Aalst, W.M.P., Rubin, V.A., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling **9**(1), 87–111 (2010)
4. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)

5. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The SCIFF framework. ACM Trans. Comput. Log. **9**(4), 29:1–29:43 (2008)
6. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Behavioral comparison of process models based on canonically reduced event structures. In: BPM. Lecture Notes in Computer Science, vol. 8659, pp. 267–282. Springer (2014)
7. Bellodi, E., Riguzzi, F., Lamma, E.: Probabilistic logic-based process mining. In: CILC. CEUR Workshop Proceedings, vol. 598. CEUR-WS.org (2010)
8. Bellodi, E., Riguzzi, F., Lamma, E.: Statistical relational learning for workflow mining. Intell. Data Anal. **20**(3), 515–541 (2016)
9. Bergami, G., Di Francescomarino, C., Ghidini, C., Maggi, F.M., Puura, J.: Exploring business process deviance with sequential and declarative patterns. CoRR **abs/2111.12454** (2021)
10. Bose, R.P.J.C., van der Aalst, W.M.P.: Discovering signature patterns from event logs. In: CIDM. pp. 111–118. IEEE (2013)
11. Brewka, G., Delgrande, J.P., Romero, J., Schaub, T.: asprin: Customizing answer set preferences without a headache. In: AAAI. pp. 1467–1474. AAAI Press (2015)
12. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
13. Camacho, A., McIlraith, S.A.: Learning interpretable models expressed in linear temporal logic. In: ICAPS. pp. 621–630. AAAI Press (2019)
14. Chesani, F., Di Francescomarino, C., Ghidini, C., Grundler, G., Loreti, D., Maggi, F.M., Mello, P., Montali, M., Tessaris, S.: Optimising business process discovery using answer set programming. Proceedings of the 16th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR 2022) (2022 To appear)
15. Chesani, F., Di Francescomarino, C., Ghidini, C., Loreti, D., Maggi, F.M., Mello, P., Montali, M., Tessaris, S.: Process discovery on deviant traces and other stranger things. IEEE Trans. on Knowledge and Data Engineering (2021), under review
16. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. Trans. Petri Nets Other Model. Concurr. **2**, 278–295 (2009)
17. Debois, S., Slaats, T.: 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. pp. 1374–1382. IEEE (2015)
18. Di Ciccio, C., Maggi, F.M., Mendling, J.: Efficient discovery of target-branched declare constraints. Inf. Syst. **56**, 258–283 (2016)
19. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. **64**, 425–446 (2017)
20. Gebser, M., Kaminski, R., Kauffman, B., Schaub, T.: Multi-shot asp solving with clingo. Theory and Practice of Logic Progr. **19**(1), 27–82 (2019)
21. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. J. Mach. Learn. Res. **10**, 1305–1340 (2009)
22. Günther, C.W.: Process Mining in Flexible Environments. Ph.D. thesis, Technische Universiteit Eindhoven (2009)
23. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In: BPM. Lecture Notes in Computer Science, vol. 4714, pp. 328–343. Springer (2007)
24. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: Business Process Management. pp. 344–359. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
25. Lamma, E., Mello, P., Riguzzi, F., Storari, S.: Applying inductive logic programming to process mining. In: ILP. Lecture Notes in Computer Science, vol. 4894, pp. 132–146. Springer (2007)

26. de León, H.P., Nardelli, L., Carmona, J., vanden Broucke, S.K.L.M.: Incorporating negative information to process discovery of complex systems. Inf. Sci. **422**, 480–496 (2018)
27. Lo, D., Khoo, S., Liu, C.: Efficient mining of iterative patterns for software specification discovery. In: KDD. pp. 460–469. ACM (2007)
28. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE. Lecture Notes in Computer Science, vol. 7328, pp. 270–285. Springer (2012)
29. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: A knowledge-based integrated approach for discovering and repairing declare maps. In: Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings. pp. 433–448 (2013)
30. Maggi, F.M., Di Ciccio, C., Di Francescomarino, C., Kala, T.: Parallel algorithms for the automated discovery of declarative process models. Inf. Syst. **74**(Part), 136–152 (2018)
31. Maruster, L., Weijters, A.J.M.M., van der Aalst, W.M.P., van den Bosch, A.: A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. Data Min. Knowl. Discov. **13**(1), 67–87 (2006)
32. Neider, D., Gavran, I.: Learning linear temporal properties. In: FMCAD. pp. 1–10. IEEE (2018)
33. Nguyen, H., Dumas, M., La Rosa, M., Maggi, F.M., Suriadi, S.: Business process deviance mining: Review and evaluation. CoRR **abs/1608.08252** (2016)
34. Partington, A., Wynn, M.T., Suriadi, S., Ouyang, C., Karnon, J.: Process mining for clinical processes: A comparative analysis of four australian hospitals. ACM Trans. Management Inf. Syst. **5**(4), 19:1–19:18 (2015)
35. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Business Process Management Workshops. Lecture Notes in Computer Science, vol. 4103, pp. 169–180. Springer (2006)
36. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). pp. 287–300. IEEE Computer Society (2007)
37. Riener, H.: Exact synthesis of LTL properties from traces. In: FDL. pp. 1–6. IEEE (2019)
38. Schunselaar, D.M.M., Maggi, F.M., Sidorova, N.: Patterns for a log-based strengthening of declarative compliance models. In: IFM. Lecture Notes in Computer Science, vol. 7321, pp. 327–342. Springer (2012)
39. Slaats, T., Debois, S., Back, C.O.: Weighing the pros and cons: Process discovery with negative examples. In: Polyvyanyy, A., Wynn, M.T., Looy, A.V., Reichert, M. (eds.) Business Process Management - 19th International Conference, BPM 2021. vol. 12875, pp. 47–64 (2021)
40. Suriadi, S., Mans, R., Wynn, M.T., Partington, A., Karnon, J.: Measuring patient flow variations: A cross-organisational process mining approach. In: AP-BPM. Lecture Notes in Business Information Processing, vol. 181, pp. 43–58. Springer (2014)
41. Suriadi, S., Wynn, M.T., Ouyang, C., ter Hofstede, A.H.M., van Dijk, N.J.: Understanding process behaviours in a large insurance company in australia: A case study. In: CAiSE. vol. 7908, pp. 449–464 (2013)
42. Tessaris, S., Di Francescomarino, C., Chesani, F.: Negdis: code for the experiments (Aug 2021). https://doi.org/10.5281/zenodo.5158527
43. van der Aalst, et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops. pp. 169–194. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
44. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering workflow models from event-based data using little thumb. Integr. Comput. Aided Eng. **10**(2), 151–162 (2003)