The Effect of Noise on Mined Declarative Constraints

Claudio Di Ciccio, Massimo Mecella, Jan Mendling

The Effect of Noise
on Mined Declarative Constraints

Claudio Di Ciccio[1], Massimo Mecella[2], and Jan Mendling[1]*

[1] Wirtschaftsuniversität Wien, Vienna, Austria
{claudio.di.ciccio|jan.mendling}@wu.ac.at
[2] SAPIENZA – Università di Roma, Rome, Italy
massimo.mecella@uniroma1.it

Summary. Declarative models are increasingly utilized as representational format in process mining. Models created from automatic process discovery are meant to summarize complex behaviors in a compact way. Therefore, declarative models do not define all permissible behavior directly, but instead define constraints that must be met by each trace of the business process. While declarative models provide compactness, it is up until now not clear how robust or sensitive different constraints are with respect to noise. In this paper, we investigate this question from two angles. First, we establish a constraint hierarchy based on formal relationships between the different types of Declare constraints. Second, we conduct a sensitivity analysis to investigate the effect of noise on different types of declarative rules. Our analysis reveals that an increasing degree of noise reduces support of many constraints. However, this effect is moderate on most of the constraint types, which supports the suitability of Declare for mining event logs with noise.

Key words: Process Mining, Declarative Workflows, Noisy Event Logs

1 Introduction

Automated process discovery is an important field of research in the area of process mining. The goal of process discovery is to generate a process model from the behavior captured in an event log. In this context, process models can be represented in different formats. There is ongoing research that aims at establishing which representations are best suited for describing the behaviour. While procedural languages like Petri nets have been found appropriate for structured processes, it is believed that declarative languages such as Declare yield a more compact representation for so-called Spaghetti processes, which are processes with a high degree of concurrency [1]. It has also been argued that Petri nets are better to communicate how a process can progress, while Declare models are good at describing the circumstances of execution of a particular activity [2, 3, 4].

Beyond these mutual strengths and weaknesses, one of the important matters of automated process discovery is robustness to noise. There has been extensive research

into techniques to abstract from noise for procedural languages, which resulted among others in the heuristics miner [5], the fuzzy miner [6], and in an approach based on genetic mining [7]. In contrast to this, a detailed discussion of the effects of noise on declarative models is missing so far. Noisy logs/traces can be natural when discovering processes in unconventional scenarios, e.g., discovering "artful processes" carried out by knowledge workers through collaboration tools and email messages [8]. In such cases, logs are derived through object matching and text mining techniques applied to communication messages, and therefore the presence of noise is inevitable.

In this paper, we address this question from two angles. First, we investigate in how far different Declare constraints are robust to noise. To this end, we develop a constraint hierarchy that builds on formal relationships between the constraint types. Second, we conduct simulation experiments in order to study the degree of robustness of different Declare constraints. Based on these two perspectives, we gain insights into general properties of Declare with respect to noise.

Against this background, the remainder of this paper is structured as follows. Section 2 discusses the background of this research. In particular, we formally define Declare constraints. Section 3 discusses formal relationships between different constraint types and defines a hierarchy, which provides the basis for formulating experimental hypotheses listed in Section 4. Section 5 defines an experimental setup, which we use to investigate the hypotheses. Section 6 discusses our findings in the light of related work. Section 7 concludes the paper.

2 Declare constraints

Process mining [1] deals with the discovery, decision support and conformance checking of business processes, based on real data. Data are meant to be provided by means of a log, i.e., a machine-readable list of traces, where each trace consists of a sequence of events. Events represent the execution of activities. Therefore, traces correspond to the recording of the enactment of process instances (a.k.a. cases). In this work, the focus is on control-flow discovery. In particular, the mined control flow is based on the process modeling notation named Declare [9, 10]. Declare is a declarative language [11], i.e., defining the control flow of processes by means of constraints. Such constraints specify the rules that must not be violated during the enactment. Every behavior which complies with such rules is acceptable. Therefore, what is not constrained is considered as permitted. The constraints are formulated on activities.

Declare defines a set of *constraint templates*, which actual constraints are instantiations of. For instance, $RespondedExistence(\rho, \sigma)$ is a template constraining the activities ρ and σ.[2] It specifies that if activity ρ is performed, then σ must be executed in the same process instance as well. $RespondedExistence$ is the constraint template for $RespondedExistence(\rho, \sigma)$. The comprehensive list of templates, along with their description, can be found in [12]. A subset of Declare constraint templates, already adopted in [13, 14], will be considered in this study (see Table 1). Considering

---

[2] For the sake of readability, we will use the following notation: ρ and σ, to indicate general activities; $a, b, c, \ldots$, to identify actual activities as well as events in the trace.

Constraint	Explanation	Example traces		
Existence constraints: cardinality constraints				
$Existence(n, \mathsf{a})$	a occurs at least n times			
$Participation(\mathsf{a})$	a occurs at least *once*	✓ bcac	✓ bcaac	× bcc
$Absence(m, \mathsf{a})$	a occurs at most $m - 1$ times			
$AtMostOne(\mathsf{a})$	a occurs at most *once*	✓ bcc	✓ bcac	× bcaac
Existence constraints: position constraints				
$Init(\mathsf{a})$	a is the *first* to occur	✓ acc	✓ abac	× cc
$End(\mathsf{a})$	a is the *last* to occur	✓ bca	✓ baca	× bc
Relation constraints				
$RespondedExistence(\mathsf{a}, \mathsf{b})$	If a occurs in the trace, then b occurs as well	✓ bcaac	✓ bcc	× caac
$Response(\mathsf{a}, \mathsf{b})$	If a occurs, then b occurs after a	✓ caacb	✓ bcc	× caac
$AlternateResponse(\mathsf{a}, \mathsf{b})$	Each time a occurs, then b occurs afterwards, before a recurs	✓ cacb	✓ abcacb	× caacb
$ChainResponse(\mathsf{a}, \mathsf{b})$	Each time a occurs, then b occurs immediately afterwards	✓ cabb	✓ abcab	× cacb
$Precedence(\mathsf{a}, \mathsf{b})$	b occurs only if preceded by a	✓ cacbb	✓ acc	× ccbb
$AlternatePrecedence(\mathsf{a}, \mathsf{b})$	Each time b occurs, it is preceded by a and no other b can recur in between	✓ cacba	✓ abcaacb	× cacbba
$ChainPrecedence(\mathsf{a}, \mathsf{b})$	Each time b occurs, then a occurs immediately beforehand	✓ abca	✓ abaabc	× bca
Coupling relation constraints				
$CoExistence(\mathsf{a}, \mathsf{b})$	If b occurs, then a occurs, and viceversa	✓ cacbb	✓ bcca	× cac
$Succession(\mathsf{a}, \mathsf{b})$	a occurs if and only if it is followed by b	✓ cacbb	✓ accb	× bac
$AlternateSuccession(\mathsf{a}, \mathsf{b})$	a and b if and only if the latter follows the former, and they alternate each other in the trace	✓ cacbab	✓ abcabc	× caacbb
$ChainSuccession(\mathsf{a}, \mathsf{b})$	a and b occur if and only if the latter immediately follows the former	✓ cabab	✓ ccc	× cacb
Negative relation constraints				
$NotChainSuccession(\mathsf{a}, \mathsf{b})$	a and b occur if and only if the latter does not immediately follows the former	✓ acbacb	✓ bbaa	× abcab
$NotSuccession(\mathsf{a}, \mathsf{b})$	a can never occur before b	✓ bbcaa	✓ cbbca	× aacbb
$NotCoExistence(\mathsf{a}, \mathsf{b})$	a and b never occur together	✓ cccbbb	✓ ccac	× accbb

Table 1: Declare constraints.

the original notation of Declare [10], we note that $Participation(\mathsf{a})$ is equivalent to $Existence(1, \mathsf{a})$ and $AtMostOne(\mathsf{a})$ is equivalent to $Absence(2, \mathsf{a})$ [15]. Constraint templates belong to types, identifying their general characteristics:

Existence constraints constrain single activities;
Cardinality constraints are existence constraints specifying the count of constrained activities;
Position constraints are existence constraints specifying the position of constrained activities;
Relation constraints constrain pairs of activities;
Coupling relation constraints are satisfied only when two relation constraints are satisfied;

Negative relation constraints negate coupling relation constraints.

As a consequence, existence constraints refer to single activities, whereas the other types constrain them in pairs. As explained in [13, 14], relation constraints are activated by the occurrence of an activity (named as "activation" in [13], or "implying" in [14]). When activated, they force the occurrence of the other activity in the pair ("target" [13] or "implied" [14]). If no activating task is performed during the process execution, the constraint imposes no condition on the rest of the enactment. For instance, if no a is performed, $RespondedExistence(\mathsf{a}, \mathsf{b})$ has no effect on the execution, and thus the occurrence of b is not required. For coupling relation constraints and negative relation constraints, both involved activities are at the same time implying and implied.

2.1 Declare constraint templates as FOL formulae

We provide the semantics of Declare templates as First Order Logic (FOL) formulae. The approach is inspired by the translation technique from Linear Temporal Logic (LTL) to FOL over finite linear ordered sequences, discussed in [16]. An exhaustive description of the rationale applied to Declare constraint templates can be found in [15]. Formulae 1a – 1r are meant to be interpreted over finite traces. Therefore, they adopt variables i, j, k and l to indicate positions of events in traces. $first$ and $last$ are constants referring to the first and last position in a trace, respectively. $Succ$ is a binary predicate specifying whether a position follows another. $InTrace$ binary predicate states whether a given event occurs in the specified position.

$$Init(\rho) \equiv InTrace(first, \rho) \tag{1a}$$

$$End(\rho) \equiv InTrace(last, \rho) \tag{1b}$$

$$Participation(\rho) \equiv \exists i.\ InTrace(i, \rho) \tag{1c}$$

$$AtMostOne(\rho) \equiv \exists i.\ InTrace(i, \rho)\ \rightarrow\ \nexists j.\ InTrace(j, \rho)\ \wedge\ j \neq i \tag{1d}$$

$$RespondedExistence(\rho, \sigma) \equiv \forall i.\ InTrace(i, \rho)\ \rightarrow \exists j.\ InTrace(j, \sigma)\ \wedge\ i \neq j \tag{1e}$$

$$Response(\rho, \sigma) \equiv \forall i.\ InTrace(i, \rho)\ \rightarrow \exists j.\ InTrace(j, \sigma)\ \wedge\ i < j \tag{1f}$$

$$AlternateResponse(\rho, \sigma) \equiv \forall i.\ InTrace(i, \rho)\ \rightarrow \exists j.\ InTrace(j, \sigma)\ \wedge\ i < j\ \wedge$$
$$\nexists l.\ InTrace(l, \sigma)\ \wedge\ i < l < j\ \rightarrow$$
$$\nexists k.\ InTrace(k, \rho)\ \wedge\ i < k < j \tag{1g}$$

$$ChainResponse(\rho, \sigma) \equiv \forall i.\ InTrace(i, \rho)\ \rightarrow \exists j.\ InTrace(j, \sigma)\ \wedge\ Succ(i, j) \tag{1h}$$

$$Precedence(\rho, \sigma) \equiv \forall j.\ InTrace(j, \sigma)\ \rightarrow \exists i.\ InTrace(i, \rho)\ \wedge\ i < j \tag{1i}$$

$$AlternatePrecedence(\rho, \sigma) \equiv \forall j.\ InTrace(j, \sigma)\ \rightarrow \exists i.\ InTrace(i, \rho)\ \wedge\ i < j\ \wedge$$
$$\nexists k.\ InTrace(k, \rho)\ \wedge\ i < k < j\ \rightarrow$$
$$\nexists l.\ InTrace(l, \sigma)\ \wedge\ i < l < j \tag{1j}$$

$$ChainPrecedence(\rho, \sigma) \equiv \forall j.\ InTrace(j, \sigma)\ \rightarrow \exists i.\ InTrace(i, \rho)\ \wedge\ Succ(i, j) \tag{1k}$$

$$CoExistence(\rho, \sigma) \equiv RespondedExistence(\rho, \sigma)\ \wedge\ RespondedExistence(\sigma, \rho) \tag{1l}$$

$$Succession(\rho, \sigma) \equiv Response(\rho, \sigma)\ \wedge\ Precedence(\rho, \sigma) \tag{1m}$$

$$AlternateSuccession(\rho, \sigma) \equiv AlternateResponse(\rho, \sigma)\ \wedge\ AlternatePrecedence(\rho, \sigma) \tag{1n}$$

$$ChainSuccession(\rho, \sigma) \equiv ChainResponse(\rho, \sigma)\ \wedge\ ChainPrecedence(\rho, \sigma) \tag{1o}$$

$$NotCoExistence(\rho, \sigma) \equiv (\forall i.\ InTrace(i, \rho)\ \rightarrow \nexists j.\ InTrace(j, \sigma)\ \wedge\ i \neq j)\ \wedge$$
$$(\forall j.\ InTrace(j, \sigma)\ \rightarrow \nexists i.\ InTrace(i, \rho)\ \wedge\ i \neq j) \tag{1p}$$

$$NotSuccession(\rho, \sigma) \equiv (\forall i.\ InTrace(i, \rho)\ \rightarrow \nexists j.\ InTrace(j, \sigma)\ \wedge\ i < j)\ \wedge$$
$$(\forall j.\ InTrace(j, \sigma)\ \rightarrow \nexists i.\ InTrace(i, \rho)\ \wedge\ i < j) \tag{1q}$$

$$NotChainSuccession(\rho, \sigma) \equiv (\forall i.\ InTrace(i, \rho)\ \rightarrow \nexists j.\ InTrace(j, \sigma)\ \wedge\ Succ(i, j))\ \wedge$$
$$(\forall j.\ InTrace(j, \sigma)\ \rightarrow \nexists i.\ InTrace(i, \rho)\ \wedge\ Succ(i, j)) \tag{1r}$$

The specification of relation constraints, coupling relation constraints and negative relation constraints (cf. Formulae 1e – 1r) are formulated either as

$$\mathcal{C}(\rho, \sigma) \equiv \bigwedge (\mathcal{A}(\alpha) \to \mathcal{T}(\beta)), \qquad \alpha, \beta \in \{\rho, \sigma\}, \alpha \neq \beta$$

or

$$\mathcal{C}(\rho, \sigma) \equiv \bigwedge (\mathcal{A}(\alpha) \to \mathcal{E}(\alpha, \beta)), \qquad \alpha, \beta \in \{\rho, \sigma\}, \alpha \neq \beta$$

where $\mathcal{A}(\cdot)$, $\mathcal{T}(\cdot)$ and $\mathcal{E}(\cdot, \cdot)$ are parts of FOL formulae disregarding quantified variables (i, j, k) and quantifiers. The suitable generalization depends on whether the implied part predicates on the argument of $\mathcal{A}(\alpha)$ (i.e., $\mathcal{E}(\alpha, \beta)$, cf. Formulae 1g, 1j, 1n) or not ($\mathcal{T}(\beta)$, cf. Formulae 1e, 1f, 1h, 1i, 1k, 1l, 1m, 1o, 1p, 1q, 1r). The activation tasks are thus defined as α variables, whereas targets are β's. It is worthwhile to remark that multiple assignments for α and β can be valid for the same constraint. For instance, $NotCoExistence(\rho, \sigma)$ is such that both ρ and σ can be indifferently assigned to α and β. This means that both (ρ, σ) and (σ, ρ) are valid pairs for activation-target assignments. For $Response(\rho, \sigma)$, instead, only one assignment holds true: therefore, ρ is the activation and σ the target.

In the following, we will refer to a constraint's valid activation and target as $\alpha(\mathcal{C})$ and $\beta(\mathcal{C})$, respectively. Table 2 lists the activations and targets for each constraint. As the table shows, coupling relation constraints and negative relation constraints are such that both constrained activities play at the same time the roles of activation and target.

3 Constraints' properties

In this section, we investigate semantics of Declare constraints in order to categorize *(i)* the effect that constraints exert on traces (Section 3.1) and *(ii)* the mutual interdependencies among constraint templates (Section 3.2). This analysis is prodromal to the formulation of ten hypotheses, relating constraints' effects and interdependencies to their reaction to noise (Section 4).

Constraint $\mathcal{C}$	$(\alpha(\mathcal{C}), \beta(\mathcal{C}))$	Constraint $\mathcal{C}$	$(\alpha(\mathcal{C}), \beta(\mathcal{C}))$
$RespondedExistence(\rho, \sigma)$	(ρ, σ)	$CoExistence(\rho, \sigma)$	$(\rho, \sigma), (\sigma, \rho)$
$Response(\rho, \sigma)$	(ρ, σ)	$Succession(\rho, \sigma)$	$(\rho, \sigma), (\sigma, \rho)$
$AlternateResponse(\rho, \sigma)$	(ρ, σ)	$AlternateSuccession(\rho, \sigma)$	$(\rho, \sigma), (\sigma, \rho)$
$ChainResponse(\rho, \sigma)$	(ρ, σ)	$ChainSuccession(\rho, \sigma)$	$(\rho, \sigma), (\sigma, \rho)$
$Precedence(\rho, \sigma)$	(σ, ρ)	$NotCoExistence(\rho, \sigma)$	$(\rho, \sigma), (\sigma, \rho)$
$AlternatePrecedence(\rho, \sigma)$	(σ, ρ)	$NotSuccession(\rho, \sigma)$	$(\rho, \sigma), (\sigma, \rho)$
$ChainPrecedence(\rho, \sigma)$	(σ, ρ)	$NotChainSuccession(\rho, \sigma)$	$(\rho, \sigma), (\sigma, \rho)$

Table 2: Activations and targets for Declare relation constraints, coupling relation constraints, and negative relation constraints. $\alpha(\mathcal{C})$ and $\beta(\mathcal{C})$ are respectively the activation and target of constraint $\mathcal{C}$
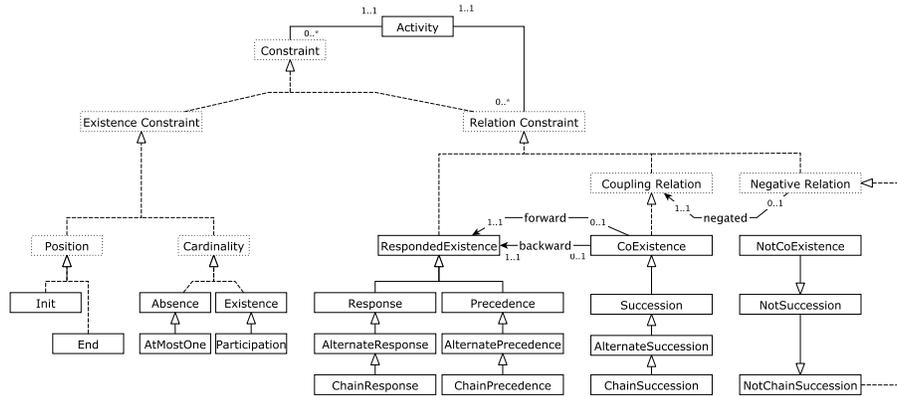
Fig. 1: The declarative process model's hierarchy of constraints. Taking into account the UML Class Diagram graphical notations, the Generalization ("is-a") relationship represents the restriction. The restricting is on the tail, the restricted on the head. The Realization relationship indicates that the constraint template (as well as the restricting ones) belong to a specific type. Constraint templates are drawn as solid boxes, whereas constraint types' boxes are dashed.

Constraint	Act.	Effect	On	Effect	On
$Participation(a)$		presence	a		
$AtMostOne(a)$	a	absence	a'		
$Init(a)$		presence	a (as first)		
$End(a)$		presence	a (as last)		
$RespondedExistence(a, b)$	a	presence	a		
$Response(a, b)$	a	presence	b (after a)		
$AlternateResponse(a, b)$	a	presence	b (after a)	absence	a' (betw. a and b)
$ChainResponse(a, b)$	a	presence	b (following a)	absence	a' (after a)
$Precedence(a, b)$	b	presence	a (before b)		
$AlternatePrecedence(a, b)$	b	presence	a (before b)	absence	b' (betw. a and b)
$ChainPrecedence(a, b)$	b	presence	a (preceding b)	absence	b' (before b)

Table 3: The effect of existence constraints and relation constraints on activities.

3.1 How constraints affect the activities

In the light of what stated by natural language (cf. Table 1) and FOL (cf. Formulae 1a – 1r), Table 3 specifies how existence constraints and relation constraints affect the execution of activities. In particular, we distinguish between *presence* and *absence* for

Restricting constraint	Restricted constraint	Restricting constraint	Restricted constraint
$Response(\rho, \sigma)$	$RespondedExistence(\rho, \sigma)$	$Succession(\rho, \sigma)$	$CoExistence(\rho, \sigma)$
$AlternateResponse(\rho, \sigma)$	$Response(\rho, \sigma)$	$AlternateSuccession(\rho, \sigma)$	$Succession(\rho, \sigma)$
$ChainResponse(\rho, \sigma)$	$AlternateResponse(\rho, \sigma)$	$ChainSuccession(\rho, \sigma)$	$AlternateSuccession(\rho, \sigma)$
$Precedence(\rho, \sigma)$	$RespondedExistence(\sigma, \rho)$	$NotCoExistence(\rho, \sigma)$	$NotSuccession(\rho, \sigma)$
$AlternatePrecedence(\rho, \sigma)$	$Precedence(\rho, \sigma)$	$NotSuccession(\rho, \sigma)$	$NotChainSuccession(\rho, \sigma)$
$ChainPrecedence(\rho, \sigma)$	$AlternatePrecedence(\rho, \sigma)$		

Table 4: Constraints under the relation of restriction.

those tasks that are involved by constraints. For instance, $AtMostOne(\mathsf{a})$ imposes that if the activating event, a, is found, not any other "a" can occur in the trace (absence). With a slight abuse of terminology, we indicate a as the activation, even though it is defined for relation constraints only, in the sense that if one a occurs, the constraint has effect on the trace. In Table 3, any *other* occurrence of a (resp. b) in the trace is pointed at by a' (b'). $Response(\mathsf{a}, \mathsf{b})$ establishes that, if a is found, then b must occur afterwards (presence). $Participation(\mathsf{a})$ has no activating event. However, it imposes the presence of a in the trace. $AlternateResponse(\mathsf{a}, \mathsf{b})$ and $ChainResponse(\mathsf{a}, \mathsf{b})$ (resp. $AlternatePrecedence(\mathsf{a}, \mathsf{b})$ and $ChainPrecedence(\mathsf{a}, \mathsf{b})$) not only constrain the presence of b (resp. a), as $Response(\mathsf{a}, \mathsf{b})$ ($Precedence(\mathsf{a}, \mathsf{b})$), but also the absence of other a's (b's), under specific conditions. For the sake of comprehensiveness, we recall here that what stated for $AtMostOne(\mathsf{a})$ and $Participation(\mathsf{a})$ in Table 3 also applies to $Absence(m, \mathsf{a})$ and $Existence(n, \mathsf{a})$, respectively.

3.2 Constraints' interdependencies

Formulae 1a – 1r show that constraint templates are not unrelated to each other. In the following, we will focus on three main interdependencies between constraints: *(i) restriction*, *(ii) conjunction*, and *(iii) activated negation*. Figure 1 sketches the interdependencies relations among constraint templates. The definition of such interdependency relations will be provided considering constraints as FOL predicates over finite linear ordered sequences (traces), coherently with Formulae of Section 2.1. Hence, we define the $\models$ relation as follows: given two constraints $\mathcal{C}$ and $\mathcal{C}'$, we say that $\mathcal{C}$ *entails* $\mathcal{C}'$ ($\mathcal{C} \models \mathcal{C}'$) when all traces allowed by $\mathcal{C}$ are also permitted by $\mathcal{C}'$. We refer to the set of all traces permitted by $\mathcal{C}$ as $\models \mathcal{C}$, logical models for a FOL predicate.

Restriction Restriction is a binary relation between constraints $\mathcal{C}$ and $\mathcal{C}'$ which holds when $\mathcal{C} \models \mathcal{C}'$. In other words, a constraint $\mathcal{C}(\rho, \sigma)$ is a restriction of another constraint $\mathcal{C}'(\rho, \sigma)$ when $\mathcal{C}(\rho, \sigma)$ allows for a subset of executions which are allowed by $\mathcal{C}'(\rho, \sigma)$. For instance, $AlternateResponse(\rho, \sigma)$ is a restriction of $Response(\rho, \sigma)$ because every process instance which is compliant to $Response(\rho, \sigma)$ is also compliant to $AlternateResponse(\rho, \sigma)$. Similarly, $ChainSuccession(\rho, \sigma)$ is a restriction of $Succession(\rho, \sigma)$. Note that the restriction relation has the property of *transitivity*. As

Coupling relation constraint	*forward*	*backward*
$CoExistence(\rho, \sigma)$	$RespondedExistence(\rho, \sigma)$	$RespondedExistence(\sigma, \rho)$
$Succession(\rho, \sigma)$	$Response(\rho, \sigma)$	$Precedence(\rho, \sigma)$
$AlternateSuccession(\rho, \sigma)$	$AlternateResponse(\rho, \sigma)$	$AlternatePrecedence(\rho, \sigma)$
$ChainSuccession(\rho, \sigma)$	$ChainResponse(\rho, \sigma)$	$ChainPrecedence(\rho, \sigma)$

Table 5: *forward* and *backward* associations for the conjunction of coupling relation constraints against relation constraints.

such, it is drawn like an "is-a" hierarchy in Figure 1. Similarly, we list the pairs of constraints in such relation, in Table 4. W.l.o.g., we specify one single restricted constraint for each restricting one, which is the closest in the hierarchy. Constrained activities are not reported in the figure. However, it is worth to recall that $Precedence(\rho, \sigma)$ restricts $RespondedExistence(\sigma, \rho)$, i.e., the activation for $Precedence$ is the target for $RespondedExistence$, and vice versa.

Conjunction Conjunction is a ternary relation among constraints $\mathcal{C}, \mathcal{C}', \mathcal{C}''$ which holds when $\mathcal{C} \models \mathcal{C}' \wedge \mathcal{C}''$. $\mathcal{C}(\rho, \sigma)$ is the conjunction of $\mathcal{C}'(\rho, \sigma)$ and $\mathcal{C}''(\rho, \sigma)$ when only those traces that comply with both $\mathcal{C}'(\rho, \sigma)$ and $\mathcal{C}''(\rho, \sigma)$ are permitted by $\mathcal{C}(\rho, \sigma)$. As an example, $Succession(\rho, \sigma)$ is the conjunction of $Response(\rho, \sigma)$ and $Precedence(\rho, \sigma)$. Table 5 report the list of conjunction relations for the Declare constraints under analysis. The conjunction relation is represented by the *forward* and *backward* associations in Figure 1. For the sake of readability, the associations are drawn only for the top elements in the hierarchy. They are meant to be inherited by the "descendant" constraints. The terms *forward* and *backward* refer to the verse in which the pairs of constrained activities become resp. activation and target for the constraints in the conjunction relation (cf. Table 2). For instance, $CoExistence(\rho, \sigma)$ is in conjunction relation with $RespondedExistence(\rho, \sigma)$ (*forward*, being ρ the activation and σ the target) and $RespondedExistence(\sigma, \rho)$ (*backward*, being σ the activation and ρ the target).

Activated negation Let $\alpha(\mathcal{C})$ be the activation of constraint $\mathcal{C}$, i a possible position of an event in a trace, $InTrace$ a binary predicate stating whether a given event occurs at the specified position (see Section 2.1). Activated negation is a binary relation among constraints $\mathcal{C}$ and $\mathcal{C}'$ which holds when
$$\models (\mathcal{C} \wedge \exists i.\ InTrace(i, \alpha(\mathcal{C}))) \quad \bigcap \quad \models (\mathcal{C}' \wedge \exists j.\ InTrace(j, \alpha(\mathcal{C}'))) \quad = \quad \emptyset.$$
$\mathcal{C}(\rho, \sigma)$ is the activated negation of another $\mathcal{C}'(\rho, \sigma)$ when no trace activating and satisfying $\mathcal{C}(\rho, \sigma)$ complies with $\mathcal{C}'(\rho, \sigma)$, and vice versa. In other terms, when a trace activates both, the former is satisfied if and only if the latter is not. As an example, $NotCoExistence(\rho, \sigma)$ is the activated negation of $CoExistence(\rho, \sigma)$. The activated negation relation is depicted by the *negated* association in Figure 1. For the sake of readability, the associations are drawn only for the constraint types. Table 6 reports the list of associated constraints for activated negation. Note that the activated negation relation is *symmetrical*. Only coupling relation constraints and negative relation con-

Negative relation constraint	$negated$
$NotCoExistence(\rho, \sigma)$	$CoExistence(\rho, \sigma)$
$NotSuccession(\rho, \sigma)$	$Succession(\rho, \sigma)$
$NotChainSuccession(\rho, \sigma)$	$ChainSuccession(\rho, \sigma)$

Table 6: *Negated* relations for *NegativeRelation* constraints

straints are listed. However, the relation extends to the relation constraints of which the coupling relation constraints are the conjunction.

We do not report formal proofs confirming the observations made so far, for the sake of space. However, they can be trivially verified by considering Formulae 1a – 1r and the textual descriptions provided in Table 1.

4 Hypotheses on the reaction of constraints to noise

Building upon the properties of Declare constraints, shown in Sections 3.1 and 3.2, we have formulated ten hypotheses, relating the characteristics of constraints discussed so far to their sensitivity or resilience to noise in logs. For the formulation of hypotheses, we have considered two specific abstractions for the effects that noise can cause on logs: *(i)* presence of spurious events in traces (insertion errors), and *(ii)* events missing in traces (absence errors). The hypotheses have driven the experiments detailed in Section 5, conducted in order to have an experimental evidence of conclusions drawn from the theoretical analysis of Declare constraints.

H1 Cardinality constraints requiring the presence of an activity are *resilient* to insertion errors and *sensitive* to deletion errors on such an activity.

H2 Cardinality constraints requiring the absence of an activity are *resilient* to deletion errors and *sensitive* to insertion errors on the referred activity.

H3 Position constraints are *resilient* to insertion errors and *sensitive* to deletion errors on the constrained activity.[3]

H4 All constraints having an activation are *resilient* to the absence of activation events.

H5 All constraints having an activation are *sensitive* to the presence of spurious activation events.

H6 All constraints requiring the presence of the target are *resilient* to the presence of spurious target events.

H7 All constraints requiring the presence of the target are *sensitive* to the absence of target events.

H8 Coupling relation constraints inherit the sensitivity of those constraints of which they are the conjunction.

H9 Negative relation constraints are *sensitive* to the presence of constrained activities and *resilient* to their absence.

---

[3] Position constraints behave like cardinality constraints requiring the presence of an activity – cf. **H1**

H10 Along the restriction hierarchy, descendant constraints are more sensitive than ancestors to the presence of noise.

5 Evaluation

In order to observe the change in the mined models due to errors in logs, we have created error-injected logs. Section 5.1 describes how we apply different categories of noise to logs complying to one constraint at a time, each representing a constraint template. Section 5.2 illustrates the experimental setup, in order to observe the reactions of constraints to different kinds of noise. Sections 5.3 to 5.10 present in detail the results for each of the hypotheses defined above. Section 5.11 summarizes the gathered insights and closes this section.

5.1 Noise categories

In order to perform a controlled injection of errors in logs, we identified four main parameters:

1. *Noise type*: it can be either one of the following: (a) insertion of spurious events in the log; (b) deletion of events from the log; (c) random insertion/deletion of events.
2. *Noise injection rate*, ranging from 0 to 100%.
3. *Noise spreading policy*; it can be either one of the following: (a) distribution of noise in every trace (*trace-based*); (b) distribution of noise over the entire log (*log-based*).
4. *Faulty activity*.

The faulty activity defines the activity whose events are subject to errors. The noise type abstracts the basic kinds of possible errors that can be in a log. The percentage of noise injection rate refers to the number of occurring targeted faulty activities. As an example, we can consider a log consisting of a single trace, like the following: $\{\langle a, a, b, a, b, a, c, d, a, b, d\rangle\}$. In such a case, taking a as the targeted faulty activity, with a noise injection rate of 20%, one error would be injected, as five a's occur $(20/100 \cdot 5)$. In case the calculated number of errors to inject results in a non-integer number, the actual amount of errors will be its round-up: e.g., if four a's occur and the noise injection rate is equal to 20%, one error is injected ($\lceil 20/100 \cdot 4 \rceil = 1$). The noise spreading policy determines where errors take place. In particular, if it is trace-based, every trace is affected by a given number of errors. This reproduces a systematic error, taking place in every recorded enactment of the process. If the noise spreading policy is log-based, instead, errors will not necessarily appear with the same recurrence in every trace. Therefore, some traces could remain untouched. Such a case simulates the presence of event-recording errors. As an example, we can consider the following log, having a as the faulty activity and a noise injection rate of 25%: $\{\langle a, a, b, a, b, a, c, d\rangle, \langle c, d, b, a, d, d, a, a, d\rangle\}$. Both traces contain four occurrences of a. If the noise spreading policy is trace-based, an error will be injected in every trace. If it is log-based, two errors will be injected in the log as well, but not necessarily one

Activities (target)	8 (1)	Noise spreading policies	3
Generating constraints	18	Noise types	3
Trace length	$[0, 30]$	Runs per combination	50
Log size	1 000	Noise injection rates	$[0, 30]$
		Total runs	167 400

Table 7: Setup of the experiments

for each trace. Furthermore, the number of errors could differ depending on the noise spreading policy. If, for instance, five a's had occurred in the first trace, and three in the second, two errors would have been injected according to the log-based noise spreading policy. However, the trace-based one would introduce three errors in the log: two in the first trace ($\lceil 25/100 \cdot 5 \rceil = 2$) and one in the second ($\lceil 25/100 \cdot 3 \rceil = 1$).

5.2 Experiment setup

We have created 18 groups of 9,300 synthetic logs each. Every group was generated in order to comply with one constraint at a time, among the 18 templates involving a, as the implying activity, and (optionally) b, as the implied (i.e., $Participation(\mathsf{a})$, $AtMostOne(\mathsf{a})$, ..., $RespondedExistence(\mathsf{a}, \mathsf{b})$, $Response(\mathsf{a}, \mathsf{b})$, ...). The alphabet comprised 6 more non-constrained activities (c, d, ..., h), totaling 8. Logs have been generated by a specifically ad-hoc developed software module that utilizes the `dk.bricks.automaton` library.[4] This Java tool is capable of generating random strings that comply with user-defined Regular Expressions (REs). In particular, we adopted the Declare-to-RE translation map, discussed in our previous works [8, 14, 17]. We chose a as the faulty activity. The faulty activity plays thus both the role of activation in, e.g., $Response(\mathsf{a}, \mathsf{b})$, and the role of target in, e.g., $Precedence(\mathsf{a}, \mathsf{b})$. Then, we have injected errors in the synthetic logs, with all possible combinations of the aforementioned parameters: *(i)* insertion, deletion or random noise type, *(ii)* trace-based or log-based noise spreading policy, *(iii)* noise injection rate, ranging between 0% and 30%. Thereupon, we have run the technique for process discovery presented in [15], on the resulting altered logs. We have collected the results and, for each of the 18 groups of logs, analyzed the trend of the support for the generating constraint. In other words, given the only constraint which had to be verified, we have looked at how its support is lowered, w.r.t. the increasing percentage of introduced noise.

For each of the hypotheses, an experimental evidence is provided next. Hypotheses define the sensitivity of single constraint templates or constraint types. Therefore, the diagrams shown will put in evidence the trend of their support (bold lines) with respect to the noise injection rate. The following figures also draw the trend of those other constraints whose topmost computed support exceeds the value of 0.75 (thin semi-

---

[4] http://www.brics.dk/automaton/

transparent lines),[5] as they are the most likely candidates to be false positives in the discovery.

5.3 *Participation* (H1)

(a) The trend of the support of *Participation*(a), w.r.t. the percentage of spurious a's inserted along the log

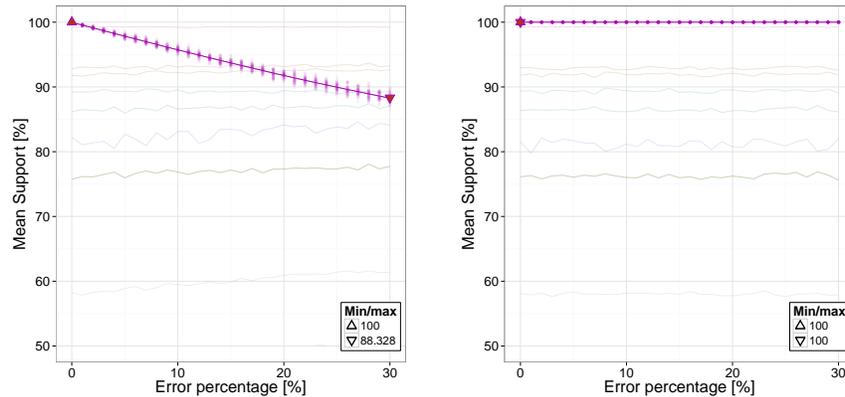(b) The trend of the support of *Participation*(a), w.r.t. the percentage of deleted a's along the log

Fig. 2: The reaction of *Participation*, w.r.t. different noise types, adopting a log-based noise spreading policy

Participation imposes the presence of the referred activity in every case. Therefore, Figure 2a and Figure 2b show that missing occurrences of a undermine the detectability of *Participation*(a) in the log. Spurious a's do not have any effect on the support of that constraint, as *Participation*(a) requires that *at least one* occurrence of a is read in every trace. Therefore, Figure 2 gives an experimental evidence of **H1**, as *Participation* is a cardinality constraints requiring the presence of the constrained activity.

5.4 *AtMostOne* (H2)

AtMostOne entails a behavior which is dual w.r.t. *Participation*, as it requires that *at least one* occurrence of a is read in every trace. This is reflected in the opposite receptiveness to the different noise types: for *AtMostOne*(a), spurious a's lower the computed support, whereas missing a's have no effect on it. This supports **H2**, as *AtMostOne* is a cardinality constraint requiring the absence of the constrained activity.

---

[5] We recall that assigning a constraint the support of 0.5 would be equivalent to asserting that such constraint would held if, tossing a coin, a cross were shown in the end. Thereby, 0.75 is the least value of the topmost half of the "reliable" range.

(a) The trend of the support of *AtMostOne*(a), w.r.t. the percentage of spurious a's inserted along the log

(b) The trend of the support of *AtMostOne*(a), w.r.t. the percentage of deleted a's along the log
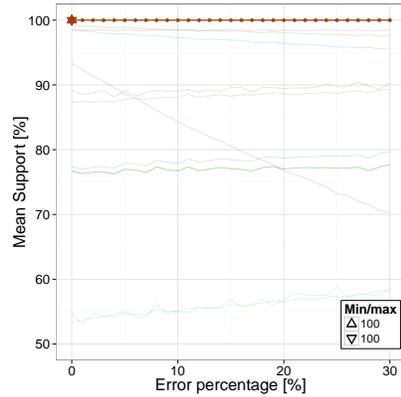
Fig. 3: The reaction of *AtMostOne*, w.r.t. different noise types, adopting a log-based noise spreading policy
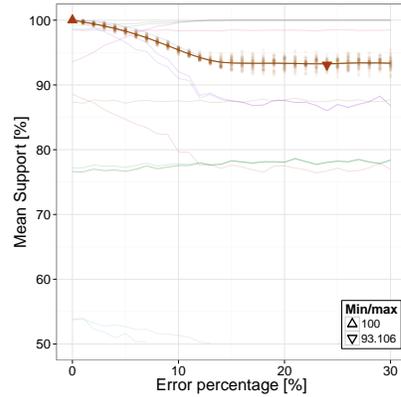
5.5 *Init* and *End* (H3)

Position constraints such as *Init* and *End* require the presence of constrained activities, resp. as the initial and final task of every trace. Disregarding the imposed position, they thus act like *Participation*. As a consequence, they are subject to the same noise type to which cardinality constraints requiring the presence of an activity are sensitive to. Figure 4 shows the trend of support for *Init*(a) and *End*(a), supporting **H3**.
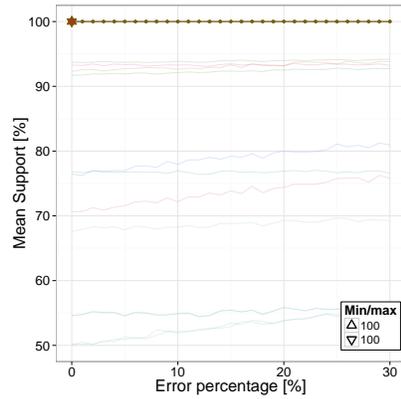
5.6 *Response* (H4, H5)

In order to have an experimental evidence of **H4** and **H5**, we considered $Response(a, b)$ as the representative constraint. We made a, i.e., the activation of $Response(a, b)$, the faulty activity. As expected, the expunction of a's did not cause any change in the support of the constraint (cf. Figure 5b). This is due to the fact that if an activation misses from the trace, the constraint has no effect on it, i.e., no further verification needs to be held to confirm whether the constraint is verified. The absence of the activation from the trace leads to what is called in literature "vacuous satisfaction" of the constraint [13]. Conversely, the insertion of spurious a's lead to a decrease in computed support. This is due to the fact that for every new a in the trace, the presence of a following b must be verified. Since the spurious a's are placed at random in the trace, the newly inserted ones are likely to lead to a violation of the constraint. This phenomenon is well documented by Figure 5.
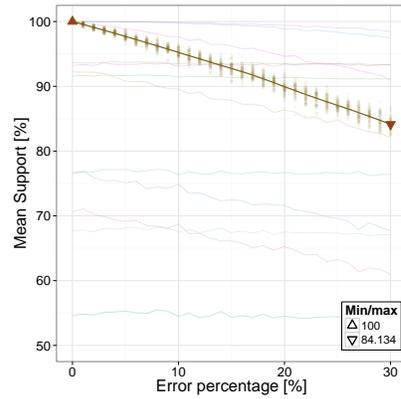
(a) The trend of the support of $Init(\mathsf{a})$, w.r.t. the percentage of spurious a's along the log

(b) The trend of the support of $Init(\mathsf{a})$, w.r.t. the percentage of deleted a's along the log

(c) The trend of the support of $End(\mathsf{a})$, w.r.t. the percentage of spurious a's along the log

(d) The trend of the support of $End(\mathsf{a})$, w.r.t. the percentage of deleted a's along the log

Fig. 4: The reaction of $Init$ and End, w.r.t. different noise types, adopting a log-based noise spreading policy

5.7 *Precedence* (H6, H7)

H6 and **H7** mention constraints requiring the presence of the target in the trace. There-fore, we take $Precedence(\mathsf{a}, \mathsf{b})$ as a representative constraint, and a as the faulty activity. As shown in Figure 6b, the expunction of a's causes the support to decrease, unlike the case of $Response$ shown before. In this case, a plays the role of the target. Therefore, its absence can entail the violation of the constraint. Conversely, having more a's does not affect the validity of the constraint on the trace, due to the fact that *at least one* occurrence of the target is required.
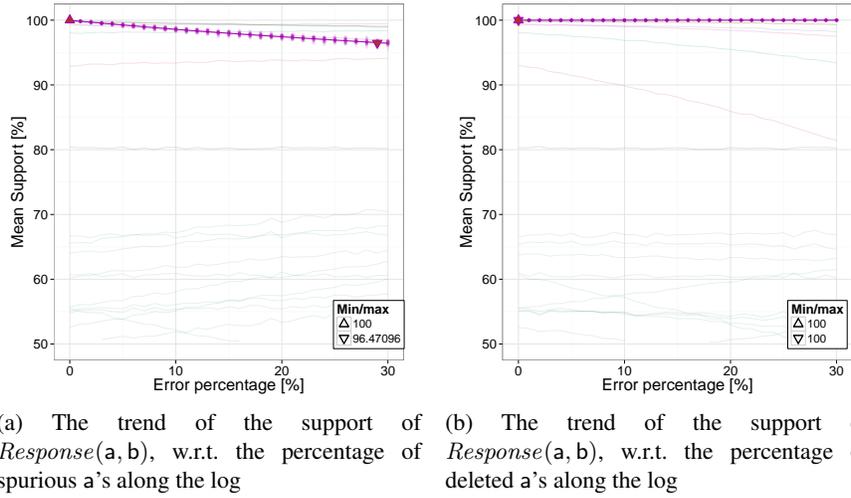
(a) The trend of the support of *Response*(a, b), w.r.t. the percentage of spurious a's along the log

(b) The trend of the support of *Response*(a, b), w.r.t. the percentage of deleted a's along the log

Fig. 5: The reaction of *Response*, w.r.t. different noise types on the activation event, adopting a log-based noise spreading policy

(a) The trend of the support of *Precedence*(a, b), w.r.t. the percentage of spurious a's along the log

(b) The trend of the support of *Precedence*(a, b), w.r.t. the percentage of deleted a's along the log
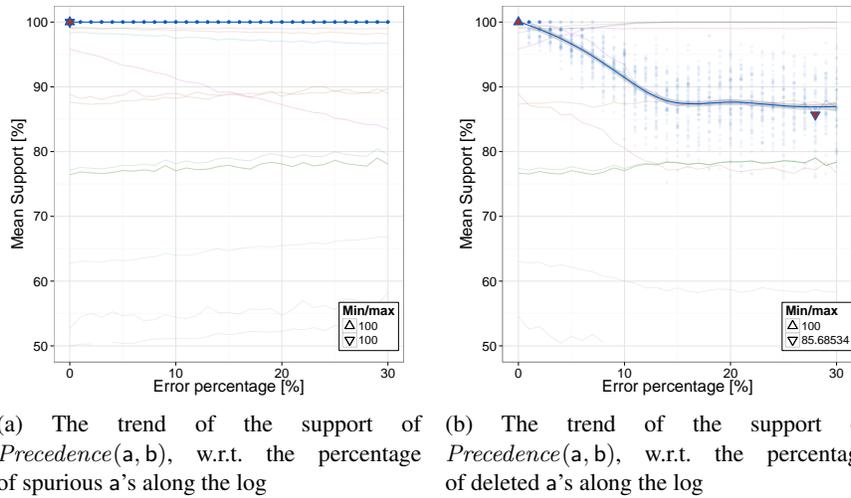
Fig. 6: The reaction of *Precedence*, w.r.t. different noise types on the target event, adopting a log-based noise spreading policy

5.8 *Succession* (H8)

Response(a, b) and *Precedence*(a, b) have been adopted to present the opposite reaction to the insertion and expunction of a's. The former is resilient to the deletion and sensitive to the insertion. The other way round, the latter is resilient to the insertion and

(a) The trend of the support of $Succession(\mathsf{a}, \mathsf{b})$, w.r.t. the percentage of spurious a's along the log

(b) The trend of the support of $Succession(\mathsf{a}, \mathsf{b})$, w.r.t. the percentage of deleted a's along the log
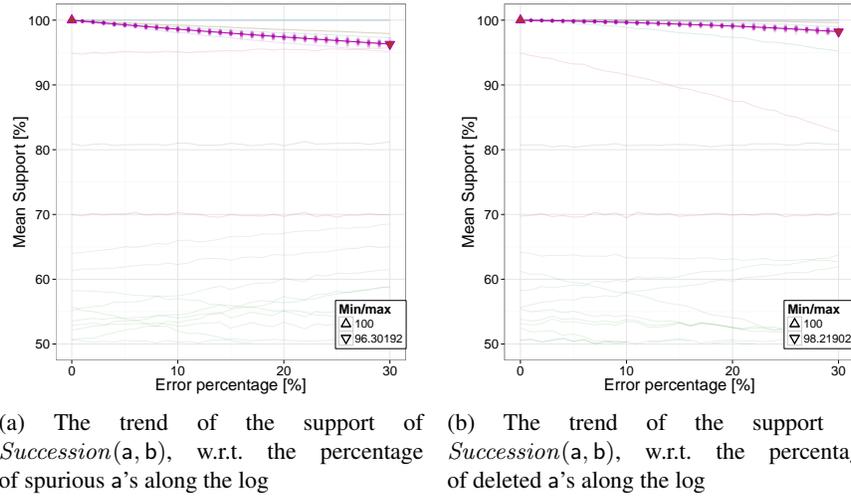
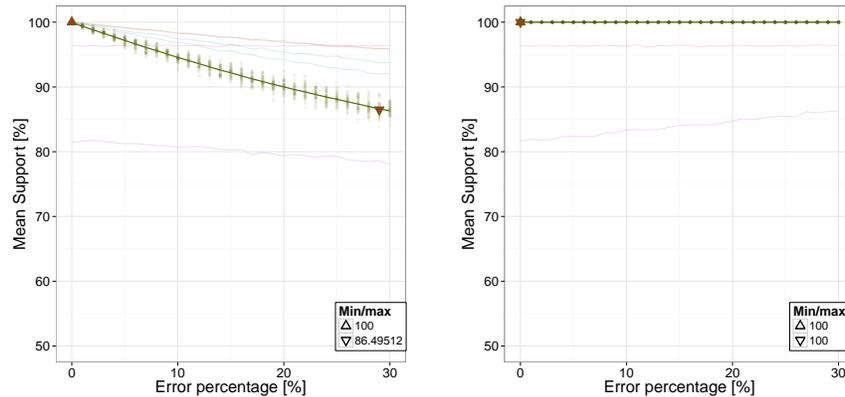Fig. 7: The reaction of $Succession$, w.r.t. different noise types, adopting a log-based noise spreading policy
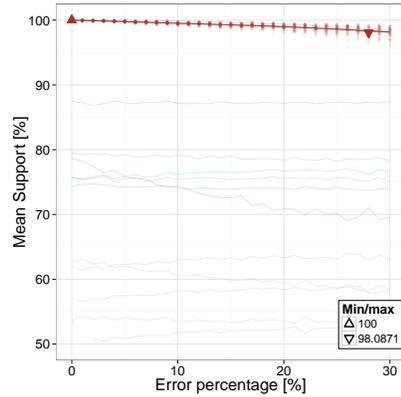
sensitive to the deletion. $Succession(\mathsf{a}, \mathsf{b})$ is the conjunction of the two. Figure 7 shows that this causes the support of $Succession(\mathsf{a}, \mathsf{b})$ to be negatively affected by both noise types, thus supporting **H8**.

5.9 $NotCoExistence$ (**H9**)

Negative relation constraints require that when one of the two referred activities occurs in the trace, the other misses, by definition. Therefore, when any of the two is missing, negative relation constraints will be more likely to be satisfied, either because none of the two is probably in the log, or because at least one of the two misses. This is the reason why Figure 8b shows that support for $NotCoExistence(\mathsf{a}, \mathsf{b})$ remains fixed to its maximum value, when expunging a's. Vice versa, the insertion of spurious a's makes the support decrease, almost linearly w.r.t. the noise injection rate (see Figure 8a). This is due to the fact that the newly inserted a's can fall into traces where a b lay. The shown behavior supports hypothesis **H9**.

5.10 The restriction hierarchy under $CoExistence$ (**H10**)

In the light of the previous discussion, coupling relation constraints are sensitive to both noise types. Therefore, the restriction hierarchy under $CoExistence(\mathsf{a}, \mathsf{b})$ has been chosen to show that descendant constraints are more sensitive than ancestors to the presence of noise (**H10**) (see Figure 1). The applied noise type is the random insertion/deletion of a. Figures 9a to 9d show how the curve drawing the trend of computed support gets steeper, from $CoExistence(\mathsf{a}, \mathsf{b})$ down to $Succession(\mathsf{a}, \mathsf{b})$, $AlternateSuccession(\mathsf{a}, \mathsf{b})$ and $ChainSuccession(\mathsf{a}, \mathsf{b})$. This is because descendants

(a) The trend of the support of $NotCoExistence(\mathsf{a}, \mathsf{b})$, w.r.t. the percentage of spurious a's along the log

(b) The trend of the support of $NotCoExistence(\mathsf{a}, \mathsf{b})$, w.r.t. the percentage of deleted a's along the log

Fig. 8: The reaction of $Succession$, w.r.t. different noise types, adopting a log-based noise spreading policy

in the restriction hierarchy impose stricter conditions than the ancestors to be verified. Figure 9 thus supports hypothesis **H10**.
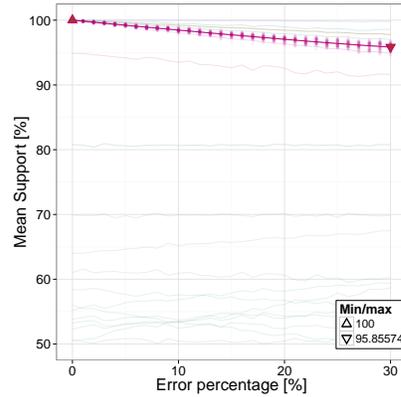
5.11 Summary of experiments

With the tests conducted, we have obtained experimental evidence for all formulated hypotheses. In particular, we have observed that constraints become less resilient to errors, in terms of trend of decreasing support compared to the increasing percentage of introduced noise, along the restriction hierarchy. In general terms, the expunction of activation tasks from traces does not diminish the support of constraints, whereas the insertion of spurious ones can cause traces to become not compliant. Constraints thus tend to be resistant to insertion errors as well as receptive to deletion errors, or vice-versa. Nevertheless, we have also seen that those constraints that are the conjunction of other two, inherit the sensitivity of both to noise. All such reactions to noise reflect the characteristics discussed in precedence, referred to the constraints' definition of activation and target, effects on activities in traces, and interdependencies. This has been extensively explained within the comments to gathered results along this section. Experimental data, though, also show that the effect of noise on support is moderate on most of the constraint types. This supports the suitability of Declare for mining event logs with noise.
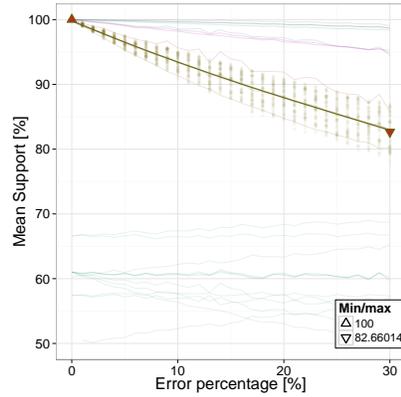
6 Related Work

Process Mining, a.k.a. *Workflow Mining* [1], is the set of techniques that allow the extraction of process descriptions, stemming from a set of recorded real executions (*event*
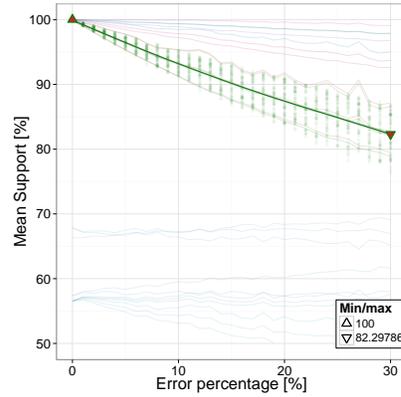
(a) The trend of the support of $CoExistence(\mathsf{a}, \mathsf{b})$, w.r.t. the percentage of random errors on a along the log

(b) The trend of the support of $Succession(\mathsf{a}, \mathsf{b})$, w.r.t. the percentage of random errors on a along the log

(c) The trend of the support of $AlternateSuccession(\mathsf{a}, \mathsf{b})$, w.r.t. the percentage of random errors on a along the log

(d) The trend of the support of $ChainSuccession(\mathsf{a}, \mathsf{b})$, w.r.t. the percentage of random errors on a along the log

Fig. 9: The reaction of coupling relation constraints, w.r.t. random noise types, adopting a log-based noise spreading policy

logs). ProM [18] is one of the most used plug-in based software environments for implementing workflow mining techniques. Process Mining mainly covers three different aspects: process discovery, conformance checking and operational support. The first aims at discovering the process model from logs. Control-flow mining in particular focuses on the causal and sequential relations among activities. The second focuses on the assessment of the compliance of a given process model with event logs, and the

possible enhancement of the process model in this regard. The third is finally meant to assist the enactment of processes at run-time, based on given process models.

From [19] onwards, many techniques have been proposed for the control-flow mining: pure algorithmic (e.g., α algorithm, drawn in [20] and its evolution α^{++} [21]), heuristic (e.g., [5]), genetic (e.g., [7]), etc. A very smart extension to the previous research work was achieved by the two-steps algorithm proposed in [22]. Differently from the former approaches, which typically provide a single process mining step, it splits the computation in two phases: *(i)* the configurable mining of a Transition System (TS) representing the process behavior and *(ii)* the automated construction of a Petri Net bisimilar to the TS [23, 24]. In the field of conformance checking, Fahland et al. [25, 26] have proposed techniques capable of realigning imperative process models to logs.

The need for flexibility in the definition of some types of process, such as the knowledge-intensive processes [27], lead to an alternative to the classical "imperative" approach: the "declarative" approach. Rather than using a procedural language for expressing the allowed sequences of activities ("closed" models), it is based on the description of workflows through the usage of constraints: the idea is that every task can be performed, except what does not respect such constraints ("open" models). The work of van der Aalst et al. [11] showed how the declarative approach (such as the one adopted by Declare [28]) could help in obtaining a fair trade-off between flexibility in managing collaborative processes and support in controlling and assisting the enactment of workflows. Maggi et al. [13] first outlined an algorithm for mining Declare processes implemented in ProM (Declare Miner), based on LTL verification over finite traces. [29] proposed an evolution of [13], to address at the same time the issues of efficiency of the computation and efficacy of the results. Logic-based approaches to declarative process mining have been proposed by [30, 31, 32, 33]. However, they rely on the presence of pre-labeled traces, stating whether they were compliant or not to the correct process execution. For further insight and details, the reader can refer to the work of Montali [34]. Di Ciccio et al. [35, 14, 15] have proposed a further alternative approach, based on heuristic-driven statistical inference over temporal and causal characteristics of the log. De Leoni et al. [36] have first proposed a framework for assessing the conformance of a declarative process to a given log.

In Process Mining, logs are thus usually considered the ground truth from which the process can be discovered. To the best of our knowledge, this is the first study aiming at systematically defining the effect of noise on mined models. In fact, Rogge-Solti et al. [37, 38, 39] have tackled the challenge of repairing logs on the basis of statistical information derived from correct logs and imperative process models. In their study, the process model is known a priori, and the objective is to derive a reliable log from one containing missing or incorrect information. Our analysis, instead, tries to shed light on what would happen when mining a previously unknown process from noisy logs, i.e., when no ground truth is provided.

In the area of control-flow mining, proposed approaches such as [5, 7] for imperative models and [29, 14, 33] for declarative ones, allowed for threshold-based techniques that filter possible outliers out of noisy logs. However, the value for such threshold is left to the choice of the user, who is probably unaware of the best setup. Furthermore,

our studies put in evidence how different constraints react with a different degree of sensitivity to noise. Therefore, a single threshold for all constraints could end up being inaccurate.

First studies on their mutual interdependencies have been reported in [40], [14] and [41]. The first two were aimed at exploiting such connections in order to make the declarative process mining result more readable, i.e., avoiding redundancies in the returned model. The third elaborated on such analysis to refine compliance models and prune irrelevant constraints out. This paper instead builds upon the characteristics of constraints, in order to have theoretical bases on top of which the level of resilience of constraints to noise is estimated. Experimental results actually support our hypotheses.

7 Conclusion

Throughout this paper, we have analyzed how much the errors affecting event logs have an impact on the discovery of declarative processes. In particular, we have formulated ten hypotheses about the resilience and sensitivity of different Declare constraints, and verified our hypotheses on a set of over 160,000 synthetically generated traces. The specific technique used for discovering control flows out of the traces has no impact on the results, therefore the presented study about the effect of noise in event logs has general validity.

Noisy logs are quite natural when applying workflow discovery techniques to unconventional scenarios, such as inferring collaboration processes out of email messages and/or social network interactions, mining of habits in smart environments (in which sensors may provide faulty measures), etc. The more process discovery techniques will be applied in such scenarios, the more existing techniques, which mainly assume error-free logs, should be improved in order to cope with noisy logs. Our study is a preliminary, yet foundational step towards the comprehension of how logs are affected by noise and how this impacts the mined constraints, thus providing a solid basement for the development of new more resilient techniques.

Starting from the present study, we aim at investigating in future work the applicability of the presented analysis to declarative languages other than Declare. We will also conduct a dedicated analysis on the effect on mined constraints of a specific category of noise that van der Spoel et al. name *sequence noise* in [42], i.e., the occurrence of events in a trace in a wrong order. The problem of defining an automated approach for the self-adjustment of user-defined thresholds in process discovery techniques, on the basis of the nature of each discovered constraint, is a future objective too. Intuitively, indeed, a more "robust" constraint should be considered valid in the log (and therefore for the process) if its support exceeds a higher threshold. However, the threshold should be diminished for more "sensitive" ones. We also aim at mixing such an approach with the analysis of different metrics, pertaining to the number of times an event occurred in the log. The intuition is that the more an event is frequent in the log, the less it can be considered subject to errors. Such metrics have been already considered in literature [29] for assessing the relevance of discovered constraints. We want to exploit them for estimating the reliability of constraints in mined processes as well.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Fahland, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of maintainability. In: BPM Workshops. (2009) 477–488
3. Fahland, D., Lübke, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of understandability. In: BMMDS/EMMSAD. (2009) 353–366
4. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: BPM Workshops (1). (2011) 383–394
5. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering **10**(2) (2003) 151–162
6. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In: BPM. (2007) 328–343
7. de Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Discov. **14**(2) (2007) 245–304
8. Di Ciccio, C., Mecella, M., Scannapieco, M., Zardetto, D., Catarci, T.: MailOfMine – analyzing mail messages for mining artful collaborative processes. In: Data-Driven Process Discovery and Analysis. Volume 116. Springer (2012) 55–81
9. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: BPM Workshops. (2006) 169–180
10. Pesic, M.: Constraint-based Workflow Management Systems: Shifting Control to Users. PhD thesis, Technische Universiteit Eindhoven (2008)
11. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D **23**(2) (2009) 99–113
12. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM. (2006) 1–23
13. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM, IEEE (2011) 192–199
14. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM, IEEE (2013) 135–142
15. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Transactions on Management Information Systems (2014)
16. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI. (2013) 854–860
17. Prescher, J., Di Ciccio, C., Mendling, J.: From declarative processes to imperative models. In: SIMPDA. Volume 1293., CEUR-WS.org (2014) 162–173
18. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, E., Weijters, T.: ProM: The process mining toolkit. In: BPM (Demos). (2009)
19. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Advances in Database Technology – EDBT'98. Volume 1377. Springer Berlin / Heidelberg (1998) 467–483
20. 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) (2004) 1128–1142
21. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. **15**(2) (2007) 145–180

22. van der Aalst, W.M.P., , Rubin, V., Verbeek, E., 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** (2010) 87–111
23. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving petri nets from finite transition systems. IEEE Trans. Comput. **47**(8) (1998) 859–882
24. Desel, J., Reisig, W.: The synthesis problem of petri nets. Acta Informatica **33** (1996) 297–315
25. Fahland, D., van der Aalst, W.M.P.: Repairing process models to reflect reality. In: BPM. (2012) 229–245
26. Fahland, D., van der Aalst, W.M.: Model repair – aligning process models to reality. Information Systems (2013)
27. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive Processes: Characteristics, requirements and analysis of contemporary approaches. Journal on Data Semantics (2014) 1–29
28. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: EDOC. (2007) 287–300
29. 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. (2012) 270–285
30. Lamma, E., Mello, P., Riguzzi, F., Storari, S.: Applying inductive logic programming to process mining. In: ILP. (2007) 132–146
31. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. T. Petri Nets and Other Models of Concurrency **2** (2009) 278–295
32. Bellodi, E., Riguzzi, F., Lamma, E.: Probabilistic logic-based process mining. In: CILC. (2010)
33. Bellodi, E., Riguzzi, F., Lamma, E.: Probabilistic declarative process mining. In: KSEM. (2010) 292–303
34. Montali, M.: Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach. Volume 56. Springer (2010)
35. Di Ciccio, C., Mecella, M.: Mining constraints for artful processes. In: BIS. Volume 117., Springer (2012) 11–23
36. de Leoni, M., Maggi, F.M., van der Aalst, W.M.P.: An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. Information Systems (2014)
37. Rogge-Solti, A., Mans, R., van der Aalst, W.M.P., Weske, M.: Repairing event logs using timed process models. In: OTM Workshops. (2013) 705–708
38. Rogge-Solti, A., Mans, R., van der Aalst, W.M.P., Weske, M.: Improving documentation by repairing event logs. In: PoEM. (2013) 129–144
39. Rogge-Solti, A.: Probabilistic Estimation of Unobserved Process Events. PhD thesis, Hasso Plattner Institute at the University of Potsdam, Germany (2014)
40. 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: CAiSE. (2013) 433–448
41. Schunselaar, D.M.M., Maggi, F.M., Sidorova, N.: Patterns for a log-based strengthening of declarative compliance models. In: IFM. (2012) 327–342
42. van der Spoel, S., van Keulen, M., Amrit, C.: Process prediction in noisy data sets: A case study in a dutch hospital. In: SIMPDA. (2012) 60–83