Interactive and Minimal Repair of Declarative Process Models

Carl Corea[1(✉)], Sabine Nagel[1], Jan Mendling[2], and Patrick Delfmann[1]

[1] University of Koblenz-Landau, Koblenz, Germany
{ccorea,snagel,delfmann}@uni-koblenz.de
[2] WU Vienna, Vienna, Austria
jan.mendling@wu.ac.at

Abstract. We present an approach for resolving inconsistencies in declarative process models while guaranteeing a minimal information loss (w.r.t. the number of deleted elements). To this aim, we show how smallest correction sets, i.e., the smallest sets of constraints that need to be deleted in order to resolve inconsistencies, can be computed via an application of Reiter's hitting set theorem. In this context, as deleting certain constraints might be highly sensitive or not plausible in a real-life sense, we extend our approach with functionalities for enabling a close human-in-the-loop interaction, such as prioritizing constraints, as well as metrics that offer modelers insights into the impact of deleting constraints. Furthermore, we implement our approach and show that our inconsistency resolution approach outperforms existing approaches in terms of runtime and information loss in experiments with real-life data sets.

Keywords: Declarative process models · Inconsistency resolution · Minimal correction sets · Hitting sets

1 Introduction

While declarative process models allow to specify flexible processes, the logic-based nature of declarative constraints leaves models prone to logical inconsistencies [3,5,12]. As a simple example, consider the declarative process model D_1 (we will formalize syntax and semantics later), defined via

$$D_1 = \{\textsc{Init}(a), \textsc{Response}(a,b), \textsc{Response}(b,c), \textsc{NotResponse}(a,c)\},$$

with the intuitive meaning that 1) a process must start with a task a, 2) a task a must be eventually followed by a task b, which must 3) eventually be followed by a task c, and 4) a task a must never be followed by a task c. D_1 is inconsistent, as it demands contradictory reactions to the occurrence of task a. Therefore, the

Part of the research project "Handling Inconsistencies in Business Process Modeling", funded by the German Research Association (reference number: DE1983/9-1).

A. Polyvyanyy et al. (Eds.): BPM Forum 2021, LNBIP 427, pp. 3–19, 2021.
https://doi.org/10.1007/978-3-030-85440-9_1

shown model is unsatisfiable and the inconsistency must be resolved in order to use the model for its intended purpose of governing compliant company behavior. Mind that contradictions in models of real-life complexity are difficult to spot, since they often arise from combinations of several constraints. Here, modelers need to be supported at design-time in order to resolve such inconsistencies.

In response, recent works [3,4] have presented methods for inconsistency repair. While this is a clear benefit for companies, a current limitation is that those approaches cannot guarantee a minimal information loss, i.e., that only the smallest possible set of constraints is actually deleted. For example, as the approach in [4] is only an approximation algorithm, it can easily occur that twice as many constraints are deleted compared to the optimum. Here, new methods are needed to counteract this risk of unnecessary information loss (**R1**).

Furthermore, existing approaches are geared towards *automated* resolution. However, as deleting constraints might be highly sensitive, automated approaches might yield implausible results. For example, if a system computes that inconsistency can be resolved by deleting only one constraint, this result is of no use if that constraint is business-critical and must be retained. This calls for a close integration of human experts in determining suitable resolution strategies (**R2**).

To address the research problems raised above, the contribution of this work is consequently twofold:

- **(R1) Minimal Information Loss.** We show how *minimal correction sets* (MCS) can be computed via a reduction to a hitting set enumeration problem, i.e., an application of *Reiter's hitting set theorem* [16]. This allows to compute the cardinality-smallest set(s) of constraints that have to be deleted to restore consistency, i.e., inconsistency is resolved while guaranteeing the smallest possible information loss w.r.t. the number of deleted constraints.
- **(R2) Human-in-the-loop Integration.** We extend our computation approach with a human-in-the-loop perspective, allowing users to impact the computation of minimal correction sets, e.g., by prioritizing certain constraints. Also, to support modelers in understanding different resolution options, novel metrics are presented that help modelers understand the impact of their choices, e.g., the effectiveness of customized resolution strategies.

The remainder of this work is structured as follows. In Sect. 2, we provide preliminaries on declarative process models and discuss limitations of related works addressed in this paper. Section 3 presents our approach for computing MCS via hitting sets. Our approach is then implemented and evaluated in experiments with real-life data sets in Sect. 4. We conclude in Sect. 5.

2 Preliminaries and Related Works

In this section, we discuss preliminaries on declarative process models, the notion of inconsistency in declarative models, as well as related work.

2.1 Declarative Process Models

Declarative process models are constituted of a set of constraints, that confine the allowed behavior of company activities [14]. As opposed to traditional process models, this allows for a high degree of flexibility within these set bounds.

Definition 1 (Declarative Process Models). *A declarative process model is a tuple* $\boldsymbol{M} = (\boldsymbol{A}, \boldsymbol{T}, \boldsymbol{C})$*, where* $\boldsymbol{A}$ *is a set of activities,* $\boldsymbol{T}$ *is a set of constraint templates, and* $\boldsymbol{C}$ *is the set of actual constraints, which instantiate the template elements in* $\boldsymbol{T}$ *with tasks in* $\boldsymbol{A}$*. We denote* $\mathfrak{M}$ *as the universe of all such models.*

In this paper, we consider DECLARE [14], which is a declarative process modeling language and notation. DECLARE offers easy-to-use predefined templates, that can be parametrized with company activities in order to specify declarative constraints. For example, the DECLARE constraint RESPONSE(a, b) states that if a task a occurs, it must be eventually followed by a task b. An advantage of DECLARE is that the semantics of template types can be defined with temporal logic, allowing to exploit the amenities of temporal logic checking, while hiding complexity from the end user. We define the semantics of DECLARE constraints with the temporal logic LTL_p [13]. An LTL_p formula is given by the grammar

$$\varphi ::= a | (\neg\varphi) | (\varphi_1 \wedge \varphi_2) | (\bigcirc\varphi) | (\varphi_1 \mathbf{U} \varphi_2) | (\ominus\varphi) | (\varphi_1 \mathbf{S} \varphi_2).$$

Each formula is built from atomic propositions $\in \mathbf{A}$ (relative to a declarative process model), and is closed under the boolean connectives, the unary temporal operators $\bigcirc$ (next) and $\ominus$ (previous), and the binary temporal operators $\mathbf{U}$ (until) and $\mathbf{S}$ (since). For any such formula, the semantics is then defined relative to a trace t. Due to space limitations, we omit a presentation of the concrete semantics and refer the reader to [5]. Based on such LTL_p formulae, the semantics of individual DECLARE constraints can then be defined. A standard set of DECLARE templates and corresponding semantics can be found in [3].

2.2 On the Notion of Inconsistency in Declarative Process Models

Based on the LTL_p semantics, it can be verified whether a constraint c is satisfied by a trace t by checking if c evaluates to true over t [12]. Given a declarative model $\mathbf{M}$, let $\mathfrak{T}^{\mathbf{A}}$ denote the set of all possible sequences that can be constructed based on the activities $\mathbf{A} \in \mathbf{M}$. An evaluation of a declarative model M over a trace t is thus a function $\epsilon : \mathfrak{M} \times \mathfrak{T}^{\mathbf{A}} \to \{\top, \bot\}$, defined via

$$\epsilon(\mathbf{M}, t) = \begin{cases} \top \text{ if for all } c \in \mathbf{C} : c \text{ evaluates to true for } t \\ \bot \text{ otherwise} \end{cases}$$

We define the *language* $\mathcal{L}$ of a model $\mathbf{M}$ as all traces that satisfy $\mathbf{M}$, i.e.,

$$\mathcal{L}(\mathbf{M}) = \{t \in \mathfrak{T}^{\mathbf{A}} \mid \epsilon(\mathbf{M}, t) = \top\}.$$

Thus, an inconsistent declarative process model is a model where $\mathcal{L} = \emptyset$, i.e., it cannot accept any traces.

Consider the exemplary declarative process models M_1, M_2, defined via

$$M_1 = \{\textsc{Init}(a), \textsc{Response}(a,b), \textsc{NotResponse}(a,b)\}$$
$$M_2 = \{\textsc{Response}(a,b), \textsc{NotResponse}(a,b)\}$$

In M_1, the constraint Init(a) confines that a trace must start with an event a. The remaining two constraints Response(a,b) and NotResponse(a,b) would also have to be satisfied in the same trace. As the latter two constraints are contradictory, there can exist no trace that satisfies M_1, i.e. $\mathcal{L}(M_1) = \emptyset$.

The notion of classical inconsistency was recently extended with the concept of *quasi-inconsistency* [5,6]. In the declarative model M_2 there exist two "contradictory" constraints, but there is no confinement regarding the occurrence of an activity a. In result, M_2 can accept an arbitrary amount of traces, i.e., any trace that does not contain the activity a . Thus, as $\mathcal{L}(M_2) \neq \emptyset$, M_2 is not classically inconsistent. Yet, the constraints in M_2 are highly problematic, as they will always be activated together, but yield contradictory conclusions. Following [5], M_2 is, therefore, *quasi*-inconsistent. For a formal definition, we first need some notation on reactive constraints and constraint activation.

Considering Declare constraints such as $\textsc{Response}(a,b)$, we see that such constraints describe a form of cause and reaction relation between the tasks a and b, i.e., given an activity a, the reaction should be b. Thus, following works such as [2], declarative constraints can be rewritten as so-called *reactive constraints*.

Definition 2 (Reactive Constraints [2]). *Given a declarative process model $\boldsymbol{M} = (\boldsymbol{A}, \boldsymbol{T}, \boldsymbol{C})$, let $\alpha \in \boldsymbol{A}$ be an activation and φ be an LTL_p formula over $\boldsymbol{A}$. Then, a reactive constraint (RCon) Ψ is a pair (α, φ), denoted as $\Psi = \alpha \Rightarrow \varphi$.*

As an example, $\textsc{Response}(a,b)$ can be rewritten as $a \Rightarrow \Diamond b$. For a constraint c, we denote $A_a(c)$ and $A_r(c)$ as the respective activating and reacting activities. For a declarative process model, if the reaction of a constraint c is an activation to a constraint c', we also say that the $A_a(c)$ transitively activates c' [5].

Consequently, we define quasi-inconsistency and issues as follows.

Definition 3 (Quasi-Inconsistent Subset [5]). *For a constraint set C, a quasi-inconsistent subset is defined as a pair $(\boldsymbol{A}, \boldsymbol{C})$, s.t. $\boldsymbol{C} \subseteq C$, $\boldsymbol{A}$ activates $\boldsymbol{C}$ and $\boldsymbol{A} \cup \boldsymbol{C} \models \bot$.*

To clarify, we consider constraints that will a) always be activated together, and b) yield an inconsistency, should they be activated.

Example 1. We recall M_2. Then, we have the quasi-inconsistent subset q, with

$$q = (\{a\}, \{\textsc{Response}(a,b), \textsc{NotResponse}(a,b)\})$$

For a model M, the set $\mathsf{MIS}(M)$ is the set of minimal quasi-inconsistent subsets, i.e., subsets where removing any constraint resolves the issue.

In this section, we have introduced the concepts of inconsistency and quasi-inconsistency. We acknowledge that [12] also introduce the notion of conflicting sets, which are conflicting constraints that were activated by a given trace. However, in this work, we only consider the introduced forms of "inconsistencies", i.e., "inconsistencies" that arise independent of specific traces and need to be resolved at design-time. For the remainder of this paper, we will refer to both quasi-inconsistencies and "classical" inconsistencies as "issues", for readability. We also denote MIS as all minimal quasi-inconsistent subsets and all "classical" minimal inconsistent subsets[1] by a slight misuse of notation. Basically, we are interested in all (potential) inconsistencies, as these issues need to be resolved.

2.3 Related Works and Contributions

Consider the model M_3 in Fig. 1, which has six issues that need to be resolved. Here, several works have presented means for inconsistency resolution [3,4,12].

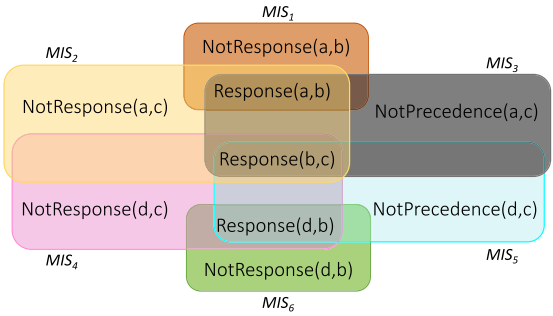

Fig. 1. Exemplary model M_3, containing six minimal issues (highlighted)

A central approach is to start by deleting constraints that have the highest number of overlaps, as this maximizes the number of minimal issues being resolved by deleting one constraint. Hence, approaches such as [4,12] would delete the constraint $\textsc{Response}(b, c)$, as it is part of the most overlapping sets (4). This resolves all issues but MIS_1 and MIS_6, which means two more constraints have to be deleted (i.e., three deletions in total). However, this is not the optimal solution, as inconsistency could be resolved by deleting only two constraints ($\textsc{Response}(a, b)$, $\textsc{Response}(d, b)$) (these two constraints are part of less overlaps, still, it would be an optimal solution to start with deleting these constraints). This shows that while the existing solutions produce "minimal" solutions/repair sets (in terms of set inclusion), they do not always yield the *cardinality-smallest* solutions, i.e., they can run into local optima due to the

[1] Given a model $\mathbf{M}$ and a corresponding constraint set $\mathbf{C}$, a minimal inconsistent subset is defined as a set $m \subseteq \mathbf{C}$, s.t. $\mathcal{L}(m) = \emptyset$ and $\nexists m' \subset m$ with $\mathcal{L}(m') = \emptyset$.

approach designs as greedy algorithms. This can result in unnecessary information loss[2]. In this work, we present an approach to compute the cardinality-smallest set(s) of constraints that have to be deleted to restore consistency, i.e., our approach guarantees a minimal information loss w.r.t. the number of deleted constraints. This is achieved by means of so-called *hitting sets*. Please note that the mentioned existing approaches [3,4,12] do not use hitting sets to resolve inconsistencies (cf. above for the resulting limitations). Furthermore, existing approaches [3,4] are geared towards *automated* resolution. However, as deleting certain constraints might be highly sensitive, such automated approaches might yield implausible results in a real-life sense. Here, new results are needed that extend inconsistency resolution with a human-in-the-loop perspective. In this regard, we, therefore, present novel means, that allow the user to impact the computation of resolution strategies, while being supported with insights to understand the impact of different resolution options.

In general, our work is also related to to other works that use hitting sets for the diagnosis and repair of knowledge representation formalisms. This idea by Reiter [16] has been applied in various other logical formalisms, e.g., first-order logic [7], propositional logic [10] or non-monotonic logics [1]. Here, this work is the first to investigate Reiter's hitting set theorem in the context of declarative process models. Also, this works introduces novel concepts towards the "customization" of hitting set computation according to company needs, e.g., allowing users to define preference relations of rules as a basis for computation.

We also acknowledge that there are works investigating inconsistency resolution in declarative processes at run-time [11,12], which is beyond the scope of this report, as we focus on design-time analysis of declarative models.

3 Minimal and Interactive Repair

In this work, we present an approach to resolve inconsistencies (i.e., minimal issues) in declarative process models by deleting (the smallest possible set of) constraints. Importantly, as the plausibility of automatedly computed resolution operations has to be carefully considered, the approach is geared towards a semi-automated resolution, allowing humans to understand, evaluate and select suitable resolution strategies. Our approach overview is shown in Fig. 2. At first, all inconsistent subsets are detected. Then, viable repair operations are computed based on minimal correction sets, i.e., minimal hitting sets (cf. Sections 3.1, 3.2). Last, to select suitable repair operations, our approach provides metrics and further means to support modelers in evaluating possible solutions (cf. Section 3.3).

[2] The approach in [3] would behave analogously, except not by deleting constraints but iteratively building a new, maximally consistent model, which could also "drop" more constraints than necessary.

Fig. 2. Approach overview

3.1 Inconsistency Resolution Based on Minimal Correction Sets

For determining resolution strategies, experts need to identify which constraints should be removed from a model to resolve all minimal issues. To avoid unnecessary information loss, it is especially interesting to identify minimal sets of constraints that can be removed to resolve inconsistency. In the following, we refer to such sets as *minimal correction sets*, i.e., a set of constraints that – when deleted – resolves all minimal issues, and is minimal in terms of set-inclusion.

Definition 4 (Minimal Correction Sets). *Given a constraint set* $\boldsymbol{C}$, $C \subseteq \boldsymbol{C}$ *is a minimal correction set of* $\boldsymbol{C}$, *if* $\mathit{MIS}(\boldsymbol{C} \backslash C) = \emptyset$, *and* $\forall C' \subset C : \mathit{MIS}(\boldsymbol{C} \backslash C') \neq \emptyset$. *Let* X *be a constraint set or a declarative process model, we denote* $\mathit{MCS}(X)$ *as the set of minimal correction sets for the constraints in* X.

Such minimal correction sets can be computed by considering so-called hitting sets, following [16].

In set theory, a set H is called a *hitting set* of a set of sets $S = \{S_1, ..., S_n\}$ iff $H \cap S_i \neq \emptyset$ for every $i = 1, ..., n$.

Example 2. Consider the set of sets $S' = \{\{1, 2, 3\}, \{1, 3, 5\}, \{4, 5, 6\}\}$. Furthermore, consider the following exemplary sets $H_1 - H_3$, defined via

$$H_1 = \{1, 4\} \qquad H_2 = \{1, 2, 3, 4, 5, 6\} \qquad H_3 = \{1, 2, 3\}.$$

H_1 and H_2 are hitting sets w.r.t. S', as $H_{1/2} \cap S_i \neq \emptyset$ for all $i = 1..3$. However, we see that H_2 is not minimal, as we could remove several elements and H_2 would still be a hitting set. Also, we see that H_3 is not a hitting set for S', as it has no elements in common with the last inner set of S'.

To compute minimal correction sets, we consequently propose to consider minimal hitting sets, by adapting Reiter's hitting set theorem as follows:

Theorem 1 (Hitting Set-Based MCS (adapted from [16])). *Given a constraint set* $\boldsymbol{C}$, $C \subseteq \boldsymbol{C}$ *is a minimal correction set of* $\boldsymbol{C}$ *iff* C *is a minimal hitting set w.r.t.* $\mathit{MIS}(C)$.

Example 3. Consider the following constraint set M_5, defined via

$$M_5 = \begin{array}{lll} \textsc{NotResponse}(a, b) & \textsc{ChainResponse}(a, b) & \textsc{Response}(a, b) \\ \textsc{NotResponse}(c, d) & \textsc{ChainResponse}(c, d) & \textsc{Response}(c, d) \end{array}$$

Then we have:

$$\begin{aligned}\mathsf{MIS}(M_5) &= \{\mu_1, \mu_2, \mu_3, \mu_4\}\\ \mu_1 &= \{\textsc{NotResponse}(a,b), \textsc{ChainResponse}(a,b)\}\\ \mu_2 &= \{\textsc{NotResponse}(a,b), \textsc{Response}(a,b)\}\\ \mu_3 &= \{\textsc{NotResponse}(c,d), \textsc{ChainResponse}(c,d)\}\\ \mu_4 &= \{\textsc{NotResponse}(c,d), \textsc{Response}(c,d)\}\end{aligned}$$

Consider the exemplary hitting sets $H_4 - H_6$, defined via

$$\begin{aligned}H_4 &= \{\textsc{NotResponse}(a,b), \textsc{NotResponse}(c,d)\}\\ H_5 &= \{\textsc{NotResponse}(c,d), \textsc{ChainResponse}(a,b), \textsc{Response}(a,b\}\\ H_6 &= \{\textsc{NotResponse}(a,b), \textsc{NotResponse}(c,d), \textsc{ChainResponse}(a,b)\}.\end{aligned}$$

$H_4 - H_6$ are hitting sets for $\mathsf{MIS}(M_5)$. However, H_6 is not minimal, as we could remove $\textsc{ChainResponse}(a,b)$ and H_6 would still be a hitting set. Correspondingly, only H_4 and H_5 are minimal correction sets for M_5.

While minimality of hitting sets enforces that the hitting sets are not reducible, this does not mean "smallest" per se. In the scope of deleting only the smallest possible amount of constraints, it could, however, be interesting to consider the cardinality-smallest minimal correction sets.

Definition 5 (Smallest viable repair). *Given a declarative process model* $\boldsymbol{M}$ *and the set of minimal correction sets* $\mathsf{MCS}(\boldsymbol{M})$*, the set of smallest minimal correction sets is defined as* $\mathsf{MCS}_{MIN}(\boldsymbol{M}) = \{S \subseteq \mathsf{MCS}(\boldsymbol{M}) : |S| = min(\mathsf{MCS}(\boldsymbol{M}))\}$.

Corollary 1. *The smallest possible number of constraints that have to be deleted from a declarative model* $\boldsymbol{M}$ *to resolve all minimal issues is* $min(\mathsf{MCS}(\boldsymbol{M}))$.

In turn, this allows to present users a list of all possible smallest viable repairs.

In this section, we have presented means to compute minimal correction sets based on hitting set enumeration. While this approach can be used to identify the cardinality-smallest sets of constraints for inconsistency resolution, there is still a major conceptual problem, namely that of plausibility: Even if algorithms can compute a (smallest) set of constraints that could be deleted to resolve the inconsistency, this does not mean that these solutions are plausible in a real-life sense. We therefore propose to extend inconsistency resolution with a human-in-the-loop perspective.

3.2 Human-in-the-Loop Features

To allow for a human-in-the-loop integration, repair operations should not be automatedly applied, but rather recommended to the user. Also, experts should be able to influence or constrain the actual computation of viable correction sets. Therefore, we raise the following two requirements for a human-in-the-loop integration in inconsistency resolution:

1. To compute a recommendation of repair operations, it should be possible to rank minimal corrections, e.g., by an arbitrary quality metric.
2. The computation of which constraints to delete should be relative to a user-definable configuration, e.g., by allowing to prohibit the deletion of certain constraints or provide superiority relations.

As motivated in Sect. 2.3, related work on inconsistency resolution in declarative process models cannot satisfy these requirements. We consequently address these issues in the following.

First, via Definition 5, correction sets can already be ranked by their size. Next to the correction set size, this can be generalized for arbitrary measures γ to allow for a general ranking of correction sets.

Definition 6 (γ-Repair Ranking). *Let a declarative process model $\boldsymbol{M}$ and the set of minimal correction sets $\mathsf{MCS}(\boldsymbol{M})$. Then, considering a measure $\gamma : \mathsf{MCS}(\boldsymbol{M}) \rightarrow \mathbb{R}^{\infty}_{\geq 0}$ that assigns to a minimal correction set a non-negative numerical value, a γ-repair ranking over all $m \in \mathsf{MCS}(\boldsymbol{M})$ is any ranking $\langle m_1, ..., m_n \rangle$ that satisfies $\gamma(m_1) \leq ... \leq \gamma(m_n)$.*

Thus, this ranking can sort all MCS relative to a measure γ. Importantly, the semantics of the ranking are defined such that the ranking sorts all minimal correction sets from "best" to "worst" option, relative to γ.

Continuing, it should be possible to confine the deletion of certain constraints in order to leverage the computation of plausible correction sets following requirement 2. Here, we consider whitelists, i.e., a whitelist $W \subseteq M$ is a list of constraints not to be deleted. Intuitively, a whitelist can "block" certain minimal correction sets, as these could include whitelisted constraints. Therefore, it is necessary to provide means to present users with the (next) best viable repair, while also considering the whitelist. To this aim, we adapt the notion of smallest viable repairs and extend this for arbitrary quality measures. This allows to determine the set of best possible γ-repairs relative to a whitelist.

Definition 7 (Best Viable γ-Repair). *Given a declarative process model $\boldsymbol{M}$, a measure $\gamma : \mathsf{MCS}(\boldsymbol{M}) \rightarrow \mathbb{R}^{\infty}_{\geq 0}$ that assigns to a minimal correction set a non-negative numerical value, and a whitelist $W \subseteq \boldsymbol{M}$, the best viable γ-repair w.r.t. W is defined as $\mathsf{MCS}^{W}_{\gamma}(\boldsymbol{M}) = \{S \subseteq \mathsf{MCS}(\boldsymbol{M}) \mid \forall s \in S : \nexists x \notin S \text{ s.t. } \gamma(x) < \gamma(s), \text{ and } s \notin W\}$.*

The best viable γ-repair thus finds the "best" minimal correction sets w.r.t. a measure γ, (e.g., the correction set size) while also considering the whitelist. Given a declarative process model $\mathbf{M}$ and a measure γ, we denote the set of best viable γ-repairs as BCS_{γ}.

Example 4. Consider the minimal correction sets $MCS_1 - MCS_4$:

$$MCS_1 = \{\textbf{NotResponse(a,b)}, \textsc{NotResponse}(c,d)\}$$
$$MCS_2 = \{\textbf{NotResponse(a,b)}, \textsc{ChainResponse}(c,d), \{\textsc{Response}(c,d)\}.$$
$$MCS_3 = \{\textsc{ChainResponse}(a,b), \textsc{Response}(a,b), \textsc{ChainResponse}(c,d), \textsc{Response}(c,d)\}.$$
$$MCS_4 = \{\textsc{NotResponse}(c,d), \textsc{ChainResponse}(a,b), \textsc{Response}(a,b)\}$$

Assume a whitelist $W = \{\textbf{NotResponse(a,b)}\}$, i.e., this constraint should not be deleted. This prohibits to select the correction sets MCS_1 and MCS_2, as deleting the corresponding constraints would violate the whitelist constraints. Considering again the correction set sizes, i.e., $\gamma(M) = |M|$, the best viable γ-repair BCS_γ would therefore be $\mathsf{BCS}_\gamma = \{MCS_4\}$. Note that MCS_3 is not part of the best viable repair: While it satisfies the whitelist constraints, it does not satisfy the first condition that there should be no other remaining correction sets with lower γ value (here: set size).

Intuitively, a large whitelist might overly restrict the set of viable γ-repairs. Thus, next to entirely blocking certain constraints in a binary manner, modelers should rather also have the possibility of a more fine-grained configuration that allows for more flexibility. Therefore, we propose to allow users to weight constraints, and calculate the fitting correction sets accordingly.

Definition 8 (Weighted Declarative Process Model). *A weighted declarative process model is a tuple $\boldsymbol{M} = (\boldsymbol{A}, \boldsymbol{T}, \boldsymbol{C}, w)$, where $\boldsymbol{A}$ is a set of activities, $\boldsymbol{T}$ is a set of constraint templates, $\boldsymbol{C}$ is the set of actual constraints, which instantiate the template elements in $\boldsymbol{T}$ with tasks in $\boldsymbol{A}$, and $w : \boldsymbol{C} \rightarrow \mathbb{R}^{\infty}_{\geq 0}$ is a weighting function for constraints.*

A weighted declarative process model extends declarative models with definable constraint weights. In order to allow modeling superiority relations between constraints, arbitrary weights can be defined manually or derived automatically. This allows to compute weighted correction sets.

Definition 9 (Correction Set Weight). *Given a weighted declarative process model $\boldsymbol{M} = (\boldsymbol{A}, \boldsymbol{T}, \boldsymbol{C}, w)$ and the corresponding minimal correction sets $\mathsf{MCS}(\boldsymbol{C})$, the weight $w(M)$ of any $M \in \mathsf{MCS}(\boldsymbol{C})$ is defined as $\sum_{c \in M} w(c)$.*

As the correction set weight can essentially be used as an assessment function γ for correction sets, it is therefore possible to compute a repair ranking via Definition 6 using the correction set weights, i.e., a larger correction set weight indicates a higher "cost" to remove this correction set.

Example 5. We recall the correction sets $MCS_1 - MCS_4$ from Example 4. Furthermore, assume the expert has determined the following constraint weights:

$$\textsc{NotResponse}(a,b) = 3 \qquad \textsc{ChainResponse}(a,b) = 1$$
$$\textsc{Response}(a,b) = 1 \qquad \textsc{NotResponse}(c,d) = 3$$
$$\textsc{ChainResponse}(c,d) = 1 \qquad \textsc{Response}(c,d) = 1$$

In the example, the expert has prioritized two constraints. In turn, we have that

$$w(MCS_1) = 3 + 3 = 6 \qquad w(MCS_2) = 3 + 1 + 1 = 5$$
$$w(MCS_3) = 1 + 1 + 1 + 1 = 4 \qquad w(MCS_4) = 3 + 1 + 1 = 5.$$

We see that while MCS_1 is smaller than MCS_2-MCS_4 (and in general it would be favorable to select smaller correction sets), the "costs" of selecting MCS_1 are higher as this would mean to delete two highly prioritized constraints. This information can thus be used for considering the trade-off between selecting correction sets of smaller size or keeping constraints of higher priority.

3.3 Understanding Support

In the previous section, we introduced means to enable a close human-in-the-loop integration. While this allows users to provide a fine-grained configuration for correction set computation, it also places an increased pressure on the human to ultimately choose which correction sets to select. Given that there can easily be multiple best viable repairs, users must be supported in understanding the consequences of choosing between these correction sets, in order to determine suitable resolution strategies. We, therefore, propose a metric to assess the quality of a correction set selection, as well as means to understand the behavioral changes resulting from applying a certain correction set, explained as follows.

Assuming a modeler is comparing the sizes of different correction sets, a smaller correction set can in general be considered as better than a larger correction set. However, it might not be plausible to apply the smallest correction set, as a user might deem that the respective constraints must be kept. Thus, the user might be forced to select a larger correction set. However, if the next viable correction set is too large, a user might have to carefully consider whether keeping certain constraints is "worth" deleting a (much) higher number of other constraints. Especially when considering correction set measures other than the size, e.g., complex correction set weights, deciding and balancing such a decision is a difficult task for experts. Here, we propose to compute distance-based metrics to support users in understanding the trade-off between different choices.

For a declarative model $\mathbf{M}$, consider any quality measure $\gamma : \mathsf{MCS}(\mathbf{M}) \rightarrow \mathbb{R}^{\infty}_{\geq 0}$, where a higher value indicates a higher "cost" of removing the individual correction set. Then, the smallest possible cost is the minimum over all best viable γ-repairs $\mathsf{BCS}_{\gamma}(\mathbf{M})$, i.e., the smallest possible cost (w.r.t. γ) $\mathrm{MIN}_{\gamma}(\mathbf{M}) = min_{B \in \mathsf{BCS}_{\gamma}(\mathbf{M})}\gamma(B)$. This allows to compute an absolute distance metric for assessing arbitrary correction sets.

Definition 10 ((Distance-based) Additional Correction Set Costs from Baseline). *Given a declarative process model $\boldsymbol{M}$ and a correction set measure γ, the additional cost $c_{add_{\gamma}}$ of any correction set M relative to the smallest possible cost is defined as $c_{add_{\gamma}}(\boldsymbol{M}, M) = \gamma(M) - \mathrm{MIN}_{\gamma}(\boldsymbol{M})$.*

This metric provides an assessment of correction sets for determining the additional costs relative to the smallest possible costs w.r.t. γ .

Example 6. We recall the correction set $MCS_1 - MCS_4$ from Example 4. Judging from a set size perspective, i.e., $\gamma(M) = |M|$, we have that $\gamma(MCS_1) = 2$, $\gamma(MCS_2) = \gamma(MCS_4) = 3$ and $\gamma(MCS_3) = 4$. Thus, the smallest possible costs MIN_γ are 2. Here, the additional costs of selecting MCS_2 or MCS_4 would be 1, and 2 for selecting MCS_3. If the additional costs from the baseline become too large, they might outweigh the costs of keeping certain constraints. The proposed distance-based metric can thus support users in making an informed decision as to whether the whitelist or rule weights should be altered.

While the distance-based additional cost metric can produce valuable insights, an ultimate selection of specific correction sets might not only depend on numeric factors such as the number of deleted constraints, but rather on the actual behavioral consequences following the deletion of a specific correction set. Here, experts need to be supported in understanding the behavioral consequences of the different available options. To this aim, we propose so-called fragment-based language profiles in order to present modelers the *exact difference in behavioral changes* for different resolution options.

To understand behavioral changes, given a declarative model **M**, one could theoretically compute the language of **M** and the language of any **M'** derived by deleting a set of constraints from **M**. Then, one could simply compare the languages of **M** and **M'** in order to identify all behavioral changes, i.e., differences in accepted traces. However, this is not feasible, as the languages can be infinitely large. Instead, we propose to consider only fragments of the possible languages, explained as follows.

Consider the constraint set $\mathbf{M} = \{\text{Response}(a, b), \text{NotResponse}(a, b)\}$, which is quasi-inconsistent. Then, consider a correction set $C = \{\text{Response}(a, b)\}$, indicating that this constraint could be deleted to resolve the issue in **M**. The question then arises which behavioral changes would follow deleting this constraint, i.e., given a model $\mathbf{M'} = \mathbf{M} \setminus C$, what would be the difference $\mathcal{L}(M') - \mathcal{L}(M)$? Regardless of any actual or possible trace for **M**, in the example, any changes in language for **M'** only apply for any trace that contains a or b. For instance, a trace cde would behave identically for **M** and **M'**, whereas the trace a could not satisfy **M** but possibly satisfy **M'**. Therefore, only the permutations of the distinct events within the correction set constraints need to be considered. For example, for the above correction set $C = \{\text{Response}(a, b)\}$, the distinct events are a, b, so all possible event combinations, i.e., trace fragments, would be a, b, ab, ba. By evaluating these fragments against the original model **M** and a corresponding altered model **M'**, changes in the different language profiles following a deletion of the correction set relative to the original rule base can be identified, as shown in Fig. 3. By deleting the correction set $C = \{\text{Response}(a, b)\}$, the two trace fragments a and ba would become possible (which were not possible before). The expert can thus inspect whether this is deemed as appropriate behavior. For example, if the sequence ba should never occur in the company processes, the expert could see that the current correction set would result in unwanted behavior, and seek for a different solution.

Trace Fragments	M	M' = M\C
a	✗	✓
b	✓	✓
ab	✗	✗
ba	✗	✓

Fig. 3. Fragment-based visualization of behavioral changes

To compute the actual behavioral differences between the models, we encode the satisfiability of the individual trace fragments relative to a model as a so-called *language profile*. Let a finite list of trace fragments be $t = (t_1, ..., t_n)$. Then, for a model M, a language profile is a $1 \times$ n matrix

$$\lambda M^t = \begin{bmatrix} \lambda t_1 \\ \vdots \\ \lambda t_n \end{bmatrix}, \text{ with every } \lambda t_i = \begin{cases} 1, \text{ if } M \models t_i \\ 0 \text{ otherwise} \end{cases}$$

Definition 11 (Behavioral Change Profile). *Given a set of trace fragments t and two models M, M', the behavioral change profile is defined as $\lambda M^t - \lambda M'^t$, where an index 1 indicates a change in behavior, and an index 0 indicates an identical behavior of the two models for the corresponding trace fragment.*

Example 7. Consider the trace fragments and models M, M' shown in Fig. 3. Then, behavioral change profile $b = [1, 0, 0, 1]$ (transposed), indicating a behavior change for the trace fragments a and ba.

Considering behavioral change profiles provides important insights to modelers, as it enables experts to understand the changes in behavior between two models, e.g., to inspect whether deleting certain constraints could lead to unwanted or non-compliant process behavior.

A possible limitation of applying behavioral change profiles could be the amount of fragments that need to be considered. For correction sets and the number of contained events, the number of permutations/fragments that need to be computed could grow factorial. However, this is only a problem if the correction sets would contain a very high number of constraints. Based on the overall goal to mitigate unnecessary deletions of constraints, our approach intuitively favors smaller correction sets by design. To anticipate our empirical results from Sect. 4, we also found that correction sets were generally small for real-life data sets, i.e., 3–5 constraints, of which only the distinct events have to be considered. Also, only fragments that contain at least one activation over the constraints in the correction set need to be considered, as traces without an activation will not be affected by the deletion of the correction set. Therefore, the number of fragments can be further confined, e.g., in Fig. 3, the trace b would technically

not need to be considered. Thus, the computation of behavioral change profiles is feasible for smaller correction sets, e.g., as in the analyzed real-life data sets (cf. Section 4). For settings with a large number of fragments, efficient algorithms should be investigated in future work.

4 Tool Support and Evaluation

We implemented our inconsistency repair approach as a proof-of-concept. The project can be viewed online[3]. Also, an online-demo is available[4]. Here, users can upload their declarative models, view the model as a 3d-graph, scan for any minimal issues and compute minimal correction sets directly in the browser. The computation of minimal issues is based on our previous work in [5].

At the core, our approach is strongly dependent on the performance of the hitting set enumeration. Fortunately, this computation task has gained recent momentum and powerful enumeration algorithms are available [8,15]. In our implementation, we integrated the PySat library[5] for computing hitting sets, which has been broadly studied and evaluated. However, these libraries have been mostly tested in more theoretical contexts, such as SAT solving.

To evaluate the plausibility of applying our proof-of-concept in a BPM setting, we conducted runtime experiments with real-life data-sets from the Business Process Intelligence (BPI) Challenge[6]. Here, we used data sets from the last four years, i.e., logs of a loan application process (BPI 17, 31.509 cases), a governmental funding process (BPI 18, 43.809 cases), a purchase order process (BPI 19, 251.734 cases), and a domestic travel expense refund process (BPI 20, 10.500 cases). From these logs, we mined DECLARE models using the declarative process discovery tool Minerful [3]. As mining parameters, we selected a support factor of 75%, as well as confidence and interest factors of 12.5%, following the experiment setup in [3]. Note that as shown in [5], these parameters allow for contradicting constraints to be added to the initial model, which is needed for our evaluation. In the future, it might be interesting to further examine the effects of mining parameters on the resulting inconsistencies and repeat the evaluation with different parameter configurations. We applied our proof-of-concept implementation to all models to compute the smallest viable repairs. As a baseline, we compared our approach to the approach in [4] (approximation algorithm) to test how many unnecessary deletions could be avoided by using an exact approach as proposed in this work[7]. The experiments were run on a machine with 3 GHz

[3] https://bit.ly/38kyxD0.
[4] https://bit.ly/38lSU2N.
[5] https://pysathq.github.io/docs/html/api/examples/hitman.html.
[6] https://icpmconference.org/2020/bpi-challenge/.
[7] We acknowledge that the approach in [3] could have also been considered as a baseline; however, that approach cannot resolve quasi-inconsistencies and is therefore not fully comparable. Also, as the approach in [3] is also an approximation algorithm, it can be expected to also not compute the smallest possible number of deletions for all cases, which is why we consider the selected baseline [4] as representative.

Intel Core i7 processor, 16 GB RAM (DDR3) under macOS. Table 1 shows the experiment results for the analyzed real-life data sets. As the model mined from the BPI19 log did not yield any minimal issues, it was omitted for readability.

Table 1. Overview of evaluation results for the analyzed real-life data sets

Log	Constraints |	# of MIS	# of Deleted constraints			Runtime		
			Baseline [4] |	This work	△	Baseline [4] |	This work	△
BPI 17	305	28954	5	3	**40% (2)**	92243 ms	30782 ms	**67%**
BPI 18	70	25303	7	4	**43% (3)**	18093 ms	13733 ms	**24%**
BPI 20	357	747	7	5	**29% (2)**	1952 ms	795 ms	**59%**

For all declarative models, our algorithm was able to resolve all minimal issues by deleting less constraints compared to the baseline from [4]. More specifically, the information loss could be lowered by up to 43% (BPI 18). The runtime of our algorithm was also lower for all cases, with a time reduction of up to 67% (BPI 17). Thus, for the analyzed real-life data sets, our proposed approach was noticeably faster, and could reduce the number of deletions, i.e., resolve inconsistency with less information loss. Regarding the reduced number of deletions, this result is generalizable, as existing methods are prone to follow a non-optimal solution due to running into local optima (cf. Section 2.3). Thus, our approach guarantees to delete less or equal amounts of constraints for any model compared to [3,4,12]. Regarding runtime, we do not see a conceptual reason for the faster results, therefore, more experiments are needed in future works. The faster runtime could be attributed to the use of the PySat library, which might have faster inconsistent subset computation than [4].

5 Conclusion

In this paper, we have presented an approach for minimal and interactive inconsistency repair of declarative process models, where users can customize the computation of repair solutions and are supported in assessing different viable options with metrics and behavioral change analysis. Our evaluation indicates that our proposed approach can outperform existing means w.r.t. runtime and information loss. In this context, we see the following limitations of our work.

Our work implicitly uses the number of deleted constraints as an information loss measure. Here, other information loss measures have been investigated [9] and might be applicable for temporal logics. For example, instead of minimizing the *number* of deleted constraints, it could be beneficial to delete those constraints that have a low impact on the number of allowed traces. Note, however, that our approach already supports modelers towards this aim via behavioral change profile analysis of possible repairs.

Furthermore, our work only considers repair via deletion. While we argue that this can be plausible in the scope of inconsistency resolution, other change patterns such as weakening have also been proposed [9] and should be investigated in future work (e.g., relaxing a constraint ChainResponse to Response).

A central limitation of using the proposed approach based on hitting set diagnosis is that the constraints are viewed as abstract elements of a set. Here, it might be necessary to develop further means for distinguishing minimal inconsistent subsets in DECLARE based on the specific temporal constraints. In this way, it would be possible to further assess the severity of inconsistencies and to implement a more fine-grained prioritization of detected problems. In this context, it is also noteworthy that this work is limited to standard DECLARE templates with at most two parameters. Thus, it should be investigated how arbitrary constraints (e.g., using logical operators) must be handled, especially regarding their effect on behavioral changes.

In this work, the repair was geared towards inconsistencies. For future work, we aim at extending our approach to also consider other types of problematic structures in declarative models, such as hidden dependencies [17]. For any type of minimal structure, it can be expected that computing "minimal repair sets" via Reiter's hitting set theorem will be applicable. We aim to evaluate our proposed approach in experiments with human participants, especially in regard to the cognitive effects of the proposed metrics and behavioral change analysis. Also, we aim to implement and evaluate our proposed approach of behavioral change profile analysis.

References

1. Brewka, G., Thimm, M., Ulbricht, M.: Strong inconsistency. Artif. Intell. **267**, 78–117 (2019)
2. Cecconi, A., Di Ciccio, C., De Giacomo, G., Mendling, J.: Interestingness of traces in declarative process mining: the Janus LTLp$_f$ approach. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 121–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_8
3. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. **64**, 425–446 (2017)
4. Corea, C., Deisen, M., Delfmann, P.: Resolving inconsistencies in declarative process models based on culpability measurement. In: 2019 Proceedings der 14. International Tagung der WI, Siegen, Germany, pp. 139–153. AISeL (2019)
5. Corea, C., Delfmann, P.: Quasi-inconsistency in declarative process models. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNBIP, vol. 360, pp. 20–35. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26643-1_2
6. Corea, C., Thimm, M.: On quasi-inconsistency and its complexity. AI **284**, 103276 (2020)
7. Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M.: Consistency-based diagnosis of configuration knowledge bases. Art. Intell. **152**(2), 213–234 (2004)
8. Gainer-Dewar, A., Vera-Licona, P.: The minimal hitting set generation problem: Algorithms and computation. SIAM J. Discret. Math. **31**(1), 63–100 (2017)
9. Grant, J., Hunter, A.: Measuring consistency gain and information loss in stepwise inconsistency resolution. In: Liu, W. (ed.) ECSQARU 2011. LNCS (LNAI), vol. 6717, pp. 362–373. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22152-1_31

10. Jabbour, S.: On inconsistency measuring and resolving. In: 2019 22nd European Conference on Artificial Intelligence, The Hague, Netherlands. Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1676–1677. IOS Press (2016)
11. López, M.T.G., Gasca, R.M., Rinderle-Ma, S.: Explaining the incorrect temporal events during business process monitoring by means of compliance rules and model-based diagnosis. In: 2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops, Vancouver, Canada, pp. 163–172. IEEE Computer Society (2013)
12. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of LTL-based declarative process models. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 131–146. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_11
13. Markey, N.: Past is for free: on the complexity of verifying linear temporal properties with past. Acta Informatica **40**(6), 431–458 (2004). https://doi.org/10.1007/s00236-003-0136-5
14. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 2007 11th International Enterprise Distributed Object Computing Conference, Annapolis, USA, pp. 287–300. IEEE Computer Society (2007)
15. Pill, I.H., Quaritsch, T., Wotawa, F.: On the practical performance of minimal hitting set algorithms from a diagnostic perspective. Int. J. Progn. Health Manage. **7**(2), 1–15 (2016)
16. Reiter, R.: A theory of diagnosis from first principles. AI **32**(1), 57–95 (1987)
17. De Smedt, J., De Weerdt, J., Serral, E., Vanthienen, J.: Discovering hidden dependencies in constraint-based declarative process models for improving understandability. Inf. Syst. **74**, 40–52 (2018)