Information Systems 89 (2020) 101482

Contents lists available at ScienceDirect

Information Systems

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

Automated discovery of declarative process models with correlated data conditions

Volodymyr Leno [a,b,*], Marlon Dumas [a], Fabrizio Maria Maggi [a], Marcello La Rosa [b], Artem Polyvyanyy [b]

[a] University of Tartu, Liivi 2, 50409, Tartu, Estonia
[b] University of Melbourne, Parkville, VIC, 3010, Australia

ARTICLE INFO

Article history:
Received 15 October 2019
Received in revised form 3 December 2019
Accepted 4 December 2019
Available online 9 December 2019
Recommended by Dennis Shasha

Keywords:
Process mining
Automated process discovery
Declarative process model
Rule mining

ABSTRACT

Automated process discovery techniques enable users to generate business process models from event logs extracted from enterprise information systems. Traditional techniques in this field generate procedural process models (e.g., in the BPMN notation). When dealing with highly variable processes, the resulting procedural models are often too complex to be practically usable. An alternative approach is to discover declarative process models, which represent the behavior of the process as a set of constraints. Declarative process discovery techniques have been shown to produce simpler models than procedural ones, particularly for processes with high variability. However, the bulk of approaches for automated discovery of declarative process models focus on the control-flow perspective, ignoring the data perspective. This paper addresses the problem of discovering declarative process models with data conditions. Specifically, the paper tackles the problem of discovering constraints that involve two activities of the process such that each of these two activities is associated with a condition that must hold when the activity occurs. The paper presents and compares two approaches to the problem of discovering such conditions. The first approach uses clustering techniques in conjunction with a rule mining technique, while the second approach relies on redescription mining techniques. The two approaches (and their variants) are empirically compared using a combination of synthetic and real-life event logs. The experimental results show that the former approach outperforms the latter when it comes to re-discovering constraints artificially injected in a log. Also, the former approach is in most of the cases more computationally efficient. On the other hand, redescription mining discovers rules with higher confidence (and lower support) suggesting that it may be used to discover constraints that hold for smaller subsets of cases of a process.

1. Introduction

Automated process discovery techniques take as input an event log recording the execution of instances of a business process over a period of time, and produce as output a process model that captures the behavior observed in the log. The process models produced by automated process discovery techniques mainly fall into two categories: procedural and declarative. The dichotomy procedural versus declarative when choosing the most suitable language to represent the output of a process discovery technique has been widely studied [1,2]: procedural languages are suitable for processes with low variability (i.e., relatively low number of variants), whereas declarative languages are suitable for processes with high variability [3–6].

This paper focuses on the problem of discovering declarative process models from event logs. The paper tackles a still open challenge in this field, namely that of discovering multi-perspective declarative process models, i.e., models that take into account both the control-flow perspective (which events occur and in which order) and the data perspective, specifically the (data) conditions that hold when a given event occurs. For example, in the context of a loan application process, we aim at discovering rules like: "when an applicant having a salary lower than 24 000 euros per year submits a loan application, eventually an assessment of the application will be carried out, and the type of the assessment is complex". In this example, we have that a *response* DECLARE constraint (the submission of an application is eventually followed by an assessment) is satisfied only when both a condition on the payload of the activation (i.e., the amount associated to the submission of the application)

* Corresponding author at: University of Melbourne, Parkville, VIC, 3010, Australia.
E-mail addresses: leno@ut.ee (V. Leno), marlon.dumas@ut.ee (M. Dumas), f.m.maggi@ut.ee (F.M. Maggi), marcello.larosa@unimelb.edu.au (M. La Rosa), artem.polyvyanyy@unimelb.edu.au (A. Polyvyanyy).

https://doi.org/10.1016/j.is.2019.101482

and a condition on the target (on the type of assessment) is satisfied. The former condition is called an activation condition, the latter is called a target condition, and when both conditions are present in a DECLARE constraint, we say that the constraint contains two *correlated data conditions*.

The paper proposes two approaches for automated discovery of declarative process models with correlated data conditions. Both techniques start by discovering a set of *frequent constraints* from an event log. A frequent constraint is a constraint having a high number of *constraint instances*, i.e., pairs of events (one activation and one target) satisfying it. In the first approach, we cluster the target payloads to find groups of targets with similar payloads. These groups are used as labels for rule mining. In particular, the labels together with the features extracted from the activation payloads are used as input of a classification problem to discover correlations between the activation payloads and the target payloads. In the second approach, we apply *redescription mining* to sets of activation and target payloads to find the rules that correlate these two sets.

This article is an extended and revised version of a conference paper [7]. The conference version focused on the first type of approach (clustering followed by rule discovery). In this article, we present an alternative approach based on redescription mining and we empirically evaluate the tradeoffs between these two approaches and their variants. This article also includes a more extensive validation that considers both the accuracy and scalability of the proposed approaches.

The paper is structured as follows. Section 2 provides the necessary background to understand the rest of the paper, and presents related work. Section 3 introduces a running example used to illustrate the proposed techniques. Section 4 presents the proposed techniques, while Section 5 discusses their evaluation on synthetic and real-life logs. Finally, Section 6 concludes the paper and spells out directions for future work.

2. Background and related work

This section introduces the notion of event log (Section 2.1). It then defines the declarative process modeling notations used in the rest of the paper (Sections 2.2 and 2.3) and provides an overview of related work (Section 2.4).

2.1. Event log

The starting point for process mining is an event log. Event logs record the executions of businesses processes as sequences of events. Each event in a log refers to an *activity* (i.e., a well-defined step in a business process) and is related to a particular *case* (i.e., a *process instance*). Events that belong to a case are *ordered* and constitute a single "run" of the process (often referred to as a *trace* of events). Event logs may store additional information about events such as *resources* (i.e., people and/or devices) executing or initiating the activities, *timestamps* indicating when the events occur, and *data elements* associated with the events. Data elements stored in the log can be either event attributes, i.e., data produced by the activities of a business process, or case attributes, namely data that are associated to the whole process instance. In this paper, we assume that all attributes are globally visible and can be accessed/manipulated by all activity instances executed inside the case.

Table 1
Semantics for DECLARE templates.

Template	LTL semantics	Activation
Responded existence	$\mathbf{G}(A \rightarrow (\mathbf{O}B \vee \mathbf{F}B))$	A
Response	$\mathbf{G}(A \rightarrow \mathbf{F}B)$	A
Alternate response	$\mathbf{G}(A \rightarrow \mathbf{X}(\neg A\mathbf{U}B))$	A
Chain response	$\mathbf{G}(A \rightarrow \mathbf{X}B)$	A
Precedence	$\mathbf{G}(B \rightarrow \mathbf{O}A)$	B
Alternate precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}(\neg B\mathbf{S}A))$	B
Chain precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}A)$	B
Not responded existence	$\mathbf{G}(A \rightarrow \neg(\mathbf{O}B \vee \mathbf{F}B))$	A
Not response	$\mathbf{G}(A \rightarrow \neg\mathbf{F}B)$	A
Not precedence	$\mathbf{G}(B \rightarrow \neg\mathbf{O}A)$	B
Not chain response	$\mathbf{G}(A \rightarrow \neg\mathbf{X}B)$	A
Not chain precedence	$\mathbf{G}(B \rightarrow \neg\mathbf{Y}A)$	B

2.2. Declare

DECLARE is a declarative process modeling language originally introduced by Pesic and van der Aalst in [3]. Instead of explicitly specifying the flow of the interactions among process activities, DECLARE describes a set of constraints that must be satisfied throughout the process execution. The possible orderings of activities are implicitly specified by constraints and anything that does not violate them is possible during execution. In comparison with procedural approaches that produce "closed" models, i.e., all that is not explicitly specified is forbidden, DECLARE models are "open" and tend to offer more possibilities for the execution. In this way, DECLARE enjoys flexibility and is very suitable for highly dynamic processes characterized by high complexity and variability due to the changeability of their execution environments.

A DECLARE model consists of a set of constraints over activities. Constraints, in turn, are based on templates. Templates are patterns that define parameterized classes of properties, and constraints are their concrete instantiations (we indicate template parameters with capital letters and concrete activities in their instantiations with lower case letters). Templates have graphical representations and their semantics has been captured using various formalisms [4], making them verifiable and executable. Each constraint inherits the graphical representation and semantics from its template.

Table 1 summarizes some DECLARE templates (the reader can refer to [8] for a full description of the language). Here, the **F**, **X**, **G**, and **U** LTL (future) operators have the following intuitive meaning: formula $\mathbf{F}\phi_1$ means that ϕ_1 holds sometime in the future, $\mathbf{X}\phi_1$ means that ϕ_1 holds in the next position, $\mathbf{G}\phi_1$ says that ϕ_1 holds forever in the future, and, lastly, $\phi_1\mathbf{U}\phi_2$ means that sometime in the future ϕ_2 will hold and until that moment ϕ_1 holds (with ϕ_1 and ϕ_2 LTL formulas). The **O**, **Y**, and **S** LTL (past) operators have the following meaning: $\mathbf{O}\phi_1$ means that ϕ_1 holds sometime in the past, $\mathbf{Y}\phi_1$ means that ϕ_1 holds in the previous position, and, lastly, $\phi_1\mathbf{S}\phi_2$ means that sometime in the past ϕ_2 holds and since that moment ϕ_1 holds.

The major benefit of using templates is that analysts do not have to be aware of the underlying logic-based formalization to understand the models. They work with the graphical representation of templates, while the underlying formulas remain hidden.

Consider, for example, the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$. This constraint indicates that if *a occurs*, *b* must eventually *follow*. Therefore, this constraint is satisfied for traces such as $\mathbf{t}_1 = \langle a, a, b, c\rangle$, $\mathbf{t}_2 = \langle b, b, c, d\rangle$, and $\mathbf{t}_3 = \langle a, b, c, b\rangle$, but not for $\mathbf{t}_4 = \langle a, b, a, c\rangle$ because, in this case, the second instance of *a* is not followed by a *b*. Note that, in $\mathbf{t}_2$, the considered response constraint is satisfied in a trivial way because *a* never occurs. In this case, we say that the constraint is *vacuously satisfied* [9].

Table 2
Semantics for MP-Declare constraints.

Template	MFOTL semantics
Responded existence	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow (\mathbf{O}_I(B \wedge \exists y.\varphi_c(x,y)) \vee \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y)))))$
Response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y))))$
Alternate response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{X}(\neg(A \wedge \varphi_a(x))\mathbf{U}_I(B \wedge \exists y.\varphi_c(x,y)))))$
Chain response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{X}_I(B \wedge \exists y.\varphi_c(x,y))))$
Precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{O}_I(A \wedge \exists y.\varphi_c(x,y))))$
Alternate precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{Y}(\neg(B \wedge \varphi_a(x))\mathbf{S}_I(A \wedge \exists y.\varphi_c(x,y)))))$
Chain precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y))))$
Not responded existence	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg(\mathbf{O}_I(B \wedge \exists y.\varphi_c(x,y)) \vee \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y)))))$
Not response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg\mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y))))$
Not precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{O}_I(A \wedge \exists y.\varphi_c(x,y))))$
Not chain response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg\mathbf{X}_I(B \wedge \exists y.\varphi_c(x,y))))$
Not chain precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y))))$

In [10,11], the authors introduce the notion of *semantical vacuity detection* according to which a constraint is non-vacuously satisfied in a trace when it is activated in that trace. Intuitively, an *activation* of a constraint in a trace is an event whose occurrence imposes, because of that constraint, some obligations on other events (targets) in the same trace. For example, a is the activation for the response constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$ and b is the target, because the execution of a forces b to be executed, eventually. In Table 1, for each template, the corresponding activation is specified.

An activation of a constraint can be a *fulfillment* or a *violation* for that constraint. When a trace is perfectly compliant with respect to a constraint, every activation of the constraint in the trace leads to a fulfillment. Consider, again, the response constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$. In trace $\mathbf{t}_1$, the constraint is activated and fulfilled twice, whereas, in trace $\mathbf{t}_3$, the same constraint is activated and fulfilled only once. On the other hand, when a trace is not compliant with respect to a constraint, an activation of the constraint in the trace can lead to a fulfillment but also to a violation (at least one activation leads to a violation). In trace $\mathbf{t}_4$, for example, the response constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$ is activated twice, but the first activation leads to a fulfillment (eventually b occurs) and the second activation leads to a violation (b does not occur subsequently). An algorithm to discriminate between fulfillments and violations for a constraint in a trace is presented in [12].

Tools implementing process mining approaches based on Declare are presented in [13]. The tools are implemented as plug-ins of the process mining framework ProM.

2.3. Multi-Perspective Declare

In this section, we illustrate a multi-perspective version of Declare (MP-Declare) introduced in [14]. This semantics is expressed in Metric First-Order Linear Temporal Logic (MFOTL) and is shown in Table 2.

We describe here the semantics informally and we refer the interested reader to [14] for more details. To explain the semantics, we introduce some preliminary notions.

The first concept we use is the one of *payload* of an event. Consider, for example, that the execution of an activity Submit Loan Application (S) is recorded in an event log and, after the execution of S at timestamp τ_S, the attributes *Salary* and *Amount* have values 12 500 and 55 000. In this case, we say that, when S occurs, two special relations are valid $event(S)$ and $p_S(12\,500, 55\,000)$. In the following, we identify $event(S)$ with the event S itself and we call (12 500, 55 000), the *payload* of S. Note that, as already mentioned, we consider all attributes as global variables accessible by all events in a case. The difference between case attributes ad event attributes is that case attributes have constant values from the beginning to the end of the case, whereas event attributes are modifiable by the single events and can change while the case progresses.

Note that all the templates in MP-Declare in Table 2 have two parameters, an activation and a target (see also Table 1). The standard semantics of Declare is extended by requiring two additional conditions on data, i.e., the *activation condition* φ_a and the *correlation condition* φ_c. As an example, we consider the response constraint "activity Submit Loan Application is always eventually followed by activity Assess Application" having Submit Loan Application as activation and Assess Application as target. The activation condition is a relation (over the variables corresponding to the global attributes in the event log) that must be valid when the activation occurs. If the activation condition does not hold the constraint is not activated. The activation condition has the form $p_A(x) \wedge r_a(x)$, meaning that when A occurs with payload x, the relation r_a over x must hold. For example, we can say that whenever Submit Loan Application occurs, and the amount of the loan is higher than 50 000 euros and the applicant has a salary lower than 24 000 euros per year, eventually an assessment of the application must follow. In case Submit Loan Application occurs but the amount is lower than 50 000 euros or the applicant has a salary higher than 24 000 euros per year, the constraint is not activated.

The correlation condition is a relation that must be valid when the target occurs. It has the form $p_B(y) \wedge r_c(x,y)$, where r_c is a relation involving, again, variables corresponding to the (global) attributes in the event log but, in this case, relating the payload of A and the payload of B. A special type of correlation condition has the form $p_B(y) \wedge r_c(y)$, which we call *target condition*, since it does not involve attributes of the activation.

In this paper, we aim at discovering constraints that correlate an activation and a target condition. For example, we can find that whenever Submit Loan Application occurs, and the amount of the loan is higher than 50 000 euros and the applicant has a salary lower than 24 000 euros per year, then eventually Assess Application must follow, and the assessment type will be *Complex* and the cost of the assessment higher than 100 euros.

Finally, in MP-Declare, also a time condition can be specified through an interval ($I = [\tau_0, \tau_1)$) indicating the minimum and the maximum temporal distance allowed between the occurrence of the activation and the occurrence of the corresponding target.

2.4. Discovery of data-aware declarative process models

In [15], the authors propose a data-aware technique for the discovery of declarative models. The technique uses a data-aware extension of the Declare language defined in terms of LTL-FO (First Order Linear Temporal Logic). Given a control-flow constraint, the approach can be used to discover activation conditions that discriminate between cases in which the constraint is satisfied and cases in which it is violated. Our approach is able to

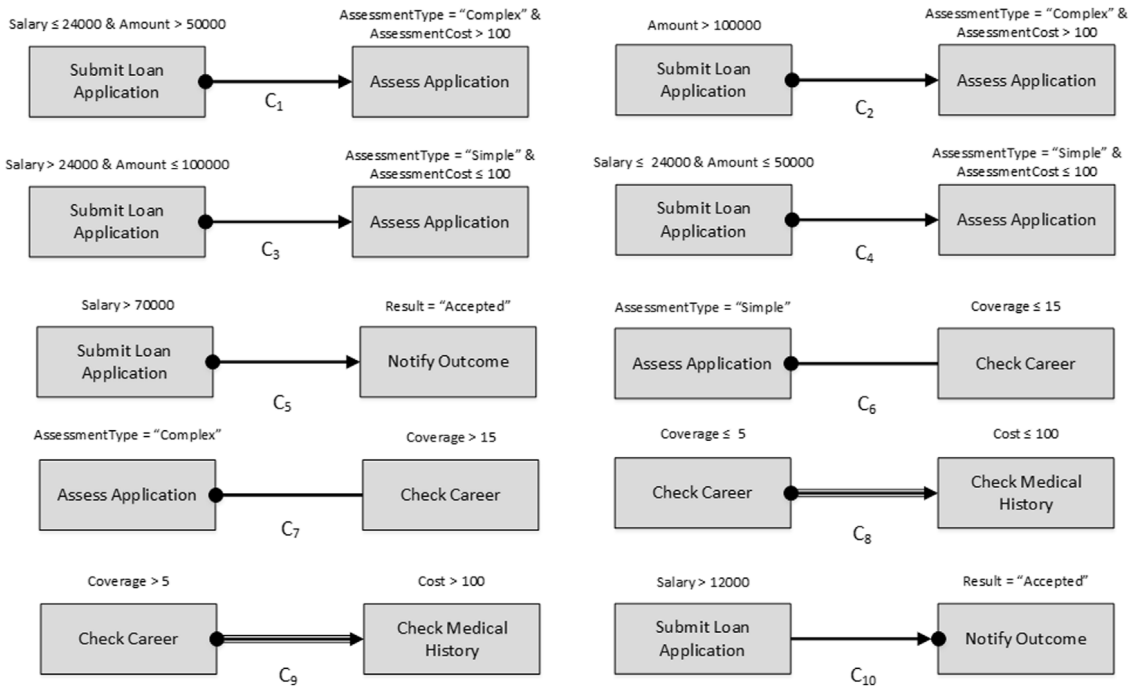


Fig. 1. Running Example.

discover the same type of discriminative activation conditions and, in addition, it is able to correlate them with conditions on the target.

In [16], the authors use correlations to prune a discovered declarative model and to disambiguate event associations. As a result, the discovered process models only show the most meaningful constraints and become more readable. This approach can be considered as a naive approach to find correlated conditions since all combinations of comparable attributes are generated. Our approach, instead, directly identifies only the meaningful correlated conditions without the need of generating and checking all the possible correlations.

The work in [17] presents another approach for the multi-perspective discovery of declarative process models. This work is based on RelationalXES, a relational database architecture for storing event log data. Once stored, relational event data can be queried with conventional SQL. Queries capture the semantics of MP-Declare and can be customized. However, the queries have to be manually specified.

The work in [18] presents a technique for the discovery of conditions in the decision rules (a.k.a. branching points) in imperative process models. The technique combines existing methods for the discovery of imperative process models (e.g., Petri nets) and decision trees, and the identified conditions compare a single variable with some constant values. The technique presented in [19] extends the work in [18] and can be used to discover conditions over more than one variable. Our approach works in a similar way, but it focuses on the discovery of conditions in declarative process models.

3. Running example

Fig. 1 shows a fictive MP-Declare model that we will use as a running example throughout this paper.

This example models a process for loan applications in a bank. When an applicant submits a loan application with an amount higher than 50 000 euros and she has a salary lower than 24 000 euros per year, eventually an assessment of the application will be carried out. The assessment will be complex and the cost of the assessment higher than 100 euros. This behavior is described by *response* constraint C_1 in Fig. 1. When an applicant submits a loan application with an amount higher than 100 000 euros, eventually a complex assessment with cost higher than 100 euros is performed (C_2). In other cases, the simple assessment will be carried out with the cost of assessment not exceeding 100 euros (constraints C_3 and C_4). The outcome is always positive when a salary of an applicant is greater than 70 000. This is reflected in *response* constraint C_5. Outside the application assessment there are 2 additional checks that can be performed before or after the assessment: the career check and the medical history check. A career check with a coverage lower than 15 years is required if the application assessment is simple (*responded existence* constraint C_6). The career of the applicant should be checked with a coverage higher than 15 years if the application assessment is complex (*responded existence* constraint C_7). If the career check covers less than 5 years, a medical history check should be performed immediately after and its cost is lower than 100 euros (*chain response* constraint C_8). If the career check covers more than 5 years, the medical history check is more complex and more expensive (its cost is higher than 100 euros). This behavior is described by *chain response* constraint C_9 in Fig. 1. If the outcome of an application assessment is notified and the result of the outcome is accepted, then this event is always preceded by an application submission whose applicant has a salary higher than 12 000 euros per year (*precedence* constraint C_{10}).

4. Enhancing Declare rules with data-aware conditions

We propose two alternative approaches to discover data-aware rules from event logs. The outline of these two approaches is shown in Fig. 2.

Both approaches start from the extraction of instances of Declare constraints discovered from the log. The discovery of Declare constraints is carried out using the approach presented in [20]. In this paper, we assume that the discovered Declare constraints are already given and we enhance them with data-aware conditions. A *constraint instance* is a pair of an activation and the corresponding target activity of a Declare constraint.

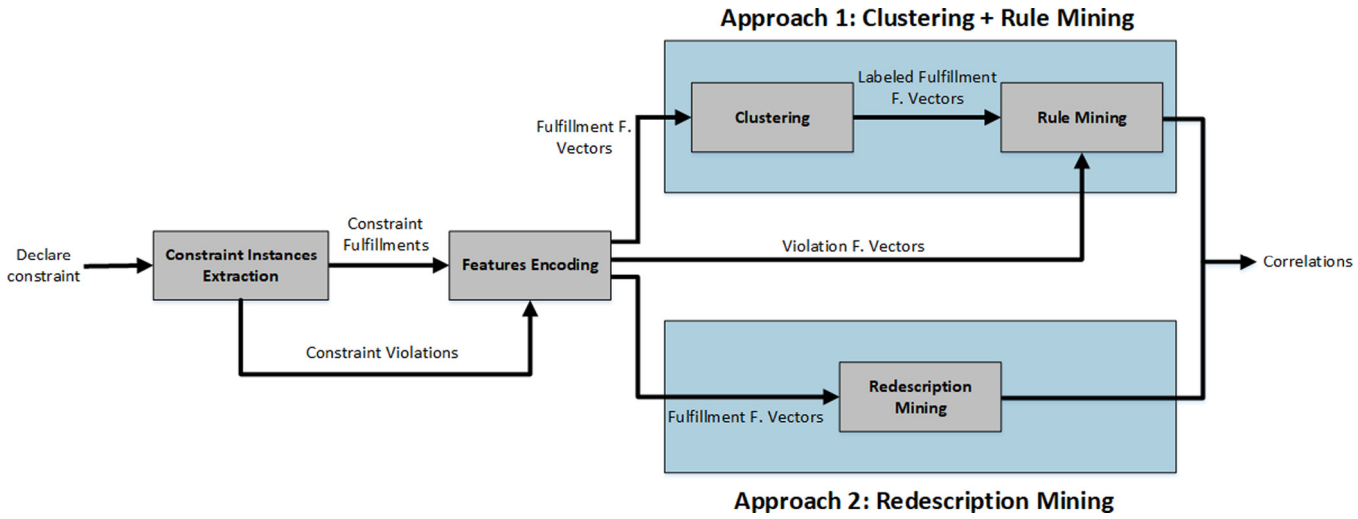


Fig. 2. Outline of the proposed approaches. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Constraint instances of type (S, A) in trace t.

Candidate constraint	Constraint instances
Response(S, A)	$\{S_1A_1\}$, $\{S_2A_1\}$, $\{S_3A_1\}$, $\{S_4A_2\}$, $\{S_5A_3\}$, $\{S_6A_3\}$
Chain Response(S, A)	$\{S_3A_1\}$, $\{S_4A_2\}$, $\{S_6A_3\}$
Alternate Response(S, A)	$\{S_3A_1\}$, $\{S_4A_2\}$, $\{S_6A_3\}$
Precedence(S, A)	$\{S_3A_1\}$, $\{S_4A_2\}$, $\{S_6A_3\}$
Chain Precedence(S, A)	$\{S_3A_1\}$, $\{S_4A_2\}$, $\{S_6A_3\}$
Alternate Precedence(S, A)	$\{S_3A_1\}$, $\{S_4A_2\}$, $\{S_6A_3\}$
Responded Existence(S, A)	$\{S_1A_1\}$, $\{S_2A_1\}$, $\{S_3A_1\}$, $\{S_4A_1\}$, $\{S_5A_1\}$, $\{S_6A_1\}$

Table 4
Constraint instances of type (A, S) in trace t.

Candidate constraint	Constraint instances
Response(A, S)	$\{A_1S_4\}$, $\{A_2S_5\}$
Chain Response(A, S)	$\{A_1S_4\}$, $\{A_2S_5\}$
Alternate Response(A, S)	$\{A_1S_4\}$, $\{A_2S_5\}$
Precedence(A, S)	$\{A_1S_4\}$, $\{A_2S_5\}$, $\{A_2S_6\}$
Chain Precedence(A, S)	$\{A_1S_4\}$, $\{A_2S_5\}$
Alternate Precedence(A, S)	$\{A_1S_4\}$, $\{A_2S_5\}$
Responded Existence(A, S)	$\{A_1S_1\}$, $\{A_2S_1\}$, $\{A_3S_1\}$

Both activation and target are also equipped with the corresponding payloads. We extract activation and target payloads of each constraint instance to form a set of (unlabeled) *fulfillment feature vectors*. Activations that cannot be associated to any target (representing a violation of the constraint) are put into *violation feature vectors* and labeled as *violated*.

In the first approach, the target payloads of previously obtained fulfillment feature vectors are clustered to find groups of targets with similar payloads. For each cluster its description is discovered. These descriptions are used as labels in combination with the activation payloads to generate a set of *labeled fulfillment feature vectors*. The labeled violation and fulfillment feature vectors are then used as input for rule mining. This procedure allows for finding correlations between the activation payloads and the target payloads.

In the second approach, we apply redescription mining algorithms using both the features of activation and target payloads. There are two major types of redescription mining algorithms. The first family of algorithms proposes to grow two decision trees in an alternating way and then join them in the leaves. By traversing the obtained trees, we can get the correlations between activation and target payloads. An alternative technique is to extend the correlations greedily starting from the simplest ones (containing one literal from each side). In this paper, we consider both approaches. Note that the core parts of our approaches (highlighted with blue rectangles in Fig. 2) are independent of the procedure used to extract constraint instances. Thus they can be used in combination with any approach for constraint instances extraction.

4.1. Constraint instances extraction

The first step of the algorithm is to extract the instances of a given Declare constraint. This is done by creating two vectors idx_1 and idx_2 that represent activation and target occurrences. Then, the algorithm processes each trace in the input log to find the events corresponding either to the activation or to the target of the constraint, and their indexes are collected in the corresponding vector. For example, for trace t = SSSASASSA and Response(S,A), we have $idx_1 = (1; 2; 3; 5; 7; 8)$ and $idx_2 = (4; 6; 9)$. Then, based on the template, the constraint instances are identified as follows:

- **(Not) Response.** For each element idx_{1i}, from the activation vector idx_1, we take the first element idx_{2j} from the target vector idx_2 that is greater than idx_{1i}.
- **(Not) Chain Response.** Here, we check the existence of pairs (i,j) from idx_1 and idx_2 where $j - i = 1$.
- **Alternate Response.** In this case, for each element idx_{1i} from idx_1, we take the first element idx_{2j} from idx_2 that is greater than idx_{1i}. However, we identify a constraint instance only if there are no elements from idx_1 that lie between idx_{1i} and idx_{2j}.
- **(Not) Precedence.** For precedence, chain precedence and alternate precedence, the logic is almost the same as for their response counterparts. However, for precedence rules, the idx_1 is considered as target vector and idx_2 as activation vector. In addition, for each element idx_{2j}, from the activation vector idx_2, we take the last element idx_{1i} from the target vector idx_1 that is smaller than idx_{2j}.
- **(Not) Responded Existence.** We associate each element idx_{1i} from the activation vector idx_1, with the first element from the target vector idx_2.

Note that, here, to extract constraint instances, we consider the closest pairs of activation and target occurrences. However, the approach is easily adaptable to use other strategies for constraint instances extraction. If we enumerate the occurrences of

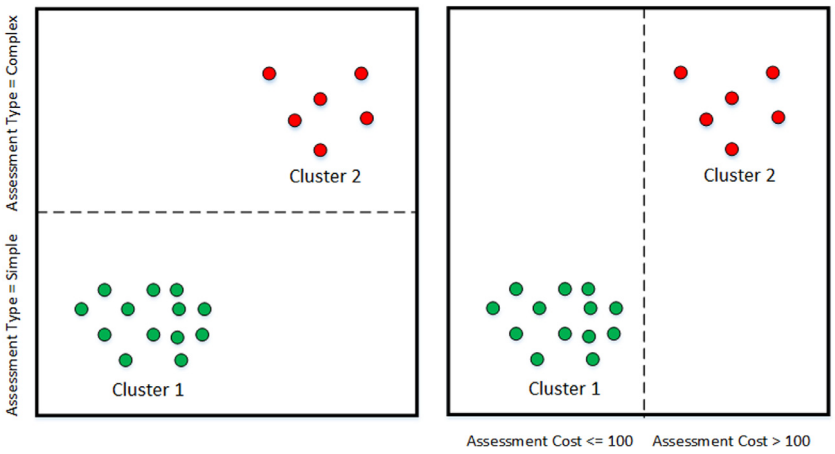


Fig. 3. Bidimensional descriptions of clusters of fulfillment feature vectors in terms of characteristics of target payloads. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

S and A in t, we have $t = S_1S_2S_3A_1S_4A_2S_5S_6A_3$. The constraint instance that consists of activation and target is called *constraint fulfillment*. The constraint fulfillments of the standard DECLARE templates instantiated with activities (A, S) and (S, A) are listed in Table 3 and in Table 4.

The procedures used to identify constraint fulfillments are also used to identify constraint violations (i.e., activations that cannot be associated to any target). The constraint fulfillments are used to create fulfillment feature vectors, while constraint violations are used to generate violation feature vectors. We stress again that these procedures only provide an example of how to identify temporal patterns in a log. Any semantics (also beyond standard DECLARE) can be used to identify frequent constraints.

4.2. Features encoding

Fulfillment and violation constraints are used to create a set of fulfillment and violation feature vectors, respectively. Fulfillment feature vectors consist of the payloads of activations and targets, while violation feature vectors consist of the payloads of activations only. Violation feature vectors do not have a corresponding target and are labeled as "violated". Assume to have a constraint instance where the activation SUBMIT LOAN APPLICATION has a payload (12 500, 55 000) (see Section 2.3). If this activation cannot be associated to any target, we generate the violation feature vector:

$$V_{viol} = [(12\,500, 55\,000); \textit{violated}]. \quad (1)$$

If the same activation is part of a constraint instance with target ASSESS APPLICATION and payload (*Complex*, 140), we generate the (unlabeled) fulfillment feature vector:

$$V_{ful} = [(12\,500, 55\,000); (\textit{Complex}, 140)]. \quad (2)$$

Violation and fulfillment feature vectors are then used as the main input data for the core parts of our approaches.

4.3. Approach 1: Clustering + Rule mining

Starting from the fulfillment feature vectors,[1] we use clustering to find groups of payloads that are similar. In particular, a modification of the K-Medoids clustering algorithm is used. With this modification, we can handle categorical attributes as well as numerical ones. In order to compute the distance between two feature vectors, we use the Gower distance

$$d(i,j) = \frac{1}{n}\sum_{f=1}^{n} d_{i,j}^{(f)}, \quad (3)$$

where n denotes the number of features, while $d_{i,j}^{(f)}$ is the distance between feature vectors i and j, when considering only feature f. $d_{i,j}^{(f)}$ is a normalized distance. For nominal attributes, we calculate the distance as follows:

$$d_{i,j}^{(f)} = \begin{cases} 0, & \text{if } x_i^{(f)} = x_j^{(f)} \\ 1, & \text{if } x_i^{(f)} \neq x_j^{(f)}. \end{cases} \quad (4)$$

For interval scaled attribute values, we use the distance:

$$d_{i,j}^{(f)} = \frac{|x_i^{(f)} - x_j^{(f)}|}{max - min}, \quad (5)$$

where max and min are the maximum and minimum observed values of attribute f.

The K-Medoids algorithm starts by selecting the initial medoids randomly. Then, at each iteration, it computes, for each cluster identified by a medoid determined in the previous iteration, a centroid containing: (1) for categorical attributes, the most frequent value in the cluster; (2) for numerical attributes, the average value in the cluster. For each computed centroid, the closest feature vector is assigned as being a medoid of the current iteration. After obtaining the medoids, each feature vector is assigned to the cluster defined by the closest medoid and the next iteration starts. The clustering stops when medoids do not change anymore or after N iterations, where N is given as input parameter.

Once the clusters have been constructed, we apply a direct rule-based classification algorithm called RIPPER (Repeated Incremental Pruning to Produce Error Reduction) to search for their distinct features that could be used to describe them. In particular, we build the classifier by using as feature vectors the projections of the fulfillment feature vectors on the target payloads and the clusters ids as labels. In this way, we can describe each cluster in terms of characteristics of target payloads in that cluster. The algorithm builds the rules greedily by adding a new condition using the conjunction operator as long as information gain improves. The initially obtained rule set is then pruned and simplified. The output of RIPPER is a decision table. For a 2-class problem, RIPPER selects one class as positive and the other as negative, and then learns rules for the positive class. The

[1] In our experiments, we also build the fulfillment feature vectors with the target payloads only instead of using the combination of activation and target payloads.

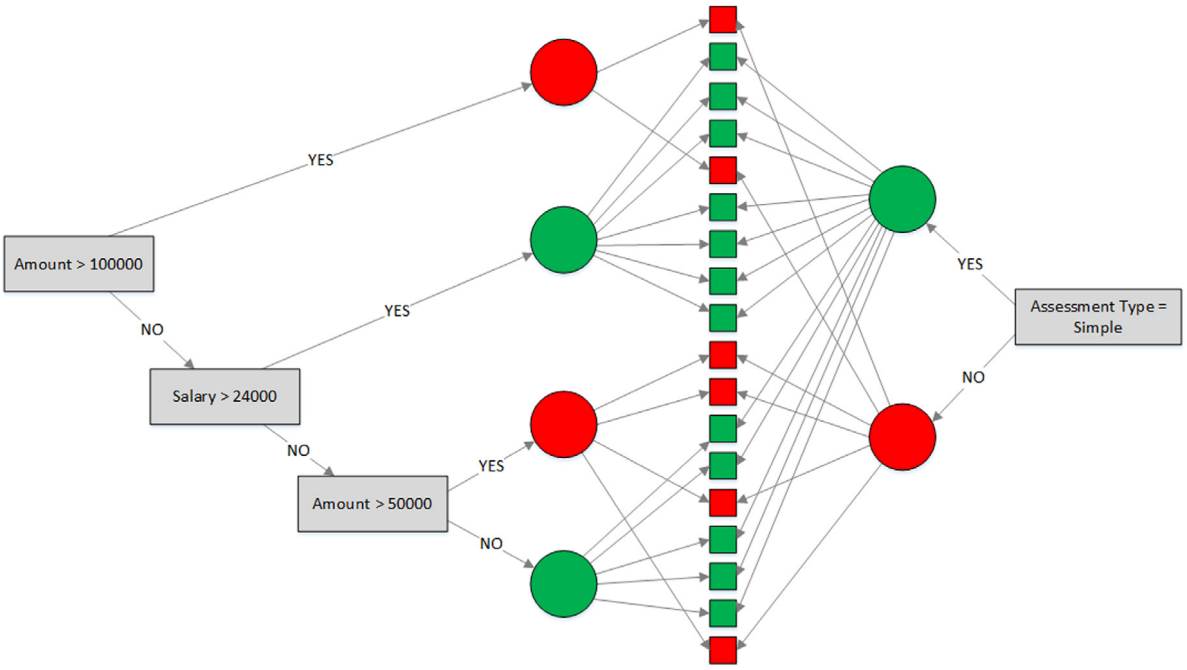


Fig. 4. Tree-based approach to redescription mining. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

negative class is described by the default rule (none of the above rules are satisfied). For multi-class problems, it picks the class with the smallest prevalence (fraction of instances that belong to a class) and considers it as the positive class, while all the other classes are considered to be negative. In such a way, it transforms a multi-class problem into a 2-class problem. When the rules for the positive class are discovered, this class is not considered anymore and the algorithm picks the next smallest class as positive class, while correspondingly treating the other classes as negative. The procedure repeats until 2 classes are left. Then it solves the 2-class problem as it was described before and the class with the largest representation becomes the negative class.

Fig. 3 shows two clusters (*Cluster*1 and *Cluster*2) associated with the *response* constraint with target Assess Application (colored in green and red respectively). We use a bidimensional representation here to show that the clusters can be characterized using the Assessment Cost attribute (*Assessment Cost* $\leq$ 100 and *Assessment Cost* $>$ 100, respectively) and the Assessment Type attribute (*Assessment Type* $=$ *Simple* and *Assessment Type* $=$ *Complex*, respectively). In particular, RIPPER would find that the conjunction *Assessment Cost* $\leq$ 100 $\wedge$ *Assessment Type* $=$ *Simple* characterizes *Cluster*1, whereas the conjunction *Assessment Cost* $>$ 100 $\wedge$ *Assessment Type* $=$ *Complex* characterizes *Cluster*2. These clusters/conditions are used as labels to build labeled fulfillment feature vectors. The features of the labeled fulfillment feature vectors come from the projections of the fulfillment feature vectors on the activation payloads.

Assume again to have a constraint instance where the activation Submit Loan Application has a payload (12 500, 55 000). If this activation is part of a constraint instance with target Assess Application and payload (*Complex*, 140), we generate the labeled fulfillment feature vector:

$$V'_{ful} = [(12\,500, 55\,000); \mathit{Cluster}2]. \quad (6)$$

Labeled fulfillment feature vectors and violation feature vectors are used as input data to RIPPER again. This time, the output of RIPPER provides a set of rules that correlates payloads of activations and targets of a given Declare constraint.

4.4. Approach 2: Redescription mining

Redescription mining is a family of unsupervised descriptive knowledge discovery approaches that aim at finding correlations between subsets of elements in a dataset by using two or more disjoint sets of descriptive attributes. In particular, the input of a redescription mining algorithm is a tuple (E, V_L, V_R, A_L, A_R), where E denotes a set of entities that are characterized by two different views V_L and V_R, respectively. These views are described by two sets of attributes A_L and A_R. In our case, the entities are the fulfillment feature vectors, the views V_L and V_R represent activations and targets of these vectors, and A_L and A_R are the attributes of their corresponding payloads.

The output of the algorithm is a set of redescriptions R that describe relations between the two different views. In particular, a redescription $r \in R$ is a logical formula that consists of two parts, r_L and r_R, where r_L contains literals from A_L, and r_R consists of literals from A_R, respectively. For example, for a redescription *Amount* $>$ 100 000 $\Rightarrow$ *Assessment Type* $\neq$ *Simple*, *Amount* $>$ 100 000 represents r_L, and *Assessment Type* $\neq$ *Simple* is r_R.

There are two types of redescription mining approaches. The first approach is based on classification and regression trees. The main idea is to iteratively grow two decision trees (one for each view) that will be joined in their leaves. The trees grow in an alternating way, meaning that the prediction vector derived for one tree in a certain step is used to grow the other tree in the next step.

Fig. 4 illustrates the application of this approach for the *response* constraint with activation Submit Loan Application (left hand side) and target Assess Application (right hand side). As can be seen, there are two classes of behaviors (colored in green and red), i.e., when *Amount* $>$ 100 000, or *Amount* $>$ 50 000 and *Salary* $\leq$ 24 000, then *Assessment Type* $\neq$ *Simple*, in all the other cases *Assessment Type* $=$ *Simple*. The redescriptions are obtained by traversing the trees from the root node of the first tree to the root node of the second one. Table 5 shows the redescriptions derived from the trees in Fig. 4.

An alternative approach is to grow the redescriptions greedily, starting from a pair of singleton queries (i.e., one variable on each

Table 5
Redescriptions discovered from the trees in Fig. 4.

	Redescriptions
1	$Amount > 100\,000 \Rightarrow Assessment\ Type \neq Simple$
2	$Amount \leq 100\,000$ & $Salary > 24\,000 \Rightarrow Assessment\ Type = Simple$
3	$Amount \leq 100\,000$ & $Salary \leq 24\,000$ & $Amount > 50\,000 \Rightarrow Assessment\ Type \neq Simple$
4	$Amount \leq 100\,000$ & $Salary \leq 24\,000$ & $Amount \leq 50\,000 \Rightarrow Assessment\ Type = Simple$

Table 6
Characteristics of the real-life logs.

Dataset	No. of cases	No. of events	No. of activities	Payload size	No. of case attributes	No. of event attributes
BPIC 2011	1140	149 730	622	9	4	5
BPIC 2013 (closed problems)	1487	6660	7	8	3	5
BPIC 2013 (incidents)	7554	65 533	13	8	3	5
BPIC 2013 (open problems)	819	2351	5	8	3	5
BPIC 2017	31 509	561 671	26	13	3	10
Sepsis	868	12 857	18	28	24	4
Traffic fines process	150 370	561 470	11	13	8	5

side) and extending them by appending a literal on either side using conjunctions or disjunctions. This procedure can be stopped when the maximum length of a query is reached or when the addition of a new condition does not improve the accuracy of the redescription. In this paper, we will consider a tree-based redescription approach called SplitT [21] and a greedy algorithm called ReReMi [22].

5. Evaluation

We implemented the Clustering + Rule Mining approach as an open-source prototype tool,[2] while, for the Redescription Mining approach, we configured an existing redescription mining tool, i.e., Siren.[3] We used these two tools to conduct a series of experiments aimed at understanding the relative strengths of the two discovery approaches. In particular, we wanted to examine their capability to rediscover the behavior injected into an artificial log, and assess their scalability (using both artificial and real-life logs) and their applicability to real-life event logs. Accordingly, we investigated the following three research questions:

- **RQ1.** Which of the approaches better rediscovers constraints artificially injected into an event log?
- **RQ2.** How do the approaches perform relative to each other in terms of execution time?
- **RQ3.** What are the characteristics of the constraint sets generated by the proposed approaches when applied to real-life logs?

RQ1 focuses on the evaluation of the rediscovery accuracy of the proposed approaches. RQ2 investigates the scalability of the approaches. RQ3 deals with the validation of the discovery approaches in real scenarios. The characteristics evaluated with RQ3 are the number of data conditions discovered for a single original control-flow constraint, their complexity, and also support and confidence of the enhanced constraints. In this evaluation, we call *support* of a constraint the percentage of feature vectors of that constraint where both activation and target conditions hold (over the total number of feature vectors for that constraint), and *confidence* the percentage of feature vectors where both activation and target conditions hold over the number of feature vectors where the activation condition holds.

5.1. Datasets

The artificial logs used in this evaluation were generated using the MP-Declare Log Generator tool [23]. To answer RQ1, we generated a log by simulating the model in Fig. 1. This log contains 5000 cases with average length of 5 events, leading to a total of 25 000 events. In order to have a sufficient number of feature vectors available, the activities that are involved in the constraints of the model (Submit Loan Application, Assess Application, Notify Outcome, Check Career, Check Medical History) appear in every case, leading to over 5000 feature vectors for each constraint.

There are two factors that affect the execution time of the algorithms (investigated with RQ2): the number of feature vectors and the payload size. In order to assess how the former factor affects the execution time, we created a set of artificial logs with growing number of feature vectors for one specific MP-Declare constraint (1000, 5000, 10 000, 20 000, 30 000, 50 000 and 100 000 feature vectors), with a default payload size of 6 attributes. To assess the impact of the latter factor, we generated a set of artificial logs with increasing event payload size (5, 10, 15, 20, 25 and 30 attributes), with a default number of feature vectors equal to 1000.

In addition to these artificial logs, we used seven real-life logs for answering RQ2 and RQ3. We considered all real-life event logs from the 4TU Data Center collection[4] having at least three event payload attributes. These are BPIC 2011,[5] all the logs of BPIC 2013,[6,7,8] BPIC 2017,[9] Sepsis,[10] and the Road Traffic Fine Management log.[11] These logs cover various domains such as healthcare, IT support management, banking and public administration. The logs were preprocessed by deleting those cases that contain missing values in the event payloads and by removing redundant attributes (i.e., case attributes that always have the same value in a case or attributes that do not store any valuable information that can be used to discriminate the different clusters of behaviors, e.g., the "Variant" attribute in the Sepsis log or the "EventID" attribute in BPIC 2017). For the BPIC 2017 log, we also

2 Available at https://doi.org/10.6084/m9.figshare.10565906.v2.
3 Available at http://siren.gforge.inria.fr/main/index.html.
4 https://data.4tu.nl/repository/collection:event_logs_real
5 doi:10.4121/d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
6 doi:10.4121/c2c3b154-ab26-4b31-a0e8-8f2350ddac11
7 doi:10.4121/10.4121/500573e6-accc-4b0c-9576-aa5468b10cee
8 doi:10.4121/10.4121/3537c19d-6c64-4b1d-815d-915ab0e479da
9 doi:10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
10 doi:10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
11 doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

Table 7
Rediscovery results: K-Medoids + RIPPER (k = 2 for K-Medoids, clustering target payloads only).

Template	Activation/Target	Activation/Target condition	Support	Confidence
(C_1) Response	**Submit Loan Application** **Assess Application**	***Salary*** $\leq$ **24 000 &** ***Amount*** $\geq$ **51 000** ***Assessment Cost*** $\geq$ **110**	**0.31**	**1.0**
(C_2) Response	**Submit Loan Application** **Assess Application**	***Amount*** $\geq$ **101 000** ***Assessment Cost*** $\geq$ **110**	**0.28**	**1.0**
(C_3, C_4) Response	**Submit Loan Application** **Assess Application**	$\neg$(***Amount*** $\geq$ **101 000**) & $\neg$(***Salary*** $\leq$ **24 000** & ***Amount*** $\geq$ **51 000**) ***Assessment Cost*** $<$ **110**	**0.56**	**1.0**
(C_5) Response	**Submit Loan Application** **Notify Outcome**	***Salary*** $\geq$ **71 000** ***Result*** $=$ ***Accepted***	**0.09**	**1.0**
Response	Submit Loan Application Notify Outcome	$\neg$(*Salary* $\geq$ 71 000) *Result* $\neq$ *Accepted*	0.61	0.67
(C_7) Responded Existence	Assess Application Check Career	*Assessment Cost* $\geq$ 110 *Coverage* $\geq$ 14	0.44	1.0
(C_6) Responded Existence	Assess Application Check Career	$\neg$(*Assessment Cost* $\geq$ 110) *Coverage* $<$ 14	0.53	0.95
(C_9) Chain Response	Check Career Check Medical History	*Coverage* $\geq$ 6 *Cost* $\geq$ 150	0.47	0.80
(C_8) Chain Response	Check Career Check Medical History	$\neg$(*Coverage* $\geq$ 6) *Cost* $<$ 150	0.41	1.0
Precedence	Submit Loan Application Notify Outcome	*Salary* $\leq$ 35 000 *Result* $=$ *Rejected*	0.33	0.54
(C_{10}) Precedence	Submit Loan Application Notify Outcome	*Salary* $>$ 35 000 *Result* $\neq$ *Rejected*	0.26	0.67

Table 8
Rediscovery results: K-Medoids + RIPPER (k = 2 for K-Medoids, clustering activation and target payloads).

Template	Activation/Target	Activation/Target condition	Support	Confidence
(C_1) Response	**Submit Loan Application** **Assess Application**	***Salary*** $\leq$ **24 000** & ***Amount*** $\geq$ **51 000** ***Assessment Cost*** $\geq$ **110**	**0.31**	**1.0**
(C_2) Response	**Submit Loan Application** **Assess Application**	***Amount*** $\geq$ **101 000** ***Assessment Cost*** $\geq$ **110**	**0.28**	**1.0**
(C_3, C_4) Response	**Submit Loan Application** **Assess Application**	$\neg$(***Amount*** $\geq$ **101 000**) & $\neg$(***Salary*** $\leq$ **24 000** & ***Amount*** $\geq$ **51 000**) ***Assessment Cost*** $<$ **110**	**0.56**	**1.0**
(C_5) Response	**Submit Loan Application** **Notify Outcome**	***Salary*** $\geq$ **71 000** ***Result*** $=$ ***Accepted***	**0.09**	**1.0**
Response	Submit Loan Application Notify Outcome	$\neg$(*Salary* $\geq$ 71 000) *Result* $\neq$ *Accepted*	0.61	0.67
(C_7) Responded Existence	**Assess Application** **Check Career**	***Assessment Cost*** $\geq$ **110** ***Coverage*** $\geq$ **16**	**0.44**	**1.0**
(C_6) Responded Existence	**Assess Application** **Check Career**	$\neg$(***Assessment Cost*** $\geq$ **110**) ***Coverage*** $<$ **16**	**0.56**	**1.0**
(C_8) Chain Response	**Check Career** **Check Medical History**	***Coverage*** $\leq$ **5** ***Cost*** $\leq$ **100**	**0.41**	**1.0**
(C_9) Chain Response	**Check Career** **Check Medical History**	$\neg$(***Coverage*** $\leq$ **5**) ***Cost*** $>$ **100**	**0.59**	**1.0**
(C_{10}) Precedence	Submit Loan Application Notify Outcome	*Salary* $\geq$ 71 000 *Result* $=$ *Accepted*	0.09	0.23
Precedence	Submit Loan Application Notify Outcome	*Salary* $<$ 71 000 *Result* $\neq$ *Accepted*	0.61	1.0

removed duplicated events. An overview of the logs' characteristics is provided in Table 6. All the logs used in evaluation as well as instructions on how to reproduce experiments are publicly available.[12]

5.2. RQ1: Rediscovery accuracy

The objective of this first experiment was to test whether the proposed approaches are able to rediscover the original constraints that were injected into a synthetic log. The constraints in question are those shown in Fig. 1.

We evaluated two different variants of the K-Medoids + RIPPER approach: one (described in Section 4.3), where the fulfillment feature vectors to be clustered contain both the activation and target payloads, and a variant of it, where the fulfillment feature vectors only contain target payloads. In this way, we tested whether the information about activation payloads in the clustering phase is relevant to improve the accuracy of the results. For the Redescription Mining approach, we used the ReReMi algorithm with default settings and the SpliT algorithm with maximum tree depth of 100 and minimum size of a node equal to 5% of the number of feature vectors.

To measure the rediscovery accuracy we used recall, precision, and F-score. In this setting, *recall* is the percentage of injected constraints that were correctly rediscovered, while *precision* represents the fraction of discovered constraints that match the

12 https://doi.org/10.6084/m9.figshare.10565906.v2

Table 9
Rediscovery results: ReReMi.

Template	Activation/Target	Activation/Target condition	Support	Confidence
(C_3) Response	**Submit Loan Application** **Assess Application**	***Salary*** $\geq$ **25 000** & ***Amount*** $\leq$ **99 000** ***Assessment Type*** $=$ ***Simple***	**0.55**	**1.0**
(C_3) Response	**Submit Loan Application** **Assess Application**	***Salary*** $\geq$ **25 000** & ***Amount*** $\leq$ **99 000** ***Assessment Cost*** $\leq$ **100**	**0.55**	**1.0**
Response	Submit Loan Application Assess Application	*Amount* $\leq$ 138 000 *Assessment Cost* $\leq$ 210	0.76	0.92
Response	Submit Loan Application Assess Application	*Amount* $\leq$ 99 000 *Assessment Type* $=$ *Simple*	0.56	0.78
Response	Submit Loan Application Assess Application	*Salary* $\leq$ 24 000 *Assessment Cost* $\geq$ 110	0.31	0.96
Response	Submit Loan Application Assess Application	*Amount* $\geq$ 139 000 *Assessment Cost* $\geq$ 220	0.05	0.29
Response	Submit Loan Application Notify Outcome	*Salary* $\leq$ 70 000 *Result* $=$ *Rejected*	0.61	0.67
(C_6) Responded Existence	**Assess Application** **Check Career**	***Assessment Type*** $=$ ***Simple*** ***Coverage*** $\leq$ **15**	**0.56**	**1.0**
(C_7) Responded Existence	**Assess Application** **Check Career**	***Assessment Type*** $=$ ***Complex*** ***Coverage*** $\geq$ **16**	**0.44**	**1.0**
(C_6) Responded Existence	**Assess Application** **Check Career**	***Assessment Cost*** $\leq$ **100** ***Coverage*** $\leq$ **15**	**0.56**	**1.0**
(C_7) Responded Existence	**Assess Application** **Check Career**	***Assessment Cost*** $\geq$ **110** ***Coverage*** $\geq$ **16**	**0.44**	**1.0**
(C_8) Chain Response	**Check Career** **Check Medical History**	***Coverage*** $\leq$ **5** ***Cost*** $\leq$ **100**	**0.41**	**1.0**
(C_9) Chain Response	**Check Career** **Check Medical History**	***Coverage*** $\geq$ **6** ***Cost*** $\geq$ **110**	**0.59**	**1.0**
Precedence	Submit Loan Application Notify Outcome	*Salary* $\leq$ 70 000 *Result* $=$ *Rejected*	0.61	1.0

Table 10
Rediscovery results: SplitT.

Template	Activation/Target	Activation/Target condition	Support	Confidence
(C_1) Response	**Submit Loan Application** **Assess Application**	***Amount*** $\geq$ **51 000** & $\neg$(***Salary*** $\geq$ **25 000**) & $\neg$(***Amount*** $\geq$ **101 000**) ***Assessment Type***! $=$ ***Simple***	**0.158**	**1.0**
(C_2) Response	**Submit Loan Application** **Assess Application**	***Amount*** $\geq$ **101 000** ***Assessment Type*** $\neq$ ***Simple***	**0.28**	**1.0**
(C_3) Response	**Submit Loan Application** **Assess Application**	***Salary*** $\geq$ **25 000** & $\neg$(***Amount*** $\geq$ **101 000**) ***Assessment Type*** $=$ ***Simple***	**0.55**	**1.0**
(C_4) Response	**Submit Loan Application** **Assess Application**	$\neg$(***Amount*** $\geq$ **51 000**) & $\neg$(***Salary*** $\geq$ **25 000**) & $\neg$(***Amount*** $\geq$ **101 000**) ***Assessment Type*** $=$ ***Simple***	**0.01**	**1.0**
(C_6) Responded Existence	**Assess Application** **Check Career**	***Assessment Type*** $=$ ***Simple*** $\neg$(***Coverage*** $\geq$ **16**)	**0.56**	**1.0**
(C_7) Responded Existence	**Assess Application** **Check Career**	$\neg$(***Assessment Type*** $=$ ***Simple***) ***Coverage*** $\geq$ **16**	**0.44**	**1.0**
Precedence	Submit Loan Application Notify Outcome	$\neg$(*Salary* $\geq$ 71 000) *Result* $=$ *Rejected*	0.61	1.0
(C_{10}) Precedence	Submit Loan Application Notify Outcome	*Salary* $\geq$ 71 000 $\neg$(*Result* $=$ *Rejected*)	0.09	0.23

injected behavior. *F-score* is the harmonic mean of precision and recall.

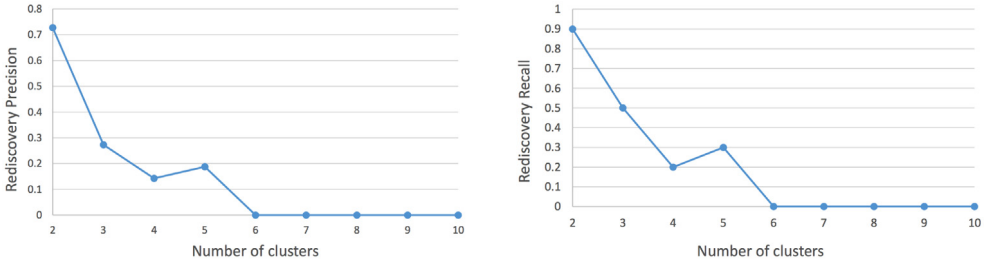
The constraints discovered by the algorithms are shown in Tables 7–10. The constraints that were correctly re-discovered are marked in bold and the id identifying the constraint in the original model in Fig. 1 is specified between brackets. Sometimes a constraint was not correctly re-discovered, but instead a semantically similar version of it was discovered. In this case, the id identifying the constraint in the original model is specified between brackets, but the row is not marked in bold.

As we can see from the tables, sometimes the rediscovered constraints were not the same as the injected ones, but they were semantically identical (e.g., "Simple Assessment" always goes in pair with "Assessment Cost" lower than or equal to 100, therefore these two conditions are interchangeable). The variant of K-Medoids + RIPPER that clusters only target payloads discovered a less accurate set of constraints when compared to its alternative (clustering of both activation and target payloads). In particular, often, in the former, the wrong separating point between clusters (e.g., *Cost* $<$ 150 and *Cost* $\geq$ 150) was selected.

The SplitT algorithm did not discover any constraints involving numerical attributes only (e.g., C_8, C_9). On the other hand, ReReMi discovered several redundant constraints describing the same behavior from different angles (see e.g., the first two constraints in Table 9). Although SplitT discovered the smallest number of constraints, the complexity of their data conditions is the highest. In contrast, ReReMi discovered the largest number of constraints with the smallest average length of the data conditions (both activation and target conditions mostly consist of one atomic condition only).

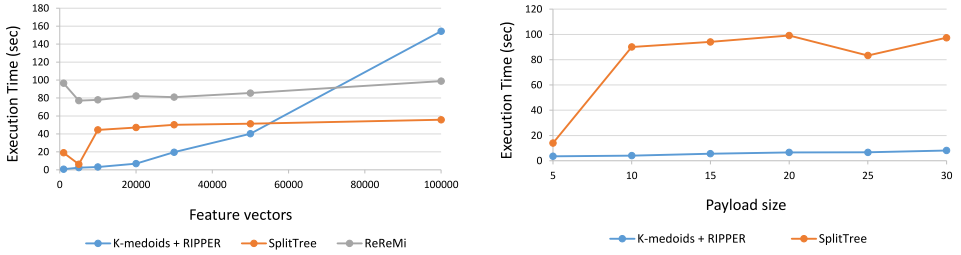
Table 11
Rediscovery accuracy.

Approach	Rules discovered	Recall	Precision	F-score
K-Medoids + RIPPER (clustering target payloads only)	11	0.5	0.36	0.43
K-Medoids + RIPPER (clustering activation + target payloads)	11	**0.9**	0.72	**0.81**
ReReMi	14	0.5	0.43	0.46
SplitT	8	0.6	**0.75**	0.67



(a) Number of clusters VS Recall (b) Number of clusters VS Precision

Fig. 5. Number of clusters VS Rediscovery Accuracy.



(a) Varying the number of feature vectors (b) Varying the payload size

Fig. 6. Scalability results (artificial logs).

Table 12
Scalability results (real-life logs).

Dataset	K-medoids + RIPPER (k = 2)	K-medoids + RIPPER (k = 5)	K-medoids + RIPPER (k = 10)	SplitT	ReReMi
BPIC 2011	**7.004**	12.904	22.489	60.612	128.220
BPIC 2013 (closed problems)	**3.727**	4.781	6.133	42.814	167.327
BPIC 2013 (incidents)	**13.852**	20.549	38.667	112.569	205.404
BPIC 2013 (open problems)	**2.967**	4.222	5.469	61.849	145.925
BPIC 2017	**26.927**	65.419	133.208	76.925	237.278
Sepsis	4.434	6.339	11.517	**1.032**	3.359
Traffic fines process	175.944	542.911	1364.352	38.669	**31.598**

Table 11 reports the rediscovery accuracy in terms of number of constraints discovered, recall, precision, and F-score for all four approaches. The results show that K-Medoids + RIPPER with clustering of activation and target payloads was able to rediscover almost all the original constraints (with a recall of 0.9). Meanwhile, SplitT achieved the highest rediscovery precision (0.75). Overall, K-Medoids + RIPPER with clustering of activation and target payloads has the highest F-score. Given that clustering of both activation and target payloads clearly leads to higher rediscovery precision and recall than clustering of only the target payloads, in the next experiments, we only consider the former variant.

We also checked how rediscovery precision and recall are affected by the number of clusters for the K-Medoids + RIPPER approach (with clustering of both activation and target payloads). The results are shown in Fig. 5. We observe that the number of clusters significantly affects the rediscovery accuracy. As the number of clusters increases starting from the original one (in this particular case, 2), precision and recall decrease to the point that it is no longer possible to rediscover any originally-injected behavior.

5.3. RQ2: Scalability

We tested the scalability of the three approaches (K-Medoids + RIPPER clustering of activation and target payloads, ReReMi and SplitT) on both artificial and real-life logs. Specifically, we measured how the execution time varies based on the size of the log in terms of number of feature vectors and event payload size. The experiments have been run on a Windows 10 ×64 machine, equipped with Intel Core i5-52000U CPU 2.20 GHz, using 16 GB of memory.

The results are shown in Fig. 6. We can see that the execution time of K-Medoids + RIPPER has a stronger dependency on the total number of feature vectors than the two Redescription Mining approaches. In particular, the execution time for K-Medoids

Table 13
Rule-set characteristics on real-life logs.

Dataset	Approach	Avg. No. of conditions	Avg. condition length	Avg. confidence	Avg. support
BPIC 2011	K-medoids + RIPPER (k = 2)	**2.75**	2.97	0.943	**0.379**
	K-medoids + RIPPER (k = 5)	4.00	2.86	0.764	0.177
	K-medoids + RIPPER (k = 10)	3.74	3.35	0.752	0.146
	SplitT	32.35	**2.72**	**0.993**	0.152
	ReReMi	59.00	3.77	0.973	0.172
BPIC 2013_1	K-medoids + RIPPER (k = 2)	**2.33**	2.99	**0.982**	**0.445**
	K-medoids + RIPPER (k = 5)	4.06	3.05	0.806	0.194
	K-medoids + RIPPER (k = 10)	4.22	3.03	0.760	0.151
	SplitT	21.83	**2.20**	0.966	0.222
	ReReMi	98.72	3.50	0.921	0.260
BPIC 2013_2	K-medoids + RIPPER (k = 2)	**2.00**	**2.13**	0.948	**0.527**
	K-medoids + RIPPER (k = 5)	3.47	2.98	0.835	0.243
	K-medoids + RIPPER (k = 10)	4.33	3.43	0.763	0.174
	SplitT	22.56	2.24	**0.954**	0.202
	ReReMi	99.33	4.11	0.953	0.177
BPIC 2013_3	K-medoids + RIPPER (k = 2)	**2.27**	2.68	**0.977**	**0.482**
	K-medoids + RIPPER (k = 5)	4.09	3.08	0.841	0.212
	K-medoids + RIPPER (k = 10)	4.64	3.33	0.801	0.155
	SplitT	27.45	**2.09**	0.974	0.193
	ReReMi	95.09	2.78	**0.977**	0.160
BPIC 2017	K-medoids + RIPPER (k = 2)	**2.32**	**2.49**	0.803	**0.394**
	K-medoids + RIPPER (k = 5)	3.32	4.20	0.616	0.185
	K-medoids + RIPPER (k = 10)	2.74	3.89	0.462	0.134
	SplitT	21.63	2.90	**0.990**	0.228
	ReReMi	50.89	4.85	0.940	0.296
Sepsis	K-medoids + RIPPER (k = 2)	**1.16**	**2.63**	**0.888**	**0.812**
	K-medoids + RIPPER (k = 5)	1.47	2.84	0.647	0.527
	K-medoids + RIPPER (k = 10)	1.82	3.48	0.573	0.419
	SplitT	–	–	–	–
	ReReMi	–	–	–	–
Traffic Fines	K-medoids + RIPPER (k = 2)	**2.00**	**2.46**	0.814	**0.457**
	K-medoids + RIPPER (k = 5)	3.00	4.30	0.661	0.238
	K-medoids + RIPPER (k = 10)	3.29	4.12	0.667	0.201
	SplitT	8.57	4.24	**0.853**	0.293
	ReReMi	8.07	3.88	0.843	0.442

+ RIPPER rapidly rises with the increase of the total number of feature vectors. On the other hand, the execution time for K-Medoids + RIPPER is less sensitive to the payload size. Note that, in Fig. 6(b), we do not report the execution time of ReReMi since it is exponentially growing from around 300 s for a payload of size 5 to more than 1 h for a payload with 15 attributes.

For evaluating the scalability of the approaches on the real-life logs, we used K-Medoids + RIPPER with three different settings (k equal to 2, 5 and 10), ReReMi with default settings and SpliT with maximum tree depth of 100 and minimum size of a node equal to 5% of the number of feature vectors. The results are presented in Table 12.

In most cases, K-Medoids + RIPPER outperforms the other approaches in terms of time performance. This can be observed for all logs except for the Road Traffic Fines Management and Sepsis logs. This can be related to the fact that these logs contain a weaker signal (smaller event payloads for Sepsis, shorter traces for Road Traffic Fines Management), which also leads Redescription Mining approaches to discover less data conditions.

5.4. RQ3: Characteristics of the discovered data conditions using real-life logs

To answer RQ3, we used the real-life logs to discover a set of control-flow DECLARE constraints with the Declare Miner [20]. Then, for each discovered control-flow constraint, we ran all the proposed approaches, recording the number of discovered data conditions, their average length, and the average support and confidence of the enhanced constraints derived from the original control-flow constraint. Finally, we computed the average of each of the recorded metrics over all the discovered DECLARE constraints. The setup of each approach is the same as the one used to answer RQ2 (see Section 5.3). The results are presented in Table 13.

From Table 13, we can see that K-Medoids + RIPPER with k equal to 2 discovers the smallest number of data conditions, while keeping the constraint confidence high (always greater than 0.8). Overall, K-Medoids + RIPPER produces less conditions than the two Redescription Mining approaches. This is due to the fact that RIPPER minimizes the number of conditions discovered, and these conditions do not overlap. On the other hand, ReReMi and SpliT try to discover all possible conditions according to the input parameters. Often, this leads to discovering redundant rules. On the other hand, the Redescription Mining approaches tend to discover rules with very high confidence, while we can see that, for K-Medoids + RIPPER, the confidence of the discovered constraints decreases as k increases.

5.5. Threats to validity

A possible threat to internal validity is posed by the use of a single log to address RQ1 (rediscovery accuracy). The log is generated using a limited set of constraints and these constraints are relatively simple, not involving a high number of attributes at the same time. To mitigate this threat, we decided to address RQ1 at a qualitative level instead of approaching this question quantitatively. In particular, we tried to rediscover constraints of different types, and with data conditions involving both categorical and numerical attributes.

A further threat to internal validity is the limited use of parameter values to configure the approaches at hand. Specifically,

in the case of ReReMi and SpliT, we used default parameters and increased the maximal depth of the trees only, while in the case of K-Medoids + RIPPER, we used k equal to 2, 5 and 10.

A potential threat to external validity is given by the use of a limited number of real-life logs (seven). However, these logs originate from different domains and exhibit different characteristics, providing a good representative set of real-life event logs. To ensure the full reproducibility of the results, we have released all the preprocessed real-life logs as well as the ones artificially generated used in our experiments.

6. Conclusion

We presented two approaches to enhance DECLARE constraints with data conditions that relate the occurrence of pairs of events in a case of an event log (correlated data conditions). The first approach combines clustering and rule mining techniques, while the second approach relies on Redescription Mining.

Overall, the experimental results show that the clustering-based approach outperforms Redescription Mining in terms of its ability to rediscover constraints artificially injected in a log, in terms of number of conditions discovered (lower number of conditions), and in terms of computational efficiency, when the number of feature vectors is not significantly high. However, the experiments showed that the accuracy of the clustering-based technique is highly dependent on the number of clusters given in input. Hence, this technique requires careful parameter tuning.

The experiments also showed that the Redescription Mining approaches discover constraints with higher confidence (and lower support). This latter observation suggests that these techniques may be used to effectively discover outlier behaviors, i.e., constraints that are less frequently activated, but, when activated, are in most of the cases satisfied.

While we have shown that the proposed techniques address the problem of discovering DECLARE constraints with correlated data conditions and that they scale up to handle real-life event logs, the usefulness of the discovered rules in practical settings still needs to be studied. In this respect, a possible avenue for future work is to conduct case studies to determine if the sets of constraints produced by the proposed techniques can provide insights to business analysts, beyond what can be achieved by simply discovering plain (non-data-enhanced) DECLARE constraints or, alternatively, by discovering procedural process models enhanced with data conditions.

Declaration of competing interest

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

Acknowledgments

This work is partly supported by the Estonian Research Council (IUT20-55) and by the Australian Research Council (DP180102839).

References

[1] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, H.A. Reijers, Imperative versus declarative process modeling languages: An empirical investigation, in: BPM Workshops, 2011, pp. 383–394.
[2] H.A. Reijers, T. Slaats, C. Stahl, Declarative modeling-An academic dream or the future for BPM? in: BPM, 2013, pp. 307–322.
[3] M. Pesic, H. Schonenberg, W.M.P. van der Aalst, DECLARE: Full support for loosely-structured processes, in: EDOC, 2007, pp. 287–300.
[4] M. Montali, M. Pesic, W.M.P. van der Aalst, F. Chesani, P. Mello, S. Storari, Declarative specification and verification of service choreographies, ACM Trans. Web 4 (1) (2010).
[5] M. Weidlich, A. Polyvyanyy, J. Mendling, M. Weske, Causal behavioural profiles - efficient computation, applications, and evaluation, Fund. Inform. 113 (3–4) (2011) 399–435.
[6] A. Polyvyanyy, M. Weidlich, R. Conforti, M. La Rosa, A.H.M. ter Hofstede, The 4C Spectrum of fundamental behavioral relations for concurrent systems, in: PETRI NETS 2014, 2014, pp. 210–232.
[7] V. Leno, M. Dumas, F.M. Maggi, Correlating activation and target conditions in data-aware declarative process discovery, in: BPM 2018, 2018, pp. 176–193.
[8] W.M.P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Comput. Sci. Res. Dev. 23 (2) (2009) 99–113.
[9] O. Kupferman, M.Y. Vardi, Vacuity detection in temporal model checking, Softw. Tools Technol. Transfer 4 (2) (2003) 224–233.
[10] F.M. Maggi, M. Montali, C. Di Ciccio, J. Mendling, Semantical vacuity detection in declarative process mining, in: Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings, 2016, pp. 158–175.
[11] C. Di Ciccio, F.M. Maggi, M. Montali, J. Mendling, On the relevance of a business constraint to an event log, Inf. Syst. 78 (2018) 144–161.
[12] A. Burattin, F.M. Maggi, W.M.P. van der Aalst, A. Sperduti, Techniques for a posteriori analysis of declarative processes, in: EDOC, Beijing, 2012, pp. 41–50.
[13] F.M. Maggi, Declarative process mining with the Declare component of ProM, in: BPM (Demos), 2013.
[14] A. Burattin, F.M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, Expert Syst. Appl. 65 (2016) 194–211.
[15] F.M. Maggi, M. Dumas, L. García-Bañuelos, M. Montali, Discovering data-aware declarative process models from event logs, in: BPM, Vol. 8094, 2013, pp. 81–96.
[16] R.P.J.C. Bose, F.M. Maggi, W.M.P. van der Aalst, Enhancing Declare maps based on event correlations, in: Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings, 2013, pp. 97–112.
[17] S. Schönig, C. Di Ciccio, F.M. Maggi, J. Mendling, Discovery of multi-perspective declarative process models, in: ICSOC, 2016, pp. 87–103.
[18] A. Rozinat, W.M.P. van der Aalst, Decision mining in ProM, in: BPM 2006, 2006, pp. 420–425.
[19] M. de Leoni, M. Dumas, L. García-Bañuelos, Discovering branching conditions from business process execution logs, in: FASE 2013, 2013, pp. 114–129.
[20] F.M. Maggi, R.P.J.C. Bose, W.M.P. van der Aalst, Efficient discovery of understandable declarative process models from event logs, in: CAiSE 2012, 2012, pp. 270–285.
[21] T. Zinchenko, E. Galbrun, P. Miettinen, Mining predictive redescriptions with trees, in: ICDMW 2015, 2015, pp. 1672–1675.
[22] E. Galbrun, P. Miettinen, From black and white to full color: extending redescription mining outside the Boolean world, Stat. Anal. Data Min. 5 (4) (2012) 284–303.
[23] V. Skydanienko, C. Di Francescomarino, C. Ghidini, F.M. Maggi, A tool for generating event logs from multi-perspective Declare models, in: Dissertation Award, Demonstration, and Industrial Track at BPM 2018, 2018, pp. 111–115.