Open access • Book Chapter • DOI:10.1007/978-3-319-98648-7_11

Correlating Activation and Target Conditions in Data-Aware Declarative Process Discovery — Source link

Volodymyr Leno, Volodymyr Leno, Marlon Dumas, Fabrizio Maria Maggi

Institutions: University of Tartu, University of Melbourne

Published on: 09 Sep 2018 - Business Process Management

Topics: Process mining, Business process discovery, Business Process Model and Notation, Business process and Process modeling

Correlating Activation and Target Conditions in Data-Aware Declarative Process Discovery*

Volodymyr Leno[1,2], Marlon Dumas[1], and Fabrizio Maria Maggi[1]

[1] University of Tartu, Estonia
{leno,marlon.dumas,f.m.maggi}@ut.ee
[2] University of Melbourne, Australia

Abstract. Automated process discovery is a branch of process mining that allows users to extract process models from event logs. Traditional automated process discovery techniques are designed to produce procedural process models as output (e.g., in the BPMN notation). However, when confronted to complex event logs, automatically discovered process models can become too complex to be practically usable. An alternative approach is to discover declarative process models, which represent the behavior of the process in terms of a set of business constraints. These approaches have been shown to produce simpler process models, especially in the context of processes with high levels of variability. However, the bulk of approaches for automated discovery of declarative process models are focused on the control-flow perspective of business processes and do not cover other perspectives, e.g., the data, time, and resource perspectives. In this paper, we present an approach for the automated discovery of multi-perspective declarative process models able to discover conditions involving arbitrary (categorical or numeric) data attributes, which relate the occurrence of pairs of events in the log. To discover such correlated conditions, we use clustering techniques in conjunction with interpretable classifiers. The approach has been implemented as a proof-of-concept prototype and tested on both synthetic and real-life logs.

1 Introduction

Process mining is a family of techniques for analyzing business processes starting from their executions as recorded in event logs [20]. Process discovery is the most prominent process mining technique. A process discovery technique takes an event log as input and produces a model without using any a-priori information. The dichotomy procedural versus declarative when choosing the most suitable language to represent the output of a process discovery technique has been largely discussed [15, 17]: procedural languages can be used for predictable processes working in stable environments, whereas declarative languages can be used for unpredictable, variable processes working in highly unstable environments. A still open challenge in the discovery of declarative process models is to develop techniques taking into consideration not only the control flow perspective of a business process but also other perspectives like the data, time, and resource perspectives.

* Work supported by the Estonian Research Council (IUT20-55)

In the current contribution, we present an approach that tries to address this challenge. We base our approach on DECLARE a declarative language to represent business processes [14]. In particular, we use the multi-perspective extension of DECLARE, MP-DECLARE, presented in [5]. The proposed approach can be seen as a step forward with respect to the one presented in [12]. In this preliminary work, the discovered models include data conditions to discriminate between cases in which a constraint is satisfied and cases in which the constraint is violated. For example, in a loan application process, we could discover a constraint telling that the submission of an application is eventually followed by a medical history check when the submission has an amount higher than 100 000 euros. Otherwise, the medical check is not performed. In the example above, we have that a response DECLARE constraint (the submission of an application is eventually followed by a medical history check) is satisfied only when a certain condition on the payload of the activation (on the amount associated to the submission of the application) is satisfied.

In this paper, we present an approach to infer two correlated conditions on the payloads of the activation (activation condition) and of the target (target condition) of a constraint. For example, we can discover behaviors 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. The approach starts with the discovery of a set of *frequent constraints*. 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. Starting from the constraint instances of a frequent constraint, the algorithm clusters the target payloads to find groups of targets with similar payloads. Then, these groups are used as labels for a classification problem. These labels together with the features extracted from the activation payloads are used to train an interpretable classifier (a decision tree). This procedure allows for finding correlations between the activation payloads and the target payloads. The proposed technique is agnostic on how the input set of frequent constraints is derived. In the context of this paper, we identify these constraints using the semantics of MP-DECLARE. The approach has been validated with several synthetic logs to show its ability to rediscover behaviors artificially injected in the logs and its scalability. In addition, the approach has been applied to 6 real-life logs in the healthcare and public administration domains.

The paper is structured as follows. Section 2 provides the necessary background to understand the rest of the paper. Section 3 presents an exemplifying MP-DECLARE model. Section 4 illustrates the proposed discovery approach and Section 6 its evaluation. Finally, Section 7 provides some related work, and Section 8 concludes the paper and spells out directions for future work.

2 Preliminaries

In this section, we first introduce the XES standard (Section 2.1), then we give some background knowledge about Declare (Section 2.2) and MP-DECLARE (Section 2.3).

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

2.1 The XES Standard

The starting point for process mining is an event log. XES (eXtensible Event Stream) [1, 22] has been developed as the standard for storing, exchanging and analyzing event logs. Each event in a log refers to an *activity* (i.e., a well-defined step in some process) and is related to a particular *case* (i.e., a *process instance*). The events belonging to a case are *ordered* and can be seen as one "run" of the process (often referred to as a *trace* of events). Event logs may store additional information about events such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event. In XES, data elements can be event attributes, i.e., data produced by the activities of a business process and case attributes, namely data that are associated to a 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.

Let Σ be the set of unique activities in the log. Let $t \in \Sigma^*$ be a trace over Σ, i.e., a sequence of activities performed for one process case. An event log E is a multi-set over Σ^*, i.e., a trace can appear multiple times.

2.2 Declare

DECLARE is a declarative process modeling language originally introduced by Pesic and van der Aalst in [14]. 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 applied to 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). They have a graphical representation understandable to the user and their semantics can be formalized using different logics [13], the main one being LTL over finite traces, 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 [21] 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**, and **Y** LTL (past) operators have the following meaning: $\mathbf{O}\phi_1$ means that ϕ_1 holds sometime in the past, and $\mathbf{Y}\phi_1$ means that ϕ_1 holds in the previous position.

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* [10]. In [6], the authors introduce the notion of *behavioral vacuity detection* according to which a constraint is non-vacuously satisfied in a trace when it is activated in that trace. 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 an activation for the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$ and b is a 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 [6].

Table 2: Semantics for Multi-Perspective 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	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y))))$

Tools implementing process mining approaches based on DECLARE are presented in [11]. 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 [5]. 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 [5] for more details. To explain the semantics, we have to introduce some preliminary notions.

The first concept we use here 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 itself S and we call $(12\,500, 55\,000)$, the *payload* of S.

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, and a time condition. 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.

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. In case the applicant submits a loan application with an amount lower than 50 000 euros or she has a salary higher than 24 000 euros per year, eventually a simple assessment will be carried out and the cost of the assessment will be lower than or equal to 100 euros. This behavior is described by response constraint C_3. 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. This behavior is described by response constraint C_2 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. This behavior is described by precedence constraint C_6. 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_5). 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_4). 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

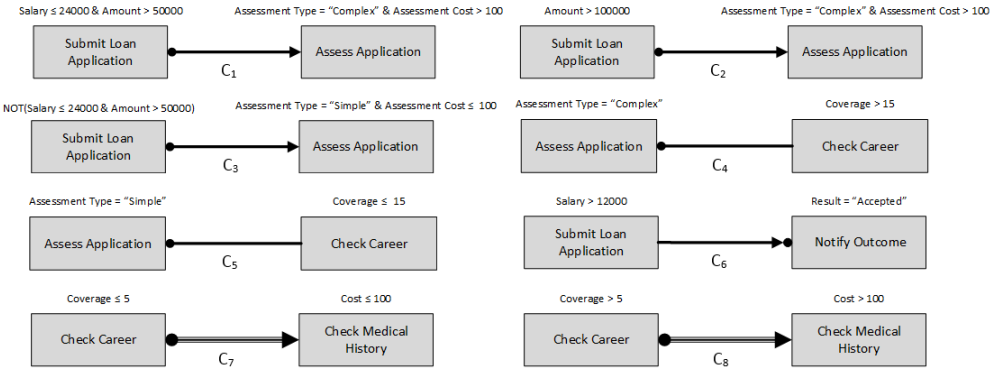


Fig. 1: Running Example.

constraint C_7). 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_8 in Fig. 1.

4 Discovery Approach

The proposed approach is shown in Fig. 2. The approach starts with the discovery of a set of *frequent constraints*. 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 addition, for each frequent constraint, also activations that cannot be associated to any target (representing a violation for the constraint) are identified. Feature vectors are extracted from the payloads of these activations and associated with a label indicating that they correspond to violations of the constraint (*violation feature vectors*). (Unlabeled) feature vectors are also extracted by using the payloads of targets of the constraint instances identified in the first phase. These feature vectors are then clustered using DBSCAN clustering [8] to find groups of targets with similar payloads. Then, these clusters are used as labels for a classification problem. These labels together with the features extracted from the activation payloads are used to generate a set of *fulfillment feature vectors*. Violation and fulfillment feature vectors are used to train a decision tree. This procedure allows for finding correlations between the activation payloads and the target payloads. Note that the core part of our approach (highlighted with a blue rectangle in Fig. 2) is independent of the procedure used to identify frequent constraints and can be used in combination with other techniques for frequent constraint mining (also based on semantics that go beyond MP-Declare).

4.1 Frequent Constraints Discovery

The first step of our discovery algorithm is to identify a set of frequent constraints. In particular, the user specifies a Declare template (e.g., response)

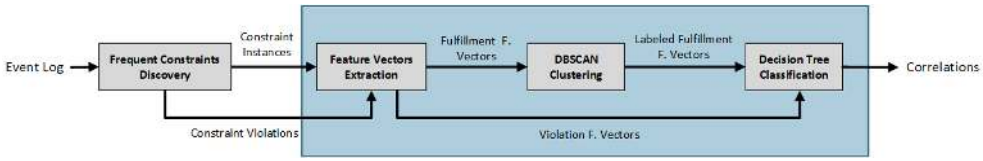


Fig. 2: Proposed Approach.

and, starting from the input template and the input log, a set of candidate constraints is instantiated. To generate the set of candidate constraints, we use the idea behind the well-known Apriori algorithm presented in [4]. In particular, the algorithm first searches for frequent individual activities in the input log. We call the absolute frequency of an activity in the log *activity support*. Individual activities with an activity support lower than an input threshold $supp_{min}$ are filtered out.

The input template is instantiated with all the possible combinations of frequent activities thus identifying a set of candidate constraints. Note that, unlike the classical Apriori algorithm that works with unordered itemsets, the order inside a candidate constraint plays an important role (i.e., RESPONSE(S,A) is not the same as RESPONSE(A,S)). In particular, we work with pairs where the first element is the activation of the constraint and the second element is the target. At this point, the algorithm evaluates, for each candidate constraint, the number of its constraint instances in the log. We call this measure *constraint support*. In particular, this measure is calculated by creating two vectors idx_1 and idx_2 that represent the activation and target occurrences. Then, the algorithm processes each trace in the input log to find the events corresponding either to the activation or the target of the candidate 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 number of constraint instances is computed 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}.
- **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, the idx_1 vector has to be reversed.
- **(Not) Responded Existence.** We associate each element idx_{1i} from the activation vector idx_1, with first element from the target vector idx_2.

If we enumerate the occurrences of S and A in t, we have $t = S_1S_2S_3A_1S_4A_2S_5S_6A_3$. The constraint instances of the standard DECLARE templates instantiated with activities (A, S) and (S, A) are listed in Table 3 and in

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)	$\{A_1S_1\}$, $\{A_2S_1\}$, $\{A_3S_1\}$

Table 4. Note that, these procedures can also be used to identify constraint violations (i.e., activations that cannot be associated to any target). We filter out candidate constraint with a support that is lower than $supp_{min}$ thus obtaining a set of frequent constraints. The constraint instances of the frequent constraints are used for creating fulfillment feature vectors. Activations that do not have a target 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 Feature Vectors Extraction

Violation feature vectors consist of the payloads of activations of frequent constraints that 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:

$$V_{viol} = [12\,500; 55\,000; violated]. \tag{1}$$

If the same activation is part of a constraint instance of a frequent constraint with target Assess Application and payload $(Complex, 140)$, we generate the (unlabeled) fulfillment feature vector:

$$V_{ful} = [Complex; 140]. \tag{2}$$

The violation feature vectors are used for interpretable classification (see section 4.4). Fulfillment feature vectors are used for clustering with DBSCAN.

4.3 Clustering with DBSCAN

Starting from the fulfillment feature vectors, we use the Density Based Spatial Clustering of Application with Noise (DBSCAN) [8] to find groups of payloads that are similar.

Given a set of points in some space DBSCAN groups together vectors that are closely packed, marking as outliers vectors that lie in low-density regions. To do this, we use the Gower distance

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

where n denotes the number of features, while $d_{i,j}^{(f)}$ is a distance between data points i and j when considering only feature *f*. $d_{i,j}^{(f)}$ is a normalized distance. For nominal attribute values, we calculate the normalized Edit Levenshtein distance [9]. Edit distance is a way of quantifying how dissimilar two strings are by counting the minimum number of operations (i.e., removal, insertion, substitution of a character) required to transform one string into other. The normalized edit distance is calculated as:

$$d_{i,j}^{(f)} = \frac{EditDistance(x_i^{(f)}, x_j^{(f)})}{maxEditDistance(x_i^{(f)}, x_j^{(f)})}. \tag{4}$$

For interval scaled attribute values, we use the distance:

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

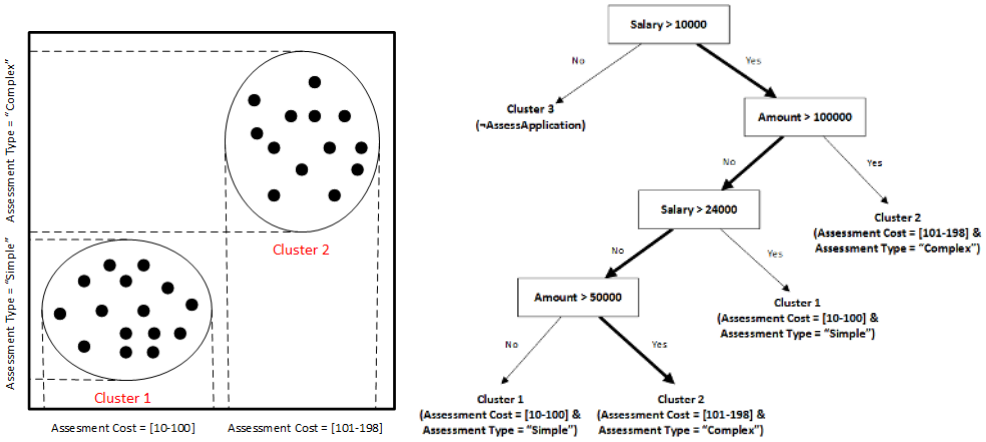
where min and max are the maximum and minimum observed values of attribute f. For boolean attribute values, we use the distance:

$$d_{i,j}^{(f)} = \begin{cases} 0, & \text{if } x_i^{(f)} = x_j^{(f)} \\ 1, & \text{if } x_i^{(f)} \neq x_j^{(f)}. \end{cases} \tag{6}$$

When obtained the clusters, we project the target payload attributes in order to describe the characteristics of the elements of the clusters. For numerical attributes, the projection results in the range [min-max], where *min* and *max* are the minimum and maximum values of the attribute. When projecting onto categorical attributes, we take the most frequent value. For example, Fig. 3a shows two clusters associated to a frequent constraint with target Assess Application. One of them is characterized by the condition $Assessment\ Cost = [10 - 100]$ & $Assessment\ Type = Simple$, while the second one by the condition $Assessment\ Cost = [101 - 198]$ & $Assessment\ Type = Complex$. These clusters/conditions are used as labels to build labeled fulfillment feature vectors. 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:

$$V'_{ful} = [12\,500; 55\,000; Cluster2]. \tag{7}$$

Labeled fulfillment feature vectors and violation feature vectors are used for interpretable classification using decision trees.



(a) Clusters projections (b) Interpretable classification

Fig. 3: Core steps of the proposed approach.

4.4 Interpretable Classification

After having created labeled fulfillment feature vectors and violation feature vectors, we use them to train a decision tree. The C4.5 algorithm is used to perform the classification [16]. The data is split in a way that the resulting parts are more homogenous. The algorithm uses entropy and information gain to choose the split. We can express the overall entropy as:

$$H(Y) = -\sum_{a \in Dom(Y)} Pr[Y = a] * log_2(Pr[Y = a]). \tag{8}$$

The information gain can be calculated as:

$$Gain = H_S(Y) - \sum_{i=1}^{k} \frac{|S_i|}{|S|} * H_{S_i}(Y), \tag{9}$$

where $|S|$ the size of the dataset and $|S_i|$ is the size of the split i. $H_S(Y)$ denotes the entropy computed over S and $H_{S_i}(Y)$ denotes the entropy computed over S_i. The information gain measures how much the split makes the target value distribution more homogenous. We select the split with maximal information gain. The splitting is stopped either when information gain is 0 (we will not improve the results when splitting) or when the size of the split is smaller than an input support threshold.

Fig. 3b shows a decision tree generated from our running example. Correlations in the activation and target payloads are found by correlating the activation conditions derived from paths from the root to the leaves of the decision tree with the target conditions labeling each leaf of the decision tree. For example,

from the path highlighted with thicker arcs in Fig. 3b, we can extract the activation condition $Salary \leq 24000$ & $Amount > 50000$ & $Amount \leq 100000$. If the frequent constraint was a response constraint, the discovery algorithm produces the MP-DECLARE constraint RESPONSE(SUBMIT LOAN APPLICATION, ASSESS APPLICATION) with activation condition $Salary \leq 24000$ & $Amount > 50000$ & $Amount \leq 100000$ and target condition $Assessment\ Cost = [101 - 198]$ & $Assessment\ Type = Complex$.

Each leaf of the decision tree (and therefore each pair of discovered correlated conditions) is associated with a *support* and a *confidence*. Support represents the number of feature vectors that follow the path from the root to the leaf and that are correctly classified; *confidence* is the percentage of vectors correctly classified with respect to all the vectors following that specific path.

5 Algorithm complexity

The complexity for generating the set of candidate constraints is $O(a^2)$, where a is the number of distinct activities in the log. To calculate the constraint support of a candidate, we need $O(e)$ time, where e is the size of the log. Thus, the complexity of the frequent constraints discovery can be computed as $T = O(ae) + O(a^2e) = O(a^2e)$. The complexity of the feature vectors extraction is the same and equal to $O(a^2e)$. The average complexity of DBSCAN algorithm is $O(nlog(n))$, and $O(n^2)$ in the worst case, when the vectors have one feature [8]. Thus, the total complexity of the DBSCAN clustering is equal to $O(mn^2)$ in the worst case, where m is the number of features (size of the feature vectors) and n is the number of feature vectors. Since DBSCAN is applied to all frequent constraints, its complexity is equal to $O(a^2mn^2)$. In order to avoid the distances recomputations, a distance matrix of size $(n^2 - n)/2$ can be used. However, this needs $O(n^2)$ memory. In general, the runtime cost to construct a balanced binary tree is $O(mnlog(n))$, where m is the number of features and n is the number of feature vectors. Assuming that the subtrees remain approximately balanced, the cost at each node consists of searching through $O(m)$ to find the feature that offers the largest reduction in entropy. This has a cost of $O(mnlog(n))$ at each node, leading to a total cost over the entire tree of $O(mn^2log(n))$.[3] Considering the fact that we have a^2 candidate constraints, the complexity is $O(a^2mn^2log(n))$. Thus, the total complexity of the algorithm is $T = O(a^2e) + O(a^2e) + O(a^2mn^2) + O(a^2mn^2log(n)) = O(a^2mn^2log(n))$.

6 Experiments

The evaluation reported in this paper aims at understanding the potential of the proposed discovery approach. In particular, we want to examine the capability of the discovery approach to rediscover behavior artificially injected into a set of synthetic logs. In addition, we want to assess the scalability of the approach and its applicability to real-life datasets. In particular, we investigate the following three research questions:

[3] http://scikit-learn.org/stable/modules/tree.html

Table 5: Experimental results: Rediscovery

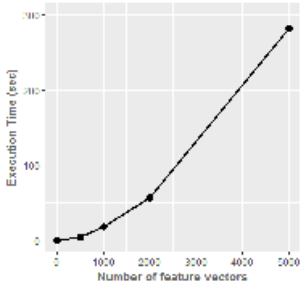
Template	Activation/Target	Activation/Target Condition	Support	Confidence
(C_2) Response	**Submit Loan Application** **Assess Application**	$\boldsymbol{Amount > 100000}$ $\boldsymbol{AssessmentCost = [101 - 198]}$ & $\boldsymbol{Assessment\ Type = Complex}$	**0.17**	**0.99**
(C_3) Response	Submit Loan Application Assess Application	$Amount \leq 100000$ $Assessment\ Cost = [10 - 100]$ & $Assessment\ Type = Simple$	0.76	0.91
(C_3) Response	Submit Loan Application Assess Application	$Salary > 24000$ & $Amount > 50000$ &$Amount \leq 100000$ $Assessment\ Cost = [10 - 100]$ & $Assessment\ Type = Simple$	0.15	0.95
(C_1) Response	**Submit Loan Application** **Assess Application**	$\boldsymbol{Salary \leq 24000}$ & $\boldsymbol{Amount > 50000}$ &$\boldsymbol{Amount \leq 100000}$ $\boldsymbol{Assessment\ Cost = [101 - 198]}$ & $\boldsymbol{Assessment\ Type = Complex}$	**0.05**	**1.0**
Response	Submit Loan Application Check Career	$Amount > 100000$ $Coverage = [16 - 25]$	0.09	0.52
Response	Submit Loan Application Check Career	$Amount \leq 100000$ $Coverage = [0 - 20]$	0.75	0.9
Response	**Submit Loan Application** **Notify Outcome**	$\boldsymbol{Salary \leq 12000}$ $\boldsymbol{Result = Rejected}$	**0.23**	**1.0**
Response	Submit Loan Application Notify Outcome	$Salary > 12000$ $Result = Accepted$	0.39	0.5
Response	**Assess Application** **Check Medical History**	$\boldsymbol{Assessment\ Cost > 100}$ $\boldsymbol{Cost = [101 - 199]}$	**0.23**	**1.0**
Response	**Assess Application** **Check Medical History**	$\boldsymbol{Assessment\ Cost \leq 100}$ $\boldsymbol{Cost = [10 - 200]}$	**0.77**	**0.99**
Response	Submit Loan Application Check Medical History	$Amount > 156868$ $Cost = [128 - 199]$	0.048	0.47
Response	Submit Loan Application Check Medical History	$Amount \leq 156868$ $Cost = [10 - 200]$	0.76	0.85
(C_4) Responded Existence	**Assess Application** **Check Career**	$\boldsymbol{Type = Complex}$ $\boldsymbol{Coverage = [16 - 30]}$	**0.22**	**1.0**
(C_5) Responded Existence	**Assess Application** **Check Career**	$\boldsymbol{Type = Simple}$ $\boldsymbol{Coverage \leq [0 - 15]}$	**0.77**	**0.99**
Precedence	**Submit Loan Application** **Notify Outcome**	$\boldsymbol{Result = Rejected}$ $\boldsymbol{Salary = [1022 - 99829]}$ & $\boldsymbol{Amount = [10028 - 249847]}$	**0.61**	**1.0**
(C_6) Precedence	**Submit Loan Application** **Notify Outcome**	$\boldsymbol{Result \neq Rejected}$ $\boldsymbol{Salary = [12001 - 99229]}$ & $\boldsymbol{Amount = [10267 - 248013]}$	**0.39**	**1.0**
Precedence	Submit Loan Application Check Career	$Coverage \leq 15$ $Salary = [1022 - 99822]$ & $Amount = [10028 - 100000]$	0.73	0.95
Precedence	Submit Loan Application Check Career	$Coverage > 15$ $Salary = [2551 - 24000]$ & $Amount = [160103 - 246486]$	0.04	0.2
Precedence	**Assess Application** **Check Career**	$\boldsymbol{Coverage > 15}$ $\boldsymbol{AssessmentCost = [101 - 198]}$ & $\boldsymbol{AssessmentType = Complex}$	**0.22**	**1.0**
Precedence	**Assess Application** **Check Career**	$\boldsymbol{Coverage \leq 15}$ $\boldsymbol{AssessmentCost = [10 - 100]}$ & $\boldsymbol{AssessmentType = Simple}$	**0.77**	**0.99**
Precedence	**Check Career** **Check Medical History**	$\boldsymbol{Cost > 100}$ $\boldsymbol{Coverage = [6 - 30]}$	**0.46**	**0.99**
Precedence	**Check Career** **Check Medical History**	$\boldsymbol{Cost \leq 100}$ $\boldsymbol{Coverage = [0 - 5]}$	**0.33**	**1.0**
Precedence	**Submit Loan Application** **Assess Application**	$\boldsymbol{AssessmentCost > 100}$ $\boldsymbol{Salary = [1145 - 24000]}$ & $\boldsymbol{Amount = [50246 - 249847]}$	**0.22**	**0.99**
Precedence	Submit Loan Application Assess Application	$AssessmentCost \leq 100$ $Salary = [1022 - 99829]$ & $Amount = [10028 - 100000]$	0.76	0.98
(C_8) Chain Response	**Check Career** **Check Medical History**	$\boldsymbol{Coverage > 5}$ $\boldsymbol{Cost = [101 - 200]}$	**0.46**	**0.99**
(C_7) Chain Response	**Check Career** **Check Medical History**	$\boldsymbol{Coverage \leq 5}$ $\boldsymbol{Cost = [10 - 100]}$	**0.52**	**1.0**

– **RQ1.** Does the proposed approach allow for rediscovering behavior artificially injected into a set of synthetic logs?

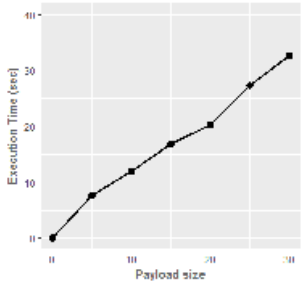
– **RQ2.** What is the time performance of the proposed approach when applied to logs with different characteristics?

– **RQ3.** Is the proposed approach applicable in real-life settings?

RQ1 focuses on the evaluation of the quality of the constraints returned by the proposed approach. **RQ2** investigates, instead, the scalability of the approach. Finally, **RQ3** deals with the validation of the discovery approach in real scenarios.



(a) Execution Time vs. Number of Feature Vectors



(b) Execution Time vs. Payload Size

Fig. 4: Experimental results: Scalability.

6.1 Rediscovering Injected Behavior (RQ1)

The most standard approach to test a process discovery algorithm is to artificially generate a log by simulating a process model and use the generated log to rediscover the process model. The log used for this experiment was generated by simulating the model in Fig. 1. The log contains 1 000 cases with 5 distinct activities (Submit Loan Application, Assess Application, Notify Outcome, Check Career, Check Medical History).[4]

The results of the rediscovery are shown in Table 5. The table shows that the approach manages to rediscover mostly all the constraints that generated the log. These constraints are among the ones with the highest confidence (0.99 or 1, highlighted in bold) and are explicitly indicated in the table. The only exception is constraint C_3 (including a disjunctive activation condition) that is included in the behavior described by 2 discovered constraints with a lower confidence.

6.2 Scalability (RQ2)

The scalability of the approach was tested by generating 5 synthetic logs with growing number of constraint instances for one specific MP-Declare constraint C (100, 500, 1 000, 2 000 and 5 000 constraint instances), with a default payload size of 10 attributes. In addition, we generated 6 synthetic logs with growing payload sizes (5, 10, 15, 20, 25 and 30 attributes), with a default number of constraint instances of 1 000 for C. Fig. 4 shows the execution times is in seconds needed for the rediscovery of C (averaged over 5 runs). The time required for the discovery from 5 000 feature vectors (with a payload of size 10) is of around 4 minutes and the time required for the discovery from 1 000 feature vectors (with a payload of size 30) is of around 32 seconds. Therefore, we can say that the execution times are reasonable when the discovered models are not extremely large.

[4] For generating the log we used the log generator based on MP-Declare available at https://github.com/darksoullock/MPDeclareLogGenerator.

6.3 Validation in Real Scenarios (RQ3)

For the validation of the proposed approach in real scenarios, we used six datasets provided for the BPI Challenges 2011 [2] and 2015 [3]. The first dataset is an event log pertaining to the treatment of patients diagnosed with cancer in a Dutch academic hospital. The remaining five datasets were provided for the BPI Challenge 2015 by five different Dutch Municipalities. The logs contain all building permit applications received over a period of four years. The full results obtained from these logs are not reported here for space limitations and can be downloaded from [5]. In the hospital log, the most important correlations link together medical exams and treatments. In the Municipality logs, there are several correlations between certain actors and the successful completion of a request handling.

7 Related Work

Two contributions are available in the literature that are close to the work presented in this paper. They have been presented in [12, 19]. The work in [12] is similar to the one presented in the current contribution but only consider activation conditions that discriminate between positive and negative cases without correlating activation and target conditions.

In [19], the authors present a mining approach that works with RelationalXES, a relational database architecture for storing event log data. The relational event data is queried with conventional SQL. Queries can be customized and cover the semantics of MP-Declare. Differently from [19], in the current contribution, we do not check the input log with user-specified queries, but we automatically identify correlations.

8 Conclusion and Future Work

We presented a technique for the automated discovery of multi-perspective business constraints. The technique allows us to discover conditions that relate the occurrence of pairs of events in the event log. The technique has been implemented as an open-source tool[6] and evaluated using both synthetic and real-life logs. The evaluation shows that the technique is able to rediscover behavior injected into a log and that it is sufficiently scalable to handle real-life datasets.

As future work, we aim to improve the efficiency of the approach. This can be done, for example, by using clustering techniques such as Canopy [18] that can be used when the clustering task is challenging either in terms of dataset size, or in terms of number of features, or in terms of number of clusters. Finally, the current approach works also with pure correlation conditions (involving attributes of activation and target together). However, these correlations should be specified manually as features that can possibly be discovered. Techniques based on Daikon [7] could help to automatically discover this type of conditions.

[5] https://bitbucket.org/volodymyrLeno/correlationminer/downloads/results.pdf
[6] Available at https://github.com/volodymyrLeno/CorrelationMinerForDeclare

References

1. *IEEE Task Force on Process Mining: XES Standard Definition*. 2013.
2. 4TU Data Center. BPI Challenge 2011 Event Log, 2011.
3. 4TU Data Center. BPI Challenge 2015 Event Log, 2015.
4. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94*, pages 487–499, 1994.
5. Andrea Burattin, Fabrizio Maria Maggi, and Alessandro Sperduti. Conformance checking based on multi-perspective declarative process models. *Expert Syst. Appl.*, 65:194–211, 2016.
6. Andrea Burattin, Fabrizio Maria Maggi, Wil M.P. van der Aalst, and Alessandro Sperduti. Techniques for a Posteriori Analysis of Declarative Processes. In *EDOC*, pages 41–50, Beijing, 2012.
7. Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Trans. Software Eng.*, 27(2):99–123, 2001.
8. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.
9. Wael H. Gomaa and Aly A. Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013.
10. Orna Kupferman and Moshe Y. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, 2003.
11. Fabrizio Maria Maggi. Declarative process mining with the Declare component of ProM. In *BPM (Demos)*, 2013.
12. Fabrizio Maria Maggi, Marlon Dumas, Luciano García-Bañuelos, and Marco Montali. Discovering data-aware declarative process models from event logs. In *BPM*, pages 81–96, 2013.
13. Marco Montali, Maja Pesic, Wil M. P. van der Aalst, Federico Chesani, Paola Mello, and Sergio Storari. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, 4(1), 2010.
14. Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. DECLARE: Full support for loosely-structured processes. In *EDOC*, pages 287–300, 2007.
15. Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo A. Reijers. Imperative versus declarative process modeling languages: An empirical investigation. In *BPM Workshops*, pages 383–394, 2011.
16. J. Ross Quinlan. *C4.5: Programs for Machine Learning*. M. Kaufmann Publishers Inc., 1993.
17. Hajo A. Reijers, Tijs Slaats, and Christian Stahl. Declarative modeling-An academic dream or the future for BPM? In *BPM*, pages 307–322, 2013.
18. Lior Rokach and Oded Maimon. *Clustering Methods*, pages 321–352. Springer US, 2005.
19. Stefan Schönig, Claudio Di Ciccio, Fabrizio Maria Maggi, and Jan Mendling. Discovery of multi-perspective declarative process models. In *ICSOC*, pages 87–103, 2016.
20. Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.
21. Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.
22. H. M. W. Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. XES, XESame, and ProM 6. In *Information Systems Evolution - CAiSE Forum 2010*, volume 72, pages 60–75.