A Framework to Improve the Accuracy of Process Simulation Models

Francesca Meneghello[1], Claudia Fracca[1], Massimiliano de Leoni[2], Fabio Asnicar[1], and Alessandro Turco[1]

[1] ESTECO SpA, Trieste, Italy
{meneghello, asnicar, turco}@esteco.com
[2] University of Padua, Italy
deleoni@math.unipd.it, claudia.fracca@phd.unipd.it

Abstract. Business process simulation is a methodology that enables analysts to run the process in different scenarios, compare the performances and consequently provide indications into how to improve a business process. Process simulation requires one to provide a simulation model, which should accurately reflect reality to ensure the reliability of the simulation findings. This paper proposes a framework to assess the extent to which a simulation model reflects reality and to pinpoint how to reduce the distance. The starting point is a business simulation model, along with a real event log that records actual executions of the business process being simulated and analyzed. In a nutshell, the idea is to simulate the process, thus obtaining a simulation log, which is subsequently compared with the real event log. A decision tree is built, using the vector of features that represent the behavioral characteristics of log traces. The tree aims to classify traces as belonging to the real and simulated event logs, and the discriminating features encode the difference between reality, represented in the real event log, and the simulation model, represented in the simulated event logs. These features provide actionable insights into how to repair simulation models to become closer to reality. The technique has been assessed on a real-life process for which the literature provides a real event log and a simulation model. The results of the evaluation show that our framework increases the accuracy of the given initial simulation model to better reflect reality.

Keywords: Business Process Simulation · BPMN Model · Decision Tree · Declarative Language · Event Log Comparison

1 Introduction

Business process simulation refers to techniques for the simulation of business process behavior on the basis of a process simulation model, a process model extended with additional information for a probabilistic characterization of the different run-time aspects (case arrival rate, task durations, routing probabilities, resource utilization, etc.). Simulation provides a flexible approach to analyse and improve business processes. Through simulation experiments, various 'what if' questions can be answered, and redesigning alternatives can be compared with respect to some key performance indicators. The main idea of business process simulation is to carry out a significantly large number

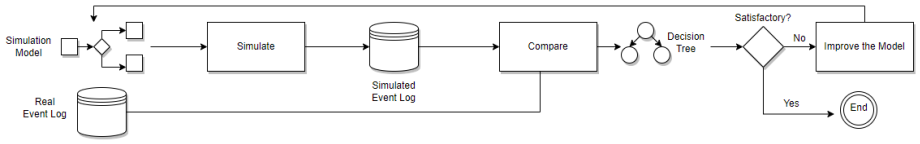


Fig. 1: The framework of the proposed approach.

of runs, in accordance with a simulation model. Statistics over these runs are collected to gain insight into the processes, and to determine the possible issues (bottlenecks, wastes, costs, etc.). By applying different changes to the simulation model, one can assess the consequences of these changes without putting them in production, and consequently can explore dimensions to possible process improvements.

A successful application of business process simulation for process improvement relies on a simulation model that reflects the real process behavior; conclusions are drawn on an unrealistic simulation model lead to process redesigns that may not yield improvements, or even may worsen the performances.

This paper puts forward a framework to assess and improve the accuracy (i.e., the realism) of a simulation model M starting from the differences figured out by our framework.

Along with M, the framework requires an event log $\mathcal{L}_r$ that records executions of the process modelled by M. The framework is based on the idea that many process runs are carried out using M, thus generating a simulated event log $\mathcal{L}_s$. If $\mathcal{L}_s$ is similar to $\mathcal{L}_r$, then M is accurate. To compute the similarity, the framework builds a decision tree that classifies the traces of $\mathcal{L}_r$ or $\mathcal{L}_s$. If the two logs are similar, the decision tree is unable to discriminate and, hence, correctly classify. The decision tree features encode the behavior observed in traces, such as whether two casually dependent activities follow in traces, activity durations, or certain declarative rules based on Linear Temporal Logic (LTL) formula.

Section 2 further introduces the framework, using an intuitive example. Section 3 introduces the basic concepts that are used throughout the paper. Section 4 reports on the discriminating features used to create the decision tree model from the two logs, and on a final data preprocessing step before training the model. Section 5 discusses on the use of decision trees for event logs discrimination. Section 6 illustrates the results of our evaluation, while 7 discusses related work. Section 8 summarizes the contributions and outlines future work directions.

2 Overall Idea and Motivation

The framework proposed in this paper can be summarized as in Figure 1. The starting point is an initial simulation model and a real event log. The simulation model can be drawn by process analysts on the basis of insights from stakeholders, or it can be discovered using combinations of Process Mining techniques [9, 11]. The simulation model is used to generate the traces composing a simulated log. After generating the simulating model, the framework aims to compare the two logs for differences; to do

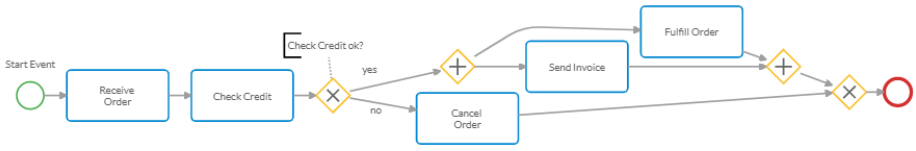
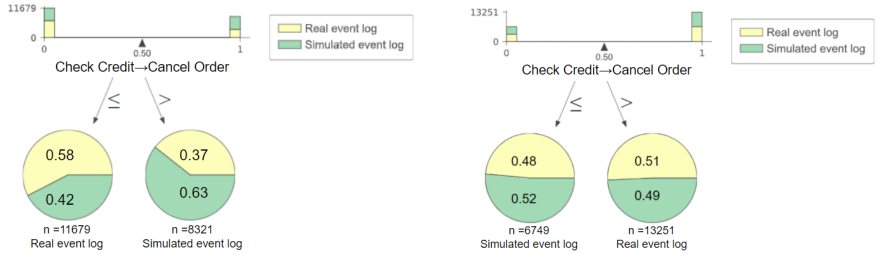


Fig. 2: BPMN model of process order.



(a) (b)

Fig. 3: Decision trees obtained from the comparison between the real log and the simulation log originated from the simulation model over the BPMN in Figure 2.

so, a decision tree is built that highlights the differences and indicates how close our simulation model represents reality. If the decision tree indicates that the simulation model accurately reflects reality, the application of the framework concludes, and the model is deemed as appropriate. If the decision tree highlights significant differences, the model is modified, using the differences as guidelines for improvement. The new simulation model can be used to generate a new simulation log and compared with the real log for the difference. It follows that the simulation model is improved iteratively through a sequence of improvement steps: the framework more and more improves the accuracy of the simulation model, until the differences are considered negligible.

As an example, let us consider the BPMN model in Figure 2, which refers to a purchase-order process composed of five activities. In particular, the exclusive gateway routes the control flow of the process based on the result of the previous activity *Check Credit*. If the *Check Credit* is successful, the order is carried; otherwise, the order is canceled. Let us suppose to have a real event log $\mathcal{L}_r$. The simulation model, composed by the BPMN in Figure 2 and the simulation parameters (e.g., case arrival rate, branching probabilities), is simulated as many times as the number of traces in $\mathcal{L}_r$ in order to generate a simulated event log $\mathcal{L}_s$. The framework builds a decision tree that is trained via a multiset of feature vectors, one vector for each trace in $\mathcal{L}_r$ and $\mathcal{L}_s$. Let us suppose to obtain the decision tree in Figure 3a. The root of the tree contains the feature `CheckCredit→CancelOrder`. For a given trace σ, this feature value is equal to the number of times that the activity *Check Credit* is followed by the activity *Cancel Order* and the opposite never happens in σ. Traces of $\mathcal{L}_r$ and $\mathcal{L}_s$ are then associated to decision tree leaves, depending on the values of the features that appear in the traces. Each leaf of the tree is represented as a pie chart, which describes the fractions of the

traces inside the leaf that belong to $\mathcal{L}_r$ and $\mathcal{L}_s$. The right leaf in Figure 3a contains the traces with the root feature equal to 1, i.e., the traces containing the activity *Check Credit* followed by *Cancel Order*. This leaf contains the traces for orders that are canceled, while the left for completed orders. The pie chart of the right leaf contains more traces from $\mathcal{L}_s$ log with respect to $\mathcal{L}_r$ log, which indicates that the simulation model sets a too high probability to cancel more orders than the real process. Starting from this rule, we can improve and fix the simulation model in order to have the same percentage of completed orders. In this case, the mistake is very likely associated with the branching probability of the exclusive gateway *Check Credit*. We can hence improve the model by better tuning the probability at the gateway. The new simulation model is run to obtain a simulated log to repeat the comparison based on the decision tree construction. The new decision tree is, e.g., as in Figure 3b: now the leaves almost contain the same numbers of traces from $\mathcal{L}_r$ and $\mathcal{L}_s$. It follows that the decision tree model is not able to well discriminate whether a trace belongs to the real or simulated event log, thus positively confirming the accuracy of the simulated model.

3 Preliminary

This section introduces the preliminary concepts to later illustrate the technique's details. First, we present the concepts of events, traces and event logs, and some related notations.

Definition 1 (Events). *Let $\mathcal{A}$ be a set of activity labels. Let $\mathcal{T}$ be the universe of timestamps. Let $\mathcal{I} = \{\texttt{start}, \texttt{complete}\}$ be the life-cycle information. An event $e \in \mathcal{A} \times \mathcal{T} \times \mathcal{I}$ is a tuple consisting of an activity label, a timestamp of occurrence, and the life-cycle information.*

In the remainder, given an event $e = (a, t, i)$, $act(e) = a$ returns the activity label, $time(e) = t$ returns the timestamp, and $life(e) = i$ is the information whether e refers to the starting or completion of an activity. In practice, several event logs are composed of events where the life-cycle information is not present. In this case, we assume that those events refer to the completion. There might be events other than related to the starting or completion of activities: those events are simply ignored. Events also carry a payload consisting of attributes taking on values: they are also ignored.

Definition 2 (Traces and Event Logs). *Let $\mathcal{E}_\mathcal{A}$ the universe of the events defined over a set $\mathcal{A}$ of activity labels. A trace $\sigma = \langle e_1, \ldots, e_m \rangle \in \mathcal{E}_\mathcal{A}^*$ is a sequence of events, with the constraint that, for all $0 < i < j \leq m$, $time(e_i) \leq time(e_j)$. An event log $\mathcal{L}_\mathcal{A}$ is a set of traces, namely $\mathcal{L}_\mathcal{A} \subset \mathcal{E}_\mathcal{A}^*$.*

In the remainder, we use the shortcut $e \in \mathcal{L}_\mathcal{A}$ to indicate that there is a trace $\sigma \in \mathcal{L}_\mathcal{A}$ such that $e \in \sigma$. Also, we drop the subscript $\mathcal{A}$ when it is clear from the context. Finally, given a trace σ, the notation $complete(\sigma)$ refers to the sequence of events in σ after removing the events referring to the starting of activities, and retaining the same order, i.e. $complete(\sigma) = \{e \in \sigma |\ life(e) = \texttt{complete}\}$. Table 1 shows an example of an event log related to the management of order requests.

Case ID	Activity	Timestamp	Life-cycle
12	Receive Order	16-08-20 08:30	start
12	Receive Order	16-08-20 08:45	complete
13	Receive Order	16-08-20 10:30	start
13	Receive Order	16-08-20 10:45	complete
12	Check Credit	17-08-20 12:27	start
12	Check Credit	17-08-20 12:32	complete
12	Fulfill Order	17-08-20 14:40	start
12	Fulfill Order	17-08-20 14:50	complete
13	Check Credit	17-08-20 16:40	start
13	Check Credit	17-08-20 17:40	complete
13	Cancel Order	17-08-20 17:56	start
14	Receive Order	17-08-20 18:00	start
14	Receive Order	17-08-20 18:45	complete
13	Cancel Order	17-08-20 18:56	complete
12	Send Invoice	18-08-20 10:40	start
12	Send Invoice	18-08-20 10:42	complete
14	Check Credit	18-08-20 13:11	start
14	Check Credit	18-08-20 13:32	complete
14	Fulfill Order	18-08-20 14:25	start
14	Send Invoice	18-08-20 14:27	complete
14	Send Invoice	18-08-20 14:40	complete
14	Fulfill Order	20-08-20 10:50	complete

Table 1: A fragment of an event log of a process about dealing with orders.

Constraints	Description
$Init(a)$	a should be the first activity in a trace
$End(a)$	a should be the last activity in a trace
$CoExistence(a, b)$	If one of the activities a or b is executed, the other one also has to be executed
$Response(a, b)$	When a is executed, b has to be executed after a
$AlternateResponse(a, b)$	When a is executed, b has to be executed after a and no other a can be executed in between
$Precedence(a, b)$	b has to be preceded by a
$AlternatePrecedence(a, b)$	b has to be preceded by a and another b cannot be executed between a and b
$Succession(a, b)$	a occurs if and only if it is followed by b

Table 2: List of DECLARE constraints used in our techniques.

Some of the differences can be given as constraints of DECLARE, i.e., a declarative process modeling language [1]. This language indeed defines the behavior of the business process as a set of constraints. An example of DECLARE constraint is $Init(a)$ and it states that every instance must start with the execution of activity a. Another example can be $Precedence(a, b)$, and it imposes that the activity b occurs only if preceded by the activity a. The full list of DECLARE constraints can be found in [1]. The list of constraints and the related descriptions that are used in this paper are listed in Table 2. Given this framework, we can notice that each DECLARE constraint can be formally defined as a LTL formula, which can ultimately be represented as a final state automaton over the alphabet of activities.

Definition 3 (DECLARE Constraint as Final State Automaton). *Let $\mathcal{A}$ be a set of activity labels. The* DECLARE *constraint can be formulated as a final state automaton $\mathcal{C} = (\mathcal{A}, Q, q_0, \delta, E)$ over a set $\mathcal{A}$ of activities, where: (i) $\mathcal{A}$ is the activity labels; (ii) Q is a finite, non-empty set of states; (iii) $q_0 \in Q$ is an initial state; (iv) $\delta \in Q \times \mathcal{A} \rightarrow Q$ is the state-transition function; (v) $E \subseteq Q$ is the set of final states.*

The automaton that represents a DECLARE constraint accepts all and only those log traces that satisfy the constraint. Note how the state-transition function is total: log traces can always be replied on the automaton.

Example 1. Given the set of activity labels related to the event log in the Table 1. Figure 4 shows the automaton $\mathcal{C} = (\mathcal{A}, Q, q_0, \delta, E)$ for the DECLARE constraint *CoExistence (Receive Order, Fulfill Order)*. This DECLARE constraint states that if one of the two

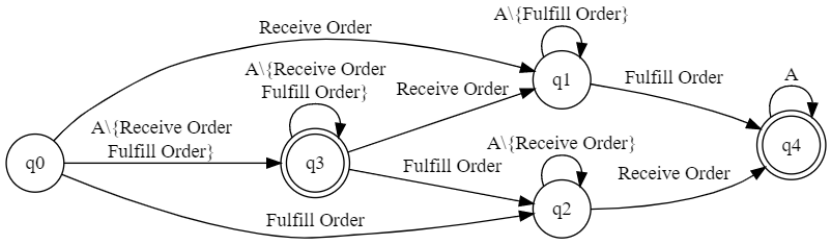


Fig. 4: This figure shows the automaton for the DECLARE constraint *CoExistence(Receive Order, Fulfill Order)*, where $\mathcal{A}$ is the who set of activity labels.

activities, *Receive Order* or *Fulfill Order*, is executed, then the other also has to be executed. As an example, the trace with the CASE ID $= 12$ in the Table 1 is accepted by the automaton, while the trace with CASE ID $= 13$ is not accepted.

4 Our Framework

Our framework requires a simulation model and a real event log $\mathcal{L}_r$ as input. The basic idea is to verify the quality of the simulation model by generating an event log $\mathcal{L}_s$ from the latter and find the event log features that discriminate $\mathcal{L}_r$ and $\mathcal{L}_s$. These features can range from the occurrences of activities and their sequencing to DECLARE constraints, and temporal features i.e. the activity durations. The discriminating features pinpoint the differences between the two event logs and provide valuable insights into repairing the simulation models.

In this structure, we employ decision tree learning to find the discriminating features. Given a set of features $\mathcal{F}$, and their potential values $\mathcal{V}$. Decision trees are learned from a multiset of pairs (x, y) where $x \in \mathcal{F} \to \mathcal{V}$ is a function that assigns values to (a subset of) the features, and y is the class value. The label y is whether the trace belongs to the real or simulated event log. In our framework, the feature function is extracted from an event log via a customizable mapping function:

Definition 4 (Trace-to-Feature Mapping Function). *Let $\mathcal{L}$ be an event log. Let $\mathcal{F}$ be a set of features. Let $\mathcal{V}$ be the set of potential values. A trace-to-features mapping function is a function $\rho^{\mathcal{L}} : \mathcal{L} \to (\mathcal{F} \to \mathcal{V})$ such that, for each trace $\sigma \in \mathcal{L}$, it returns a feature-to-value function $z = \rho^{\mathcal{L}}(\sigma)$ that assigns a value $z(f) \in \mathcal{V}$ to each feature $f \in \mathcal{F}$.*

With these concepts at hand, given a real event log $\mathcal{L}_r$ and a simulated event log $\mathcal{L}_s$ the training set $T_{\mathcal{L}_r,\mathcal{L}_s}$ of the decision tree is constructed as follows where `real` and `sim` denote the two possible class values for respectively real and simulated log trace:[3]

$$T_{\mathcal{L}_r,\mathcal{L}_s} = \biguplus_{\sigma_r \in \mathcal{L}_r} (\rho^{\mathcal{L}}(\sigma_r), \texttt{real}) \uplus \biguplus_{\sigma_s \in \mathcal{L}_s} (\rho^{\mathcal{L}}(\sigma_s), \texttt{sim}) \tag{1}$$

[3] Symbol $\uplus$ indicates the union of multisets where duplicates are retained.

In the remainder, from Sections 4.1 to 4.3, we introduce several trace-to-features mapping functions supported in our implementation to construct the training sets for the decision tree algorithm, i.e., *Basic Features, Extended Features for Declare Rules and Temporal Features*.

4.1 Basic Features

In this section, we describe the features related to the control-flow perspective. First of all, we define the features related to the activities occurrences, i.e., if an activity is performed in the given trace or not.

Definition 5 (Activity Function). *Let $\mathcal{L}$ be an event log over the set $\mathcal{A}$ of activities. Let define a trace-to-function mapping function $\rho^{\mathcal{L}}_{activity} : \mathcal{L} \rightarrow (\mathcal{F} \rightarrow \mathcal{V})$ in which the set of features $\mathcal{F}$ is $\mathcal{A}$. For each trace $\sigma \in \mathcal{L}$, it returns a function $z = \rho^{\mathcal{L}}_{activity}(\sigma)$ that assigns a value $z(f) \in \mathcal{V}$ for each activity feature $f \in \mathcal{F}$. The value $z(f)$ is the number of times an activity a is executed within the trace, i.e. $|e \in complete(\sigma) : act(e) = a|$.*

Example 2. Let σ the trace with the CASE ID $= 12$ in the Table 1. Let fix the set of features $\mathcal{F}$ equal to the set $\mathcal{A}$, i.e. $\mathcal{F} = \{$`ReceiveOrder, CheckCredit, FulfillOrder, SendInvoice, CancelOrder`$\}$. Given the trace σ, the function $\rho^{\mathcal{L}}_{activity}$ maps the trace into a function z that assigns for each $f \in \mathcal{F}$ a value in $\{0, \ldots, n\}$. The trace contains the activities: *Receive Order*, *Check Credit*, *Fulfill Order*, and *Send Invoice*. Therefore we have that *z(*`ReceiveOrder`*)=1, z(*`CheckCredit`*)= 1, z(*`FulfillOrder`*)=1, z(*`SendInvoice`*)=1, and z(*`CancelOrder`*)=0.*

In the following, we define the second type of features related to the control-flow perspective, i.e., these features encoded the causality relation between activities:[4]

Definition 6 (Causality Relation). *Given an event log $\mathcal{L}$ defined over a set $\mathcal{A}$ of activities. $a \rightarrow_{\mathcal{L}} b$ is a causality relation in $\mathcal{L}$ iff there is a trace $\sigma = \langle e_1, \ldots, e_m \rangle \in \mathcal{L}$ s.t $\langle e_i, e_{i+1} \rangle \subseteq complete(\sigma) \mid act(e_i) = a \wedge act(e_{i+1}) = b$ and $\nexists \sigma' = \langle e'_1, \ldots, e'_m \rangle \in \mathcal{L}$ s.t $\langle e'_i, e'_{i+1} \rangle \subseteq complete(\sigma') \mid act(e'_i) = b \wedge act(e'_{i+1}) = a$.*

Definition 7 (Causality Relation Function). *Let $\mathcal{L}$ be an event log defined over a set $\mathcal{A}$ of activities. We introduce a trace-to-feature mapping function $\rho^{\mathcal{L}}_{\rightarrow} : \mathcal{L} \rightarrow (\mathcal{F} \rightarrow \mathcal{V})$ in which the set of features $\mathcal{F}$ coincides with the set of causality relation in $\mathcal{L}$. The causality relation function $\rho^{\mathcal{L}}_{\rightarrow}(\mathcal{L})$ returns a function $z(f)$ that, to each causality relation $f = (a \rightarrow_{\mathcal{L}} b) \in \mathcal{F}$, assigns the numbers of times an event for activity a is followed by an event for activity b in $\mathcal{L}$.*

Example 3. Let σ the trace with the CASE ID $= 12$ in the Table 1. Let fix the set of features $\mathcal{F}$ equal to the set of all possible causality relation in $\mathcal{L}$, i.e. $\mathcal{F} = \{$ `ReceiveOrder`$\rightarrow$`CheckCredit,...,CheckCredit`$\rightarrow$`CancelOrder`$\}$. In this case we have for instance that *z(*`CheckCredit`$\rightarrow$`FulfillOrder`*)* $= 1$, and *z(*`CheckCredit`$\rightarrow$ `CancelOrder`*)* $= 0$.

[4] Given two sequences s and s', $s' \subseteq s$ indicates that s' is a sub-sequence of s.

4.2 Extended Features for Declare Rules

We also want to support more complex features related to the control-flow perspective using the DECLARE constraints. Given a real event log $\mathcal{L}_r$ and the simulated event log $\mathcal{L}_s$ as in our framework, we compute the sets of DECLARE constraints over these two event logs via MinerFul Miner [8]. We denote these constraint sets are D_r and D_s respectively. As mentioned in Section 3, we only support the constraints in Table 2, excluding the constraints already covered or extended by the *Basic Features* discussed in Section 4.1. For instance, the *Activity Features* already cover the constraint *Participation(a)* and *AtMostOne(a)*. The first constraint requires that the activity a occurs at least once and the other that the activity a occurs at most once. However, the *Activity Feature* related to the number of occurrences of the activity a is certainly more detailed. Also, we decide to remove the negative DECLARE constraints because they might be overly complex and not give useful insight. After removing these constraints, we perform the symmetrical difference between the two sets of constraints, i.e., $(D_r \cup D_s) \setminus (D_r \cap D_s) = D_t$, obtaining the final set of constraints to consider. In fact, the constraints in common cannot pinpoint differences. Note also that for each $d_j \in D_t$ we have the corresponding constraint automaton $\mathcal{C}_j = (\mathcal{A}^j, Q^j, q_0^j, \delta^j, E^j)$.

Definition 8 (DECLARE Function). *Let $\mathcal{L}$ be an event log over a set $\mathcal{A}$ of activities. Let define a trace-to-function mapping function $\rho^{\mathcal{L}}_{declare} : \mathcal{L} \to (\mathcal{F} \to \mathcal{V})$ in which the set of feature $\mathcal{F}$ is the set of* DECLARE *constraints D_t. For each trace $\sigma \in \mathcal{L}$, it returns a function $z = \rho^{\mathcal{L}}_{declare}(\sigma)$ that assigns a value $z(f) \in \mathcal{V}$ to each* DECLARE *constraint $f \in \mathcal{F}$. The value $z(f) \in \{\texttt{True}, \texttt{False}\}$ corresponds to the evaluation of the trace $complete(\sigma)$ on the respective automaton, i.e. set to `True` if the automaton of the* DECLARE *constraint accepts the trace, and `False` otherwise.*

Example 4. Let σ the trace with the CASE ID $= 12$ in the Table 1. Let fix the set of features $\mathcal{F}$ equal to the set of the DECLARE constraints D_t. For instance, let take the DECLARE constraint *CoExistence(Receive Order, Fulfill Order)* represented in the automaton in the Figure 4. In this case we have that *z(`CoExistence(ReceiveOrder, FulfillOrder)`* is equal to `True`.

4.3 Temporal Features

In this section, we describe the features related to the time perspective. For each $a \in \mathcal{A}$, we define the activity duration feature p_a that represents the duration of the activity a in the trace. Let define with P the set of all the activity duration features related to $\mathcal{A}$, i.e., $P = \bigcup_{a \in \mathcal{A}} p_a$.

Definition 9 (Activity Duration Function). *Let σ a trace in $\mathcal{L}$. Let define a trace-to-function mapping function $\rho^{\mathcal{L}}_{duration} : \mathcal{L} \to (\mathcal{F} \to \mathcal{V})$ in which the set of features $\mathcal{F}$ is equal to the set of activity duration P. For each trace $\sigma \in \mathcal{L}$, it returns a function $z := \rho^{\mathcal{L}}_{duration}(\sigma)$ that assigns a value $z(f) \in \mathcal{V}$ for each feature $f \in \mathcal{F}$. The value $z(f) \in [-1, \infty)$ corresponds to the activity duration.*

Example 5. Let σ the trace with the CASE ID $= 12$ in the Table 1. Let fix the set of features $\mathcal{F}$ equal to the set P, i.e. $\mathcal{F} = \{$`time:ReceiveOrder,time:CheckCredi`

time:Fulfill Order	...	Check Credit→Cancel Order	...	CoExistence(Fulfill Order, Send Invoice)	...	Class Label
10 min	...	0	...	true	...	real
-1	...	1	...	false	...	sim
25 min	...	0	...	true	...	real

Table 3: The corresponding training set of the fragment event log in Table 1.

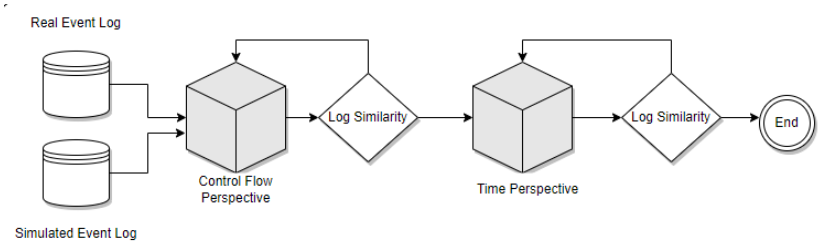


Fig. 5: The framework of the proposed iterative approach. The inputs are the real and the simulated event logs. We have two phases, related to the control-flow and the time perspectives respectively. The blacks-box refer to the framework in Figure 1.

t,time:FulfillOrder,time:SendInvoice,time:CancelledOrder}. Therefore, for example, we have that $z(\texttt{time:ReceiveOrder}) = 15\ minutes$, and that $z(\texttt{time:CancelledOrder}) = -1$ due to the absence of the activity *Cancel Order* in the trace.

Using these trace-to-features mapping functions, we can construct the multiset $T_{\mathcal{L}_r,\mathcal{L}_s}$ as in Equation 1. Table 3 represents an example of the resulted multiset for the traces contained in the Table 1.

4.4 Application of the Framework using Different Features

As mentioned in Section 2, our framework improves the simulation model by repeatedly comparing the same real event log with different simulated event logs obtained via more and more accurate simulation models. Since the control-flow of the simulation models (the activities, events, gateways, etc.) is tightly correlated to the temporal features (e.g., the time elapsed between the execution of two non-consecutive activities), we separately employ the framework using control-flow and temporal features.

The framework is firstly applied to consider the *Basic* and *Extended features for Declare rules* (cf. Sections 4.1 and 4.2). Subsequent iterations are carried out to improve the simulation model, until a decision tree is constructed that shows an accurate model (namely the tree is unable to distinguish the traces of the simulation log from those of the real event log), as in Figure 5. Then, we reapply the framework, focusing on the temporal features discussed in Section 4.3. Section 6 discusses a case-study assessment where we indeed employ the framework where we first focus on the control flow of the simulation model, and then on the temporal features.

4.5 Feature Selection and Normalization

The multiset $T_{\mathcal{L}_r,\mathcal{L}_s}$ obtained from the Equation 1 encoding basic, extended and/or temporal features is used to train a decision tree model. As last step beforehand, we preprocess $T_{\mathcal{L}_r,\mathcal{L}_s}$ to improve the quality of the trained decision tree. Initially, we perform feature selection and remove those features do not show sufficient variability. Features over discrete and numerical domains (e.g., the number of occurrence of activities in a trace) are filtered out if more than $X\%$ of the set elements take on the same value, where X is customizable and typically is around 90%. Any feature defined over continuous domains (e.g., temporal features) is removed if the mean and standard deviation considered for the multiset elements related to the real event-log traces are distant less than $(1 - X)\%$ from the mean and standard deviations for the elements related to the simulated event-log traces, with X customizable. Then, we normalize the values of the features of the elements of the multiset, using a traditional z-score normalization[2]: the values are transformed into distribution with a mean of 0 and a standard deviation of 1, i.e., a common scale. For each value, we subtract the mean value and divide it by the standard deviation of the respective feature. Normalization may be useful when learning a decision tree because the learning algorithms tend to give more weight and importance to features characterized by larger values.

5 Discussion

Our technique aims at a business simulation model that is able to generate traces that exhibit all and only the behavioral characteristics of the traces of the real event log. The behavioral characteristics of interest can be customized, such as with those in Section 4. Note that, however, the real event log does not need to contain every potential trace, but it is enough to just contain traces that exhibit these behavioral characteristics. It follows that the simulation model does not need to generalize beyond the real event log. This explains why we use the entire feature vector multiset for training: the tree needs to just classify the traces in the real and simulated event log, and can ignore potential future traces that exhibit characteristics not observed in the real and simulated event log. We however acknowledge the importance to have simulated models that generalize beyond the real event log, and we aim to investigate the generalization question as future work.

Training on the entire dataset might lead to an overfitting decision-tree model, namely with an excessive number of nodes and with leaves associated to single traces. We implemented decision-tree pruning by limiting the maximum depth of the tree and the maximum number of leaves, in order to mitigate overfitting. Pruning also improves the decision-tree clarity and readability.

Recall that we aim at a decision tree that cannot distinguish traces of the two event logs. Considering the real and simulated event log have the same number of traces, the most favourable tree is such that each leaf is associated to the same number of real and simulated log traces. This leads to a metrics for log similarity that considers the weighted average of the distribution of the classes in the leaves. The metric can take on values between 0 and 1. Value 0 means that the decision tree is able to distinguish each trace of the real-life event log from each trace of the simulated one, i.e., the simulation

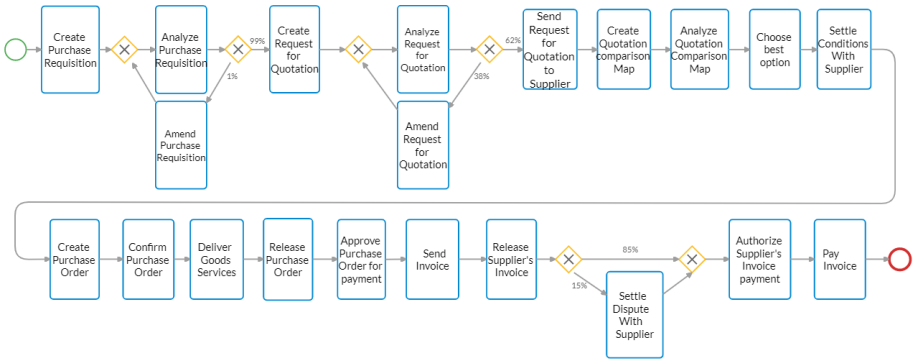


Fig. 6: The initial BPMN model for the simulation of an order purchase process [5], used in our case study.

model does not reflect the real process behavior. In the opposite case, Value 1 means that the two event logs are indistinguishable.

Definition 10 (Log Similarity). *Let $B = \{\beta_1, \ldots, \beta_n\}$ the set of all leaves in the decision tree model $\mathcal{DT}$. Let be $\eta_r \in T_{\mathcal{L}_r,\mathcal{L}_s}$ instances of multiset with the class label equal to* `real` *and $\eta_s \in T_{\mathcal{L}_r,\mathcal{L}_s}$ instances with* `sim` *as class label. For each $\beta_i \in B$, m_{β_i} is the total number of instances in β_i, of which η_{r_i} and η_{s_i} are the percentage of instances with* `real` *and* `sim` *label, respectively. The similarity of $\mathcal{L}_r, \mathcal{L}_s$ with respect the decision tree $\mathcal{DT}$ is the follow:*

$$Log\ Similarity(\mathcal{DT}^{\mathcal{L}_r,\mathcal{L}_s}) = \frac{(1-|\eta_{r_1}-\eta_{s_1}|)\cdot m_{\beta_1}+\ldots+(1-|\eta_{r_n}-\eta_{s_n}|)\cdot m_{\beta_n}}{m_{\beta_1}+\cdots+m_{\beta_n}}$$

Example 6. The log similarity related to the decision tree model in Figure 3a is:

$$Log\ Similarity(\mathcal{DT}^{\mathcal{L}_r,\mathcal{L}_s}) = \frac{(1-|0.58-0.42|)\cdot 11679+(1-|0.37-0.63|)\cdot 8321}{11679+8321} = 0.80$$

Note how the above-defined metrics is tightly coupled with the typical decision-tree accuracy metrics: the higher is the log-similarity metrics, the lower is the accuracy. Indeed, less accuracy means that the decision tree has lower capabilities to classify traces. This implies that simulated and real logs are similar, and consequently the simulation model is capable to generate all and only the traces in the real event log.

6 Implementation and Experiments

Our approach has been implemented as a Python command-line tool.[5] Python language provides the main libraries and frameworks for process mining and machine learning. In particular, the libraries scikit-learn and PM4py are used to implement our approach.[6]

[5] https://github.com/francescameneghello/A-Framework-to-Improve-the-Accuracy-of-Process-Simulation-Models.git

[6] scikit-learn: https://scikit-learn.org/stable/, PM4py: https://pm4py.fit.fraunhofer.de/

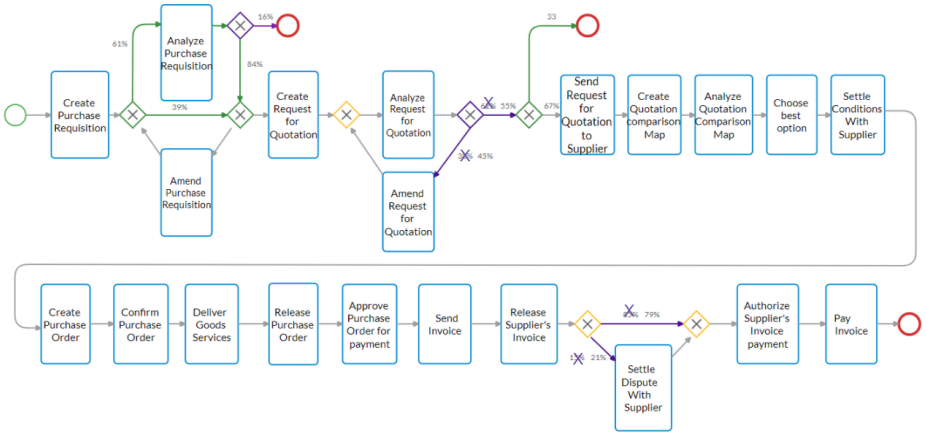
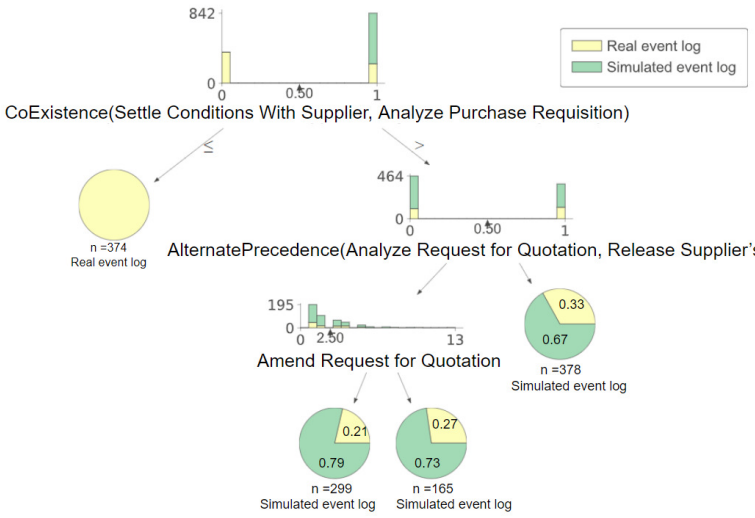


Fig. 7: The initial BPMN model for the simulation. The elements colored with green represent the changes derived from the first iteration, while those colored with purple are derived from the second iteration. The last, third iteration produced no change.

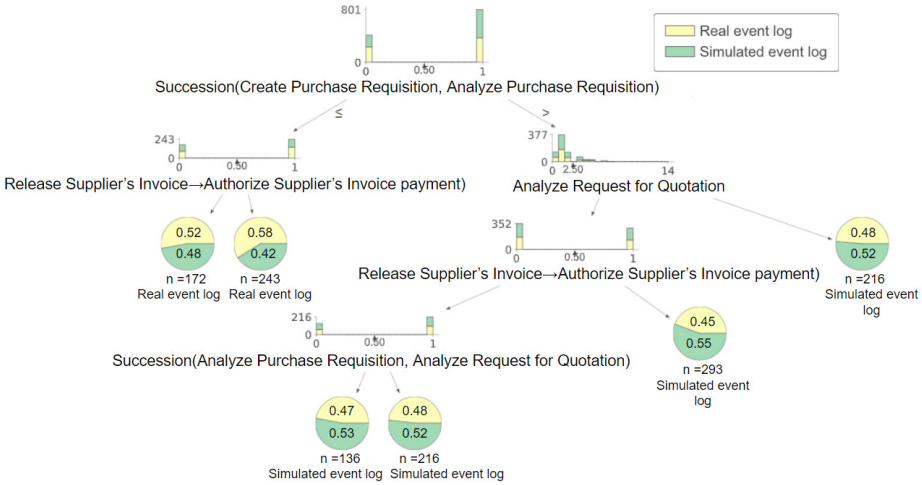
As input, the tool takes the real and the simulated event logs in XES format, together with the respective declare-constraints set in CSV format. These sets were discovered via MinerFul Miner [8]. The output of our implementation is a decision tree model such as those in Figure 8.In order to test the applicability of our approach, we conducted a case study related to an order purchase process. We used a real event log and an existing accordant simulation model coming from literature by Camargo et al. [5].[7] The latter is composed of the BPMN model in Figure 6 and several simulation parameters. Hence, we run this simulation model to obtain the simulated event log. In the remainder, we iteratively apply our technique to further improve the existing simulation model.

First Step: Control Flow Perspective. The first step of the improving iterative process is the analysis of the control-flow perspective using the *Basic Features* and the *Extended Features for Declare Rules*. Figure 8a shows the first comparison result a Log Similarity equal to 0.38, which illustrates the need to improve the simulation model. We leveraged on the rules discovered by the decision tree model in Figure 8a. The CoExistence constraint in the root highlights that for 374 traces in the real log, the activities *Settle Conditions With Supplier* and *Analyze Purchase Requisition* do not coexist while the model requires both activities to occur. In fact, by analyzing the real log, we observed that the *Analyze Purchase Requisition* activity is not always executed. Hence, an exclusive gateway is introduced before this activity to make it optional (see Figure 7). When both activities are performed, the decision tree pinpoints via the node of the AlternatePrecedence constraint that the activity *Release Supplier's Invoice* is more often preceded by *Analyze Request For Quotation* in the simulated log, compared with the real event log. This is actually caused by having more traces in the real event log where *Analyze Re-*

[7] The event log is available at http://fluxicon.com/academic/material/ while the accordant simulation model is available at https://github.com/AdaptiveBProcess/Simod



(a) The decision tree obtained at the first iteration (Log Similarity 0.38).



(b) The final decision tree obtained at the third and last iteration (Log Similarity 0.91).

Fig. 8: Decision trees generated at the first and last iteration of the framework, using the *Basic Features* and the DECLARE features.

quest For Quotation is not followed by the other activity. This might trigger at a first glance to make *Release Supplier's Invoice* optional via an exclusive gateway. We tried to do so, and we saw that an alternate-precedence constraint remained in the decision tree between *Analyze Request For Quotation* and any of the other activities following it (e.g., *Send Invoice*). We thus concluded that the real process allows for termination after *Analyze Request For Quotation*: therefore, we added an exclusive gateway before

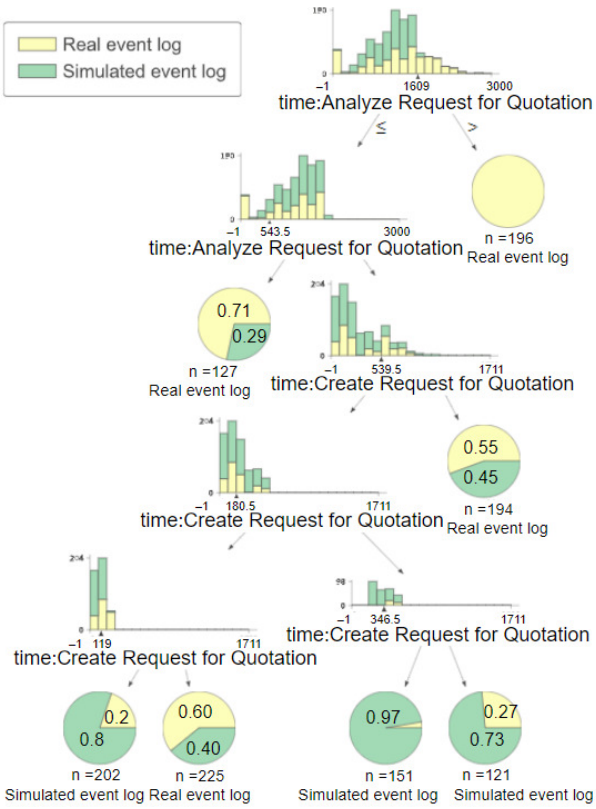


Fig. 9: Decision trees are generated at the first iteration of the second step of our iterative framework, using the *Activity Duration Features.*

Send Request for Quotation to Supplier to the model, along with a BPMN's end event (see Figure 7).

We iterated once more the procedure. Space limitation prevents us from showing the new decision tree and discussing it. The new iteration led to some changes in the routing probabilities, as shown with purple in Figure 7. This second iteration was followed by a third associated with the decision tree in Figure 8b: even if the decision tree contains a few branches, one can see that the distribution of traces at leaves is well balanced. Indeed, the Log Similarity is 0.91. So, we proceeded with the second phase regarding the analysis of the time perspective through the *Activity Duration Features*.

Second Step: Time Perspective. The first iteration of time analysis produced the decision tree in Figure 9 with Log Similarity equal to 0.42, so we tried to improve the simulation model. The root's feature `time:AnalyzeRequestForQuotation` evidences that only in the real log the activity *Analyze Request For Quotation* takes more than 26 minutes to complete. The remaining nodes pinpoint several deviations also about the processing time of the activity *Create Request for Quotation*. Moreover, each

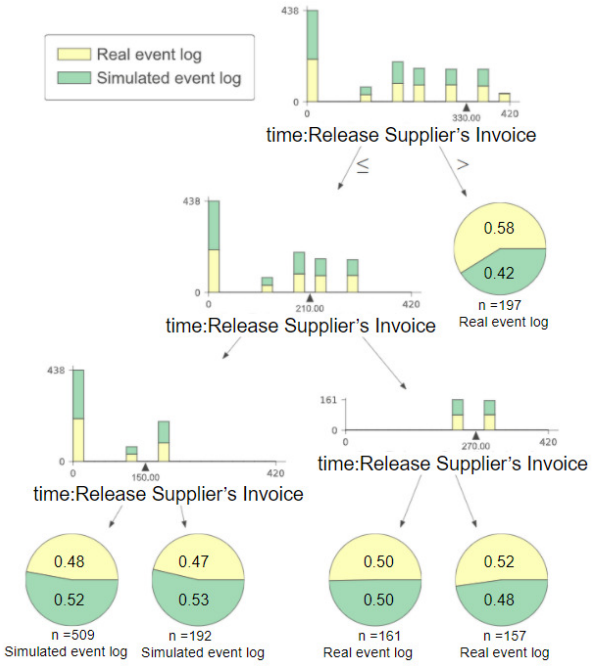


Fig. 10: Decision trees are generated at the last iteration of the second step of our iterative framework, using the *Activity Duration Features.*

node presents a histogram that underlines the variations in the distribution of processing times for both logs. For the first activity, we changed the distribution of activity duration from a uniform distribution to a custom discrete distribution, that assigns a sample of points to the corresponding probability. For the second one, we retained the same normal distribution and we only adjusted the mean and standard deviation. After two iterations, we obtained the decision tree in Figure 10 with Log Similarity equal to 0.94.

7 Related Works

Our technique is centered around comparing a real and a simulated event log to pinpoint common, behavioural differences. Several approaches exist for the comparison of two event logs from the same process [3, 4, 6, 10, 12]. Bolt et al. propose an approach where pairs of event logs are shown as automata and statistically-significant differences are highlighted through different colors [4]. Since transitions systems explicitly represent all the interleavings of execution of activities, differences are usually captured at low level. Low-level differences are also returned by the technique Nguyen et al. [10], where a differential graph of the differences between two event logs is constructed. As also highlighted in the evaluation presented in Taymouri's work [12], the two aforementioned research works yield an explosion of differences, which provide few actionable

insights into how to improve a simulation model. In contrast, a decision tree model is able to detect the main differences between the logs, i.e., differences affecting a significant number of traces, and to return them in form of compact and high-level behavioral rules, which help process analysts to improve a simulation model. Beest et al. [3] present a method for diagnosing the differences between two event logs via natural language statements capturing behavior present in one log but not in other, but they only consider differences related to the ordering of activities and to branching probabilities. Taymouri et al. [12] proposed a hybrid machine learning approach for process variant analysis based on Discrete Wavelet Transformation (DWT). This approach uses a Support Vector Machine (SVM) to extract the features that provide enough information to discriminate the two process variations, and they are plotted using directly-follows graphs. The above-mentioned research stop at considering the *Basic Features* as introduced in Section 4.1, thereby ignoring DECLARE features, which allow one to capture and compare more complex behavioral patterns. Cecconi et al. [6] is the only work that consider DECLARE features when comparing two event logs. However, it does not consider the *Temporal Features*, nor does it attempt to filter out non-discriminating DECLARE rules. The latter also implies that a DECLARE miner may potentially mine redundant and inconsistent rules, which are subsequently returned (see [7]).

8 Conclusion

A successful application of process simulation to analyze and improve a process passes through a realistic simulation model, namely which accurately represents the potential real process executions. This enables analysts to improve the real process and not the supposed one. This paper has proposed a framework to assess and improve the accuracy of a simulated model to reflect the real behavior. The input is the simulation model and an event log that records real process executions. The simulation model is run to generate simulated event logs that are compared with the real log for differences. Differences are shown in form of a decision tree that classifies traces from the real event log and those from the simulated log. The tree provides actionable insights into how to modify the model to be more accurate. By repeatedly comparing with more and more accurate models and by using different behavioral dimensions of comparison, the framework aims to obtain an accurate simulation model which analysts can rely on.

The framework has been applied on a purchasing process, for which a simulation model and event log were available from literature. Our framework was able to further improve the accuracy of the simulation model, thus illustrating the benefits of the framework proposed.

There are multiple directions of future work. First, we want to investigate the question of simulation-model generalizability (cf. Section 5). Second, we aim to extend the set of decision tree features available in our operationalization of the framework, which now refers to control-flow and time: we also want to explicitly consider the resource, cost, and data perspectives. Second, we want to investigate how statistical approaches determine the number of traces in the event log: we presently simulate as many traces as the real event log, but it is possible to a small number would be statistically sufficient. Fourth, we want to extend the framework to provide concrete recommendations

on how to modify the simulation model: the current framework focuses on providing insights, but the effort of transforming those insights into actual modifications is left to the process analysts.

References

1. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM (2006)
2. Al Shalabi, L., Shaaban, Z., Kasasbeh, B.: Data mining: A preprocessing engine. Journal of Computer Science **2** (2006)
3. van Beest, N.R., Dumas, M., García-Bañuelos, L., Rosa, M.L.: Log delta analysis: Interpretable differencing of business process event logs. In: International Conference on Business Process Management. pp. 386–405. Springer (2016)
4. Bolt, A., de Leoni, M., van der Aalst, W.M.: Process variant comparison: Using event logs to detect differences in behavior and business rules. Information Systems **74** (2018)
5. Camargo, M., Dumas, M., González-Rojas, O.: Automated discovery of business process simulation models from event logs. Decision Support Systems **134** (2020)
6. Cecconi, A., Augusto, A., Di Ciccio, C.: Detection of statistically significant differences between process variants through declarative rules. In: International Conference on Business Process Management. pp. 73–91. Springer (2021)
7. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Information Systems **64** (2017)
8. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Transactions on Management Information Systems (TMIS) **5** (2015)
9. Martin, N., Depaire, B., Caris, A.: The use of process mining in a business process simulation context: Overview and challenges. In: 2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM). pp. 381–388. IEEE (2014)
10. Nguyen, H., Dumas, M., Rosa, M.L., ter Hofstede, A.H.: Multi-perspective comparison of business process variants based on event logs. In: International Conference on Conceptual Modeling. pp. 449–459. Springer (2018)
11. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.: Discovering simulation models. Information systems **34** (2009)
12. Taymouri, F., Rosa, M.L., Carmona, J.: Business process variant analysis based on mutual fingerprints of event logs. In: International Conference on Advanced Information Systems Engineering. pp. 299–318. Springer (2020)