Open access • Journal Article • DOI:10.1109/TKDE.2013.130

Determining Process Model Precision and Generalization with Weighted Artificial Negative Events — Source link

Seppe vanden Broucke, Jochen De Weerdt, Jan Vanthienen, Bart Baesens

Institutions: Katholieke Universiteit Leuven

Published on: 01 Aug 2014 - IEEE Transactions on Knowledge and Data Engineering (IEEE)

Topics: Conformance checking, Process mining, Business process modeling, Process modeling and Business process management

Determining Process Model Precision and Generalization with Weighted Artificial Negative Events

Seppe K.L.M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen and Bart Baesens

Abstract—Process mining encompasses the research area which is concerned with knowledge discovery from event logs. One common process mining task focuses on conformance checking, comparing discovered or designed process models with actual real-life behavior as captured in event logs in order to assess the "goodness" of the process model. This paper introduces a novel conformance checking method to measure how well a process model performs in terms of precision and generalization with respect to the actual executions of a process as recorded in an event log. Our approach differs from related work in the sense that we apply the concept of so-called weighted artificial negative events towards conformance checking, leading to more robust results, especially when dealing with less complete event logs that only contain a subset of all possible process execution behavior. In addition, our technique offers a novel way to estimate a process model's ability to generalize. Existing literature has focused mainly on the fitness (recall) and precision (appropriateness) of process models, whereas generalization has been much more difficult to estimate. The described algorithms are implemented in a number of ProM plugins, and a Petri net conformance checking tool was developed to inspect process model conformance in a visual manner.

Index Terms—process mining, conformance checking, artificial negative events, precision, generalization.

1 Introduction

Process mining encompasses the research area concerned with knowledge discovery from event logs [1, 2, 3, 4]. One common process mining task pertains to process conformance checking, where existing process models are compared with real-life behavior as captured in event logs so as to measure how well a process model performs with respect to the actual executions of the process at hand [5, 6]. As such, the "goodness" of a process model is typically assessed over the following four quality dimensions [7, 8]: fitness (or: recall, sensitivity), indicating the ability of the process model to correctly replay the observed behavior; precision (or: appropriateness), i.e. the model's ability to disallow unwanted behavior; generalization, which indicates the model's ability to avoid overfitting; and finally, simplicity (or: structure, complexity), stating that simpler process models should be preferred above more complex ones if they are able to fit the observed behavior just as well, thus embodying the principle of Occam's Razor.

This paper focuses on the precision and generalization dimensions. We propose a novel conformance checking approach in order to measure how well a process model performs with respect to the actual executions of a process as recorded in an event log. Our approach offers the following contributions. First, although the use of negative events for conformance checking itself was first proposed in [9], an improved strategy for artificial negative event induction is applied [10], extending it with a novel weighting method in order to tackle the problem of event log completeness, so that obtained conformance assessments are more robust when dealing with less complete event logs (i.e. logs containing only a subset of all possible process execution behavior). Apart from utilizing an improved and weighted artificial negative event induction strategy, the complexity of the original generation algorithm was significantly reduced by implementing a suffix tree based generation procedure. Second, the concept of weighted artificial negative events is used as the basis for two new conformance checking metrics: *Weighted Behavioral Precision* (p_B^w) and *Weighted Behavioral Generalization* (g_B^w). Most existing literature has focused on the fitness and precision of process models, whereas the ability to generalize has been much more difficult to estimate or describe. Our method is able to assess both precision and generalization, taking into account the inherent tradeoff between these two dimensions in an explicit manner for the evaluation of process models. Finally, all described algorithms have been implemented in a number of ProM[1] plugins; a Petri net conformance checking tool was developed to inspect model quality, conformance and deviations in a visual

Seppe vanden Broucke, Jan Vanthienen and Bart Baesens are affiliated with the Department of Decision Sciences and Information Management, KU Leuven, Naamsestraat 69, B-3000 Leuven, Belgium. E-mail: seppe.vandenbroucke@kuleuven.be
Jochen De Weerdt is affiliated with the Business Process Management Group, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia.
Bart Baesens is also affiliated with the School of Management, University of Southampton, Highfield, Southampton, SO17 1BJ, United Kingdom.

1. Similarly to the WEKA toolset for data mining, ProM, a plugin framework for process mining, is available as free, open source software for use within the process mining community. See: http://www.processmining.org.

manner, using both a heuristic as well as a model-log alignment based replay technique [11] for the calculation of the proposed metrics.

The remainder of the paper is structured as follows. Section 2 provides preliminaries in order to clarify some basic terms and concepts. Section 3 introduces weighted artificial negative events, together with an experiment in order to illustrate the validity of the proposed extension. In Section 4, the approach towards estimating process model precision and generalization is put forward; Section 5 empirically evaluates the proposed approach and compares the technique with related methodologies. An assessment of the scalability of the proposed metrics is provided in this section as well. The paper is concluded in Section 6, where remarks regarding future work are also provided.

2 PRELIMINARIES

This section provides an overview of important concepts and notations used throughout the paper, together with a concise description of the basic (non-weighted) artificial negative event generation procedure.

2.1 Notations and Definitions

We assume readers to be familiar with well-known concepts used in the context of process mining, such as event logs and the semantics of Petri net models (see e.g. [12, 13]), as we will apply Petri nets to represent process models in this paper. Remark, however, that every process model representation with event-granular execution semantics can be utilized with the described techniques just as well. During the remainder of this paper, we will use the following definitions and notations. Let A_L with size $|A_L|$ denote the finite set of activities that can occur in an event log L. Let L be a multiset of traces. The cardinality of an event log $|L|$ denotes the total number of traces in the log (including duplicates). $|L_D|$ denotes the distinct cardinality of the event log, i.e. the number of unique traces. A trace (denoted with a Greek letter) $\sigma \in L$ is a finite sequence with length $|\sigma|$ and with $\sigma_i \in A_L$ for all $0 < i \leq |\sigma|$, with σ_i the event (i.e. activity) at position i in trace σ, so that $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_{|\sigma|} \rangle$.

Fig. 1 depicts an event log together with a set of process models expressed as Petri nets to illustrate the impact of the four quality dimensions, similarly as done in [8]. Process model *perfectModel* shows a high quality, "perfect" model: all traces contained in the event log can be replayed by the model (fitness), the model does not allow for extra behavior not found in the event log (precision), the model does not overfit the event log (generalization) and is structurally simple and easy to understand (simplicity). The second process model *singleModel* represents only one "single path" from start to finish and is thus unable to generalize. In addition, only a small subset of traces can be replayed correctly. The third model (*flowerModel*) permits any sequence of transitions (between starting and ending transitions). As such, flower models are simple, generalize well and are able to fit all traces, but score low on precision since a lot of additional behavioral not found in the event log is allowed. Next, *connectedModel* is comparable to a flower model with regards to the four quality dimensions, except that here, a fully connected Petri net is used with an abundance of invisible "routing" transitions in order to allow any sequence of activities[2], making the model much harder to understand. Finally, *stackedModel* shows a model where each trace in the event log is modeled as a separate path of transitions between the start and end place (remark the occurrence of duplicate transitions). Although well performing in terms of fitness and precision, this model heavily overfits the event log and is thus not able to generalize. In addition, the model is not simple to interpret. Although the structural logic behind a stacked model looks simple enough, finding out which path should be followed in order to replay a trace is more difficult, especially during execution where the future behavior of a process instance is likely unknown at the current point in time.

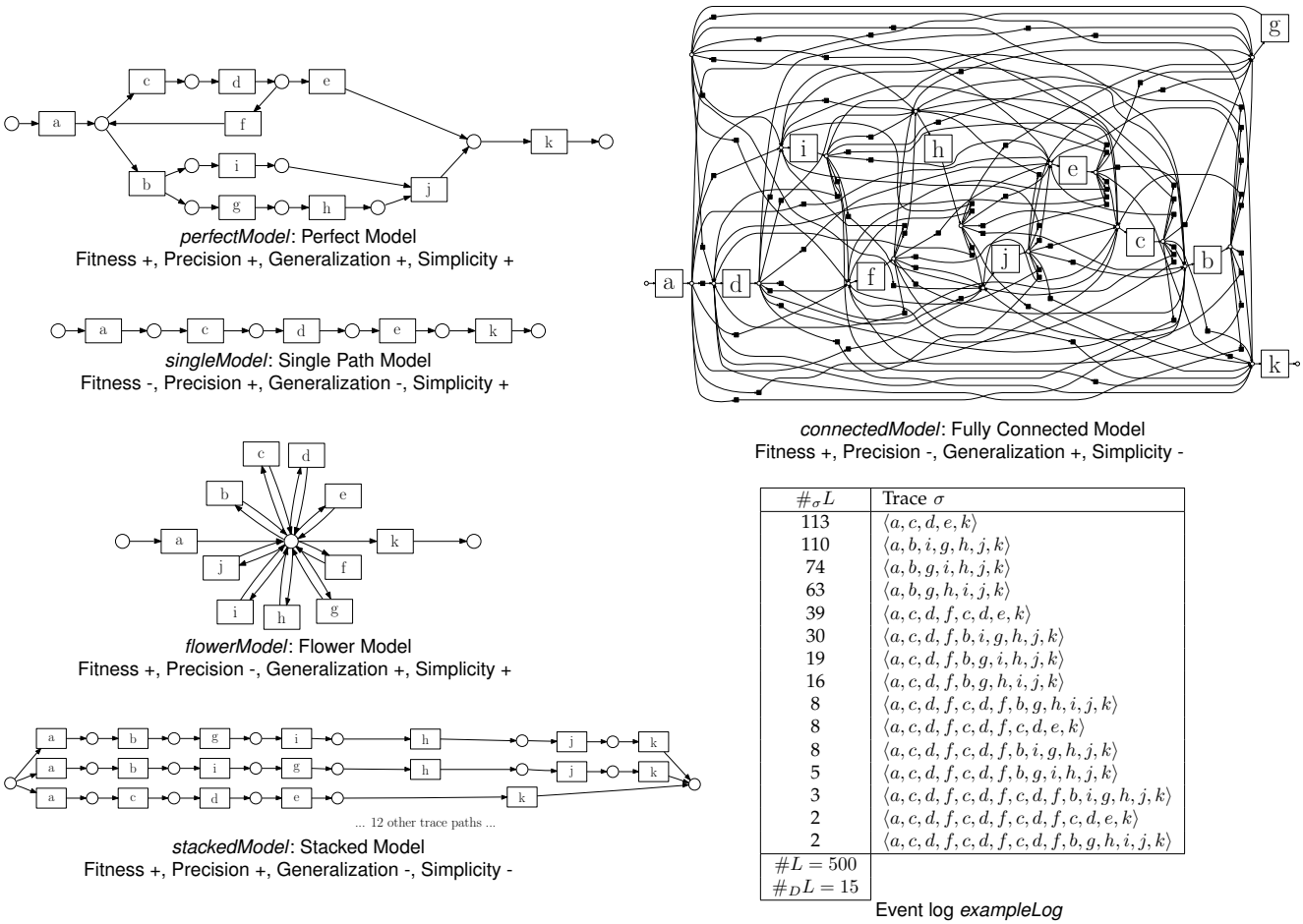
2.2 Artificial Negative Events

Our precision and generalization evaluation approach is based on the use of weighted artificial negative events. The definition of which expands on earlier work on artificial negative events, with the initial formulation found in [9] and subsequent improvements detailed in [10]. A concise overview on (non-weighted) artificial negative events is given below so as to provide background before continuing with our new extension.

Negative events represent information about activities that were prevented from taking place. Such events are rarely logged in real-life life logs [8], so that in [9], a generation algorithm is proposed to induce negative events in an artificial manner in an event log, which can be summarized as follows. Negative events record that at a given position in a trace, a particular event cannot occur. Thus, at each position in each trace in the event log, it is examined which negative events can be induced, by checking whether traces exist in the event log where the negative event under consideration does occur and was preceded with a similar execution history. If this check fails, no counter-evidence for the candidate negative event could be found in the event log, and the negative event is inserted in the original trace.

As an example, consider the event log *exampleLog* from Fig. 1. After running the artificial negative generation algorithm, negative events are inserted into the traces, so that the trace $\langle a, c, d, e, k \rangle$ for instance now reads as: $\langle (b^-, c^-, d^-, e^-, f^-, g^-, h^-, i^-, j^-, k^-), a, (a^-, d^-, e^-, f^-, g^-, h^-, i^-, j^-, k^-), c, \ldots, (a^-, b^-, c^-, d^-, e^-, f^-, g^-, h^-, i^-, j^-), k \rangle$ (a super scripted

2. The fully connected model may appear rather unrealistic. However, note that Causal Nets [14], another representational form for process models, may end up looking like *connectedModel* after converting such models to Petri nets.



$\#_\sigma L$	Trace σ
113	$\langle a, c, d, e, k\rangle$
110	$\langle a, b, i, g, h, j, k\rangle$
74	$\langle a, b, g, i, h, j, k\rangle$
63	$\langle a, b, g, h, i, j, k\rangle$
39	$\langle a, c, d, f, c, d, e, k\rangle$
30	$\langle a, c, d, f, b, i, g, h, j, k\rangle$
19	$\langle a, c, d, f, b, g, i, h, j, k\rangle$
16	$\langle a, c, d, f, b, g, h, i, j, k\rangle$
8	$\langle a, c, d, f, c, d, f, b, g, h, i, j, k\rangle$
8	$\langle a, c, d, f, c, d, f, c, d, e, k\rangle$
8	$\langle a, c, d, f, c, d, f, b, i, g, h, j, k\rangle$
5	$\langle a, c, d, f, c, d, f, b, g, i, h, j, k\rangle$
3	$\langle a, c, d, f, c, d, f, c, d, f, b, i, g, h, j, k\rangle$
2	$\langle a, c, d, f, c, d, f, c, d, f, c, d, e, k\rangle$
2	$\langle a, c, d, f, c, d, f, c, d, f, b, g, h, i, j, k\rangle$
$\#L = 500$	
$\#_D L = 15$	

Event log *exampleLog*

Fig. 1. An event log *exampleLog* together with five process models. The process models illustrate the impact of the four quality dimensions: fitness, precision, generalization and simplicity.

minus symbol is used to indicate negative events; the negative events before a positive event are inserted as a set).

In [9] and [15] respectively, a fitness and precision metric are proposed in order to evaluate process models on these two quality dimensions, using event log traces in which artificially generated negative events have been induced. In accordance with the construction of a confusion matrix in the field of data mining, sets of true and false positive and negative events are defined as follows. Let $NE(\sigma_i)$ be a function denoting the set of negative events which can be induced before σ_i in a trace σ. Each trace $\sigma \in L$ is replayed on a process model after inducing negative events with $NE(\sigma_i)$ for each $\sigma_i \in \sigma$. During trace replay, whenever a positive event is encountered, a check is performed to determine whether a model element can be found which can be executed to fire the corresponding positive event. For Petri nets, we thus check if a transition mapped to the positive event at hand is enabled, in which case the event is added to the set of true positive events; if not, the event is added to the set of false negatives, followed with a forced firing of a disabled transition mapped to the positive event. Negative events on the other hand are treated differently from positive ones when evaluating a process model. As negative events denote activities which should be prevented from being executed by the process model at hand given a certain context (i.e. a marking for a Petri net), it is thus checked whether it would be possible to fire a transition mapped to the negative event at hand, given the marking obtained so far (if so, the negative event is a false positive; if not, a true negative), but the corresponding model element itself is not fired.

3 Weighted Artificial Negative Events

Our precision and generalization evaluation approach is based on the use of weighted artificial negative events. Weighted artificial negative events extend the concept of artificial negative events with a scoring mechanism in order to make the evaluation of process models more robust as event logs become less complete.

3.1 Rationale and Formulation

Generating a robust set of negative events boils down to finding an optimal set of negative examples under

the counteracting objectives of correctness (prevent generation of incorrect negative events which would lead to pessimistic conformance scores) and completeness (induce "non-trivial" negative events which are based on constraints imposed by complex structural behavior). The existence of the tradeoff between these two goals is due to the completeness assumption made over an event log, which comes into play for each process mining task. Under its most strict form, a completeness assumption requires that the given event log contains all possible traces that can occur. Without some assumption regarding the completeness of a given event log, it would be impossible to induce any negative events at all, since no single candidate negative event can be introduced in the knowledge that the given log does not cover all possible cases. Process discovery algorithms make a similar assumption in order to derive models which are not overly general.

Assuming a strict completeness assumption as defined above is often unrealistic in practice. The original version of the negative event induction algorithm [9, 10] therefore allows to configure the completeness assumption made on an event log by means of a window size parameter and takes concurrent and recurrent behavior into account as well when testing whether a negative event can be induced by generating additional varying behavior based on existing traces. This approach, however, comes with two drawbacks. First, even when taking into account existing techniques to estimate the completeness of a given event log without an a-priori known process model [16], which could be applied in order to guide the artificial negative event generation algorithm, the problem still remains that negative events would be either induced or not, without some kind of strength or confidence associated to their presence based on the structure contained in the given log. Second, generating trace variants in order to provide more behavior which can be used as counter-evidence towards the presence of a negative event is a time-consuming step, requiring the discovery of structural properties in the event log followed with the recursive generation of the variants.

To resolve these issues, we propose a scoring method which can be used to weight negative events in terms of their confidence. That is, the higher the weighting of a negative event, the less likely it is deemed that this negative event will be refuted by additional traces as generated by the real underlying, unknown process and vice versa. The calculation of this weight is done in the following manner (formalized as a definition for function $NE(\sigma_i)$ in Algorithm 1): we calculate the score for a negative event with activity $a \in A_L \backslash \{\sigma_i\}$ (lines 3-4). For each trace υ containing activity a at υ_j (lines 5-7), the event window before υ_j is compared with the event window before σ_i in the original trace σ in order to obtain the "unmatching window ratio", i.e. the length of the unmatching window divided by the total window length in σ, working backwards from σ_i and υ_j: $\frac{|window|-|matching\ window|}{|window|}$ (lines 8-15). Remark that it is possible for a single trace υ to contain multiple activities equal to the negative event under consideration, so that one trace can give rise to multiple comparisons, and thus different unmatching window ratios. Remark also that the current $\sigma \in L$ itself may also contain the activity for the candidate negative event under consideration. Finally, to obtain the final weighting for a negative event, the minimum unmatching window ratio is taken over all comparisons performed (line 16): $Min_{\forall window\ comparisons}(\frac{|window|-|matching\ window|}{|window|})$. To induce all weighted artificial negative events, function $NE(\sigma_i)$ is called for each $\sigma_i \in \sigma, \sigma \in L$.

Algorithm 1 Weighted artificial negative event generation algorithm.

1) given an activity σ_i in trace $\sigma \in L$, event log L with activities A_L
2) function $NE(\sigma_i)$ % Induce set of neg. events
3) let $N := \emptyset$ % Neg. events induced at this position
4) for each $a \in A_L \setminus \{\sigma_i\}$ do
5) let $s := 1$ % Score for this neg. event
6) for each $\upsilon \in L$ do
7) for each $\upsilon_j \in \upsilon : \upsilon_j = a$ do
8) % Calculate unmatching window ratio:
9) let $ws := i - 1$ % Window size
10) let $mw := 0$ % Matching window size
11) let $l := 1$
12) while $l < Min(i, j) - 1 \wedge \sigma_{i-l} = \upsilon_{j-l}$ do
13) let $mw := mw + 1$
14) let $l := l + 1$
15) let $uwr := \frac{ws-mw}{ws}$ % Unmatching window ratio for this comparison
16) let $s := Min(s, uwr)$
17) let $N := N + \{a_s^-\}$ % Neg. event with activity a and weight s
18) return N

An example can help to clarify the weighting procedure. Consider the trace $\sigma \in L = \langle a, b, c, x, d \rangle$; we wish to obtain the unmatching window ratio for a negative event, say y, inserted before $\sigma_4 = x$ by comparing the window before σ_4 (i.e.: $\langle a, b, c \rangle$, $|window| = 3$) with the window before υ_5 in the trace $\upsilon = \langle e, a, f, c, y, g \rangle$, which is another trace in L that does contain event y. The window in υ is thus: $\langle e, a, f, c \rangle$. These two windows ($\langle a, b, c \rangle$ and $\langle e, a, f, c \rangle$) are now compared as follows. We calculate the matching window length, starting backwards from both windows. The first pair of events, $\sigma_3 = \upsilon_4 = c$, are indeed equal, so the matching window length is incremented with 1. The next pair, $(\sigma_2 = b) \neq (\upsilon_3 = f)$, however, is not equal, so the comparison is ended at this point, even though the next pair $\sigma_1 = \upsilon_2 = a$ is equal again. The unmatching window ratio for this result thus amounts to $\frac{3-1}{3} = 0.66$. The final weighting for this negative event is obtained by taking the minimum unmatching window ratio over all similar window comparisons which could be performed.

The weighting for each negative event can be sum-

marized as follows: the longer a matching prefix can be found equal to the prefix before the negative event under consideration, the smaller the unmatching window will become and a lower weight will be given to the negative event. A weighting of 0 (minimum) indicates that a trace was found containing the same full prefix as seen before the negative event under consideration, indicating that this behavior in fact did occur and as such cannot be supported at all as being disallowed (i.e. negative) behavior. A weighting of 1 (maximum) indicates that there did not exist any trace where the candidate negative event's activity occurred and was preceded by a matching prefix (even of length 1). Strong evidence then exists that the behavior represented by the candidate negative event should indeed be disallowed.

The scalability of the weighted negative artificial event induction procedure as described above is weak. The complexity of inducing all weighted artificial negative events in an event log L with Algorithm 1 can be expressed as follows: $\mathrm{O}\big((|L| \times |\mu|) \times (|A_L|) \times (|L| \times |\mu| \times |\mu|)\big)$. That is, for each trace in the event log L, and for each position in that trace ($|\mu|$ is equal to the length of the longest trace in the event log L and forms an upper bound for the number of positions iterated), function $NE(\sigma_i)$ is evaluated to induce the negative events for this position, which considers each activity in A_L as a candidate negative event. For each such candidate event, the traces in the event log are iterated again together with its activities, and a window comparison is performed whenever an activity is encountered which is equal to the current candidate negative event (worst case meaning every position in every trace with a maximum window length of $|\mu|$). This evaluates to $\mathrm{O}\big(|L|^2 \times |\mu|^3 \times |A_L|\big)$. To deal with the problem of scalability, we utilize Ukkonen's algorithm [17] in order to construct a suffix tree over the event log in order to quickly perform window lookups. As an example, consider the trace $\langle a, b, c, d, e, f, g\rangle$. A suffix tree over this trace yields the following retrievable suffixes: $\langle a, b, c, d, e, f, g\rangle$ (the trace itself), $\langle b, c, d, e, f, g\rangle$, $\langle c, d, e, f, g\rangle$, $\langle d, e, f, g\rangle$, $\langle e, f, g\rangle$, $\langle f, g\rangle$ and $\langle g\rangle$. Suppose now we want to find the matching length for the window $\langle x, c, d\rangle$ (i.e. 2). As it stands, the suffix tree does not contain an entry point for x and as such, the matching length is deemed to be 0. One solution is to iterate over the trace and add the suffixes of $\langle a, b, c, d, e, f, g\rangle, \langle b, c, d, e, f, g\rangle, \ldots, \langle g\rangle$, but the more optimal and preferred approach is to construct the suffix tree over the reversed trace, i.e. $\langle g, f, e, d, c, b, a\rangle$ resulting in the following suffixes: $\langle g, f, e, d, c, b, a\rangle$, $\langle f, e, d, c, b, a\rangle$, $\langle e, d, c, b, a\rangle$, $\langle d, c, b, a\rangle$, $\langle c, b, a\rangle$, $\langle b, a\rangle$ and $\langle a\rangle$. To find the matching window length of $\langle x, c, d\rangle$, the window is traversed backwards so that now, $\langle d, c\rangle$ is found with a length of 2. Observe that this reversal has the effect of losing the online property of Ukkonen's algorithm, but retains linear-time construction complexity. The complexity of the implemented weighted artificial negative event generation algorithm to generate all weighted artificial negative events in an event log is thus as follows: $\mathrm{O}\big((|L| \times |\mu|) + (|L| \times |\mu|) \times (|A_L|) \times (|\mu|)\big)$; the suffix tree construction step is executed once and is linear in the size of the alphabet (in this case, the length of the longest trace times the log size is an upper bound) [17]. Next, every trace in the event log is iterated once more at each position, and each activity is still evaluated as a candidate negative event. However, the lookup of the matching window now scales linearly with the length of the longest trace. As such, the overall complexity evaluates to $\mathrm{O}\big(|L| \times |\mu|^2 \times |A_L|\big)$, so that the induction algorithm now scales linearly with the size of the event log and the activity alphabet.

This contribution regarding the weighting of negative events and the application of suffix trees greatly improves the robustness of the negative event induction to varying levels of event log completeness and the time needed to generate artificial negative events compared to existing techniques, especially since the trace variant generation step can be dropped without a significant loss of accuracy in the weighting of a negative event, as will be shown in the following subsection.

3.2 Empirical Validation

To validate our weighted artificial negative event approach, we apply the proposed generation technique (Algorithm 1) on a number of process event logs in a controlled environment for which the reference model is known beforehand. Having such a reference model allows us to construct a complete and correct set of negative events, as defined by the process model itself, by which the artificially generated set of events can then be evaluated. Note that, in real-life process mining settings, a true reference model is, naturally, almost always unavailable, so that these models are only applied in this section as a means to validate the performance of the weighted artificial negative event generation procedure.

Twenty-five different event logs, containing a variety of structural constructs and differing in complexity, have been utilized. Twenty of these logs were used before by Alves de Medeiros et al. [18] and have since become part of a widely used benchmarking set in the field of process mining. Additionally, another log, *complex*, was built, containing complex behavior (e.g. nested loops and parallelism). Table 1 lists the main characteristics of these event logs. The overview also includes four real-life logs which will be used in further sections.

The first validation task investigates whether the weight given to generated artificial negative events is able to correctly differentiate between correct and incorrect negative events as indicated by the reference model. Our tests indeed show that this is the case for the logs included in our experiment. Fig. 2 depicts the distribution of artificial negative event weights for event log *driversLicenseLoop*. The distributions for the other logs are similar and omitted for brevity. For most candidate negative events, the weighting technique is able to correctly and unambiguously (i.e. with absolute

Table 1
Characteristics of event logs and reference models used in the weighted artificial negative event validation setup.

Event Log	$\|A_L\|$	$\|L\|$	$\|L_D\|$	parallelism?	loops?	invisibles?	non-free choice?	duplicates?
a10skip	12	300	6	✓		✓		
a12	14	300	5	✓				
a5	7	300	13	✓	✓	✓		
a6nfc	8	300	3	✓		✓	✓	
a7	9	300	14	✓				
a8	10	300	4	✓				
betaSimplified	13	300	4			✓	✓	✓
choice	12	300	16			✓		
driversLicense	9	2	2			✓		
driversLicenseLoop	11	350	87	✓	✓	✓	✓	✓
herbstFig3p4	12	32	32	✓	✓			
herbstFig5p19	8	300	6	✓		✓		✓
herbstFig6p18	7	300	153		✓	✓		
herbstFig6p31	9	300	4					✓
herbstFig6p36	12	300	2			✓	✓	
herbstFig6p38	7	300	5	✓				✓
herbstFig6p41	16	300	12	✓				
l2l	6	300	10		✓	✓		
l2lOptional	6	300	9		✓	✓		
l2lSkip	6	300	8		✓	✓		
complex	19	6107	1006	✓	✓			✓
hospital (real-life)	626	1143	981	–	–	–	–	–
incident (real-life)	18	24770	1174	–	–	–	–	–
telecom (real-life)	42	17812	1908	–	–	–	–	–
ticketing (real-life)	9	276599	3140	–	–	–	–	–

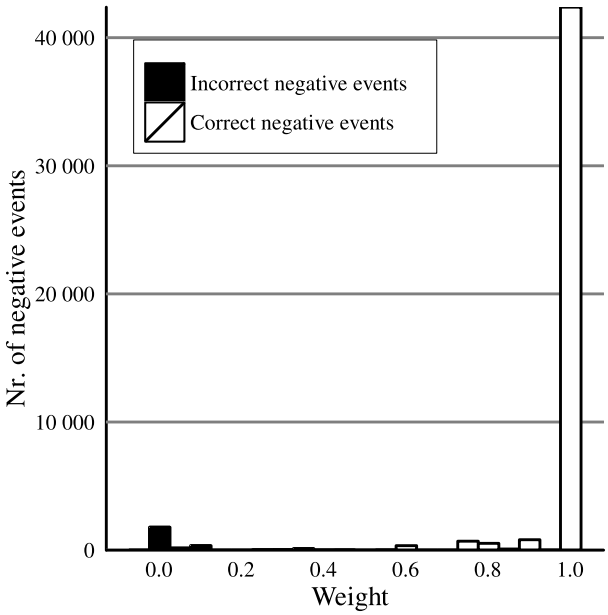


Fig. 2. Overview of generated artificial negative events with their calculated weights for event log *driversLicenseLoop*. White colored bars represent correct negative events as indicated by the reference model, whereas black coloring indicates incorrect negative events.

confidence) indicate whether the negative event is valid (weight of 1) or invalid (weight of 0). As logs become more complex or less complete, more candidate negative events will occur for which it is impossible (based on the given log) to unambiguously derive whether these candidates are valid or not. They are given a weight between 0 and 1. Our experiment shows that these weights correspond to our initial requirement, namely the assignment of a lower weight to incorrect negative events (as determined by the reference model available in this controlled setup) and a higher weight to correct negative events. While it is the case that – for sufficiently complete logs – the correctness of the multitude of negative events can be determined in a straightforward manner based on the given log (i.e. all negative events with weight equaling 1), it should be noted that the negative examples which cannot be derived as easily often reveal the most discerning information, such as the presence of non-free choice constructs, for example, where a choice in a process model is bounded by choices made earlier in the process, and where one thus desires the induced negative events to reflect this behavior. This underlines the strength of the weighting mechanism as described above, as it now becomes possible to consider all candidate negative events in further analysis tasks, taking into account the confidence measure given to their existence.

The second validation task investigates how the weighting given to negative events evolves in comparison to the completeness of the given input event log. Fig. 3 depicts the evolution of negative event weights versus the completeness of the log (shown again for *driversLicenseLoop*). To generate the logs containing all distinct traces, CPNTools[3] was used to simulate complete event logs (bounded in the number of loops allowed) from the reference Petri nets. Starting from this complete log, distinct traces were randomly removed to obtain smaller sized, less complete logs. More precisely, to generate the artificial negative events, the input log in which negative events are induced is kept constant over all runs, corresponding with the logs listed in Table 1, whereas the log used to build the suffix tree is modified for each run and set equal to the differently sized logs. This ensures that the same negative events are generated in each run, allowing to better compare the evolution of their weights. If the differently sized logs themselves would have been used to induce the negative events herein, their varying sizes would impact the average weight for the correct and incorrect negative events, as an increase in log size could create a large additional amount of negative events, making it impossible to track their global weight evolution. This operation was repeated twenty times. The results again support our requirement that correct negative events are generally given a higher weight than incorrect ones, and that this difference further increases as the input event log becomes more complete. For some logs, however, the

3. See: http://cpntools.org.

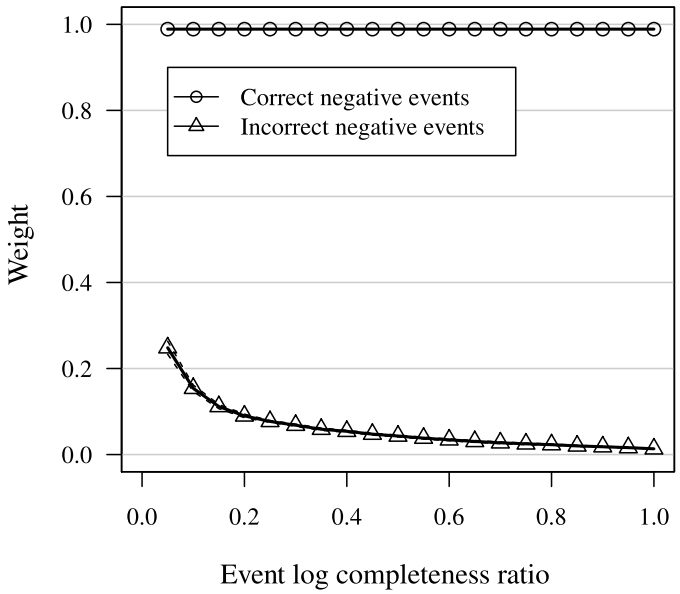


Fig. 3. Evolution of artificial negative event weights in comparison with event log completeness for *driversLicenseLoop*, averaged over twenty iterations with the 95% confidence interval shown above and below the average. Negative events with low weights are refuted as additional traces are added (i.e. reach a weight of 0) whereas negative events with higher weights remain stable.

mean weight obtained for the incorrect negative events remains high when these logs are very incomplete. This behavior is somewhat expected and desired, as it could be argued that the reference model can no longer be accepted as the ideal solution in terms of quality when the input event log gets too small. As such, the aim of assigning a weighting to artificially induced negative events is not to uncover the true underlying reference model behind a log, even when the log is very incomplete (this would be impossible, in fact), but rather to weaken the completeness requirement in a robust and correct manner, so that less complete event logs can still be used to evaluate more complex process models without negatively impacting the quality assessment. Remark that all artificial logs listed in Table 1 are sufficiently complete, to ensure that their corresponding reference model remains the optimal solution quality-wise.

4 Checking Precision & Generalization

Based on the definition of weighted artificial negative events as explained in the previous section, we now introduce two new metrics, *Weighted Behavioral Precision* (p_B^w) and *Weighted Behavioral Generalization* (g_B^w), to assess the conformance of process models in accordance with a given event log.

To calculate the precision and generalization of a process model in comparison with a given log, all traces $\sigma \in L$ are replayed on the given process model. For each such trace, we calculate values according with the sets of true positive events TP (positive events which could be replayed without error), false positive events FP (negative events which could be replayed and are thus erroneously permitted by the process model), allowed generalizations AG (generalized events which could be replayed without error and confirm the model's ability to generalize) and disallowed generalizations DG (generalized events which could not be replayed by the process model). The concept of allowed and disallowed generalizations requires some further explanation. Consider the trace $\sigma = \langle a\,,\ b\,,\ d\,,\ e\rangle$ from a log L with $A_L = \{a, b, c, d, e\}$. We induce artificial negative events, so that $\sigma^- = \langle (b^-\,,\ c^-\,,\ d^-\,,\ e^-)\,,\ a\,,\ (a^-\,,\ d^-\,,\ e^-)\,,\ b\,,\ (a^-\,,\ b^-\,,\ c^-\,,\ e^-)\,,\ d\,,\ (a^-\,,\ b^-\,,\ c^-\,,\ d^-)\,,\ e\rangle$ (assume the weight of each negative event equal to 1, i.e. complete confidence). Consider now the positive event $\sigma_2 = b$. The set of negative events preceding this positive event is $NE(\sigma_i) = \{a^-, d^-, e^-\}$. By comparing these events with the full activity alphabet A_L, it is possible to deduce which alternative events should also be permitted by the process model after the execution of a (i.e. before $\sigma_2 = b$), namely $A_L \backslash \{b, a, d, e\} = \{c\}$ (the activity alphabet minus the positive event and its preceding negative events). The process model can then be queried to investigate whether these deduced generalized events are indeed accepted or not. We have assumed the negative events in this example to have a weight of 1; it follows that the strength of a negative event also indicates something about the strength of this candidate event towards generalization. In short: negative events with a weight of 1 are unusable to assess generalization capabilities of a process model, whereas negative events with a weight of 0 (the ones not listed in trace σ^-) are completely unusable to assess precision, but fully valid to determine a model's capability to generalize. Consequently, negative events with a weight between the extrema of 0 and 1, impact both precision and generalization. From now on, we will thus assume that, when artificial negative events are generated in traces, all activities in A_L are inserted before each positive event, excluding the positive event itself, together with their associated weights.

The values for TP, FP, AG and DG are now calculated as follows between an event log L and a process model. Each trace $\sigma \in L$ is replayed on the process model. For every positive event $\sigma_i \in \sigma$, function $NE(\sigma_i)$ is called to induce the set containing all artificial negative events with their weights at this position. Starting from the state reached so far in the trace replay (e.g. the current marking in case of a Petri net), we inspect whether each negative event $n \in NE(\sigma_i)$ could be fired by the process model. If this is indeed the case, the value of FP is incremented with the weight of the negative event $Weight(n)$, and the value of AG is incremented with $1 - Weight(n)$. If the process model is unable to parse the negative event, DG is incremented with $1 - Weight(n)$. Remark that we only inspect the possibility of executing each negative event, without actually firing the corresponding process model element. After evaluating the negative events at the current

position towards checking precision and generalization conformance, the positive event itself is parsed. When this event could be fired without error, the value of TP is increased by 1.

Concerning the implementation of trace replay, a heuristic procedure has been developed to evaluate event logs on Petri nets, similar to other approaches described in literature [6, 9]. However, evaluating traces containing negative events requires a modified trace replay procedure, which explores from the marking obtained after firing the last encountered positive event all invisible transition "paths" in order to determine if a negative event can be enabled or not, an aspect which is not taken into account in earlier techniques. Additionally, some words should be devoted to the aspect of force firing. As the force firing of transitions produces additional tokens, it might follow that these tokens subsequently lead to the undesirable enabling of negative events. Adriansyah et al. have proposed a replay technique which avoids force firing altogether by allowing the process model and log to move independently (thus resulting in a model-log alignment). As such, we have added a second replay procedure to establish a best fitting firing sequence based on the alignment technique as described in [19, 20], which evaluates positive events based on whether the event under consideration could be successfully aligned with a transition execution in the process model. For negative events, the same evaluation procedure as before is utilized (with invisible path exploration), starting from the marking obtained after the last aligned model-log move. In our heuristic approach, we skip the evaluation of the set of negative events following immediately after a force fired positive event. More details on how to replay an event log with negative events in a Petri net are described in a technical report [21].

The precision and generalization metrics are now calculated as:

$$p_B^w = \frac{TP}{TP + FP} \qquad g_B^w = \frac{AG}{AG + DG}$$

Remark that the weight of a negative event (its impact on precision) was defined as $Min_{\forall window\ comparisons}(\frac{|window|-|matching\ window|}{|window|})$, i.e. the minimal unmatching window ratio over all window comparisons performed. As such, the impact of a negative event on generalization equals $1 - Min_{\forall window\ comparisons}(\frac{|window|-|matching\ window|}{|window|})$ or, also, $Max_{\forall window\ comparisons}(\frac{|matching\ window|}{|window|})$ i.e. the maximal matching window ratio found over all window comparisons performed.

We have, thus far, presented a unified technique to calculate precision and generalization based on event traces with weighted artificial negative events. Note finally that fitness can still be calculated as in [9] (i.e. using the true positive and false negative values).

All described techniques and methods are implemented in a series of ProM plugins. For the sake of brevity, we do not describe in depth further details regarding the architecture, parameters and structure of the various objects included in the plugins. Instead, binaries and source code, together with installation instructions can be retrieved from http://processmining.be/neconformance.

5 Experimental Evaluation

This section presents a comparative analysis of our proposed *Weighted Behavioral Precision* (p_B^w) and *Weighted Behavioral Generalization* (g_B^w) metrics in respect to various other approaches found in the literature.

5.1 Setup

The following techniques are included in the experimental setup. For precision, the *Advanced Behavioral Appropriateness* ($a_B^{'}$) metric is included, as defined by Rozinat et al. [6]. Although this metric provides a theoretically sound method in order to evaluate the precision of a process model, it has been argued that a number of drawbacks exist, among which an exhaustive calculation requirement and an implementation which is only approximate [7]. Second, the *ETC Precision* metric (etc_p) as proposed by Muñoz-Gama and Carmona [22] evaluates precision based on the concept of so-called escaping edges. Adriansyah et al. define a similar precision metric as *ETC Precision* (etc_p) in [11], based on their model-log alignment replay technique, denoted here as *Alignment Based Precision* (p_A). The same authors have also defined *One Align Precision* (a_p^1) and *Best Align Precision* (a_p) metrics [23, 24], which combine the concept of model-log alignments with the metrics defined by Muñoz-Gama and Carmona. Adriansyah et al. also propose a generalization metric in [11] where a Bayesian estimator is applied in order to derive a model's ability to generalize, based on the idea that the likelihood of new, unseen behavior in a certain state depends on the number of times this state was encountered and the number of different decisions (i.e. activities) observed in this state. This technique has been included as *Alignment Based Generalization* (g_A). In [6], an *Advanced Structural Appropriateness* ($a_S^{'}$) metric is defined to determine whether a process model overfits an event log. However, this metric only takes into account certain structural properties of a process model (redundant invisible transitions and alternative duplicate transitions), so that it is not fully able to evaluate generalization. More precisely, it is argued that this metric better fits with the fourth quality dimension – simplicity, as redundant and duplicate transitions mainly hinder a model's ability to be well and easily understood than its ability to parse event traces. As such, we do not include this entry in our setup. Finally, we compare our approach with a related technique described by De Weerdt et al. [15] and Goedertier et al. [9], where the notion of precision (*Behavioral Precision*, p_B) based on negative events was initially put forward. Since our technique extends both the generation procedure of negative

events and the replay procedure applied, we expect to gain a performance increase. This earlier approach does not define a generalization metric. Other, less related, precision and generalization metrics exist [18, 25, 26, 27], which either target a process modeling representation other than Petri nets or are unimplemented in ProM, so that the set of evaluated conformance metrics is limited to the entries described above.

The included metrics are compared with our new conformance checking approach, using both our own heuristic trace replay method and the replay method based on model-log alignments. For reasons of completeness and to confirm the capability of the developed replay techniques to correctly parse traces (based on the known reference and example models), we report *Behavioral Recall* (r_B) in the results as well (also using both replay methods).

5.2 Fixed Log Sizes

In this comparative analysis experiment, the event log test set introduced in Section 3 will be utilized. Remark that most event logs stem from a-priori known reference models and will be evaluated on the same model. As such, we expect well-performing conformance checking techniques to reach high precision and generalization scores for these combinations. For the real-life logs, no reference model is known, so that we will evaluate these logs on discovered Petri net models (using the HeuristicsMiner [28] discovery algorithm). Finally, we also incorporate the five models from Fig. 1 in our experimental setup (event log: *exampleLog*), as this allows us to evaluate which conformance checking technique is best able to punish Petri net models of extremely low quality (low precision for *connectedModel* and *flowerModel,* and low generalization for *singleModel* and *stackedModel*).

Table 2 shows the results of our experiment for the evaluated metrics on all event log-Petri net pairs. For each result, scores are reported as a number between 0 (worst) and 1 (best). Run times are given between parenthesis as seconds. Calculations were limited to 24 hours of computational time. Runs exceeding this amount of time or runs which resulted in a crash or error are left out. All experiments were executed on a workstation with 2 processors (2.53Ghz; 4 cores per processor) and 64GB of memory. The Java heap size for each individual experiment was set to 4GB. Stack size was left default, as no single stack overflow error occurred during the experimentation.

Based on the results listed in Table 2, the following observations can be made. First, many conformance checking techniques have difficulties dealing with large or complex event logs. For many real-life logs, only the artificial negative event-based conformance checking techniques and *ETC Precision* (etc_p) were able to always find a result, the others encountering out-of-time or out-of-memory errors; note that the *hospital* log forms a noteworthy exception – as the artificial negative event-based conformance checking techniques were not able to compute a result in time. Next, focusing on precision, it is found that many metrics unduly punish precision errors due to input logs being less complete. For all artificial cases, our *Weighted Behavioral Precision* metric (p_B^w) is able to find a precision value close to 1. For the real-life logs, the existing metrics return a lower value than the results obtained by *Weighted Behavioral Precision* (p_B^w). Although the true precision of these mined models is unknown, the almost-0 score obtained by *ETC Precision* (etc_p) appears to be overly pessimistic. Concerning the *exampleLog* models, we observe that for *connectedModel* and *flowerModel*, the *Weighted Behavioral Precision* metric (p_B^w) correctly punishes the many precision errors found in these models. Note that some metrics have issues when traversing the possible invisible paths in *connectedModel*, with *Behavioral Precision* (p_B) being unable to detect any imprecisions at all. Concerning generalization, the *Alignment Based Generalization* metric (g_A) returns unexpectedly high values for *stackedModel* and *singleModel* (the score of 0 for *driversLicense* is also unexpected and perhaps due to a bug). The *Weighted Behavioral Generalization* metric (g_B^w) is able to punish each generalization issue, and does so in a more strict manner. Therefore, for real-life applications, it remains recommended to first focus on model fitness and precision, followed by generalization with lower priority. A final remark should be made concerning the differences between the two implemented replay techniques (heuristic and alignment based) when using the *Weighted Behavioral Precision/Generalization* metrics (p_B^w and g_B^w). For many logs, we cannot observe a significant difference in values between the two techniques, although the run times are higher when using the alignment based replay procedure. Some interesting differences are observed for the *telecom*, *incident* and the *connectedModel* and *singleModel* logs, where the alignment based replay procedure is less punishing with regards to precision.

5.3 Varying Log Sizes

This section discusses the scalability (in terms of run time) and stability (in terms of obtained values) of the evaluated metrics under varying sizes of event logs (and thus completeness). To do so, the process model of the *complex* log was used as a basis from which twenty sets of differently sized event logs (ranging between 500 and 10 000 traces) were generated – thus totaling 400 event logs. We then evaluate each log on the reference model using the metrics discussed above.

Regarding scalability, it is investigated whether the run time performance of a metric depends heavily on the size of the evaluated log. Fig. 4 provides an overview of the required run times for each evaluated technique over various log sizes. The run times of most techniques remain within acceptable bounds. For the *Advanced Behavioral Appropriateness* metric ($a_B^{'}$), however, calculations always exceeded the allotted 24 hours. This is due to the extensive process model state space exploration

Table 2
Obtained values and run times (in seconds) for the evaluated metrics on all included event log-Petri net pairs. Calculations were limited to 24 hours of computational time. Runs exceeding this amount of time or runs which resulted in a crash or error are left out ("–"). The proposed precision and generalization metrics show consistent results across all twenty artificial logs and behave as expected with respect to the theoretical models from Fig. 1, penalizing the precision problems of the *connectedModel* and the *flowerModel* and the generalization issue of the *singleModel* and the *stackedModel*. With exception for the *hospital* log, the *Weighted Behavioral Precision/Generalization* metrics are capable to consistently compute results within the allotted time frame of 24 hours. The differences between the heuristic and alignment based replay method are limited, except for the fact that the alignment method causes increased computation times.

Event Log	Petri Net	Fitness Metric		Precision Metrics						Weighted Behavioral Precision		Generalization Metric	Weighted Behavioral Generalization	
		r_B (heuristic)	r_B (alignment)	a'_B	etc_p	p_A	a^1_p	a_p	p_B	p^w_B (heuristic)	p^w_B (alignment)	g_A	g^w_B (heuristic)	g^w_B (alignment)
a10skip	*reference model*	1.00 (0.31s)	1.00 (0.61s)	1.00 (0.33s)	0.91 (0.25s)	0.97 (0.42s)	1.00 (0.50s)	0.92 (5.60s)	1.00 (1.81s)	1.00 (0.53s)	1.00 (0.61s)	1.00 (0.48s)	0.91 (0.55s)	0.91 (0.80s)
a12	*reference model*	1.00 (0.30s)	1.00 (0.55s)	1.00 (0.35s)	1.00 (0.23s)	1.00 (0.41s)	1.00 (0.42s)	0.93 (4.38s)	1.00 (3.26s)	1.00 (0.45s)	1.00 (0.66s)	1.00 (0.39s)	0.83 (0.48s)	0.83 (0.69s)
a5	*reference model*	1.00 (0.36s)	1.00 (0.58s)	0.47 (0.38s)	0.29 (0.22s)	1.00 (0.53s)	0.93 (0.45s)	0.70 (13.50s)	1.00 (0.70s)	1.00 (0.47s)	1.00 (0.70s)	1.00 (0.55s)	0.88 (0.56s)	0.88 (0.99s)
a6nfc	*reference model*	1.00 (0.17s)	1.00 (0.39s)	0.69 (0.35s)	0.51 (0.22s)	1.00 (0.34s)	0.79 (0.41s)	0.78 (2.31s)	0.94 (0.53s)	1.00 (0.30s)	1.00 (0.49s)	1.00 (0.47s)	0.64 (0.33s)	0.64 (0.47s)
a7	*reference model*	1.00 (0.34s)	1.00 (0.55s)	1.00 (0.33s)	1.00 (0.23s)	1.00 (0.39s)	1.00 (0.48s)	0.77 (8.84s)	1.00 (0.72s)	1.00 (0.44s)	1.00 (0.72s)	1.00 (0.41s)	0.70 (0.47s)	0.70 (0.74s)
a8	*reference model*	1.00 (0.22s)	1.00 (0.37s)	1.00 (0.30s)	1.00 (0.22s)	1.00 (0.30s)	1.00 (0.33s)	0.80 (1.89s)	1.00 (0.74s)	1.00 (0.28s)	1.00 (0.45s)	1.00 (0.30s)	0.94 (0.28s)	0.94 (0.48s)
betaSimplified	*reference model*	1.00 (0.31s)	1.00 (0.47s)	1.00 (0.47s)	0.77 (0.27s)	0.98 (0.42s)	0.94 (0.49s)	0.87 (4.55s)	1.00 (7.91s)	1.00 (0.45s)	1.00 (0.62s)	1.00 (0.48s)	0.70 (0.50s)	0.70 (0.75s)
choice	*reference model*	1.00 (0.44s)	1.00 (0.70s)	1.00 (0.36s)	0.57 (0.27s)	1.00 (0.60s)	1.00 (0.63s)	0.89 (19.76s)	1.00 (1.58s)	1.00 (0.61s)	1.00 (0.91s)	1.00 (0.63s)	1.00 (0.81s)	1.00 (1.24s)
complex	*reference model*	1.00 (3.20s)	1.00 (10.34s)	–	0.55 (1.52s)	0.75 (5.33s)	0.65 (3.08s)	–	0.73 (1963.23s)	0.73 (18.54s)	0.74 (25.58s)	1.00 (5.26s)	0.51 (17.94s)	0.51 (22.37s)
driversLicense	*reference model*	1.00 (0.09s)	1.00 (0.17s)	1.00 (0.11s)	0.63 (0.03s)	1.00 (0.13s)	0.90 (0.11s)	0.80 (1.30s)	1.00 (0.13s)	1.00 (0.17s)	1.00 (0.36s)	0.00 (0.09s)	0.80 (0.14s)	0.80 (0.28s)
driversLicenseLoop	*reference model*	1.00 (1.02s)	1.00 (1.65s)	0.91 (0.53s)	0.90 (0.41s)	0.95 (1.06s)	0.90 (0.84s)	–	0.87 (10.02s)	0.97 (1.78s)	0.97 (3.78s)	1.00 (1.22s)	0.83 (1.45s)	0.83 (2.96s)
herbstFig3p4	*reference model*	1.00 (0.50s)	1.00 (0.98s)	0.70 (0.28s)	0.99 (0.22s)	0.98 (0.44s)	0.97 (0.34s)	0.92 (77.55s)	0.96 (3.50s)	0.99 (1.12s)	0.99 (2.23s)	1.00 (0.41s)	0.94 (0.97s)	0.94 (1.69s)
herbstFig5p19	*reference model*	1.00 (0.33s)	1.00 (0.44s)	1.00 (0.48s)	0.88 (0.25s)	0.96 (0.42s)	1.00 (0.34s)	0.89 (5.31s)	1.00 (0.70s)	1.00 (0.47s)	1.00 (0.66s)	1.00 (0.38s)	0.77 (0.41s)	0.77 (0.81s)
herbstFig6p18	*reference model*	1.00 (0.95s)	1.00 (4.11s)	0.59 (0.55s)	0.40 (0.34s)	0.73 (2.28s)	0.93 (1.80s)	0.88 (1239.92s)	0.91 (7.23s)	0.97 (1.95s)	0.97 (3.79s)	1.00 (2.62s)	1.00 (2.29s)	1.00 (3.87s)
herbstFig6p31	*reference model*	1.00 (0.25s)	1.00 (0.56s)	1.00 (0.41s)	1.00 (0.25s)	1.00 (0.34s)	1.00 (0.38s)	0.78 (1.25s)	1.00 (0.72s)	1.00 (0.33s)	1.00 (0.53s)	1.00 (0.45s)	0.57 (0.42s)	0.57 (0.55s)
herbstFig6p36	*reference model*	1.00 (0.34s)	1.00 (0.64s)	1.00 (0.53s)	0.75 (0.36s)	1.00 (0.58s)	0.92 (0.84s)	0.92 (3.01s)	1.00 (3.31s)	1.00 (0.55s)	1.00 (0.59s)	1.00 (0.53s)	0.57 (0.53s)	0.57 (0.87s)
herbstFig6p38	*reference model*	1.00 (0.33s)	1.00 (0.51s)	1.00 (0.52s)	0.90 (0.27s)	1.00 (0.53s)	0.95 (0.44s)	0.89 (5.31s)	1.00 (0.81s)	1.00 (0.42s)	1.00 (0.69s)	1.00 (0.58s)	0.50 (0.48s)	0.50 (0.66s)
herbstFig6p41	*reference model*	1.00 (0.47s)	1.00 (0.89s)	1.00 (0.47s)	1.00 (0.39s)	1.00 (0.55s)	1.00 (0.58s)	0.92 (16.51s)	1.00 (11.96s)	1.00 (0.69s)	1.00 (0.95s)	1.00 (0.56s)	0.73 (0.80s)	0.73 (1.00s)
l2l	*reference model*	1.00 (0.30s)	1.00 (0.48s)	0.56 (0.28s)	1.00 (0.17s)	1.00 (0.33s)	1.00 (0.41s)	0.79 (8.15s)	1.00 (0.84s)	1.00 (0.39s)	1.00 (0.59s)	1.00 (0.37s)	1.00 (0.44s)	1.00 (0.67s)
l2lOptional	*reference model*	1.00 (0.31s)	1.00 (0.47s)	0.65 (0.31s)	0.40 (0.19s)	1.00 (0.37s)	1.00 (0.31s)	0.87 (7.04s)	1.00 (0.72s)	1.00 (0.42s)	1.00 (0.66s)	1.00 (0.45s)	1.00 (0.45s)	1.00 (0.70s)
l2lSkip	*reference model*	1.00 (0.30s)	1.00 (0.50s)	0.42 (0.28s)	0.85 (0.22s)	0.93 (0.36s)	1.00 (0.39s)	0.83 (6.37s)	1.00 (0.97s)	1.00 (0.42s)	1.00 (0.61s)	1.00 (0.41s)	1.00 (0.47s)	1.00 (0.64s)
exampleLog	*perfectModel*	1.00 (0.64s)	1.00 (0.69s)	1.00 (1.00s)	1.00 (0.36s)	1.00 (0.64s)	1.00 (0.52s)	0.78 (15.33s)	1.00 (2.91s)	1.00 (0.86s)	1.00 (1.05s)	1.00 (0.58s)	0.84 (0.62s)	0.84 (0.94s)
exampleLog	*connectedModel*	1.00 (0.83s)	1.00 (0.95s)	–	0.06 (0.36s)	0.26 (1.81s)	0.26 (1.08s)	0.27 (35.93s)	1.00 (3.02s)	0.13 (1.50s)	0.13 (1.84s)	1.00 (1.78s)	1.00 (1.42s)	1.00 (1.70s)
exampleLog	*flowerModel*	1.00 (0.37s)	1.00 (0.70s)	–	0.16 (0.42s)	0.26 (0.64s)	0.16 (0.48s)	0.26 (3.67s)	0.12 (2.66s)	0.12 (0.72s)	0.12 (0.98s)	1.00 (0.55s)	1.00 (0.62s)	1.00 (0.89s)
exampleLog	*singleModel*	0.38 (0.45s)	0.45 (0.64s)	1.00 (0.70s)	1.00 (0.34s)	1.00 (0.50s)	1.00 (0.59s)	1.00 (1.06s)	0.41 (2.41s)	0.71 (0.63s)	1.00 (0.87s)	1.00 (0.47s)	0.00 (0.59s)	0.00 (0.84s)
exampleLog	*stackedModel*	0.84 (0.86s)	1.00 (0.97s)	1.00 (14.31s)	0.12 (0.37s)	1.00 (1.11s)	1.00 (0.86s)	0.71 (123.51s)	0.71 (3.23s)	0.67 (1.19s)	1.00 (1.59s)	0.98 (0.94s)	0.33 (1.22s)	0.00 (1.40s)
hospital (real-life)	*mined model*	0.92 (5460.67s)	–	–	0.03 (4.68s)	0.13 (1829.74s)	0.03 (1234.47s)	0.04 (873.27s)	–	–	–	0.29 (1853.34s)	–	–
incident (real-life)	*mined model*	0.79 (13.56s)	0.41 (9374.49s)	–	0.05 (1.65s)	–	–	–	0.68 (88.39s)	0.41 (140.83s)	0.94 (5885.06s)	–	0.52 (181.39s)	0.54 (5709.55s)
telecom (real-life)	*mined model*	0.70 (15.70s)	0.39 (16523.82s)	–	0.10 (1.53s)	–	0.47 (17439.21s)	–	0.47 (86385.41s)	0.45 (566.04s)	0.88 (18236.50s)	–	0.41 (557.31s)	0.46 (18215.54s)
ticketing (real-life)	*mined model*	1.00 (503.24s)	1.00 (701.53s)	–	0.13 (128.31s)	0.81 (15746.07s)	0.99 (15374.66s)	–	1.00 (44834.75s)	0.97 (455.13s)	0.97 (693.85s)	1.00 (20957.02s)	0.81 (429.81s)	0.81 (672.68s)

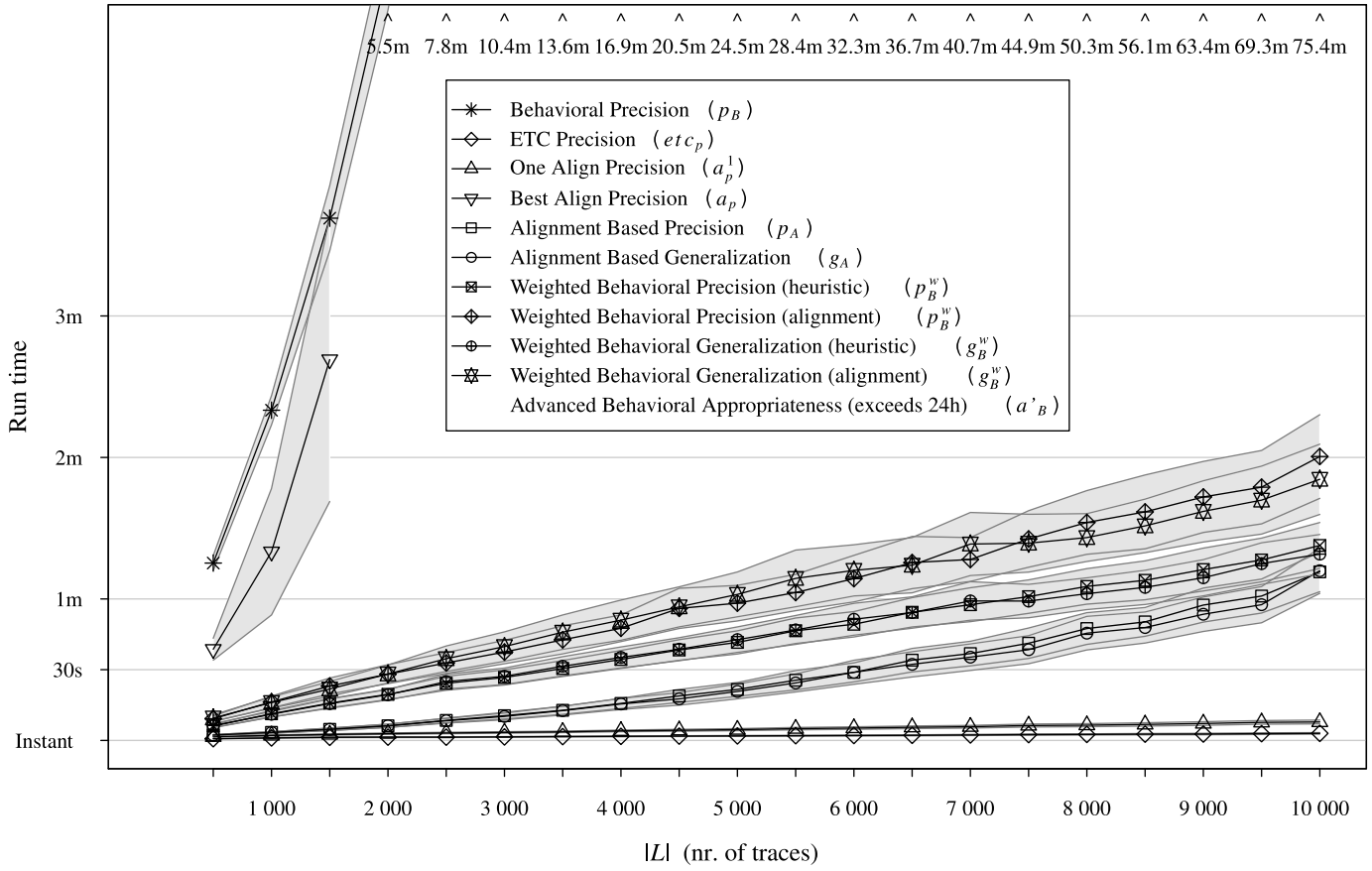


Fig. 4. Run times (averaged over twenty iterations) of evaluated conformance checking techniques over various log sizes. The grey bands indicate the 95% confidence intervals below and above the means.

which is performed by these metrics. Also, calculating *Best Align Precision* (a_p) requires exponentially more time as the log size is increased. In [23], the authors indeed acknowledge the long calculation time required towards calculating this metric. In addition, for logs containing 2 000 or more traces, the metric was not able to obtain any result due to out-of-memory errors. Concerning the calculation of the artificial negative event based metrics (*Behavioral Precision* p_B, *Weighted Behavioral Precision* p_B^w and *Weighted Behavioral Generalization* g_B^w), it is important to note that Fig. 4 includes the time taken to induce the negative events in the reported run time (note the exponential time increase for *Behavioral Precision*). Fig. 5 shows the run time required for the generation of all artificial negative events separately and illustrates the scalability benefit gained by our implementation using suffix trees to induce the weighted artificial negative events and shows linear complexity in terms of log size, as was discussed above.

Concerning stability, we evaluate whether the obtained precision and generalization values are sensitive to various levels of log completeness, as well as investigate the actual score values themselves. Fig. 6 depicts an overview of the retrieved metric values. The following remarks can be made based on the results. First, the *Weighted Behavioral Precision* metric (p_B^w) improves slightly upon the non-weighted *Behavioral Precision* (p_B) and *Alignment Based Precision* (p_A) metrics and that it highly exceeds the *ETC Precision* (etc_p) and *One Align Precision* (a_p^1) metrics (recall that, since we are dealing with simulated event logs from a reference model, we expect precision to be high as the evaluated log reaches a sufficient size). On the other hand, the *Best Align Precision* metric (a_p) outperforms our approach, although this metric comes with high run times and is less able to deal with larger event logs. We were not able to confirm the statement made in [23] regarding the similarity in results between *One Align Precision* (a_p^1) and *Best Align Precision* (a_p), since there exists a wide gap between the results for these two metrics. Finally, although the *Advanced Behavioral Appropriateness* metric (a'_B) results in the highest precision value, the result is only approximate (the state space exploration phase was canceled after 24h). Moreover, this metric only investigates basic structural relations between model and log activities, so that this metric as well presents a tendency to be overly optimistic. Concerning generalization, the *Weighted Behavioral Generalization* metric (g_B^w) applies the concept of weighted artificial negative events towards checking generalization, which takes into account full behavioral properties found in both process model and event log. Although the metric remains very stable over various log

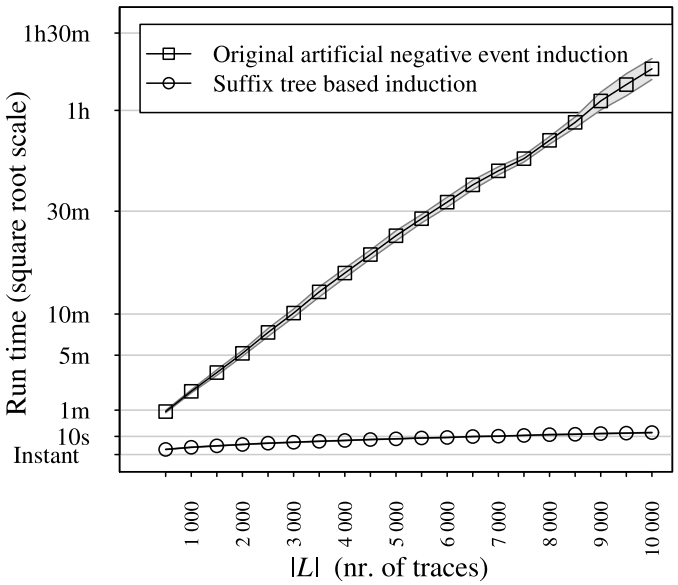


Fig. 5. Run times (averaged over twenty iterations) of (weighted) artificial negative event procedure over various log sizes, illustrating the speed benefit obtained by our technique compared to the original artificial negative induction method. The grey bands indicate the 95% confidence intervals below and above the means.

sizes (the slight drop at the beginning is due to the fact that the activity alphabet is incomplete for the very small logs), it is noted that this metric estimates generalization in a very strict manner, especially when compared with the *Alignment Based Generalization* metric (a_g). However, as seen in Subsection 5.2, this probabilistic metric also returns very high values for the experiment models with a very low generalization capability, i.e. *singleModel* and *stackedModel*, whereas *Weighted Behavioral Generalization* (g_B^w) does punish these models.

6 Conclusion & Future Work

In this paper, a novel conformance checking approach was put forward to determine how well a process model performs in terms of precision and generalization. The validity of the approach was illustrated by evaluating the proposed artificial negative event weighting method and by performing an experimental setup to benchmark our metrics in comparison with related techniques.

Our approach offers the following contributions. First, we have outlined an improved strategy for artificial negative event induction, extending it with a weighting method in order to indicate the confidence given to a candidate negative event, which helps to deal with incomplete logs, and implementing it in a manner which greatly improves the scalability of the technique. Second, these weighted artificial negative events form the basis of two novel metrics for precision and generalization. The introduction of a new generalization metric is an important contribution, due to the fact that the metric does not rely on a probabilistic estimator. Third, it was illustrated how these metrics can be applied on Petri net models and we have implemented both a heuristic and log-model alignment based approach in order to perform trace replay. All described techniques and metrics are made available in a number of ProM plugins, being the first implementation to provide a full conformance checking framework using negative events.

In closing, a number of possible improvements are put forward as the focus for future work. Although the *Weighted Behavioral Generalization* (g_B^w) metric provides a solid first step towards determining a process model's ability to generalize, it was found that the metric performs a rather strict evaluation. This is mainly due to the fact that, currently, the usefulness of a negative event towards generalization is direct inversely related to its use towards precision. Thus, the same weighted negative event can be used towards assessing both precision and generalization. It would be reasonable to constrain this behavior in future additions. Finally, the current way of calculating an artificial negative event's weight ignores the number of times a matching prefix window was encountered in the event log. More specifically, a single trace in the event log with a matching window can lead to a drop in weighting for an artificial negative event. In scenarios where noise is assumed to be absent, this poses no issue (as seen in the experiments). It is possible to take the trace frequencies into account during the construction of the suffix tree (thus retaining the scalability benefits), although modifying the weighting scheme or metrics to incorporate this information in a robust manner is not a simple task, as some event logs might – by nature – be skewed in terms of trace frequency distribution. Note also that trace frequencies are already taken into account in the calculation of the metric values themselves, i.e. a non-conforming trace with a higher frequency will have a larger impact than a low frequent one. We plan to investigate this aspect in future work.

Acknowledgment

We would like to thank the KU Leuven research council for financial support under grand OT/10/010 and the Flemish Research Council for financial support under Odysseus grant B.0915.09. This work is also supported by an ARC Discovery grant number: DP120101624.

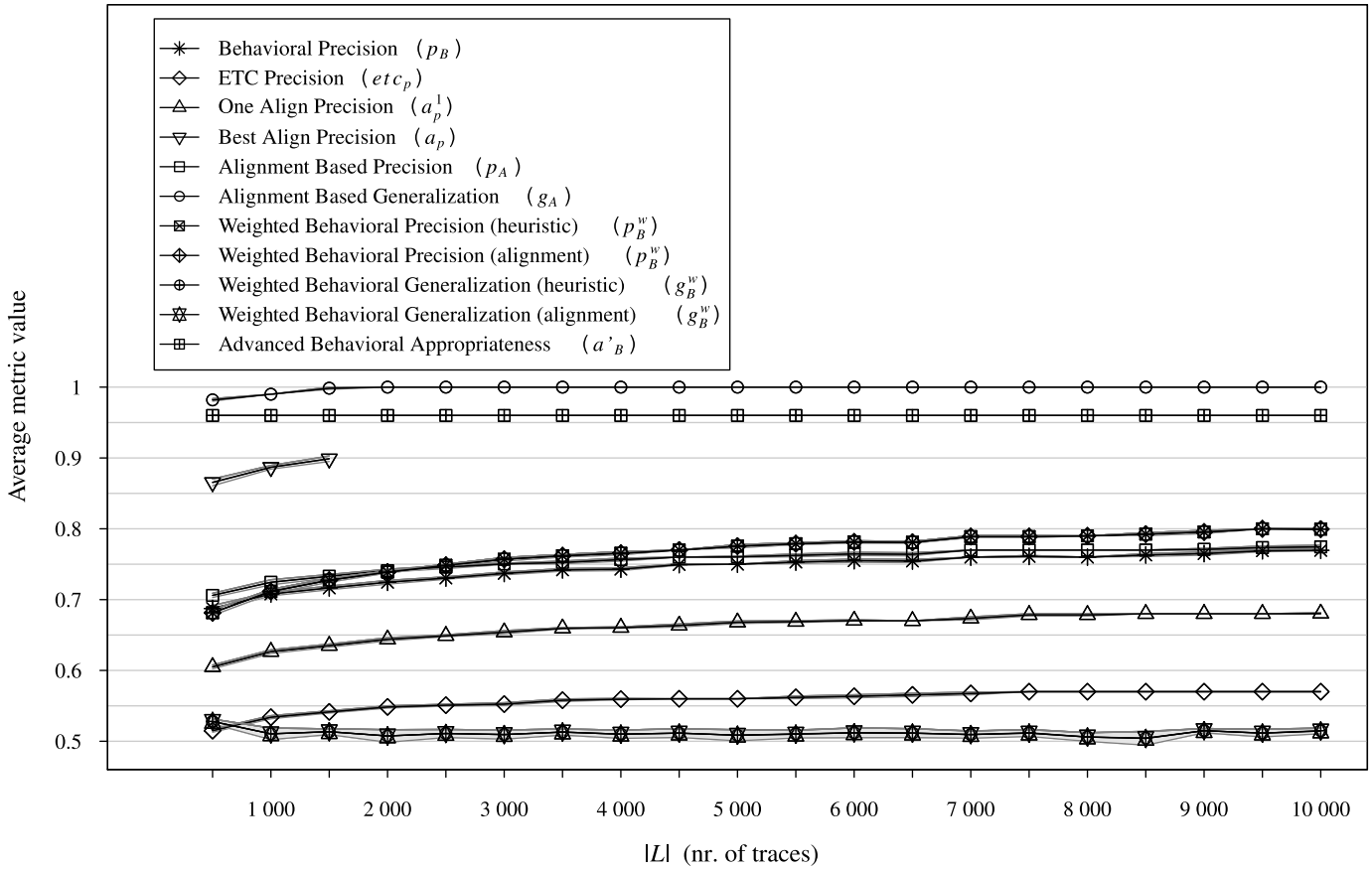


Fig. 6. Results of evaluated conformance checking techniques (averaged over twenty iterations) over various log sizes. The grey bands (narrow) indicate the 95% confidence intervals below and above the means.

REFERENCES

[1] J. Cook and A. Wolf, "Discovering models of software processes from event-based data," *ACM Transactions on Software Engineering and Methodology*, vol. 7, pp. 215–249, 1998.

[2] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," in *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98), volume 1377 of Lecture Notes in Computer Science*, pp. 467–483, Springer, 1998.

[3] A. Datta, "Automating the Discovery of AS-IS Business Process Models: Probabilistic and Algorithmic Approaches," *Information Systems Research*, vol. 9, pp. 275–301, 1998.

[4] W. van der Aalst, A. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, pp. 1128–1142, 2004.

[5] W. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Verlag, 2011.

[6] A. Rozinat and W. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Information Systems*, vol. 33, pp. 64–95, 2008.

[7] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs," *Information Systems*, vol. 37, pp. 654–676, 2012.

[8] A. Rozinat, M. Veloso, and W. van der Aalst., "Evaluating the quality of discovered process models," in *Proceedings of IPM 2008 induction of process models (ECML PKDD 2008)*, pp. 45–52, 2008.

[9] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, "Robust Process Discovery with Artificial Negative Events," *Journal of Machine Learning Research*, vol. 10, pp. 1305–1340, 2009.

[10] S. vanden Broucke, J. De Weerdt, B. Baesens, and J. Vanthienen, "Improved Artificial Negative Event Generation to Enhance Process Event Logs," in *24th International Conference on Advanced Information Systems Engineering (CAiSE'12), volume 7328 of Lecture Notes in Computer Science*, pp. 254–269, 2012.

[11] W. van der Aalst and A. Adriansyah, "Replaying history on process models for conformance checking and performance analysis," *WIREs Data Mining and Knowledge Discovery*, vol. 2, pp. 1–18, 2012.

[12] T. Murata, "Petri nets: Properties, analysis and applications," in *Proceedings of the IEEE*, vol. 77, pp. 541–580, 1989.

[13] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Prentice Hall, 1981.
[14] W. van der Aalst, A. Adriansyah, and B. van Dongen, "Causal nets: a modeling language tailored towards process discovery," in *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR 2011)*, pp. 28–42, 2011.
[15] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A robust F-measure for evaluating discovered process models," in *IEEE Symposium Series in Computational Intelligence 2011 (SSCI 2011)*, pp. 148–155, 2011.
[16] H. Yang, B. van Dongen, A. ter Hofstede, M. Wynn, and J. Wang, "Estimating Completeness of Event Logs," *BPM Center Report, 12-04-2012*, 2012.
[17] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, pp. 249–260, 1995.
[18] A. Alves de Medeiros, A. Weijters, and W. van der Aalst, "Genetic process mining: an experimental evaluation," *Data Mining and Knowledge Discovery*, vol. 14, pp. 245–304, 2007.
[19] A. Adriansyah and B. F. van Dongen, "Conformance checking using cost-based fitness analysis," in *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2011)*, pp. 55–64, 2011.
[20] A. Adriansyah, B. van Dongen, and W. van der Aalst, "Towards Robust Conformance Checking," in *BPM 2010 International Workshops and Education Track (BPM 2010), volume 66 of Lecture Notes in Business Information Processing*, pp. 122–133, 2011.
[21] S. vanden Broucke, J. De Weerdt, J. Vanthienen, and B. Baesens, "On replaying process execution traces containing positive and negative events," KU Leuven KBI technical report KBI-1311, 2013.
[22] J. Muñoz-Gama and J. Carmona, "Enhancing precision in Process Conformance: Stability, confidence and severity," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pp. 184–191, 2011.
[23] A. Adriansyah, J. Muñoz-Gama, and J. Carmona, "Alignment Based Precision Checking," *BPM Center Report, 12-10-2012*, 2012.
[24] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst, "Alignment based precision checking," in *Business Process Management Workshops* (M. L. Rosa and P. Soffer, eds.), vol. 132 of *Lecture Notes in Business Information Processing*, pp. 137–149, Springer, 2012.
[25] M. Weidlich, A. Polyvyanyy, N. Desai, J. Mendling, and M. Weske, "Process compliance analysis based on behavioural profiles," *Information Systems*, vol. 36, no. 7, pp. 1009–1025, 2011.
[26] G. Greco, A. Guzzo, L. Ponieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, pp. 1010–1027, 2006.
[27] A. Rozinat, *Process Mining: Conformance and Extension*. PhD thesis, TU Eindhoven, 2010.
[28] A. Weijters, W. van der Aalst, and A. Alves de Medeiros, "Process mining with the heuristics miner-algorithm," BETA Working Paper Series WP 166, Eindhoven University of Technology, 2006.

Seppe vanden Broucke received an MSc in Business Economics: Information Systems Engineering from KU Leuven, Belgium. He is currently employed as a PhD Candidate at the Department of Decision Sciences and Information Management at KU Leuven, and is working in such areas as business process mining and management, data mining and event sequence analysis.

Jochen De Weerdt is a Research Fellow at the Information Systems School of the Queensland University of Technology, Australia. He holds a PhD in Business Economics from KU Leuven. His research interests include business process management, process mining, data mining, artificial intelligence, and web analytics.

Jan Vanthienen received his PhD degree in Applied Economics from KU Leuven, Belgium. He is a Full Professor of Information Systems with the Department of Decision Sciences and Information Management, KU Leuven. He is the author or co-author of numerous papers published in international journals and conference proceedings. His current research interests include information and knowledge management, business intelligence and business rules, and information systems analysis and design.

Bart Baesens is an associate professor at KU Leuven, Belgium, and a lecturer at the University of Southampton, United Kingdom. He has done extensive research on predictive analytics, data mining, web analytics, fraud detection, and credit risk management. His findings have been published in well-known international journals (e.g. Machine Learning, IEEE Transactions on Neural Networks, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Evolutionary Computation, etc.).