Conformance Checking of Partially Matching Processes: An Entropy-Based Approach

Artem Polyvyanyy, Anna Kalenkova

School of Computing and Information Systems
The University of Melbourne, Parkville, VIC, 3010, Australia

Abstract

Conformance checking is an area of process mining that studies methods for measuring and characterizing commonalities and discrepancies between processes recorded in event logs of IT-systems and designed processes, either captured in explicit process models or implicitly induced by information systems. Applications of conformance checking range from measuring the quality of models automatically discovered from event logs, via regulatory process compliance, to automated process enhancement. Recently, process mining researchers initiated a discussion on the desired properties the conformance measures should possess. This discussion acknowledges that existing measures often do not satisfy the desired properties. Besides, there is a lack of understanding by the process mining community of the desired properties for conformance measures that address partially matching processes, i.e., processes that are not identical but differ in some process steps. In this article, we extend the recently introduced precision and recall conformance measures between an event log and process model that are based on the concept of entropy from information theory to account for partially matching processes. We discuss the properties the presented extended measures inherit from the original measures as well as properties for partially matching processes the new measures satisfy. All the presented conformance measures have been implemented in a publicly available tool. We present qualitative and quantitative evaluations based on our implementation that show the feasibility of using the proposed measures in industrial settings.

Keywords: Process mining, conformance checking, partial matching, properties, entropy

1. Introduction

The research discipline of *process mining* combines elements of data-centric disciplines, like data mining and machine learning, with process modeling and

URL: `http://polyvyanyy.com/` (Artem Polyvyanyy), `https://cis.unimelb.edu.au/people/anna-kalenkova` (Anna Kalenkova) analysis to tackle the problems of discovering, monitoring, and improving real-world processes [1]. To this end, process mining relies on the data on the past process executions. One of the core problems studied in process mining is *conformance checking*. Conformance checking studies methods, techniques, and tools that characterize and quantify commonalities and discrepancies between *designed processes*, e.g., (business) process models, and records of *observed processes*, e.g., event logs of IT-systems [2, 3, 4, 5, 6]. The core application of conformance checking is to measure and explain the "goodness" of the designed processes with respect to those observed in the real-world. The obtained information can be used, for example, in the context of automated process enhancement [7, 8], regulatory process compliance [9], and assessment of the quality of models automatically discovered from event logs [10].

Two main measures in conformance checking are *precision* and *recall*.[1] Given an event log and process model, precision quantifies the relation between the process information shared by the log and model and the process information encoded in the model, while recall quantifies the relation between the shared information and the information recorded in the log. Precision and recall are usually defined to take values between, and including, zero and one. The larger the precision and recall values, the more commonalities and fewer discrepancies, as per the applied technique to compute the measurements, exist between the log traces and the traces described by the model. The precision and recall values of zero represent a situation of no commonalities between the traces of the log and model, while the values of one represent the situation of no discrepancies between the traces.

Precision and recall are well-known performance measures for information retrieval systems [11]. Given a collection of documents that are *relevant* to an information query and a collection of documents *retrieved* by an information retrieval system using the same query, precision is the fraction of the number of relevant retrieved documents over the number of retrieved documents. In contrast, recall is the fraction of the number of relevant retrieved documents over the number of relevant documents. Taking the analogy of information retrieval, in conformance checking, one can address traces captured in an event log and model as *relevant* and *retrieved* units of information, respectively. Indeed, automated *process discovery* techniques [12, 13, 14, 15, 16, 17, 18] aim to construct a process model that describes traces of interest from an input event log as closely as possible. Using this interpretation, the precision of the discovered process model with respect to the input log is the fraction of the number of distinct traces from the log described in the model over the total number of distinct traces described by the model, while recall is the fraction of the number of distinct traces from the log described in the model over the number of distinct traces recorded in the log. Hence, a process discovery technique "retrieves" traces by including them in the resulting process model while aiming to reflect the relevant traces from the input event log.

[1] Recall is also often referred to as *fitness*.

The proposed above precision and recall measures for conformance checking face at least one major limitation. Unlike in information retrieval, where both collections of relevant and retrieved documents are finite, the collection of traces encoded in a (discovered) process model is often infinite. This makes it difficult to define conformance measures with the desired properties, e.g., *determinism* and *monotonicity*, as it is not immediate to define how to measure infinite collections of traces. In [4], we overcame this challenge by proposing process conformance measures of precision and recall based on the notion of *topological entropy* over regular languages [19]. The entropy-based precision and recall are deterministic and monotonic measures [4]. However, they are limited in their ability to compare event logs that contain similar, non-identical traces, e.g., traces that differ in some process steps. For instance, if every trace in an event log differs from every trace described by a model in at least one process step, the precision and recall values computed based on the notion of topological entropy proposed in [4] both equal to zero, just like in the situation when the compared traces are entirely different. In this article, we overcome this limitation. We achieve this by "diluting" the traces captured in the compared event log and process model and then comparing the diluted traces. The dilution of a trace results in the inclusion of all sub-traces of the trace into the collection of traces captured in the corresponding event log or process model. The new measures inherit most of the properties of the original measures and, in addition, possess intuitively desired properties that address the comparison of the partially matching traces. For example, the more (in the number of traces) and the longer (in the number of matched process steps) are the partial matches between the traces captured in the compared event log and process model, the larger the precision and recall values are.

This article is an extended version of our conference paper [20]. The original conference paper made these contributions:

- Extended the entropy-based precision and recall measures grounded in the exact matching of traces [4] with support of the partial matching between the traces by comparing their sub-traces;
- Analyzed inheritance of properties of the exact entropy-based measures [4] by the proposed partial matching measures;
- Demonstrated further properties of the exact matching measures that can also be applied for partial matching measures;
- Presented results of qualitative and quantitative evaluations of the extended entropy-based precision and recall measures that demonstrated their usefulness and applicability in industrial settings.

This article makes these extensions to the original conference paper:

- Introduces several optimizations of the original partial matching algorithm. The introduced optimizations aim at achieving lower memory usage by performing event log decompositions;
- Presents an improvement of the algorithms for computing the precision and recall measures that exploits the sub-trace relationships between the diluted traces;

- Presents and discusses results of an extensive evaluation of computing the precision and recall using all the devised algorithms on the real-world datasets;
- Extends the discussions of the core contributions, conducted evaluations of our work, and its usefulness and practical relevance.

The rest of the article is structured as follows. The next section discusses related work. Section 3 presents a motivating example. Section 4 introduces the basic notions from process mining and automata theory required to understand the subsequent discussions. Section 5 presents the entropy-based precision and recall measures from [4], whereas Section 6 extends them with partial matching support. Section 7 presents and discusses implementation details of several versions of our core algorithm for computing the entropy-based precision and recall measures from Section 6. Section 8 presents the results of our evaluation. Finally, Section 9 states concluding remarks and discusses future work.

2. Related Work

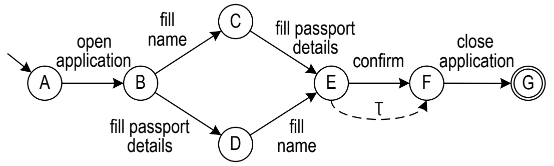
Over the past decade, a plethora of conformance checking methods have been developed and proven useful in analyzing real-world process data [3]. These methods vary in the types of process models and event logs they support and the types of results they produce. Conformance checking techniques usually either produce a single number assessing the behavioral similarity of process models and event logs (*quantitative* conformance checking) or provide rich diagnostic information highlighting deviations in the model and log behaviors (*qualitative* conformance checking). In this paper, we present and study a quantitative conformance checking technique.

Some prominent examples of conformance checking techniques include *k-order Markovian abstractions* [21], *Projected conformance checking* [22], *Anti-alignments* [23], *Escaping edges* [24], *Set difference* [25], *Negative events* [26], and *Entropy-based techniques* [4, 20, 27]. Recently, quantitative stochastic conformance checking approaches that account for model and log trace probabilities have been proposed [5, 28, 29]. Although these approaches produce a single number, they can often be supported by visual analytics to provide qualitative conformance information. For instance, the *Projected conformance checking* and the Escaping edges techniques represent analyzed behaviors in the form of finite automata models that are compared based on their commonalities and deviations. Similarly, the *Entropy-based techniques*, which are also based on automata theory, can be complemented by qualitative techniques that visualize commonalities and discrepancies in the automata models. We consider this as a direction for future work. Existing methods that combine quantitative and qualitative conformance checking techniques visualize conformance analytics on process models and are usually based on the concept of *alignment*, a *token replay* technique, or *footprint matrices*, refer to [30], [31], and [1] for details.

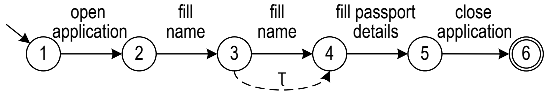
In [6], the authors study properties that the state-of-the-art quantitative conformance techniques fulfill. The entropy-based exact matching precision and recall measures [4] are shown to possess all the studied properties. Besides, as shown in [4], the entropy-based exact matching precision measure fulfills all the axioms proposed in [32]. However, as shown in the next section, analysis founded on the exact matching of model and log trace can often be seen as restrictive. In this article, we present entropy-based precision and recall measures that account for the partial matching of the compared traces and study them from theoretical and practical perspectives. We show that the partial entropy-based precision fulfills generalized versions of the properties from [32]. Additionally, we prove that the exact matching approach fulfills the strict monotonicity version of the precision axiom from [32]. Finally, we elaborate on the monotonicity properties of the proposed partial matching measures distinguishing the cases when they inherit the strict monotonicity properties from the exact matching approach. We support the explanations of the inherited and the new properties with examples.

3. Motivating Example

Consider a simple process of booking a flight ticket captured as a finite automaton shown in Fig. 1a; ignore the dashed arrow in the figure. The process starts when the user opens a booking application. Then, she fills in her name and passport details (in any order). Finally, the booking is confirmed, and the user closes the application. Suppose that the user behavior in the real-world can deviate from the one specified in the reference model. For example, a user can fill in the name twice and close the booking application without confirmation. This sequence of steps is captured in the automaton shown in Fig. 1b; again, ignore the dashed arrow. Even though the two automata presented in Fig. 1 describe similar traces, precision and recall measures that rely on exact comparisons of traces cannot appraise this similarity (i.e., the corresponding values of precision and recall equal to zero).



(a)



(b)

Figure 1: Two finite automata before (solid arrows only) and after (with dashed arrows) inserting silent steps.

To address this limitation, one can augment the automata by “mirroring” certain *observable* steps, i.e., steps that represent meaningful activities, with *silent* (τ) steps. As a result, the augmented automata should describe fresh traces that capture the sub-sequences of observable steps shared by the similar traces encoded in the original automata. For instance, if one mirrors step *confirm* in the automaton in Fig. 1a and one step *fill name* (any of the two) in the automaton in Fig. 1b with silent steps, cf. the dashed arrows in Fig. 1, then both resulting automata describe the observable trace $\langle$*open application*, *fill name*, *fill passport details*, *close application*$\rangle$; an observable trace is obtained from a trace by removing all its silent steps without changing the order of the remaining steps. The fresh trace in the augmented automata explicitly encodes the commonalities, while the original traces that induced the fresh trace encode the discrepancies between the processes captured in the automata. This idea of identifying partially matching traces through their "dilution" with silent steps lies at the heart of the solution proposed in the work at hand. Finally, note that after the insertion of the silent step in the automaton in Fig. 1a, the resulting automaton describes two fresh traces. In the subsequent sections, we will explain how we exploit this phenomenon for the benefit of identifying all the commonalities between the traces of two given automata.

According to the conformance measures proposed in this article, precision and recall values between the original automata (without dashed arrows) in Fig. 1 are 0.859 and 0.964, respectively. If one inserts step *confirm* between *fill passport details* and *close application* in the automaton in Fig. 1b, i.e., makes the traces it describes more similar to the traces captured by the automaton in Fig. 1a, the values of precision and recall increase to 0.978 and 0.973, respectively, reflecting the monotonic properties of our measures.

4. Preliminaries

This section introduces notions and notations necessary to support the subsequent discussions.

4.1. Sequences, Languages, and Event Logs

Let X be a set of elements. The *powerset* over set X, denoted by $\mathcal{P}(X)$, is the set of all subsets of X including the empty set and X.

By $\langle x_1, x_2, \ldots, x_k \rangle$, where $x_1, x_2, \ldots, x_k \in X$, $k \in \mathbb{N}_0$, we denote a *sequence* of elements over X of length k. The *empty sequence* of zero length is denoted by $\langle\rangle$. Given two sequences $x = \langle x_1, x_2, \ldots, x_k \rangle$ and $y = \langle y_1, y_2, \ldots, y_m \rangle$, by $x \cdot y$ we denote the concatenation of x and y, i.e., the sequence obtained by appending y to the end of x; it holds that $x \cdot y = \langle x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_m \rangle$. Let us define the notion of *sub-sequence* (*sub-word*) recursively: (1) if sequence x is empty, then the empty sequence is a sub-sequence of x, (2) if $x = \langle x_1 \rangle \cdot \alpha$, then $\langle x_1 \rangle \cdot \hat{\alpha}$ and $\hat{\alpha}$, where $\hat{\alpha}$ is a sub-sequence of α, are sub-sequences of x. As follows from this definition, a sub-sequence can be obtained from a sequence by skipping some of the elements' occurrences. Let us consider sequence $\langle a, a, b \rangle$, the set of all its sub-sequences is $\{\langle\rangle, \langle a \rangle, \langle b \rangle, \langle a, a \rangle, \langle a, b \rangle, \langle a, a, b \rangle\}$. X^* stands for the set of all finite sequences over X including the empty sequence. Given a sequence x and a set K, by $x|_K$, we denote a sequence obtained from x by removing all elements of x that are not members of K without changing the order of the remaining elements, e.g., it holds that $\langle 1,3,2,1,4,3\rangle|_{\{4,1\}} = \langle 1,1,4\rangle$.

An *alphabet* is a nonempty finite set. The elements of an alphabet are its *labels*. A (formal) *language* L over an alphabet Σ is a (not necessarily finite) set of *sequences*, or *words*, over Σ, i.e., $L \subseteq \Sigma^*$. By $C_n(L)$, $n \in \mathbb{N}$, we denote the set of all words in L of length n. By Ξ, we denote a universe of all *observable* labels. By τ, we denote a special *silent* label, such that $\tau \notin \Xi$. Let L_1 and L_2 be two languages. Then, $L_1 \circ L_2$ is their concatenation defined by $\{l_1 \cdot l_2 \,|\, l_1 \in L_1 \wedge l_2 \in L_2\}$.

Let $\mathcal{E}$ be a finite nonempty set of *events*. A finite language $L \subseteq \mathcal{E}^*$ is an *event log*, and its words are called *traces*. A sub-word of a trace is called its *sub-trace*. In process mining, traces are used to encode executions of business processes, with events representing occurrences of activities. The ordering of events in a trace then encodes the temporal order relation, i.e., for two events at two different positions in a trace, the one at the smaller position has occurred before the other event.

4.2. Finite Automata

We deal with a common notion of a finite automaton [33]. A *nondeterministic finite automaton* (NFA) is a 5-tuple $(Q, \Lambda, \delta, q_0, A)$, where Q is a finite nonempty set of *states*, $\Lambda \subseteq \Xi$ is a set of *labels*, such that Q and Ξ are disjoint, $\delta : Q \times (\Lambda \cup \{\tau\}) \to \mathcal{P}(Q)$ is the *transition function*, where $\tau \notin Q \cup \Lambda$, $q_0 \in Q$ is the *start state*, and $A \subseteq Q$ is the *set of accept states*.

An NFA induces a collection of computations. A *computation* of an NFA $(Q, \Lambda, \delta, q_0, A)$ is either the empty word or a word $\sigma = \langle a_1, a_2, \ldots, a_n\rangle$, $n \in \mathbb{N}$, where every a_i is a member of $\Lambda \cup \{\tau\}$, $i \in [1\,..\,n]$, and there exists a sequence of states $\langle q_0, q_1, \ldots, q_n\rangle$, where every q_j is a member of the set of states Q, $j \in [1\,..\,n]$, such that for every $k \in [1\,..\,n]$ it holds that $q_k \in \delta(q_{k-1}, a_k)$.

We say that σ *leads to* q_n. By convention, the empty sequence always leads to the start state. An NFA *accepts* a word σ iff σ is a computation of this NFA that leads to one of its accept states. The *language* of an NFA $B = (Q, \Lambda, \delta, q_0, A)$ is denoted by *lang*(B) and is the set $\{\sigma \in \Lambda^* \,|\, \exists\, \rho \in (\Lambda \cup \{\tau\})^* : ((B \textit{ accepts } \rho) \wedge (\sigma = \rho|_\Lambda))\}$, i.e., the set of all words B accepts with all the silent labels removed. We also say that B *recognizes lang*(B).

A *deterministic finite automaton* (DFA) is an NFA $(Q, \Lambda, \delta, q_0, A)$ such that for every state $q \in Q$ it holds that $\delta(q, \tau) = \varnothing$ and for every state $q \in Q$ and for every label $a \in \Lambda$ it holds that $|\delta(q, a)| \leq 1$.

A DFA $(Q, \Lambda, \delta, q_0, A)$ is *ergodic* if its underlying graph is strongly irreducible, i.e., for all $(q, p) \in Q \times Q$ there is a sequence of states $\langle q_1, \ldots, q_n\rangle \in Q^*$, $n \in \mathbb{N}$, such that $q_1 = q$, $q_n = p$, and for every $k \in [1\,..\,n-1]$ there exists $\lambda \in \Lambda$ such that $q_{k+1} \in \delta(q_k, \lambda)$.

A language $L \subseteq \Sigma^*$ is *regular* iff there exists an NFA that recognizes L. L is *irreducible* if, given two words w_1 and w_2 in L, there exists a word $w \in \Sigma^*$, such that $w_1 \cdot w \cdot w_2 \in L$. A regular language L is irreducible iff it is a language of an ergodic DFA [19].

5. Entropy-Based Conformance Checking: Exact Matching

In this article, we consider process models that describe collections of traces that can be recognized by DFAs. These can be Petri nets or BPMN models that induce finite reachability graphs, or NFAs, which can always be converted into equivalent DFAs [33]. Note that state-of-the-art process discovery algorithms often construct process models that induce finite reachability graphs [34, 18]. As a log induces a finite language, it can, as well, be encoded as a DFA. To compute precision and recall between a given process model and event log, we compare the languages the corresponding DFAs recognize. Specifically, we compute precision (recall) as the ratio of the entropy that estimates the number of distinct traces the model and log share to the entropy that estimates the number of distinct traces captured in the model (log).

Next, in Section 5.1, we present the notion of topological entropy [19]. Then, in Section 5.2, we use topological entropy to define a notion of the *short-circuit entropy* of a regular language. Finally, in Section 5.3, we present the entropy-based precision and recall between a process model and an event log.

5.1. Topological Entropy

The cardinality of a finite language $L \subseteq \Sigma^*$ is defined as the number of words in L, i.e., $|L|$. This definition can be used, for example, to decide which language contains more words. However, it is not particularly useful when it comes to comparing infinite languages. Indeed, it is arguable whether a whole and its part should be considered to be the same. Assuming they should not, in certain cases, for irreducible languages, one can use the notion of *topological entropy to make the distinction.* Given an irreducible language L, its topological entropy is defined as follows [19]:

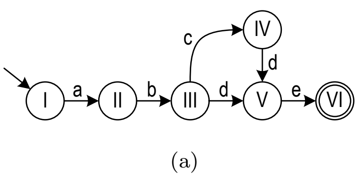
$$ent(L) = \limsup_{n \to \infty} \frac{\log |C_n(L)|}{n}.$$

Therefore, topological entropy of L estimates cardinality of L by measuring the ratio of the number of distinct words in the language to the length of these words, as the length of words approaches infinity. One can compute topological entropy of an irreducible regular language L as the logarithm of the Perron-Frobenius eigenvalue of the adjacency matrix of a DFA that recognizes L [19].

5.2. Short-Circuit Entropy

As topological entropy is defined over irreducible languages, it is not applicable for infinite non-irreducible languages or finite languages, e.g., event logs. To allow estimating cardinality of an arbitrary regular language, in [4], we proposed the notion of a *short-circuit measure* over languages.

Given a measure over languages $m : \mathcal{L} \to \mathbb{R}_0^+$, where $\mathcal{L} \subseteq \mathcal{P}(\Sigma^*)$ and $\Sigma \subset \Xi$, its *short-circuit* version $m\bullet$ is defined as $m\bullet(L) = m((L \circ \{\langle\chi\rangle\})^* \circ L)$, where $L \subseteq \Sigma^*$ and $\chi \in \Xi \setminus \Sigma$. For a given language L, it holds that $(L \circ \{\langle\chi\rangle\})^* \circ L$ is irreducible [4]. Hence, if $\mathcal{L}$ is a family of all regular languages, then $ent\bullet(L)$, $L \in$



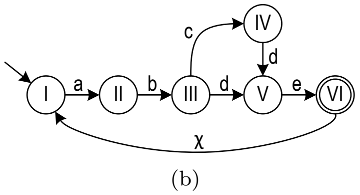


Figure 2: Two finite automata; (b) is ergodic.

$\mathcal{L}$, is well-defined and can be computed. To compute *short-circuit (topological) entropy* of a regular language L, i.e., $ent\bullet(L)$, one can transform a DFA that recognizes L by inserting additional transitions – marked by a special label χ not occurring in the words of L – that connect all the accept states of the DFA with its start state and then compute topological entropy of the language recognized by the resulting ergodic automaton; refer to [4] for details.

We assume that every state of the DFA used in the above-presented computation procedure is on a directed path from the start state to some accept state. Note that given a DFA that does not satisfy this requirement, one can remove all the states (and transitions incident with these states) that are not on a directed path from the start to some accept state, to obtain a DFA that satisfies the requirement and recognizes the language of the original DFA.

For example, $ent\bullet(L)$, where L is the language recognized by the DFA in Fig. 2a, can be computed as topological entropy of the language recognized by the ergodic DFA in Fig. 2b.

It holds that $ent\bullet(\varnothing) = 0$. Moreover, for two regular languages L_1 and L_2, such that $L_1 \subset L_2$, it holds that $ent\bullet(L_1) < ent\bullet(L_2)$, i.e., $ent\bullet$ is monotonic on the partially ordered set of regular languages induces by the subset relation; again, refer to [4] for details.

Next, we show two additional properties of the short-circuit entropy measure over regular languages. In Section 5.3, we use these properties to establish several new properties of the entropy-based precision and recall measures.

First, if for every length it holds that one regular language has no more words of that length than the other regular language, then the short-circuit entropy of the former language is less than or equal to the short-circuit entropy of the latter language.

Theorem 5.1 (Monotonicity on the number of words).
Let L_1 and L_2 be two regular languages such that $\forall\, k \in \mathbb{N} : |C_k(L_1)| \leq |C_k(L_2)|$. Then, it holds that $ent\bullet(L_1) \leq ent\bullet(L_2)$. ⌟

Proof. Let $x_n = \frac{\log|C_n(L_1)|}{n}$ and $y_n = \frac{\log|C_n(L_2)|}{n}$, $n \geq 1$. Suppose that $\{x_{n_l}\}_{l=1}^{\infty}$, where $n_1 < n_2 < \ldots$, is a subsequence of $\{x_n\}_{n=1}^{\infty}$, such that $\lim_{l\to\infty} x_{n_l} = ent\bullet(L_1)$. Consider the corresponding sequence $\{y_{n_l}\}_{l=1}^{\infty}$. As follows from the theorem conditions, $\forall l \in \mathbb{N}, l \geq 1 : x_{n_l} \leq y_{n_l}$, then $\lim_{l\to\infty} \sup x_{n_l} \leq \lim_{l\to\infty} \sup y_{n_l}$. Hence, $ent\bullet(L_1) = \lim_{l\to\infty} x_{n_l} = \lim_{l\to\infty} \sup x_{n_l} \leq \lim_{l\to\infty} \sup y_{n_l} \leq ent\bullet(L_2)$. ■

Second, in certain cases we can refine the above property as follows.

Theorem 5.2 (Strict monotonicity on the number of words).
Let L_1 and L_2 be two regular languages such that $\forall\, k \in \mathbb{N} : |C_k(L_1)| \leq |C_k(L_2)|$ and $\exists\, k_0 \in \mathbb{N} : |C_{k_0}(L_1)| < |C_{k_0}(L_2)|$. Then, it holds that $ent\bullet(L_1) < ent\bullet(L_2)$. ⌟

Proof. Language L_1 does not contain the maximum number of sequences of length k_0, because $|C_{k_0}(L_1)| < |C_{k_0}(L_2)|$. Let l be a sequence such that $l \in \Sigma^*$, $|l| = k_0$ and $l \notin L_1$. Let us consider language $\overline{L}_1 = L_1 \cup \{l\}$. Then, $ent\bullet(L_1) < ent\bullet(\overline{L}_1)$, because of the monotonicity of the short-circuit entropy measure, see [4]. According to Theorem 5.1, $ent\bullet(\overline{L}_1) \leq ent\bullet(L_2)$. Thus, $ent\bullet(L_1) < ent\bullet(L_2)$. ∎

By counting and comparing the number of words of a certain length, we operate over a different partial order than in [4]. This partial order illustrates a more general phenomenon. Note that if $L_1 \subseteq L_2$, then $\forall\, k \in \mathbb{N} : |C_k(L_1)| \leq |C_k(L_2)|$. If, additionally, $L_1 \subset L_2$, then $\exists\, k_0 : |C_{k_0}(L_1)| < |C_{k_0}(L_2)|$. Theorem 5.1 and Theorem 5.2 prove monotonicity of $ent\bullet$ on this new partial order and allow comparing regular languages that are not in the set containment relationship. According to Theorem 5.2, for example, for two regular languages $L_1 = \{\langle a,a\rangle, \langle a,a,a\rangle, \langle a,a,a,a\rangle, \ldots\}$ and $L_2 = \{\langle b\rangle, \langle b,b\rangle, \langle b,b,b\rangle, \langle b,b,b,b\rangle, \ldots\}$, it holds that $ent\bullet(L_1) < ent\bullet(L_2)$.

5.3. Precision and Recall

Let M and L be two regular languages that capture the traces of a model and log, respectively. The intersection of M and L is a regular language, refer to [33]. Therefore, one can use $ent\bullet(M \cap L)$ to estimate cardinality of the collection of all the traces shared by the model and log. Consequently, in [4], the authors define the entropy-based precision (*prec*) and recall (*recall*) between M and L as follows:

$$prec(M,L) = \frac{ent\bullet(M \cap L)}{ent\bullet(M)}, \quad recall(M,L) = \frac{ent\bullet(M \cap L)}{ent\bullet(L)}.$$

In [4], we showed that as the number of traces shared by the model and log increases, the entropy-based precision and recall also increase. Next, we demonstrate three additional properties of the entropy-based precision and recall, which also hold for their extensions presented in Section 6.

Theorem 5.3 (Monotonicity).
Let L_1 and L_2 be two event logs such that $L_1 \subseteq L_2$ and let M be a regular language. Then, it holds that $prec(M,L_1) \leq prec(M,L_2)$. Let M_1 and M_2 be two regular languages such that $M_1 \subseteq M_2$ and let L be an event log. Then, it holds that $recall(M_1,L) \leq recall(M_2,L)$. ⌟

Proof. Since it holds that $L_1 \subseteq L_2$ ($M_1 \subseteq M_2$), it also holds that $M \cap L_1 \subseteq M \cap L_2$ ($M_1 \cap L \subseteq M_2 \cap L$). Because of the monotonicity of the short-circuit entropy [4], it holds that $ent\bullet(M \cap L_1) \leq ent\bullet(M \cap L_2)$ ($ent\bullet(M_1 \cap L) \leq ent\bullet(M_2 \cap L)$). Hence, it also holds that $prec(M,L_1) \leq prec(M,L_2)$ ($recall(M_1,L) \leq recall(M_2,L)$). ∎

Theorem 5.3 shows that the "monotonicity of languages" implies the monotonicity of the corresponding precision and recall values. This result can be exploited to prove the properties and axioms proposed in [6] and [32]. Additionally, this theorem generalizes Axiom A5 proposed in [32], which states that $L_1 \subseteq L_2 \subseteq M$ is a sufficient condition for the outlined inequality of precision values. Moreover, if one claims that $L_1 \subset L_2 \subset M$, then $(L_1 \cap M) \subset (L_2 \cap M)$, and thus, according to [4], the strict inequality holds, i.e., $\mathit{prec}(M, L_1) < \mathit{prec}(M, L_2)$. Similarly, Theorem 5.3 shows the monotonicity of recall values.

Desired properties and axioms for the precision and recall measures discussed in [6, 32] rely on the partial order of languages induced by the subset relation. However, it might be feasible to extend the partial order and consider a more general case when one language is not necessarily a subset of another. To that end, we extend the partial order by considering the number of words of a specific length. The following two theorems give a more general view on the properties of precision and recall measures.

Theorem 5.4 (Generalized monotonicity).
Let L_1 and L_2 be two event logs and let M be a regular language such that $\forall k \in \mathbb{N} : |C_k(M \cap L_1)| \leq |C_k(M \cap L_2)|$. Then, it holds that $\mathit{prec}(M, L_1) \leq \mathit{prec}(M, L_2)$. Let M_1 and M_2 be two regular languages and let L be an event log such that $\forall k \in \mathbb{N} : |C_k(M_1 \cap L)| \leq |C_k(M_2 \cap L)|$. Then, it holds that $\mathit{recall}(M_1, L) \leq \mathit{recall}(M_2, L)$. ⌟

Theorem 5.4 follows from Theorem 5.1 applied to $M \cap L_1$ and $M \cap L_2$ ($M_1 \cap L$ and $M_2 \cap L$). This result can be conveniently refined into the next one.

Theorem 5.5 (Generalized strict monotonicity).
Let L_1 and L_2 be two event logs and let M be a regular language such that $\forall\, k \in \mathbb{N} : |C_k(M \cap L_1)| \leq |C_k(M \cap L_2)|$ and $\exists\, k_0 \in \mathbb{N} : |C_{k_0}(M \cap L_1)| < |C_{k_0}(M \cap L_2)|$. Then, it holds that $\mathit{prec}(M, L_1) < \mathit{prec}(M, L_2)$. Let M_1 and M_2 be two regular languages and let L be an event log such that $\forall\, k \in \mathbb{N} : |C_k(M_1 \cap L)| \leq |C_k(M_2 \cap L)|$ and $\exists\, k_0 \in \mathbb{N} : |C_{k_0}(M_1 \cap L)| < |C_{k_0}(M_2 \cap L)|$. Then, it holds that $\mathit{recall}(M_1, L) < \mathit{recall}(M_2, L)$. ⌟

Theorem 5.5 follows immediately from the application of Theorem 5.2 to languages $M \cap L_1$ and $M \cap L_2$ ($M_1 \cap L$ and $M_2 \cap L$).

Consider DFAs in Fig. 3a and Fig. 3b that recognize languages M_1 and M_2, respectively, and event logs $L_1 = \{\langle a, b\rangle, \langle b, a\rangle\}$, $L_2 = \{\langle\rangle, \langle a, b\rangle, \langle b, a\rangle, \langle a, c\rangle, \langle b, a, b\rangle\}$, and $L_3 = \{\langle b, a, d\rangle, \langle b, a, b\rangle\}$.

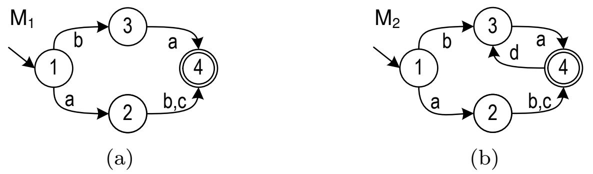


Figure 3: Two finite automata.

The entropy-based precision and recall values for all the combinations of these two regular languages and three event logs are listed in Table 1.

Table 1: The entropy-based precision and recall values.

Model	Log	*prec*	*recall*
M_1	L_1	0.874	1.000
M_1	L_2	1.000	0.745
M_1	L_3	0.000	0.000
M_2	L_1	0.754	1.000
M_2	L_2	0.863	0.745
M_2	L_3	0.000	0.000

The values in Table 1 justify, e.g., that M_1 contains all the traces from L_1, as it holds that $\mathit{recall}(M_1, L_1) = 1$, and does not contain some traces from L_2, as $\mathit{recall}(M_1, L_2) < 1$. One can also conclude that L_2 "covers" all the behavior of M_1, $\mathit{prec}(M_1, L_2) = 1$. Furthermore, M_2 does not contain some traces from L_2, $\mathit{recall}(M_2, L_2) < 1$. According to Theorem 5.3, since $L_1 \subset L_2$ ($M_1 \subset M_2$), it holds that $\mathit{prec}(M_1, L_1) \leq \mathit{prec}(M_1, L_2)$, $\mathit{prec}(M_2, L_1) \leq \mathit{prec}(M_2, L_2)$ ($\mathit{recall}(M_1, L_1) \leq \mathit{recall}(M_2, L_1)$, $\mathit{recall}(M_1, L_2) \leq \mathit{recall}(M_2, L_2)$). Moreover, Theorem 5.5 allows us obtaining strict inequalities $\mathit{prec}(M_1, L_1) < \mathit{prec}(M_1, L_2)$ and $\mathit{prec}(M_2, L_1) < \mathit{prec}(M_2, L_2)$.

Note that L_3 does not intersect with M_1 (or M_2). Consequently, despite some shared subsequences in their traces, e.g., $\langle b, a \rangle$, the corresponding precision and recall values equal to zero. This limitation is addressed in the next section.

6. Entropy-Based Conformance Checking: Partial Matching

In this section, we extend the conformance checking approach summarized in the previous section with support for partial matching of the compared model and log. We first define additional notions and discuss their properties, e.g., τ-closure of an automaton that implements the idea of diluting model or log traces, refer to Section 6.1. We then use these notions to propose new precision and recall measures, refer to Section 6.2. Finally, Section 6.3 discusses aspects related to the construction of a DFA that recognizes a given (diluted) event log.

6.1. Entropy and τ-Closure of Regular Languages

Let $B = (Q, \Lambda, \delta, q_0, A)$ be an NFA that recognizes language L, i.e., $\mathit{lang}(B) = L$. The τ-closure of B, denoted by B', is the NFA $B' = (Q, \Lambda, \gamma, q_0, A)$, where $q_2 \in \gamma(q_1, a)$ iff $(q_2 \in \delta(q_1, a)) \vee ((a = \tau) \wedge (\exists\, b : q_2 \in \delta(q_1, b)))$, $q_1, q_2 \in Q$, $a, b \in \Lambda \cup \{\tau\}$. In other words, for each two states connected via a transition in B, its τ-closure contains an additional silent transition that connects these states. We call B' and $\mathit{lang}(B')$ the *τ-closure* of B and L, respectively. For example, Fig. 4 shows the τ-closure of the automaton from Fig. 2a.

If L is the language recognized by B, by L', we denote the language recognized by B'; note that $L \subseteq L'$ and L' is regular. Next, we state two important properties of τ-closures of regular languages.

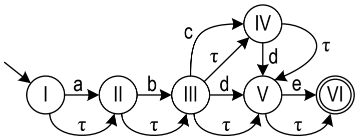


Figure 4: τ-closure of the DFA from Fig. 2a.

Theorem 6.1 (Monotonicity of τ-closure).
Let L_1, L_2 be regular languages and $L_1 \subset L_2$. Then, it holds that $L_1' \subseteq L_2'$. ⌟

Proof. For each $\alpha \in L_1$, it holds that $\alpha \in L_2$. Consequently, all the sequences obtained from α by the τ-closure operation belong to both L_1' and L_2'. This implies $L_1' \subseteq L_2'$. ∎

Next, we give a condition that ensures strict monotonicity.

Theorem 6.2 (Strict monotonicity of τ-closure).
Let L_1 and L_2 be two regular languages such that $L_1 \subset L_2$ and $L_2 \nsubseteq L_1'$. Then, it holds that $L_1' \subset L_2'$. ⌟

Proof. It follows from Theorem 6.1 that $L_1' \subseteq L_2'$. Since $L_2 \nsubseteq L_1'$, there exists a sequence α, such that $\alpha \in L_2$ and $\alpha \notin L_1'$. Hence $\alpha \in L_2'$ (because $L_2 \subseteq L_2'$), and consequently $L_1' \neq L_2'$. Since it holds that $L_1' \subseteq L_2'$ and $L_1' \neq L_2'$, it is a strong inclusion $L_1' \subset L_2'$. ∎

Theorem 6.2 gives a sufficient condition for the strong inclusion of τ-closures of languages L_1 and L_2, where $L_1 \subset L_2$. Concretely, it suffices to find a sequence in L_2 such that it cannot be obtained by "diluting" L_1, i.e., $L_2 \nsubseteq L_1'$.

6.2. Precision and Recall

Let M and L be two languages recognized by NFAs that encode the traces of a model and log, respectively. We propose to measure precision and recall between the model and log based on the τ-closures of M and L as follows:

$$prec_\tau(M, L) = \frac{ent\bullet(M' \cap L')}{ent\bullet(M')}, \quad recall_\tau(M, L) = \frac{ent\bullet(M' \cap L')}{ent\bullet(L')},$$

where M' and L' are the τ-closures of M and L, respectively. Note that first M', L', and $M' \cap L'$ are constructed, and only after that short-circuit entropy is computed. The relations between M and L and their τ-closures are visualized in Fig. 5. The intersection of M' and L' contains the intersection of M and L and, in addition, all the common sub-traces of M and L. It is possible that $M \cap L$ is empty, while $M' \cap L'$ is not. This holds, for example, if M and L are the languages recognized by the automata in Fig. 1a and Fig. 1b, respectively. This is also the reason the corresponding precision and recall values take non-zero values, refer to Section 3 for details.

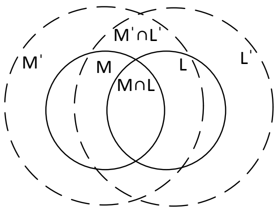


Figure 5: Intersection of two languages and their τ-closures.

Theorem 6.1 and Theorem 6.2 prove monotonicity of τ-closure operation and allow applying Theorem 5.3 to τ-closures of original languages, and hence, inheriting monotonicity properties of the exact matching precision and recall measures. As shown in Theorem 6.2, for strict monotonicity, additional conditions are to be met.

In contrast to the monotonicity based on the partial order of languages induced by the subset relation (Theorem 6.1 and Theorem 6.2), the monotonicity based on the partial order that relates languages on the number of words of a specific length is not inherited by τ-closures of languages. Consider two languages: $L_1 = \{\langle a, a\rangle, \langle a\rangle\}$ and $L_2 = \{\langle a, b\rangle\}$. According to Theorem 5.2, $ent\bullet(L_1) > ent\bullet(L_2)$, because $|C_2(L_1)| = |C_2(L_2)|$ and $|C_1(L_1)| > |C_1(L_2)|$. The τ-closures of these languages are $L'_1 = \{\langle a, a\rangle, \langle a\rangle, \langle\rangle\}$ and $L'_2 = \{\langle a, b\rangle, \langle a\rangle, \langle b\rangle, \langle\rangle\}$, respectively. According to Theorem 5.2, since $|C_2(L'_1)| = |C_2(L'_2)|$, $|C_1(L'_2)| > |C_1(L'_1)|$, and $|C_0(L'_1)| = |C_0(L'_2)|$, it holds that $ent\bullet(L'_1) < ent\bullet(L'_2)$. Although the generalized monotonicity for precision and recall measures (Theorem 5.4, Theorem 5.5) cannot be extrapolated to the partial matching approach, Theorem 5.4 and Theorem 5.5 can still be applied separately for original languages and their τ-closures.

Table 2 extends Table 1 by showing the new precision and recall values for the corresponding models and logs.

Table 2: The entropy-based precision and recall values, both original and based on the τ-closures of languages.

Model	Log	*prec*	*recall*	$prec_\tau$	$recall_\tau$
M_1	L_1	0.874	1.000	0.873	1.000
M_1	L_2	1.000	0.745	1.000	0.960
M_1	L_3	0.000	0.000	0.873	0.811
M_2	L_1	0.754	1.000	0.615	1.000
M_2	L_2	0.863	0.745	0.704	0.960
M_2	L_3	0.000	0.000	0.733	0.966

These example values show that although log L_3 has no common traces with the models, they can be partially matched. According to Theorem 6.2, it holds that $L'_1 \subset L'_2$, and according to Theorem 5.3 (applied to L'_1, L'_2, M'_1, and M'_2), new precision values for L_1 (0.873 and 0.615) are indeed less than or equal to the corresponding new precision values for L_2 (1.000 and 0.704). Interestingly, $prec_\tau(M_1, L_2) = 1$ because $M'_1 \subseteq L'_2$. Also, note that the absolute values of the reported measures are of minor importance, as those are their relations that provide useful insights. According to Theorem 6.2, $L'_1 \subset L'_3$, similarly, the new precision values for L_3 (0.873 and 0.733) are greater than or equal to the corresponding new precision values for L_1 (0.873 and 0.615).

It is easy to verify that $M'_1 \subset M'_2$, refer to Fig. 3. Then, according to Theorem 5.3, recall values for M_2 are always greater than or equal to the corresponding recall values for M_1.

Let us take a closer look at logs L_2 and L_3. None of these logs includes the other; same holds for logs L'_2 and L'_3. It holds that $M'_2 \cap L'_2 = \{\langle\rangle, \langle a\rangle, \langle b\rangle, \langle c\rangle, \langle a,b\rangle, \langle b,a\rangle, \langle a,c\rangle\}$ and $M'_2 \cap L'_3 = \{\langle\rangle, \langle a\rangle, \langle b\rangle, \langle d\rangle, \langle a,b\rangle, \langle b,a\rangle, \langle a,d\rangle, \langle b,d\rangle, \langle b,a,d\rangle\}$. The former language contains less number of sequences of the length two and three. Hence, by Theorem 5.5 applied to L'_2, L'_3, and M'_2, it must hold that $prec_\tau(M_2, L_2) < prec_\tau(M_2, L_3)$. Indeed, according to the values in Table 2: $prec_\tau(M_2, L_2) = 0.704$ and $prec_\tau(M_2, L_3) = 0.733$. Note that this property allows comparing languages over different alphabets.

6.3. Automata That Recognize τ-Closures of Event Logs

In most practical cases, a process model designed or discovered from an event log generalizes the behavior captured in the log. Consequently, its reachability graph often has a compact structure. A real-world event log, however, is usually large, and its representation involves a large number of states and transitions. Furthermore, the construction of a DFA that recognizes the τ-closure of a given event log can lead to the state space explosion [35]. To this end, we introduce two heuristics that aim at reducing the number of states generated when constructing a DFA that encodes the τ-closure of an event log.

Firstly, we take advantage of the composite nature of an event log and apply the divide-and-conquer approach. Indeed, an event log can be split into disjoint subsets of traces. Then, intermediate minimal DFAs that recognize the languages of the subsets can be constructed and then merged to obtain a DFA that recognizes the original event log. Although this does not solve the problem of state space explosion when determinizing a diluted event log, in practice, this strategy often allows reducing the number of constructed intermediate states. Consider the DFA constructed for the τ-closure of event log $L = \{\langle a,b,a\rangle, \langle b,a,a,b\rangle, \langle c,b,c\rangle\}$ shown in Fig. 6. This DFA contains groups of equivalent states, for example $\{D,F,J\}$ and $\{E,I\}$, that will be merged during the subsequent minimization. To reduce the number of states stored in memory, it may be feasible to construct models from subsets of traces, minimize them independently, and then merge. For instance, one can first construct a DFA recognizing the τ-closure of traces $\langle a,b,a\rangle$ and $\langle b,a,a,b\rangle$ (the corresponding states have a white background in Fig. 6), then minimize it by merging $\{D,F\}$ and $\{E,I\}$ groups of states, and finally add states and transitions modeling the τ-closure of trace $\langle c,b,c\rangle$ (these states are highlighted in gray in the figure).

In Section 8, we experiment with and discuss different strategies of splitting a given event log into subsets of traces to construct a DFA that recognizes the diluted version of this event log.

Secondly, when constructing an NFA that recognizes the diluted version of an event log, one can ignore those traces that do not impact the final result.

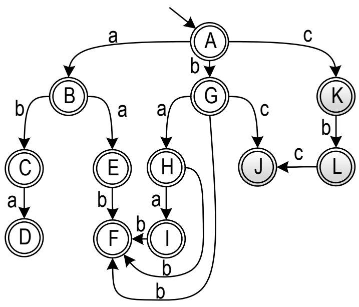


Figure 6: A DFA that recognizes the τ-closure of $L = \{\langle a,b,a\rangle, \langle b,a,a,b\rangle, \langle c,b,c\rangle\}$.

For example, one can ignore a trace that is a sub-trace of some other trace in the log, as captured in the following result.

Proposition 6.3 (Sub-trace reduction).
Let L be an event log and let $\alpha, \beta \in L$ be two traces such that β can be obtained from α by removing some events without changing the order of the remaining events, i.e., β is a sub-trace of α. Then, it holds that $L' = (L \setminus \{\beta\})'$. ⌟

Proof. By the definition of L', it holds that $L' = (L \setminus \{\beta\})' \cup \{\beta\}'$. As it holds that $\alpha \in L \setminus \{\beta\}$, it also holds that all the sub-traces of α are in $(L \setminus \{\beta\})'$. As β is a sub-trace of α, it holds that $\beta \in (L \setminus \{\beta\})'$. As every sub-trace $\hat{\beta}$ of β is also a sub-trace of α, it also holds that $\hat{\beta} \in (L \setminus \{\beta\})'$. Hence, it holds that $\{\beta\}' \subseteq (L \setminus \{\beta\})'$. Consequently, it holds that $L' = (L \setminus \{\beta\})'$. ∎

Proposition 6.3 suggests that when constructing an NFA that recognizes the diluted version of a given event log, refer to Section 6.1, one is "safe" to only use the traces that are not sub-traces of some other traces in the log and can ignore all the other traces. In practice, this often leads to a significant reduction of the input, and consequently, of the computation time and memory used to construct a DFA that recognizes the diluted version of the log. Again, we experiment with and discuss this phenomenon in Section 8.

7. Algorithms

In this section, we describe algorithms that implement conformance checking measures proposed in this article. All the presented in this section algorithms have been implemented in a publicly available tool with a command-line interface distributed as part of the jBPT library [36, 37].[2]

[2]Refer to `https://github.com/jbpt/codebase` (`jbpt-pm` module).

Algorithm 1: EntropyBasedPrecisionAndRecall [4]

Input: Two DFAs *ret* and *rel* describing retrieved and relevant behavior, respectively; for example, *ret* recognizes the language of a process model and *rel* recognizes the language of an event log.

Output: A pair $(prec, rec)$, where *prec* and *rec* are, respectively, precision and recall for *ret* and *rel*.

```
1  $mRet \leftarrow$ Minimize($ret$);                                   /* Minimize $ret$ */
2  $mRel \leftarrow$ Minimize($rel$);                                   /* Minimize $rel$ */
3  $intersection \leftarrow$ Intersection($mRet, mRel$);              /* Construct intersection */
4  $mIntersection \leftarrow$ Minimize($intersection$);              /* Minimize $intersection$ */
5  $scRet \leftarrow$ ShortCircuit($mRet$);                           /* Short-circuit $mRet$ */
6  $scRel \leftarrow$ ShortCircuit($mRel$);                           /* Short-circuit $mRel$ */
7  $scIntersection \leftarrow$ ShortCircuit($mIntersection$);  /* Short-circuit $mIntersection$ */
8  // Compute largest eigenvalues of the adjacency matrices of automata
9  $eigRet \leftarrow$ PerronFrobenius($scRet$);
10 $eigRel \leftarrow$ PerronFrobenius($scRel$);
11 $eigIntersection \leftarrow$ PerronFrobenius($scInersection$);
12 return $(\frac{eigIntersection}{eigRet}, \frac{eigIntersection}{eigRel})$;
```

Algorithm 1 was originally proposed in [4].[3] It can be used to compute precision and recall values between two DFAs recognizing languages that represent relevant and retrieved process behavior. Firstly, the input DFAs are minimized using the *Minimize* function (see lines 1–2). Then, the intersection DFA of the two minimal DFAs is constructed using the *Intersection* function (line 3), and the resulting DFA is, again, minimized (line 4). Next, the short-circuit versions of the three minimal DFAs are constructed using the *ShortCircuit* function (lines 5–7), refer to Section 5.2. Subsequently, Perron–Frobenius (largest) eigenvalues of the adjacency matrices of the resulting ergodic DFAs are computed using the *PerronFrobenius* function (lines 9–11). These eigenvalues quantify the topological entropy values of the corresponding languages. Note that topological entropy of a regular language is defined as the logarithm of the Perron–Frobenius eigenvalue of the adjacency matrix of a DFA that recognizes the language [19]. As the precision and recall measures presented in [4] inherit useful properties from the monotonicity property of the Perron–Frobenius eigenvalues, and the logarithm is a monotonic function, either eigenvalues or their logarithms can be used to define the precision and recall quotients. In [4], the choice was made in favor of eigenvalues, hence the pair of precision and recall values returned at line 12 of the algorithm; note that according to the Perron–Frobenius theorem [38], the eigenvalues are always positive.

Function *Minimize* implements the algorithm by Hopcroft [39]; its worst-case time complexity is $O(nm \log(m))$, where n is the number of states and m is the number of labels used in the automaton. Function *Intersection* imple-

[3]The corresponding algorithm in [4] takes two NFAs as input and starts by computing their deterministic versions. In this article, we use Algorithm 1 as a sub-routine that is always called for two deterministic input automata.

ments a well-known operation of intersection of regular languages in automata theory [33]. Its time complexity is $O(nm)$, where n and m are the numbers of states in the intersected automata. Function *ShortCircuit* implements the corresponding construction presented in Section 5.2. Finally, for the details on the implementation of function *PerronFrobenius*, refer to [4].

Function *TauClosure* implements the construction defined in Section 6.1 and exemplified in Fig. 4. Function *Determinize* implements the extended version of the Rabin-Scott powerset construction [33]. To mitigate the performance issues, both in terms of computation time and memory usage, that arise in practice due to the exponential worst-case time complexity of the Rabin-Scott construction, below, we introduce several versions of Algorithm 2.

Algorithm 2 summarizes the procedure presented in [20] for computing precision and recall that account for the partial matching of traces. The algorithm starts by constructing the τ-closures of the input NFAs using the *TauClosure* function (lines 1–2). Then, the resulting NFAs are determinized using the *Determinize* function (lines 3–4). Finally, the constructed DFAs are supplied as input to Algorithm 1 to compute the precision and recall values.

Algorithm 3 captures the core logic of an alternative version of Algorithm 2 that aims to improve the original algorithm by incorporating the two heuristics for constructing a DFA that recognizes the τ-closure of an event log discussed in Section 6.3. In the alternative algorithm, function *TauClosureAndDeterminize* constructs the required DFA. While it may be computationally challenging to construct a DFA that recognizes the τ-closure of an entire event log, it may be feasible to construct minimal DFAs that recognize the τ-closures of its parts (or blocks). Consequently, one can fix the size of a block (number of traces in a block), construct minimal DFAs for individual blocks, and finally merge them into the resulting DFA. Algorithm 4 implements this idea.

Algorithm 4 starts by filtering out traces from the input log that do not influence the result. According to Proposition 6.3, traces that are sub-traces of some other traces in the log are irrelevant when constructing the τ-closure of a language. These traces are removed from the log using function *FilterTraces* (line 1); the function is described in Algorithm 6. Next, the algorithm splits the input log into blocks, where the size of a block *blockSize* is provided as

Algorithm 2: PartialMatchingEntropyBasedPrecisionAndRecall [20]

```
Input: Two NFAs $ret$ and $rel$ describing retrieved and relevant behavior,
       respectively; for example, $ret$ recognizes the language of a process model
       and $rel$ recognizes the language of an event log.
Output: A pair $(prec_\tau, rec_\tau)$, where $prec_\tau$ and $rec_\tau$ are, respectively, precision and
        recall for $ret$ and $rel$ that account for the partial matching of traces.
1 $tcRet \leftarrow$ TauClosure($ret$);              /* Construct $\tau$-closure of $ret$ */
2 $tcRel \leftarrow$ TauClosure($rel$);              /* Construct $\tau$-closure of $rel$ */
3 $tcdRet \leftarrow$ Determinize($tcRet$);                  /* Determinize $tcRet$ */
4 $tcdRel \leftarrow$ Determinize($tcRel$);                  /* Determinize $tcRel$ */
5 return $EntropyBasedPrecisionAndRecall(tcdRet, tcdRel)$;
```

Algorithm 3: PartialMatchingEntropyBasedPrecisionAndRecall

```
Input: An NFA $ret$ and an event log $rel$ describing retrieved and relevant behavior,
       respectively, where $ret$ recognizes the language of a process model.
Output: A pair $(prec_\mathcal{T}, rec_\mathcal{T})$, where $prec_\mathcal{T}$ and $rec_\mathcal{T}$ are, respectively, precision and
        recall for $ret$ and $rel$ that account for the partial matching of traces.
1 $tcRet \leftarrow$ TauClosure($ret$);                          /* Construct $\tau$-closure of $ret$ */
2 $tcdRet \leftarrow$ Determinize($tcRet$);                                 /* Determinize $tcRet$ */
3 $tcdRel \leftarrow$ TauClosureAndDeterminize($rel$);   /* Construct deluted DFA for $rel$ */
4 return $EntropyBasedPrecisionAndRecall(tcdRet, tcdRel)$;
```

input, constructs the minimal DFA for each block, and incrementally adds these minimal DFAs (using the *Union* function) to the resulting automaton (lines 4–19). Since the union of two DFAs is not necessary a DFA, each time the resulting automaton gets updated, it is determinized and minimized; otherwise, the final automaton may contain many nondeterministic states leading to the state space explosion in the last phase of the algorithm. By incrementally constructing the automaton that recognizes the τ-closure of the input event log, we reduce the number of states, and thus the amount of required memory, to capture the automaton.

Given two positive numbers a and b, a mod b, see line 6 in Algorithm 4, is the remainder of the Euclidean division of a by b. Each call to the *NewAutomaton* function, refer to lines 3 and 13 in Algorithm 4, returns the empty automaton, i.e., the automaton without states and transitions. Function *Union* constructs an automaton that recognizes the union of languages recognized by two input

Algorithm 4: TauClosureAndDeterminize (fixed block size)

```
Input: An event log $L$ and size of a block $blockSize$.
Output: A DFA that recognizes the $\tau$-closure of $L$, i.e., $L'$.
1  $T \leftarrow$ FilterTraces($L$);       /* Filter out traces that do not impact result */
2  $cnt \leftarrow 1$;                                                 /* Initialize counter */
3  $A, B \leftarrow$ NewAutomaton();                                /* Initialize two automata */
4  while $cnt \le |T|$ do
5      $B \leftarrow$ AddTrace($B, T.get(cnt)$);                          /* Add next trace to $B$ */
6      if ($cnt$ mod $blockSize$) = 0 then
7          $B \leftarrow$ TauClosure($B$);                          /* Construct $\tau$-closure of $B$ */
8          $B \leftarrow$ Determinize($B$);                                        /* Determinize $B$ */
9          $B \leftarrow$ Minimize($B$);                                              /* Minimize $B$ */
10         $A \leftarrow$ Union($A, B$);                          /* Construct union of $A$ and $B$ */
11         $A \leftarrow$ Determinize($A$);                                        /* Determinize $A$ */
12         $A \leftarrow$ Minimize($A$);                                              /* Minimize $A$ */
13         $B \leftarrow$ NewAutomaton();                          /* Re-initialize automaton $B$ */
14     end
15     $cnt \leftarrow cnt + 1$;                                            /* Increment counter */
16 end
17 $A \leftarrow$ Union($A, B$);                                  /* Construct union of $A$ and $B$ */
18 $A \leftarrow$ Determinize($A$);                                                /* Determinize $A$ */
19 $A \leftarrow$ Minimize($A$);                                                      /* Minimize $A$ */
20 return $A$;
```

Algorithm 5: TauClosureAndDeterminize (decreasing block size)

```
Input: An event log $L$, size of a minimal block $minBlockSize$, and rate at which the
       block size is decreased $rate$, such that $rate \geq 2$.
Output: A DFA that recognizes the $\tau$-closure of $L$, i.e., $L'$.
1  $T \leftarrow$ FilterTraces($L$);                /* Filter out traces that do not impact result */
2  $cnt \leftarrow 1$;                                                   /* Initialize counter */
3  $blockSize \leftarrow |T| \,/\, rate$;                             /* Initialize size of a block */
4  $A, B \leftarrow$ NewAutomaton();                                /* Initialize two automata */
5  while $cnt \leq |T|$ do
6      $B \leftarrow$ AddTrace($B$, $T.get(cnt)$);                       /* Add next trace to $B$ */
7      if $(cnt \bmod blockSize) = 0$ then
8          $B \leftarrow$ TauClosure($B$);                         /* Construct $\tau$-closure of $B$ */
9          $B \leftarrow$ Determinize($B$);                                      /* Determinize $B$ */
10         $B \leftarrow$ Minimize($B$);                                            /* Minimize $B$ */
11         $A \leftarrow$ Union($A, B$);                            /* Construct union of $A$ and $B$ */
12         $A \leftarrow$ Determinize($A$);                                      /* Determinize $A$ */
13         $A \leftarrow$ Minimize($A$);                                            /* Minimize $A$ */
14         $B \leftarrow$ NewAutomaton();                           /* Re-initialize automaton $B$ */
15         if $(blockSize > minBlockSize)$ then
16             $blockSize \leftarrow blockSize \,/\, rate$;              /* Reduce size of a block */
17         end
18     end
19     $cnt \leftarrow cnt + 1$;                                             /* Increment counter */
20 end
21 $A \leftarrow$ Union($A, B$);                                    /* Construct union of $A$ and $B$ */
22 $A \leftarrow$ Determinize($A$);                                              /* Determinize $A$ */
23 $A \leftarrow$ Minimize($A$);                                                    /* Minimize $A$ */
24 return $A$;
```

automata, which is trivially achieved by “joining” the start states of the input automata. A call to function *AddTrace* at line 6 of Algorithm 4 constructs an automaton that recognizes language $lang(B) \cup \{T.get(cnt)\}$, where $T.get(cnt)$ is a trace at position *cnt* in T (lexicographical order on traces is used).

Algorithm 5 modifies Algorithm 4 to work with blocks of different sizes. Indeed, it gets computationally more demanding to process every next block at lines 7–13 of Algorithm 4 because of the operations on automaton A at lines 10–12. Hence, in Algorithm 5, we start by processing a large block and then progressively decrease the size of every next processed block until the minimal specified block size is reached. For instance, if the input variable *rate* is set to 2, then the first processed block contains half of the traces from the input event log. The second block is twice smaller than the first one. Etc. This strategy exploits the assumption that the number of states in automaton A grows fast as more traces of the log get processed.

Algorithm 6 defines function *FilterTraces*. Function *GetLongestTrace* is used to obtain a longest trace from yet unprocessed traces (line 3). Then, the current longest unprocessed trace *curTrace* is checked to be a sub-trace of some trace in T – a set of traces that are not sub-traces of any so far processed trace and of length at least of the length of *curTrace* (lines 5–11). If *curTrace* is not a sub-trace of any trace in T, then it is added to T (line 12). Once all the traces

Algorithm 6: FilterTraces

```
    Input: An event log $L$.
    Output: The filtered version of $L$.
 1  $T \leftarrow \varnothing$;                                      /* Initialize result */
 2  while $|L| > 0$ do
 3      $curTrace \leftarrow$ GetLongestTrace(L);         /* Find a longest trace in $L$ */
 4      $L \leftarrow L \setminus \{curTrace\}$;                   /* Remove $curTrace$ from $L$ */
 5      $isSubtrace \leftarrow false$;                /* Initialize control variable */
 6      foreach $trace \in T$ do
 7          if $IsSubtrace(curTrace, trace)$ then
 8              $isSubtrace \leftarrow true$;
 9              break;
10          end
11      end
12      if $isSubtrace = false$ then $T \leftarrow T \cup \{curTrace\}$;
13  end
14  return $T$;
```

in the input log are processed, set T is the filtered version of L (line 14). A call to function *IsSubtrace* at line 8 of Algorithm 6 checks if trace *curTrace* is a sub-trace of trace *trace*. It is implemented through sequential scans through both traces keeping track of matching events and runs in $O(n+m)$ worst-case time where n and m are the lengths of the two traces.

8. Experimental Results

In this section, we evaluate our implementation of the eigenvalue-based precision and recall measures presented in Section 7 using synthetic and real-world datasets. All the experiments can be reproduced using our publicly available implementation[4] and the command line tool [37].

8.1. Synthetic Dataset

In this section, we experiment with a synthetic event log and a set of corresponding process models described in [21, 23]. The log is defined as follows:

$$L = \{\langle A,B,D,E,I\rangle, \langle A,C,D,G,H,F,I\rangle, \langle A,C,G,D,H,F,I\rangle, \\ \langle A,C,H,D,F,I\rangle, \langle A,C,D,H,F,I\rangle\}.$$

The set of process models consists of ten Petri nets shown in Fig. 7.

The reachability graphs of all these ten Petri nets can be encoded as DFAs. Table 3 shows precision and recall values between languages recognized by these DFAs and L. All these values were computed in close to real-time (in milliseconds) using a computer with Intel Core i3-3110M CPU @2.40 GHz and 4 GB RAM.

[4] `https://github.com/akalenkova/eigen-measure`

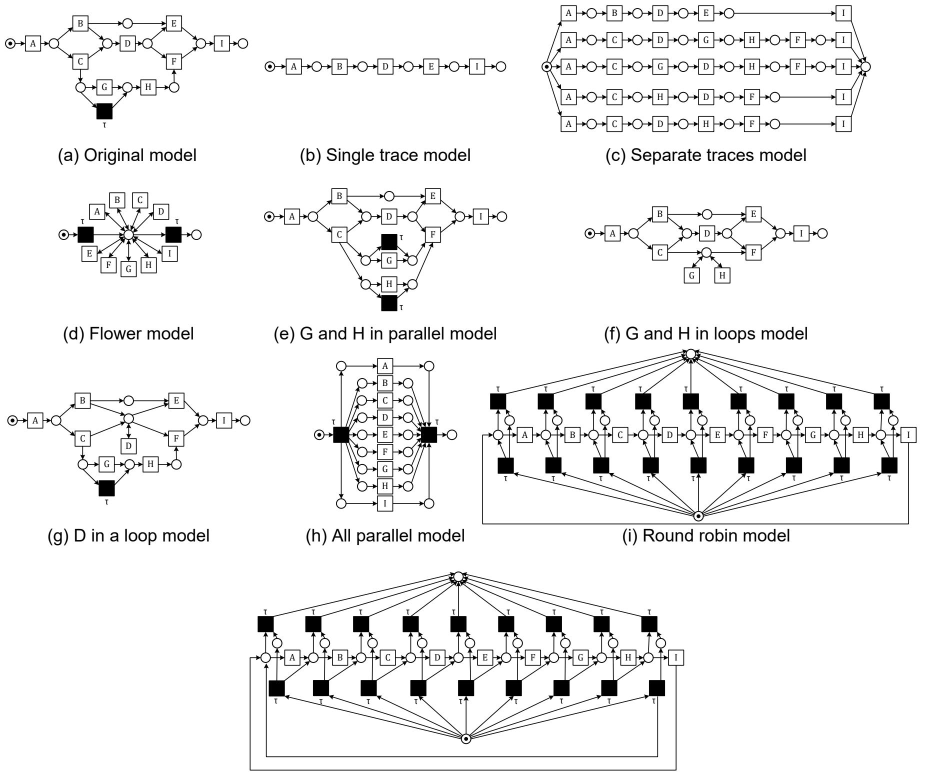

(a) Original model (b) Single trace model (c) Separate traces model
(d) Flower model (e) G and H in parallel model (f) G and H in loops model
(g) D in a loop model (h) All parallel model (i) Round robin model
(j) Round robin model (alternative)

Figure 7: Synthetic process models from [21, 23].

The values in the left-most *recall* column in Table 3 measure the share of traces from the log that are also the traces of the model. Six models can "replay" all the log traces, hence the recall values of one. The *Single trace* model accepts only one trace from the log, which leads to the recall value of less than one. The *All parallel* model, which imposes a restriction that all the labels must appear exactly once in a trace, and the two versions of the *Round robin* model (shown in Fig. 7(i) and Fig. 7(j) and introduced in [21] and [23], respectively)[5], which execute all the activities in a particular order without skipping them, do not accept a single trace from the log. Therefore, the corresponding recall values equal to zero. The τ-closures of the behaviors described by the *All parallel* and the two versions of the *Round robin* model contain traces supporting any order of events and any number of event skips. Additionally, in the *Round robin* models, events can be repeated any number of times. Hence, *All parallel* and

[5]The difference between the two versions of the *Round robin* model is in the length of traces that they accept. However, since both models give the same precision and recall values for all the evaluated measures, they are presented in a single row in Table 3 and Table 4.

Table 3: Precision and recall values for synthetic log and models.

Model	*prec*	*recall*	$prec_\tau$	$recall_\tau$
Original	0.979	1.000	0.998	1.000
Single trace	1.000	0.798	1.000	0.732
Separate traces	1.000	1.000	1.000	1.000
Flower	0.125	1.000	0.479	1.000
G and H in parallel	0.889	1.000	0.986	1.000
G and H in loops	0.568	1.000	0.933	1.000
D in a loop	0.758	1.000	0.970	1.000
All parallel	0.000	0.000	0.656	1.000
Round robin	0.000	0.000	0.479	1.000

the two versions of the *Round robin* model accept any trace from the τ-closure of L, see the corresponding recall values of one in Table 3. This example shows that the τ-closure operation can significantly extend the initial behavior. In such cases, it might be feasible to consider both measures when comparing the log and model.

Table 4: Rankings of the synthetic models based on precision values for the synthetic log: 1 – worst precision, 9 – best precision.

Model	*SD*	ETC_a	*NE*	*PCC*	*AA*	MAP^1	$MAP^{2\text{-}7}$	*EB*	EB^τ
Original	7	7	9	8	7	7	7	7	7
Single trace	8	8	6	8	8	7	8	8	8
Separate traces	8	8	8	7	8	7	8	8	8
Flower	1	1	1	1	1	1	1	3	1
G and H in parallel	6	3	7	6	6	5	6	6	6
G and H in loops	1	5	5	4	5	3	3	4	4
D in a loop	1	6	4	5	4	5	4	5	5
All parallel	1	2	2	2	3	2	2	1	3
Round robin	1	4	3	3	1	4	5	1	1

Table 4 shows ranks of the synthetic models with respect to the synthetic log derived based on the state-of-the-art precision measures; 9 stands for the best precision score, while 1 stands for the worst precision score. The last two columns in the table show rankings derived based on our techniques. Column *EB* stands for the *entropy-based* approach from Section 5, while EB^τ for its extension from Section 6. The other rankings for the same models and log were computed in [23, 21]. Concretely, these precision measures were considered (refer to columns 2–8 in the table): *Set Difference* (*SD*) [25], *Alignment-based ETC precision* (*ETCa*) [30], *Negative Events* (*NE*) [26, 40], *Projected Conformance Checking* (*PCC*) [22], *Anti-Alignment precision* (*AA*) [23, 41], and *k-order Markovian Abstraction Precisions* (MAP^k), $k \in [1..7]$ [21]. Greater values of k for the latter technique correspond to less abstraction in the encodings of models and logs and, thus, more "precise" measurements.

According to [4], the exact *EB* precision is the only precision measure that was formally shown to possess the property of the strict monotonicity, i.e., the more common traces a log and model have, the greater the measured values are. The partial EB^τ measures presented in Section 6 inherit all the properties of the *EB* measures with the only exception for strict monotonicity under the conditions stated in Theorem 6.2. The non-strict monotonicity holds for the

EB^{τ} precision immediately based on the result of Theorem 6.1. Note that all these properties of the EB and EB^{τ} measures hold for behaviors that describe arbitrary, possibly infinite, collections of traces.

The partial matching EB^{τ} precision ranks the models in the same way as the EB precision except for the *All parallel* model and the *Flower* model. Since the *All parallel* model does not accept traces of the log, the corresponding EB precision rank is the worst (along with the *Round robin* models). However, according to the EB^{τ} precision, the *Flower* model has the worst rank because the *All parallel* model after "dilution" accepts all the traces from the "diluted" event log and the share of its behavior not present in the log is less than the share of the *Flower* model behavior not present in the log.

The closest ranking to that one produced by the EB^{τ} precision is the ranking by the *Anti-Alignment precision* (AA). The difference is in the rankings of models *G and H in loops* and *D in a loop*. These two models describe all the traces from the synthetic log. However, the behavior described in the *G and H in loops* model has more variability. Indeed, for a fixed length, the *G and H in loops* model describes more traces of that length than the *D in a loop* model. This is due to the high number of possible interleavings of G and H in the *G and H in loops* model. The monotonicity of the approach reported in [23, 41] is based on finite concepts, such as maximal length of the log trace, while we propose a general approach capable of assessing infinite behaviors described in models.

Some precision measurements and ranks are less intuitive. For instance, ETC_a, MAP^1, and $MAP^{2\text{-}7}$ techniques rank the *Round robin* models among the *G and H in loops*, *D in a loop*, and the *G and H in parallel* models. Note, however, that these models are similar to the original process while the *Round robin* models accept many traces not present in the log. The NE and PCC measures distinguish the *Single trace* model and the *Separate traces* model, while in terms of precision, these models should be the same because they do not accept traces that are not in the log. Finally, the SD measure gives the same score to five different models.

Although the partially matching conformance measures, under some reasonable conditions, ensure the monotonicity of the measurements, they may yield computationally expensive. In the next subsection, we apply our partial matching techniques to real-world data and analyze the scalability of different modifications of the original partial matching algorithm proposed in [20].

8.2. Real-World Event Data

Next, we investigate the scalability of our approach to verify whether it can be applied to real-world event logs. In our experiments, we used Intel Xeon Gold 6154 CPU @3.00 GHz with 128 GB RAM. We have applied the proposed conformance checking approach and its extensions to Business Process Intelligence Challenge (BPIC) event logs [42, 43, 44], which are publicly available logs[6] of real-world IT-systems, and an event log of a booking flight system

[6]BPIC logs: `https://data.4tu.nl/repository/collection:event_logs_real`.

Table 5: Characteristics of the real-world event logs.

Event log	Name	# Traces	# Sub-traces	# Ev. Classes	# Events
1	BPIC'12	2,320	1,998	18	164,144
2	BPIC'13 closed	111	105	3	5,179
3	BPIC'13 open	45	44	3	1,403
4	BPIC'13 incid.	832	805	4	44,607
5	BFS'13	1,315	1,003	12	30,393
6	BPIC'15 1	709	60	64	25,823
7	BPIC'15 2	449	16	85	20,420
8	BPIC'15 3	756	66	56	28,482
9	BPIC'15 4	580	20	61	21,848

(BFS). Prior to the analysis, we filtered out infrequent events that appear less than in 80% of traces using *Filter Log using Simple Heuristics* Process Mining Framework (ProM) [45] plugin. Analysis of filtered event logs allows finding deviations in sequences of the most frequent event executions discarding less frequent noise events. All the filtered logs used in the evaluation are available for download together with our implementation of the measures. Characteristics of the analyzed event logs, such as the total number of traces (# Traces), number of traces that are sub-traces of some other traces of the log (# Sub-traces), number of event classes, i.e., unique event labels, (# Ev. Classes), and the overall number of events (# Events), are summarized in Table 5.

From each event log, a Petri net was discovered using the Inductive miner [17]. This discovery technique constructs bounded Petri nets, such that their reachability graphs can be encoded as DFAs. We used these DFAs as representations of model behaviors to compute the measures presented in this article. Firstly, we calculated all the precision and recall values using the exact matching approach [4]. The results are shown in Table 6. For each event log, we ran experiments five times and calculated average execution times and the corresponding 95% confidence intervals. To speed up calculations, we verified whether the model can replay all the traces from the log (this can be checked efficiently). If so, we accepted that the intersection of languages is encoded by the same DFA as the event log. Consequently, the time to build the automaton and the time to calculate the entropy for the intersection of languages was accepted to be zero. As can be seen in Table 6, analyzed event logs are modeled by DFAs with not more than several thousands of states, and precision and recall values can be obtained in less than a minute. In contrast to determinization, the entropy calculation times do not depend trivially on the sizes of automata. This can be explained by the fact that the calculation of the eigenvalues of a matrix depends not only on the size of the matrix but also on its content [4].

Next, we calculated the precision and recall values using the new partial matching approach. When using the partial matching approach, NFAs with silent transitions should be converted to equivalent DFAs. As this operation is known to be computationally expensive (especially for large NFAs constructed

Table 6: Exact matching approach (time for DFAs construction and entropy calculation in milliseconds).

Event log	Automaton	# States / # Transitions	Construct. time	Entropy calc. time	Total time	Precision / Recall
1	L	9,102 / 10,642	2,861±334	787±104	3,649±391	0.147 / 1.000
	$M \cap L$	9,102 / 10,642	0	0	0	
	M	4 / 22	5±2	27±3	32±3	
2	L	156 / 243	15±2	193±5	208±6	0.918 / 0.797
	$M \cap L$	16 / 19	30±5	82±29	111±31	
	M	3 / 4	6±2	15±2	21±2	
3	L	33 / 43	3±1	514±95	517±95	0.903 / 1.000
	$M \cap L$	33 / 43	0	0	0	
	M	3 / 4	6±2	16±0	21±2	
4	L	2,032 / 2,760	127±5	553±7	680±6	0.575 / 0.824
	$M \cap L$	6 / 5	125±32	0±0	125±32	
	M	5 / 9	7±1	1±2	8±2	
5	L	2,830 / 4,063	355±100	36,033±591	36,388±661	0.498 / 0.948
	$M \cap L$	293 / 631	126±18	4±1	130±18	
	M	514 / 3,329	36,049±7,124	363±268	36,412±7,166	
6	L	10,784 / 11,456	2,013±177	380±9	2,393±176	0.025 / 1.000
	$M \cap L$	10,784 / 11,456	0	0	0	
	M	13 / 541	72±4	1±2	73±3	
7	L	12,316 / 12,752	2,723±157	573±22	3,295±154	0.016 / 0.991
	$M \cap L$	6,482 / 6,685	1,266±162	305±3	1,571±162	
	M	15 / 552	55±3	1,041±414	1,096±416	
8	L	9,590 / 10,297	3,674±778	408±4	4,082±780	0.030 / 1.000
	$M \cap L$	8,140 / 8,751	948±64	134±11	1,082±55	
	M	29 / 1,105	62±6	3±2	66±6	
9	L	9,187 / 9,753	2,936±218	6,988±107	9,950±238	0.027 / 0.999
	$M \cap L$	7,981 / 8,478	1,272±98	5,175±2,619	1,276±99	
	M	57 / 2,353	141±7	3±2	144±8	

from real-world event logs), in this article we propose improvements that allow us, where possible, controlling the space used in computations (Section 6.3), making this operation applicable for a wider set of logs.

Table 7 compares time (in milliseconds) and memory (in GB) used by the original approach [20] (Algorithm 2), refer to column 3, Algorithm 3 instantiated with the fixed block size *TauClosureAndDeterminize* function (Algorithm 4), refer to columns 4–6, and Algorithm 3 instantiated with the decreasing block size *TauClosureAndDeterminize* function (Algorithm 5), see column 7 in the table. These experiments were performed only once to compare and select determinization techniques for the further analysis. As reported in the table, DFAs representing τ-closures of large event logs may have millions of states. For the *fixed block size* algorithm, the size of a block was set to 1, 10, and 100. For the *decreasing block size* algorithm, the *minBlockSize* and *rate* parameters

Table 7: Time and maximal RAM used for the determinization of NFAs (in milliseconds and GB, respectively).

Log	NFA / DFA # States	Time / Memory				
		Original	Fixed block size			Decreasing block size
			1	10	100	
1	9,102 / 90,557	97,270 / 7.1	743,444 / 2.1	109,583 / 2.1	45,980 / 4.1	93,885 / 2.5
2	156 / 216	29 / 1	232 / 1	134 / 1	77 / 1	105 / 1
3	33 / 17	5 / 1	51 / 1	33 / 1	33 / 1	62 / 1
4	2,032 / 24,336	1,464,409 / 88	11,303 / 1	18,008 / 4.1	112,681 / 11	14,591 / 3.5
5	2,830 / 22,359	13,008 / 3.1	168,995 / 1	21,941 / 1	8,254 / 1.5	21,083 / 2.8
6	10,784 / 4,068,472	–	284,904,733 / 90	31,101,090 / 90	8,661,413 / 90	–
7	12,316 / –	–	–	–	–	–
8	9,590 / 1,665,113	1,352,873 / 71	105,000,038 / 55	12,309,253 / 65	2,918,489 / 65	10,015,028 / 65
9	9,187 / 4,343,979	–	162,898,238 / 90	19,003,571 / 90	5,911,793 / 100	24,869,735 / 115

were set to 10 and 3, respectively; setting the *rate* parameter to 2 made this approach similar to the original one in terms of space.

The results of experiments (Table 7) show that the algorithms proposed in this article allow reducing the memory used for constructing the DFAs that recognize the τ-closures of event logs. Moreover, in some cases (for logs 6 and 9), new techniques managed to construct diluted DFAs, while the original technique faced memory limitations. These experiments also demonstrate that the size of a block influences the memory size, i.e., the smaller the block size the less memory is required to perform computations. Unfortunately, for some large event logs (such as log 7), neither the original nor the new techniques produced a result (such cases are represented by dashes). Besides controlling the memory used for computations, the new techniques can reduce the computation time. For instance, log 4 can be efficiently processed if split into blocks. After the filtering, log 4 contains only 32 traces and in this case the optimal block size is not 100 (as for larger event logs, such as log 1, log 5, log 6, and log 9), but 1, i.e., the optimal block size is proportional to the overall size of the event log. For small event logs, such as logs 2 and 3, and sometimes for large logs, such as log 8, the overhead of manipulating intermediate NFAs exceeds the total time required to build the desired DFAs. This is explained by the fact that for DFAs that encode these logs the number of intermediate states merged during the minimization is small. Note that the number of states in the DFA modeling the τ-closure of log 3 is less than the number of states in the corresponding NFA because, after the filtering of sub-traces, the "diluted" version of log 3 contains only one trace, and hence, the resulting DFA has less states.

Table 8 summarizes the overall times of computing the precision and recall values using the partial matching approach. Again, the experiments were executed five times for each event log, and average execution times with 95% confidence intervals were calculated. For each event log, the best approach to build the "diluted" DFA (based on the results in Table 7) was applied. Similarly to the exact matching approach, the intersection of the model and log languages was computed only if the model could not replay the log. Note also that, similar to the exact matching approach, the entropy calculation time does not depend on the size of the model.

To conclude, none of the new methods demonstrated better runtime than the original approach consistently. For real-world logs of characteristics similar to those used in our evaluation, we recommend using Algorithm 4 with the fixed block of size 100, as it consistently uses less memory and is often faster. This method is implemented in our command line tools.

9. Conclusion

Entropy-based precision and recall conformance measures [4] quantify the similarity between traces described in a designed process model and its corresponding executed traces recorded in an event log. While precision quantifies how well the traces of the model are represented in the log, recall measures how well the traces in the log are represented in the model. In [20], we extended these conformance measures by addressing the phenomenon of partially matching traces, i.e., non-identical traces that nevertheless describe the same sub-sequences of process steps. The proposed extension based on the addition of silent (skipping) steps to the model and log traces has proven effective but also demonstrated a significant negative impact on the efficiency of the measures. In this article, we address this problem and introduce algorithms that aim to reduce the memory required to compute the measures. Additionally, these algorithms were supplemented by a technique for filtering out traces that can be obtained from other traces of the log by skipping some of the event occurrences. We prove that this filtering does not influence the final result. All the presented algorithms were implemented and tested on synthetic and real-world datasets. The experiments confirmed that our new techniques require less memory and often run faster.

We identify several directions for future work. The proposed measures do not support systems that cannot be described as DFAs, e.g., infinite-state systems. To address this limitation, coverability graphs may yield useful. Another limitation is that the proposed approach considers distinct traces only and can be extended to take into account frequencies of traces in event logs. We also

Table 8: Partial matching approach (minimal time for NFAs determinization and entropy calculation in milliseconds).

Event log	Automaton	# States / # Transitions	Determin. time	Entropy calc. time	Total time	Precision / Recall
1	L'	90,557 / 446,847	46,150±1,762	11,033±701	57,183±2,188	0.709 / 1.000
	$M' \cap L'$	90,557 / 446,847	0	0	0	
	M'	3 / 33	2±2	18±3	19±3	
2	L'	216 / 269	29±3	258±17	287±19	0.961 / 1.000
	$M' \cap L'$	216 / 269	0	0	0	
	M'	1 / 3	0±0	14±2	14±2	
3	L'	17 / 31	5±3	174±3	179±4	0.980 / 1.000
	$M' \cap L'$	17 / 31	0	0	0	
	M'	1 / 2	2±2	15±1	17±4	
4	L'	24,336 / 72,994	11,222±235	2,074±32	13,297±252	0.995 / 1.000
	$M' \cap L'$	24,336 / 72,994	0	0	0	
	M'	1 / 3	1±2	15±2	15±3	
5	L'	22,359 / 200,254	8,348±354	3,936±1,523	12,284±1,403	0.940 / 0.895
	$M' \cap L'$	7,542 / 45,163	1,343±91	2,407±1,035	3,750±1,025	
	M'	514 / 3,340	225±4	26±15	250±16	
6	L'	4,068,472 / 66,501,653	8,496,809±246,746	9,332,802±2,692,693	17,818,611±2,642,806	0.374 / 1.000
	$M' \cap L'$	4,068,472 / 66,501,653	0	0	0	
	M'	5 / 312	4±3	4±6	8±7	
7	L'	–	–	–	–	–
	$M' \cap L'$	–	–	–	–	
	M'	3 / 252	1±2	16±2	17±2	
8	L'	1,665,113 / 24,396,479	1,356,059±15,340	1,699,562±533,044	3,055,621±538,234	0.393 / 1.000
	$M' \cap L'$	1,665,113 / 24,396,479	0	0	0	
	M'	5 / 272	2±3	1±2	4±5	
9	L'	4,343,979 / 64,701,275	6,078,928±191,831	8,451,036±2,114,810	14,529,964±2,154,036	0.376 / 1.000
	$M' \cap L'$	4,343,979 / 64,701,275	0	0	0	
	M'	5 / 297	11±5	2±2	13±7	

plan to extend the techniques by providing qualitative information on differences between designed and observed processes, including the identification and visualization of deviations. Finally, we plan to explore further partial and total orders over the set of regular languages to identify other interesting properties for conformance checking measures and study whether the entropy-based measures possess them. At this stage, we recommend using the partial matching conformance measures in offline settings, e.g., when designing new and evaluating existing process discovery algorithms. This limitation can be overcome by distributing the calculations over multiple computers, which will be explored in future work.

Acknowledgment

This work was in part supported by the Australian Research Council Discovery Project DP180102839. We want to thank Wolfgang Reisig for his input in the design of the presented optimized version of our algorithm for computing the entropy-based precision and recall that accounts for the partial matching of the compared traces.

References

[1] W. van der Aalst, Process Mining: Data Science in Action, 2016. `doi:10.1007/978-3-662-49851-4`.

[2] W. van der Aalst, A. Adriansyah, B. van Dongen, Replaying history on process models for conformance checking and performance analysis, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2 (2) (2012) 182–192. `doi:10.1002/widm.1045`.

[3] J. Carmona, B. van Dongen, A. Solti, M. Weidlich, Conformance Checking—Relating Processes and Models, Springer, 2018. `doi:10.1007/978-3-319-99414-7`.

[4] A. Polyvyanyy, A. Solti, M. Weidlich, C. D. Ciccio, J. Mendling, Monotone precision and recall measures for comparing executions and specifications of dynamic systems, ACM Trans. Softw. Eng. Methodol. 29 (3). `doi:10.1145/3387909`.

[5] S. J. J. Leemans, A. F. Syring, W. M. P. van der Aalst, Earth movers' stochastic conformance checking, in: Business Process Management Forum, Vol. 360 of Lecture Notes in Business Information Processing, Springer, 2019, pp. 127–143. `doi:10.1007/978-3-030-26643-1_8`.

[6] A. F. Syring, N. Tax, W. M. P. van der Aalst, Evaluating Conformance Measures in Process Mining Using Conformance Propositions, Springer Berlin Heidelberg, Berlin, Heidelberg, 2019, pp. 192–221.

[7] D. Fahland, W. M. P. van der Aalst, Model repair — aligning process models to reality, Information Systems 47 (2015) 220–243. `doi:10.1016/j.is.2013.12.007`.

[8] A. Polyvyanyy, W. van der Aalst, A. ter Hofstede, M. Wynn, Impact-driven process model repair, ACM Transactions on Software Engineering and Methodology 25 (4) (2017) 1–60. `doi:10.1145/2980764`.

[9] M. Weidlich, A. Polyvyanyy, N. Desai, J. Mendling, M. Weske, Process compliance analysis based on behavioural profiles, Information Systems 36 (7) (2011) 1009–1025. doi:10.1016/j.is.2011.04.002.

[10] J. C. A. M. Buijs, B. F. van Dongen, W. M. P. van der Aalst, Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity, International Journal of Cooperative Information Systems 23 (1). doi:10.1142/S0218843014400012.

[11] W. Frakes, R. Baeza-Yates, Information Retrieval: Data Structures and Algorithms, Prentice-Hall, Inc., NJ, USA, 1992.

[12] A. J. M. M. Weijters, W. M. P. van der Aalst, Rediscovering workflow models from event-based data using little thumb, Integrated Computer-Aided Engineering 10 (2) (2003) 151–162.

[13] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128–1142. doi:10.1109/TKDE.2004.47.

[14] C. W. Günther, W. M. P. van der Aalst, Fuzzy mining - adaptive process simplification based on multi-perspective metrics, in: Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings, 2007, pp. 328–343. doi:10.1007/978-3-540-75183-0_24.

[15] A. K. A. de Medeiros, A. J. M. M. Weijters, W. M. P. van der Aalst, Genetic process mining: An experimental evaluation, Data Mining and Knowledge Discovery 14 (2) (2007) 245–304. doi:10.1007/s10618-006-0061-7.

[16] J. Carmona, J. Cortadella, Process mining meets abstract interpretation, in: Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I, Vol. 6321 of Lecture Notes in Computer Science, Springer, 2010, pp. 184–199. doi:10.1007/978-3-642-15880-3_18.

[17] S. Leemans, D. Fahland, W. van der Aalst, Discovering Block-Structured Process Models from Incomplete Event Logs, in: ATPN'2014, Vol. 8489 of LNCS, Springer, 2014, pp. 91–110.

[18] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, A. Polyvyanyy, Split miner: automated discovery of accurate and simple business process models from event logs, KAIS (2018) 1–34doi:10.1007/s10115-018-1214-x.

[19] T. Ceccherini-Silberstein, A. Machì, F. Scarabotti, On the entropy of regular languages, Theor. Comp. Sci. 307 (2003) 93–102.

[20] A. Polyvyanyy, A. A. Kalenkova, Monotone conformance checking for partially matching designed and observed processes, in: International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24–26, 2019, IEEE, 2019, pp. 81–88. doi:10.1109/ICPM.2019.00022.

[21] A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, M. La Rosa, D. Reissner, Abstract-and-compare: A family of scalable precision measures for automated process discovery, in: Business Process Management, Springer International Publishing, Cham, 2018, pp. 158–175.

[22] S. Leemans, D. Fahland, W. van der Aalst, Scalable process discovery and conformance checking, Software & Systems Modeling 17 (2) (2018) 599–631.

[23] B. van Dongen, J. Carmona, T. Chatain, A unified approach for measuring precision and generalization based on anti-alignments, in: Business Process Management, Springer, Cham, 2016, pp. 39–56.

[24] J. Muñoz-Gama, J. Carmona, A fresh look at precision in process conformance, in: Business Process Management, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 211–226.

[25] G. Greco, A. Guzzo, L. Pontieri, D. Sacca, Discovering expressive process models by clustering log traces, IEEE Trans. on Knowl. and Data Eng. 18 (8) (2006) 1010–1027.

[26] J. De Weerdt, M. De Backer, J. Vanthienen, B. Baesens, A robust f-measure for evaluating discovered process models, in: CIDM, IEEE, 2011, pp. 148–155.

[27] A. Kalenkova, A. Polyvyanyy, A spectrum of entropy-based precision and recall measurements between partially matching designed and observed processes, in: Service-Oriented Computing, Springer International Publishing, Cham, 2020, pp. 337–354.

[28] A. Polyvyanyy, A. Moffat, L. García-Bañuelos, An entropic relevance measure for stochastic conformance checking in process mining, in: 2nd International Conference on Process Mining (ICPM), IEEE, 2020.

[29] S. Leemans, A. Polyvyanyy, Stochastic-aware conformance checking: An entropy-based approach, in: S. Dustdar, E. Yu, C. Salinesi, D. Rieu, V. Pant (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2020, pp. 217–233.

[30] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, W. van der Aalst, Measuring precision of modeled behavior, Inf. Syst. and e-Business Management 13 (1) (2015) 37–67.

[31] A. Rozinat, W. van der Aalst, Conformance checking of processes based on monitoring real behavior, Information Systems 33 (1) (2008) 64 – 95. `doi:10.1016/j.is.2007.07.001`.

[32] N. Tax, X. Lu, N. Sidorova, D. Fahland, W. van der Aalst, The imprecisions of precision measures in process mining, Information Processing Letters 135 (2018) 1 – 8. `doi:10.1016/j.ipl.2018.01.013`.

[33] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd Edition, Pearson International Edition, Addison-Wesley, 2007.

[34] S. Leemans, D. Fahland, W. van der Aalst, Discovering block-structured process models from event logs - A constructive approach, in: ATPN'2013 Conference, Vol. 7927 of LNCS, 2013, pp. 311–329.

[35] J. E. Hopcroft, J. D. Ullman, An n log n algorithm for detecting reducible graphs, in: Proe. 6th Annual Princeton Conf. on Inf. Sciences and Systems, 1972, pp. 119–122.

[36] A. Polyvyanyy, M. Weidlich, Towards a compendium of process technologies—the jBPT library for process model analysis, in: CAiSE Forum, Vol. 998 of CEUR Workshop Proceedings, CEUR-WS, 2013, pp. 1–8. URL `http://ceur-ws.org/Vol-998/Paper14.pdf`

[37] A. Polyvyanyy, H. Alkhammash, C. Di Ciccio, L. García-Bañuelos, A. Kalenkova, S. Leemans, J. Mendling, A. Moffat, M. Weidlich, Entropia: A family of entropy-based conformance checking measures for process mining, in: 2nd International Conference on Process Mining (ICPM). Demo Track., CEUR-WS, 2020.

[38] C. D. Meyer, Matrix Analysis and Applied Linear Algebra, SIAM, 2000. `doi:10.1137/1.9780898719512`.

[39] J. E. Hopcroft, An n log n algorithm for minimizing states in a finite automaton, Tech. rep., Stanford (1971).

[40] S. vanden Broucke, J. De Weerdt, J. Vanthienen, B. Baesens, Determining process model precision and generalization with weighted artificial negative events, IEEE Transactions on Knowledge and Data Engineering 26 (8) (2014) 1877–1889.

[41] T. Chatain, J. Carmona, Anti-alignments in conformance checking – the dark side of process models, in: ATPN'2016, pp. 240–258.

[42] B. van Dongen, Bpi challenge 2012 (2012). `doi:10.4121/UUID:3926DB30-F712-4394-AEBC-75976070E91F`.

[43] W. Steeman, Bpi challenge 2013 (2013). `doi:10.4121/UUID:A7CE5C55-03A7-4583-B855-98B86E1A2B07`.

[44] B. van Dongen, Bpi challenge 2015 (2015). `doi:10.4121/UUID:31A308EF-C844-48DA-948C-305D167A0EC1`.

[45] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, W. van der Aalst, The ProM Framework: A new era in process mining tool support, in: ATPN'2005, pp. 444–454.