A fresh look at Precision in Process Conformance

Jorge Muñoz-Gama and Josep Carmona

Universitat Politècnica de Catalunya, Spain
jmunoz@lsi.upc.edu, jcarmona@lsi.upc.edu

Abstract. Process Conformance is a crucial step in the area of Process Mining: the adequacy of a model derived from applying a discovery algorithm to a log must be certified before making further decisions that affect the system under consideration. Among the different conformance dimensions, in this paper we propose a novel measure for precision, based on the simple idea of counting these situations were the model deviates from the log. Moreover, a log-based traversal of the model that avoids inspecting its whole behavior is presented. Experimental results show a significant improvement when compared to current approaches for the same task. Finally, the detection of the shortest traces in the model that lead to discrepancies is presented.

Key words: Process Mining, Process Conformance.

1 Introduction and Related Work

Nowadays, the organizations make use of a wide variety of *Process-Aware Information Systems* (PAISs) to conduct their business processes [13]. These systems record all kind of information about the processes in *logs*, which can be used for different purposes. *Process Mining* is an area of research that aims at the discovery, analysis and extension of formal models in a PAIS, in order to support its design and maintenance.

The problem of deriving a formal model from a log is known as *Process Discovery*. For this problem, several algorithms exist which derive models that represent (maybe partially) processes detected by observing the traces in the log. In particular, the production of a Petri net [7] whose underlying behavior is related to the traces in the log has been presented extensively in the literature [17,2,14,4,18]. The Petri nets produced by many of these algorithms represent *overapproximations* of the log, i.e. the set of traces accepted in the net is a superset of the set of traces in the log. Therefore, one can not rely on the accuracy of the discovered model unless some minimality property is guaranteed ([2,4]), or a certification provided by a metric ensures its quality.

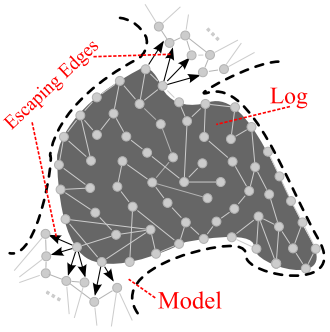
Process Conformance aims at evaluating the adequacy of a model in describing a log. Analyzing conformance is a complex task which involves the interplay of different and orthogonal dimensions [8,9]:

- ⋆ *Fitness*: indicates how much of the observed behavior is captured by (i.e. "fits") the process model.
- ⋆ *Precision*: refers to overly general models, preferring models with minimal behavior to represent as closely as possible the log.
- ⋆ *Generalization*: addresses overly precise models which overfit the given log, thus been possible to generalize.
- ⋆ *Structure*: refers to models minimal in structure which clearly reflect the described behavior.

Different algorithms for conformance checking have been presented in the literature (a complete survey can be found in [9]). In particular, some examples of approaches focused on precision are: [6] (measuring the percentage of potential traces in the model that are in the log), [5] (comparing two models and a log to see how much of the first model's behavior is covered by the second) used in [14], [19] (comparing the behavioral similarity of two models without a log), and [3] (using *minimal description length* to evaluate the quality of the model). In this work we focus on the precision between a model (a Petri net in our case) and a log. On this regard, the methods presented in [10,11] may be seen as a seminal work, were metrics for fitness, structural and precision are presented. In particular, the precision metric presented (called *advanced behavioral appropriateness*) is limited to comparing the ordering relations between events in the model with the ones from the log. This approach needs the exhaustive exploration of the model's state space, which can be impractical for large models that exhibit a high degree of concurrency, i.e. the *state-space explosion problem* arises.

Briefly, this paper presents a novel technique to measure precision, which aims at: i) complementing the precision information provided by other techniques, ii) fighting the inherent complexity that relies in checking conformance for industrial or real-life models and logs and, iii) providing useful information for later *Process Extension*, i.e. the stage where the model may be extended to better reflect the log. The initial implementation is available as the *ETConformance* plug-in in the ProM 6 framework [1].

The approach can be summarized as follows: given a model and a log, the behavior of the model restricted to the log is computed (part with gray background in the figure on the right). The border between the log's and model's behavior defines crucial points where the model deviates from the log. We call these situations *escaping edges*. By quantifying these edges and their frequency, we aim at providing an accurate measurement of the precision dimension. Moreover, the escaping edges denote inconsistencies that might be treated in the process extension phase.



In the previous figure it has been considered that the model's behavior includes all the log's behavior (perfect fitness), in order to evaluate its precision. When the fitness condition does not hold, we recommend (as in [10]), to analyse

the conformance in two phases: in the first phase the fitness is evaluated, filtering the noise and analysing the discrepancies. In the second phase, the other three dimensions are evaluated. Section 7 briefly comments how to deal with non-fitting models.

1.1 Why a new measure to quantify Precision?

A fresh look at precision We aim at providing a precision metric that estimates the effort needed to obtain an accurate model, focusing on the discrepancies detected. This contrasts with the existing approaches for precision that only provide discrepancies in the event relations [11].

Efficiency To compute the precision metric, we present a log-based traversal of the model's behavior . The technique avoids the traversal of the complete model's behavior, i.e., only the behavior of the model reflected in the log is explored (see the figure above). Hence, the approach can handle inputs that can not be handled by other approaches which traverse the complete behavior.

Granularity The closest measure to the one presented in this paper, is the *advanced behavioral appropriateness*, by Rozinat *et al* [11], denoted by a'_B. This approach consists in computing the precedence/follows relations between tasks in the model and in the log, and compares both relations, thus providing a behavioral metric for precision. However, these relations can only have three possible values: *Always*, *Never* and *Sometimes*. Our approach works directly with the behavior of the model, getting a deeper view of the precision problem.

Extensionality Finally, we feel that, in addition to the metric, it is important to provide an appropiate mechanism for the later Process Extension. On this regard, the methods presented in this paper output also the exact points of discrepancy, i.e., the traces where the model starts to deviate from the log.

The background for the understanding of this paper is presented in Section 2. Section 3 describes informally the approach, whereas Sections 4 and 5 present the algorithm to collect escaping edges and the metric for precision analysis, respectively. Section 6 introduces the notion of disconformant traces. Section 7 discuss some extensions and experimental results are presented in Section 8.

2 Preliminaries

In this section we present the two main inputs needed to perform the conformance analysis: Petri nets and logs (Event Logs). Additionally, Transitions Systems will be presented to link these two inputs.

Some mathematical notation is provided for the understanding of the paper. Given a set S, we denote $\mathcal{P}(S)$ as the powerset over S, i.e. the set of possible subsets of elements of S. Given a set T, a sequence $\sigma \in T^*$ is a called *trace*.

Given a trace $\sigma = t_1 t_2 \dots t_n$, and a natural number $0 \leq k \leq n$, $hd^k(\sigma)$ is the trace $t_1 t_2 \dots t_k$, also called the *prefix* of length k in σ. Notice that $hd^0(\sigma) = \lambda$, i.e., the empty word. Finally, given a set of traces L, we denote $Pref(L)$ the set of all prefixes for traces in L.

2.1 Petri Nets

Definition 1 (Petri Net [7]). *A Petri Net (*PN*) is a tuple (P, T, W, M_0) where P and T represent finite sets of places and transitions, respectively, with $P \cap T = \emptyset$. And $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the weighted flow relation. A marking is a mapping $P \to \mathbb{N}$. M_0 is the initial marking, i.e. defines the initial state of the system.*

A transition $t \in T$ is *enabled* in a marking M iff $\forall p \in P : M(p) \geq W(p,t)$. An enabled transition can be *fired* resulting in a new marking M' such that $\forall p : M'(p) = M(p) - W(p,t) + W(t,p)$. A marking M' is *reachable* from M if there is a sequence of firings $\sigma = t_1 t_2 \dots t_n$ that transforms M into M', denoted by $M[\sigma\rangle M'$. A sequence of transitions $\sigma = t_1 t_2 \dots t_n$ is a *feasible sequence* if $M_0[\sigma\rangle M$, for some M. The set of reachable markings from M_0 is denoted by $[M_0\rangle$, and form a graph called *reachability graph*. A PN is said to be *k-bounded* or simply *bounded* if $\forall p : M'(p)$ does not exceed a number k for any reachable marking M' from M_0. If no such number exists, it is said to be *unbounded*.

2.2 Event Logs

Event logs contain executions of a system [16]. These executions represent the ordering between different tasks, but may also contain additional information, like the task originator or its timestamp. For the purposes of this paper, all this information is abstracted:

Definition 2 (Event Log). *An* event log EL *is a set of traces, i.e.,* $\mathsf{EL} \in \mathcal{P}(T^*)$.

We denote by $|\mathsf{EL}|$ the number of traces of the log. As notation, these traces can be iterated through $\sigma_1 \dots \sigma_{|\mathsf{EL}|}$.

2.3 Relation between a Petri Net and an Event Log

Similarly as it is done in [10], tasks in the log and transitions in the Petri net must be mapped in order to establish a relation between both objects. Besides the most simple relation between only one task and one transition, the mapping may have some more complex scenarios: (i) *Duplicate tasks*, two or more tasks in the model are associated with the same task in the log. (ii) *Invisible tasks*, a task in the model is associated with no task in the log[1]. In this paper, the invisible tasks

[1] The reasons for this lack of relation may be different: non recordable steps in the process (phone calls, meetings, ...), introduced in the model for routing purposes, relaxations of the model, or invisible tasks resulting from some discovery algorithm (e.g., [14]).

in the model will be represented as transitions filled black. (iii) *Non Modelled tasks*, a task in the log is associated with no task in the model. Non modelled tasks are not relevant for the purpose of this paper. Therefore, they can be removed from the log before starting the analysis. The techniques presented in this paper cover all the scenarios above. Note that, for the sake of clarity, we refer to task, event or transition indistinctly, whenever no mistake is possible.

2.4 Transitions Systems

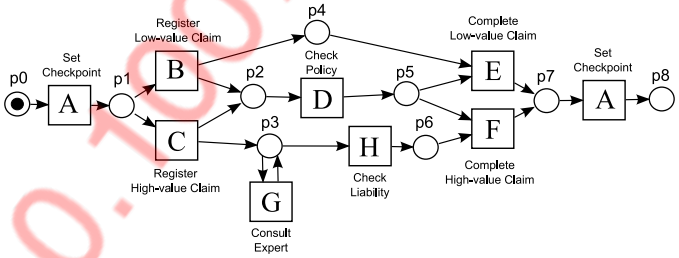
Definition 3 (Transition system). *A* transition system *(*TS*) is a tuple* (S, T, A, s_{in})*, where* S *is a set of* states*,* T *is an alphabet of* actions*,* $A \subseteq S \times T \times S$ *is a set of* (labelled) transitions*, and* $s_{in} \in S$ *is the* initial state*.*

We will use $s \xrightarrow{e} s'$ as a shortcut for $(s, e, s') \in A$, and the transitive closure of this relation will be denoted by $\xrightarrow{*}$. The *language* of a transition system TS, $L(\mathsf{TS})$, is the set of traces feasible from the initial state.

3 Problem Statement and Approach

As was said in the introduction, our goal is to measure the precision of a model with respect to a log. Moreover, we strive to locate inconsistencies between the model and the log, thus allowing the later extension of the model.

Let us introduce the PN and EL shown in Fig. 1(a) and (b), respectively, based on the examples used in [11]. We will use this example in the rest of the paper as a running example. The PN reflects the typical process of liability insurance claim in a bank. Note that, although the log represents a plausible situation, the control flow shown in the model is not realistic, i.e., the *Consult Expert* task should be executed exactly once. This inconsistence may seem obvious in this small scenario, but may be not for larger and realistic cases. Hence, we use it to illustrate the conformance analysis.



(a) *PN1*

Log Traces
A B D E A
A C D G H F A
A C G D H F A
A C G H D F A

(b) *EL1*

Fig. 1. Running example.

The precision metric introduced in this paper is computed smoothly: in contrast to other approaches that require a complete exploration of both the model

and the log [11], in our approach the model exploration is restricted to the observed log's behavior. As a consequence, the computational requirements are bounded to the log size, thus being independent of the whole model's underlying behavior. This might be crucial for models obtained from a mining algorithm that may have an underlying behavior of intractable size.

The key concept of the approach is that of *Escaping Edges*, i.e., the situations were the model allows more behavior than the log, thus exhibiting less precision. We base our measure on the relation between the number and frequency of escaping edges with respect to model's behavior restricted to the log . Hence, the more the model deviates from the log, the less precise is its precision value. It is important to stress the fact that a model can have few escaping edges (thus exhibiting good precision) but with underlying behavior substantially bigger than the log: it is not the aim of this work to compare sizes between the model and the log, but providing an estimation of the efforts required to improve the model to precisely describe the log. Figure 2 shows the *route map* of the ETConformance approach, and indicates the section where each part is explained in detail.

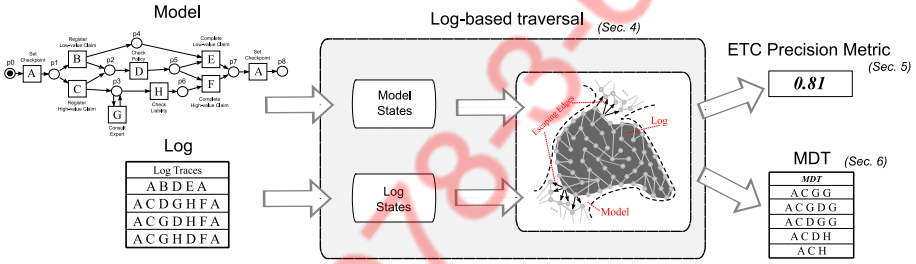


Fig. 2. Route map of ETConformance: a log and a Petri net are the inputs of the technique. Internally, log and Petri net states are identified and mapped. Finally, both the metric and the set of minimal discrepancies are reported.

4 Log-based Traversal of the Model's Behavior

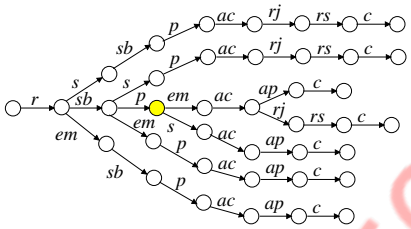
This section describes the log-based traversal of the model's behavior. We have an assumption on the input log: every trace in the log is possible in the model, i.e. it has fitness value one (see how to adapt the technique to non-fitting models in Section 7). This assumption is grounded on the correlations that exist between fitness and precision dimensions, as pointed in [10]. Hence, the fitness analysis and corresponding adjustments can be done before the precision analysis. There are some approaches to perform this task, e.g., [11].

4.1 Basic Idea

In order to perform a log-based traversal of the model behavior, it is necessary to find a common comparable domain between log and model, i.e. a domain where

1	r,s,sb,p,ac,ap,c
2	r,sb,em,p,ac,ap,c
3	r,sb,p,em,ac,rj,rs,c
4	r,em,sb,p,ac,ap,c
5	r,sb,s,p,ac,rj,rs,c
6	r,sb,p,s,ac,ap,c
7	r,sb,p,em,ac,ap,c

(a)



(b)

Fig. 3. (a) event log EL, (b) corresponding transition system TS such that $L(\mathsf{TS}) = \mathsf{EL}$.

states of the model and log states could be mapped. For performing such task, state information must be obtained for both objects (model and log). In the log side, several algorithms are presented in [15] to incorporate state information in the log. These algorithms are parametrizable with respect to the decision of a state (*past*, *future* or *both*), the representation of the information (*sequence*, *set* and *multiset*), and its horizon (*limited* or *infinite*). In particular, using the past, sequence and infinite settings allows to derive a behavioral representation of the log (a transition system) with the same language. The states of that transition system will be the states of the log.

Definition 4 (Prefix automaton, Log states). *Given an event log* EL*, let* $\mathsf{TS} = (S, T, A, s_0)$ *be the automaton derived by using the construction presented in [15] using the past, sequence and infinite settings. We call this automaton* prefix automaton*. The set* S *will be denoted as the* set of states of EL*. Given a trace* $\sigma_i = t_1 \dots t_{|\sigma_i|} \in \mathsf{EL}$*,* $s_j^i \in S$ *denotes the state in* TS *corresponding to the prefix* $t_1 \dots t_{j-1}$*, for* $0 \leq j \leq |\sigma_i| + 1$*.*

Fig. 3 shows an example of this transformation. The states of this transition system correspond to prefixes of the traces in the log: for instance, the state filled in Fig. 3 corresponds to the prefix r, sb, p, contained in traces 3, 6 and 7.

To obtain state information for a Petri net model, its reachable states can be computed. However, due to the well-known *space-explosion problem*, Petri nets can exhibit a large or even infinite behavior, making this approach impractical for these instances. Instead, the approach presented in this paper only visits those reachable markings of the net for which there is at least one state in the log (see Def. 4) mapped. Let us define formally the mapping:

Definition 5 (Mapping between log states and Petri net markings). *Let* EL *and* $\mathsf{PN} = (P, T, W, M_0)$ *be a log and a Petri net, respectively, and consider the prefix automaton* $\mathsf{TS} = (S, T, A, s_0)$ *from* EL*. A marking* M *is mapped to the state* $s \in S$*, denoted by* $M \multimap s$*, if there exists a trace* σ *such that* $s_0 \xrightarrow{\sigma} s$ *in* TS *and* $M_0[\sigma\rangle M$*.*

Due to this mapping, the Petri net traversal can be controlled to reach only markings for which there is a corresponding mapped state in the log. Moreover, for each one of the markings reached on this guided traversal, the local precision

on this marking can be measured by collecting the discrepancies between the behavior allowed in the model with respect to the behavior observed in the log:

Definition 6 (Allowed Tasks and Reflected Tasks). *Let s be a state of the prefix automaton $\mathsf{TS} = (S, T, A, s_0)$ from EL, and $\mathsf{PN} = (P, T, W, M_0)$ a Petri net. We define $A_T(s) = \{t \in T \mid M \multimap s \wedge M[t\rangle M'\}$ and $R_T(s) = \{t \in T \mid s \xrightarrow{t} s')\}$ as the set of allowed and reflected tasks in s.*

Since we are assuming fitness value one of the model with respect to the log, clearly $R_T(s) \subseteq A_T(s)$ for every state s of TS, i.e. the model overapproximates the log. Now it is possible to define the concept of escaping edge:

Definition 7 (Escaping Edges). *Let s be a state of the prefix automaton $\mathsf{TS} = (S, T, A, s_0)$ from EL, and $\mathsf{PN} = (P, T, W, M_0)$ a Petri net. The Escaping Edges (E_E) of s is defined as $E_E(s) = A_T(s) \setminus R_T(s)$.*

An example of escaping edge for the pair (*PN1*,*EL1*) in Fig. 1 is H in the log state reached after the prefix AC: here *PN1* accepts the H task, whereas this is not reflected in *EL1*. It is important to stress that the tasks reflected of a state s are the ones that appear in *any* of the traces that contain s. For instance, for the example of Fig. 1, the prefix A has two allowed tasks in the model (B and C) but given that both are reflected in the log (in different traces), no escaping edge arise in the log state after observing A. The algorithm to collect the set of escaping edges is presented as Algorithm. 1.

```
Input: EL, PN
foreach State s in EL do
    RT := outEdges (s)
    σ := prefix (s,EL)
    mark := fire (σ,PN)          // Fire σ and get the reached marking
    AT := enable (mark,PN)       // Get the enable transitions of mark
    EE := AT \ RT                // Allowed tasks minus Reflected Tasks
    register (s,EE)
```

Algorithm 1: ComputeEscapingEdges

4.2 Duplicate and Invisible tasks

Until now, the methods have been explained without considering duplicate or invisible tasks. Although the approach presented in this paper can deal with duplicate and invisible tasks, we must remark some issues concerning the potential indeterminism that may arise.

When trying to determine the marking associated to a log state, it may happen that firing the sequence of tasks cannot be done in a deterministic way, i.e., one log task is associated to two or more enabled transitions in the model,

and therefore, one of the transitions must be chosen to continue the firing. A similar situation occurs when a sequence of invisible tasks that enables a visible log task must be fired. In this cases, part of the state space of the model reachable from the current marking must be explored. To avoid producing an infinite state space (for instance, because the Petri net might be unbounded), we construct the *invisible coverability graph*: a variant of the well-known coverability graph algorithm [7] where the nodes are *ω-markings* and the edges are only invisible tasks. Informally, this graph will contain all the paths of invisible tasks that lead to the enabling of a visible one. It may occur that a visible task is enabled for several and different sequences of invisible tasks, and therefore, a guess between the different sequences must be made to continue the traversal.

In real-life scenarios, dealing with indeterminism requires the use of heuristics and therefore, the inconsistencies detected (escaping edges in our framework) might be caused by a bad guess and not by a real precision problem. In [11] a heuristic that uses the shortest sequence of invisible that enables a visible task is proposed. This heuristic tries to minimize the possibility that a invisible fired task interfere the future firing of another task. In general, the availability of several heuristics can be helpful to apply ad-hoc explorations which depend on the scenario considered.

5 Evaluating Precision

As it has been seen before, the escaping edges are a good indicator for measuring the behavior of a model compared to the behavior reflected in the log. For that reason, we propose a metric to take into account these escaping edges and their frequency. This metric also allows us to compare between models to know which one captures better the behavior reflected of a log. Let us formalize the metric:

Metric 1 (ETC Precision) *Let* $\mathsf{EL} = \{\sigma_1, \ldots, \sigma_{|\mathsf{EL}|}\}$ *and* $\mathsf{PN} = (P, T, W, M_0)$ *be a log and a Petri net, respectively. For each trace* σ_i $(1 \leq i \leq |\mathsf{EL}|)$*, state* s_j^i $(1 \leq j \leq |\sigma_i| + 1)$ *denotes the* $j-th$ *state of* σ_i *(see Def. 4). The metric is defined as follows:*

$$etc_P(\mathsf{EL}, \mathsf{PN}) = 1 - \frac{\sum_{i=1}^{|\mathsf{EL}|} \sum_{j=1}^{|\sigma_i|+1} |E_E(s_j^i)|}{\sum_{i=1}^{|\mathsf{EL}|} \sum_{j=1}^{|\sigma_i|+1} |A_T(s_j^i)|}$$

By dividing the set of escaping edges by the set of allowed tasks in the model, the metric evaluates the amount of overapproximation in each trace. Note that, for all s_j^i, $|E_E(s_j^i)| \leq |A_T(s_j^i)|$, and therefore $0 \leq \text{etc}_P \leq 1$. The fact that we take into account the frequency of the traces makes that the most used and appropriate traces would contribute with higher weight than the noisy traces and the wrong indeterministic choices. The metric value of the example for the pair (*PN1*, *EL1*) in Fig. 1 is

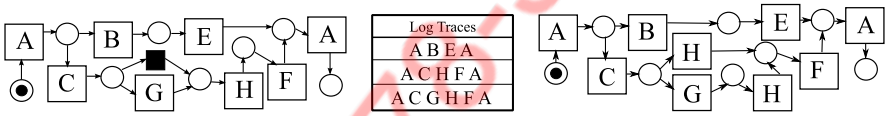
$$1 - \frac{0+3+3+2}{6+12+13+12} = 0.81$$

where every i-th summand of the numerator/denominator is processing the penalizations for the escaping edges of trace σ_i, e.g., in trace $\sigma_4 \in EL1$ there are 2 escaping edges and 12 allowed tasks.

As was done in [10], we present some of the quality requirements a good metric should satisfy and a brief justification on its fulfillment.

Validity (*The metric and the property to measure must be sufficiently correlated with each other*). In the case of Metric 1, the more escaping edges, the lower value will be provided (even closer to 0 in the worst case). This inverse correlation quantifies if a model is a precise description of a log.

Stability (*The metric must be as little as possible affected by the properties that are not measured*). In other words, the metric must measure only one dimension (precision in this case) independently of the others (e.g., fitness, structural, generalization). With regard to fitness or generalization, the metric is not stable since there is a correlation between precision and both dimensions. In contrast, by only focusing on the underlying behavior of the model, $\text{etc}_P$is independent to the structural dimension. To illustrate this, Fig.4 show *PN2* and *PN3*, two Petri Nets with the same behavior and different structure. The result provided by $\text{etc}_P$is 1 in both cases when compared with the log *EL2*.

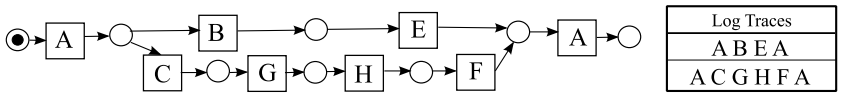


(a) *PN2* (b) *EL2* (c) *PN3*

Fig. 4. Stability of the metric with respect to structure.

Analyzability (*It relates to the properties of the measured values. In our case, the emphasis is on the requirement that the measured values should be distributed between 0 and 1, with 1 being the best and 0 being the worst value. It is important to be 1 when there is no precision problem*). The values returned by etc_P are distributed between 0 and 1. In addition, the value 1 can be reached, indicating that there are no inconsistencies. Notice that, to achieve this value it is not necessary to have only one enabled at each point of the trace (like in a_B [11]). This is because the metric does not depend on the idea of *more enabled tasks, more behavior*, but in the concept *behavior allowed vs reflected* itself. Moreover, the metric value can be 1, even if the whole behavior of the model is distributed in two or more different traces. This is because decision on escaping edges is done *globally* after processing the whole set of traces. This can be seen in *PN4* and *EL3* (cf. Fig. 5), that has a etc_P value of 1.

Localizability (*The system of measurement forming the metric should be able to locate those parts in the analyzed object that lack certain measured properties*). This is a crucial requirement in conformance analysis, due to the fact that



(a) *PN4* (b) *EL3*

Fig. 5. Global analysis of the traces leads to a global analysis of escaping edges.

providing the discrepancy points we are making possible to identify the potential points of model's enhancement. This is done in the approach of this paper through the escaping edges, identified by their marking and the task used to *escape* from the reflected behavior in the log. However, given the importance of the localizability in conformance, we go a step further and in the next section a technique to collect the traces leading to these situations is presented.

6 Minimal Disconformant Traces

Given a log and a model, we are interested in identifying these minimal traces that lead to a situation where the model starts to deviate from the log. Some of these traces may represent meaningful abstractions that arise in the model and therefore no further action is required. For the rest of traces, a decision on whereas the model or the log are wrong shall be made. In the case of an erroneous model, process extension techniques must be applied.

Definition 8 (Minimal Disconformant Traces). *Let* EL *and* $\mathsf{PN} = (P, T, W, M_0)$ *be a log and a Petri net, respectively. We define the* Minimal Disconformant Traces *(*MDT*) as the set of traces* $\sigma = \sigma' t$ *such that* $M_0[\sigma\rangle M$, $\sigma' \in \text{Pref}(\mathsf{EL})$ *and* $\sigma \notin \text{Pref}(\mathsf{EL})$.

The escaping edges computed in previous section can be used to obtain the Minimal Disconformant Traces. Algorithm 2 shows how to generate this set of traces using the escaping edges: for each state s with a escaping edge, the sequence to reach s is computed and it is concatenated with the escaping edge. Finally, Lemma 1 ensures a minimality criterion on the derived traces.

```
Input: EL
Output: M
foreach State s in EL do
    foreach Task t in E_E(s) do
        σ := prefix (s,EL)
        σ := σ · t                                   // Concatenate
        addTrace (σ, M)                    // Register σ as an MDT
return M
```

Algorithm 2: ComputeMDT

Lemma 1. *Algorithm 2 computes the Minimal Disconformant Traces.*

Proof: Let M the set of traces computed by the algorithm *ComputeMDT* and MDT the set of traces that satisfy Definition 8. To prove $M = \mathsf{MDT}$, we will prove $M \subseteq \mathsf{MDT}$ and then $\mathsf{MDT} \subseteq M$.

Let $\sigma = \sigma' t$ be any trace of M. By construction σ' is a prefix of the log. However, given the formation of R_T and E_E in Algorithm 1, σ is not a prefix of the log. Furthermore, σ' is a feasible sequence of the model because it is a prefix of the log, and all traces in the log are compliant with the model (since we assume fitness value one). In addition, by construction of A_T and E_E, σ is a feasible sequence by the model too. Therefore, $\sigma \in \mathsf{MDT}$.

Now, let $\sigma = \sigma' t$ be any trace of MDT. σ' is a prefix of the log. According to Definition 4, it must be defined a log state s after the sequence σ'. The task t must be in the A_T of s, because σ is a feasible sequence of the model. But t must not appear in R_T because σ is not a prefix of the log. By construction of R_T, t is a Escaping Edge. Therefore, $\sigma \in M$. □

MDT
A C G G
A C G D G
A C D G G
A C D H
A C H

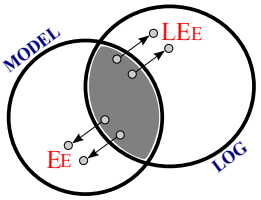
Fig. 6. MDTs

Following with the running example, Fig. 6 shows the MDTs for the model *PN1* and log *EL1* (described in Fig. 1). The MDT traces shown are result of the unseen behavior produced by the loop of G, which even allows the possibility of skipping G. Note that, the set of MDTs computed by Algorithm 2 might be seen as a log (i.e sequence of traces). Consequently, all kind of Process Mining techniques can be applied to it in order to get a general view of the information. In particular, mining methods can be used to obtain a Petri Net representing this extra behavior.

7 Extensions

Log States as Markings The first possible extension is to consider log states just as Petri net markings, i.e., two traces reaching the same marking correspond to the same state. With that approach we could have a more high-level vision of the precision, e.g., the R_T sets of two different states (with the same associated marking) now will be united in only one set corresponding to the new state. Although it could be useful in some occasion, we must be awared about all the information we are not considering. For instance, the etc_P value of (*PN1*, *EL1*) using this new approach would be 1, losing all track about the extra behavior introduced by the G loop.

Non-fitting models Symmetric to the Escaping Edges (E_E), we can define the *Log Escaping Edges*(LE_E), i.e., the points where the log deviates from the model. All these points could be evaluated, providing a metric that, in this case, would mesure fitness instead of precision.



All these extensions might be incorporated to extend the applicability of the approach presented in this paper.

8 Experimental Results

The technique presented in this paper, implemented as the *ETConformance* plug-in within ProM 6, has been evaluated on existing public-domain benchmarks [1]. The purpose of the experiments is:

- ⋆ Justify the existence of a new metric to evaluate precision, i.e. demonstrate the novelty of the concept when compared to previous approaches.
- ⋆ Show the capacity of the technique to handle large specifications.

Table 1(a) shows a comparison of the technique presented in this paper with the technique presented in [11], implemented in ProM 5.2 as the *Conformance Checker*. The rows in the table represent benchmarks with small size (few traces). The names are shortened, e.g., GFA5 represents GroupedFollowsA5. We report the results of checking precision for both conformance checkers in columns under a'_B and etc_P, respectively, for the small Petri nets obtained by the Parikh miner [18] which derived Petri nets with fitness value one. For the case of our checker, we additionally provide the number of minimal disconformant traces ($|\mathsf{MDT}|$). We do not report CPU times since checking precision in both approaches took less than one second for each benchmark.

From Table (a) one can see that when the model describes precisely the log, both metrics provide the maximum value. Moreover, when the model is not a precise description of the log, only three benchmarks provide opposite results (GFBN2, GFl2l, GFl2lSkip). For instance, the GFl2lSkip benchmark a'_B is providing a significant lower value: this is because the model contains an optional loop that is always traversed in the log. This variability is highly penalized by simply observing the tasks relations. On the other hand, metric etc_P will only penalize the few situations where the escaping edges appear in the log.

Larger benchmarks for which *Conformance Checker* cannot handle are provided in Table 1(b). For these benchmarks, we report the results (precision value, number of MDT and CPU time in seconds) for the models obtained by the Parikh miner and the RBMiner [12]. These are two miner that guarantee fitness value one. For each one of the aN benchmarks, N represents the number of tasks in the log, while the _1 and _5 suffixes denote its size: 100 and 900 traces, respectively. The $t32$ has 200 (_1) and 1800 (_5) traces. The pair of CPU times reported denote the computation of etc_P without or with the collection of MDTs (in parenthesis). Also, we provide the results of the most permissive models, i.e., models with only the transitions but without arcs or places (M_T). These models allow any behavior and thus, they have a low etc_P value, as expected.

A first conclusion on Table 1 (b) is the capability of handling large benchmarks in reasonable CPU time, even for the prototype implementation carried out. A second conclusion is the loss of precision of the metric with respect to the increase of abstraction in the mined models: as soon as the number of tasks increases, the miners tend to derive models less precise to account for the complex relations between different tasks. Often, these miners derive models with a high degree of concurrency, thus accepting a potentially exponential number of traces which might not correspond to the real number of traces in the log.

Table 1. Experimental results for small (a) and big (b) benchmarks.

(a)

Benchmark	a'_B	etc_P	|MDT|	Benchmark	a'_B	etc_P	|MDT|
GFA6NTC	1.00	1.00	0	GFl2lOpt	1.00	0.85	7
GFA7	1.00	1.00	0	GFAL2	0.86	0.90	391
GFA8	1.00	1.00	0	GFDrivers	0.78	0.89	2
GFA12	1.00	1.00	0	GFBN3	0.71	0.88	181
GFChoice	1.00	1.00	0	GFBN2	0.59	0.96	19
GFBN1	1.00	1.00	0	GFA5	0.50	0.57	35
GFParallel5	1.00	0.99	11	GFl2l	0.47	0.75	11
GFAL1	1.00	0.88	251	GFl2lSkip	0.30	0.74	10

(b)

		M_T	Parikh					RBMiner				
Benchmark	|TS|	etc_P	$\|P\|$	$\|T\|$	etc_P	|MDT|	CPU	$\|P\|$	$\|T\|$	etc_P	|MDT|	CPU
a22f0n00_1	1309	0.06	19	22	0.63	1490	0(0)	19	22	0.63	1490	0(0)
a22f0n00_5	9867	0.07	19	22	0.73	9654	0(3)	19	22	0.73	9654	0(4)
a32f0n00_1	2011	0.04	31	32	0.52	2945	0(0)	32	32	0.52	2944	0(1)
a32f0n00_5	16921	0.05	31	32	0.59	22750	2(10)	31	32	0.59	22750	2(11)
a42f0n00_1	2865	0.03	44	42	0.35	7761	0(2)	52	42	0.37	7228	0(2)
a42f0n00_5	24366	0.04	44	42	0.42	60042	5(28)	46	42	0.42	60040	6(29)
t32f0n00_1	7717	0.03	30	33	0.37	15064	1(15)	31	33	0.37	15062	1(12)
t32f0n00_5	64829	0.04	30	33	0.39	125429	9(154)	30	33	0.39	125429	8(160)

Finally, three charts are provided: the relation between the log size with respect to the CPU time, the etc_P value and the number of MDTs are shown in Fig. 7. For these charts, we selected different log sizes for different types of benchmarks (a22f0, a22f5, a32f0,a32f5 for the two bottom charts, a42f0, t32f5 and t32f9 for the top chart). For the two bottom charts, we used the Petri nets derived by the Parikh miner to perform the conformance analysis on each log, whereas we use a single Petri net for the top chart to evaluate the CPU time (without collecting MDTs) on different logs, illustrating the linear dependance of our technique on the log size. The chart on top clearly shows the linear relation between log size and CPU time for these experiments, which is expected by the technique presented in Sect. 4. The two charts on bottom of the figure show: (left) since for the a22/a32 benchmarks the models derived are very similar independently of the log, the more traces are included the less escaping edges are found. On the other hand, the inclusion of more traces contributes to the incorporation of more MDTs, as it is shown in the right chart at the bottom.

9 Conclusion

This paper has presented a low-complexity technique that allows checking the precision of a general Petri net with respect to a log. By only focusing on the underlying behavior of the Petri net that is reflected in the log, the technique

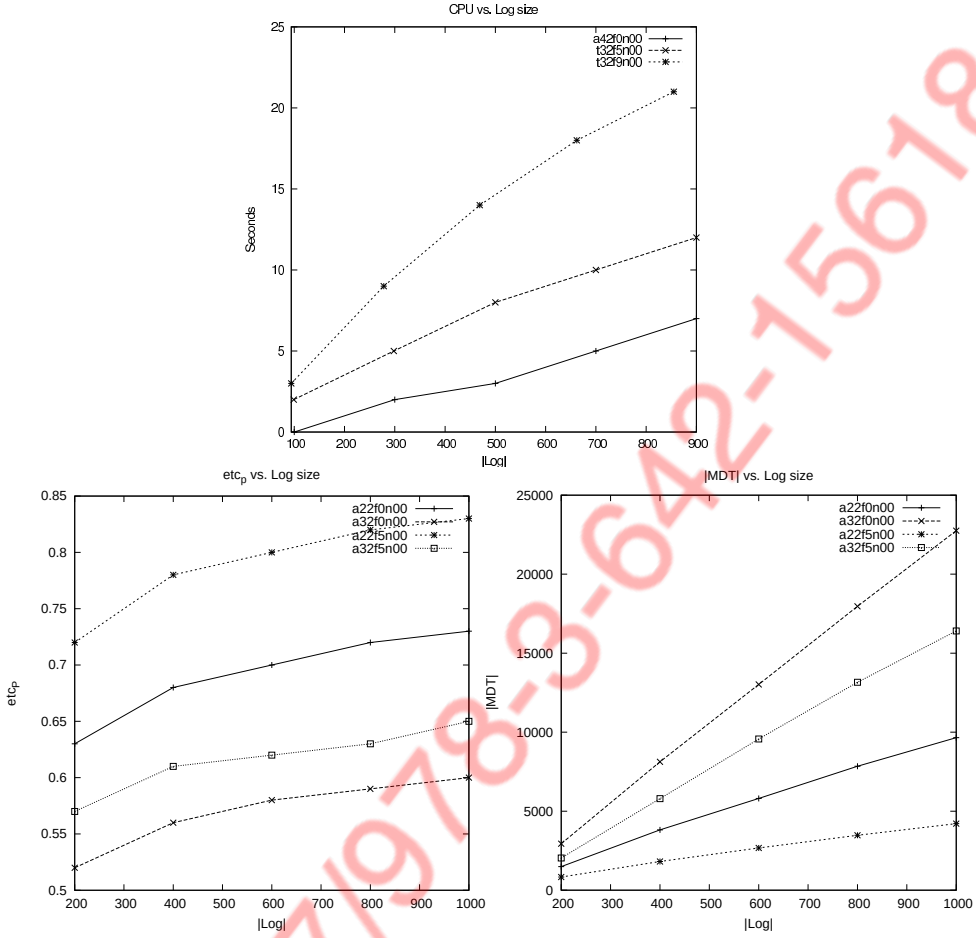


Fig. 7. CPU and etc_P versus log size for some large benchmarks.

avoids the potential state explosion that might arise when dealing with large and highly concurrent nets. The theory has been implemented as a plugin within ProM 6 and experimental results are promising. The technique is enriched with the detection of minimal disconformant traces that may be the starting point for extension of the model to better represent the log.

Acknowledgments We would like to thank M. Solé, A. Rozinat and ProM developers for their help. This work has been supported by the project FORMALISM (TIN2007-66523), and a grant by Intel Corporation.

References

1. Process mining. www.processmining.org.

2. R. Bergenthum, J. Desel, R. Lorenz, and S.Mauser. Process mining based on regions of languages. In *Proc. 5th Int. Conf. on Business Process Management*, pages 375–383, Sept. 2007.
3. T. Calders, C. W. Günther, M. Pechenizkiy, and A. Rozinat. Using minimum description length for process mining. In *SAC*, pages 1451–1455. ACM, 2009.
4. J. Carmona, J. Cortadella, and M. Kishinevsky. A region-based algorithm for discovering Petri nets from event logs. In *BPM*, volume 5240 of *LNCS*, pages 358–373. Springer, 2008.
5. A. K. A. de Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters. Quantifying process equivalence based on observed behavior. *Data Knowl. Eng.*, 64(1):55–74, 2008.
6. G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006.
7. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
8. A. Rozinat, A. K. A. de Medeiros, C. W. Günther, A. J. M. M. Weijters, and W. M. P. van der Aalst. The need for a process mining evaluation framework in research and practice. In *BPM Workshops*, volume 4928 of *LNCS*, pages 84–89, 2007.
9. A. Rozinat, A. K. A. de Medeiros, C. W. Günther, A. J. M. M. Weijters, and W. M. P. van der Aalst. Towards an evaluation framework for process mining algorithms. *BPM Center Report BPM-07-06, BPMcenter. org*, 2007.
10. A. Rozinat and W. M. P. van der Aalst. Conformance testing: measuring the alignment between event logs and process models. *BETA Working Paper Series, Eindhoven University of Technology*, WP 144:203–210, 2005.
11. A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.
12. M. Solé and J. Carmona. Process mining from a basis of state regions. In *International Conference on Application and Theory of Petri Nets and other Models of Concurrency (Petri Nets 2010)*, LNCS. Springer, 2010.
13. W. M. P. van der Aalst. Process-aware information systems: Lessons to be learned from process mining. *T. Petri Nets and Other Models of Concurrency*, 2:1–26, 2009.
14. W. M. P. van der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters. Genetic process mining. In *ICATPN*, volume 3536 of *LNCS*, pages 48–69. Springer, 2005.
15. W. M. P. van der Aalst, V. Rubin, H. M. W. E. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 2009.
16. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.
17. W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, September 2004.
18. J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In *Petri Nets*, volume 5062 of *LNCS*, pages 368–387. Springer, 2008.
19. B. F. van Dongen, J. Mendling, and W. M. P. van der Aalst. Structural patterns for soundness of business process models. In *EDOC*, pages 116–128. IEEE Computer Society, 2006.