Towards Robust Conformance Checking

A. Adriansyah[1], B.F. van Dongen[1], and W.M.P. van der Aalst[1]

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
`{a.adriansyah,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl`

Summary. The growing complexity of processes in many organizations stimulates the adoption of business process management (BPM) techniques. Process models typically lie at the basis of these techniques and generally, the assumption is made that the operational business processes as they are taking place in practice conform to these models. However, recent experience has shown that this often isn't the case. Therefore, the problem of checking to what extent the operational process conforms to the process model is increasingly important.

In this paper, we present a robust approach to get insights into the conformance of an operational process to a given process model. We use logs that carry information about which activities have being performed, in which order and we compare these logs to an abstract model. We do not only provide several different conformance metrics, but we show an efficient implementation for the calculation of these metrics.

Our approach has been implemented in the ProM framework[1], evaluated using simulated event logs and compared against an existing conformance technique based on Petri nets.

Key words: Process mining, conformance, process analysis

1 Introduction

The growing complexity of business processes has triggered a wide usage of process models. The emergence of many systems that base their functions around process models such as BPM (Business Process Management), BAM (Business Activity Monitoring), and BPI (Business Process Intelligence) shows how important process models are to organizations. Models are not only used as instruments to describe existing processes. They have become an integral part of process optimization, monitoring, and even auditing [14].

Unfortunately, process models do not always conform to reality. Even in automated processes, deviations can occur [10]. In some other cases, it is desirable to have models that allow for flexibility [8]. Hence, before performing any sort of process analysis based on process models, it is important to know in advance to what extent the models *conform* to reality.

---

[1] see `http://www.processmining.org`

Conformance checking techniques evaluate the relation between process models and reality presented in form of *event logs*. Given a process model and an event log, the following orthogonal dimensions of conformance can be measured [11]:

Fitness: is the observed behavior captured by the model?

Precision: does the model only allow for behavior that happens in reality?

Generalization: does the model allow for more behavior than encountered in reality?

Structure: does the model have a minimal structure to describe its behavior?

Many existing conformance checking techniques require process models in the form of Petri nets (e.g. [2,7,11]). Given a Petri net and an event log, various conformance metrics are calculated by replaying the log in the net. However, there are at least two drawbacks of Petri net-based conformance checking techniques. First, their metrics are often based on notions that only exist in Petri nets such as tokens and "invisible" transitions and second, Petri-net-based conformance checking techniques may produce "false negative" results. Thus, without in depth knowledge about the language and the algorithm used, it is difficult to utilize the metrics for further analysis.

In Figure 1, we show the result of applying conformance checking technique in [11] to a Petri net and an event log. The event log was obtained by simulating the net, hence it conforms fully to the model. The positive number in each place indicates the number of remaining tokens after replay and negative number indicates missing tokens. The existence of missing and remaining tokens leads to a fitness value less than 100%, although it should be 100% [9]. In this case, the false negative is caused by the invisible transitions that model an OR-split [9].

In [3], problems of Petri-net-based conformance checking are solved by using fuzzy models that have very relaxed semantics. For these fuzzy models, conformance calculations are again made by replaying the log in the model. However, the problem with this conformance is that it is difficult to perform further analysis given a conformance value, because the semantics of fuzzy models are too relaxed.

In this paper, we propose a new way of looking at conformance in the context of event logs. In Section 2, we introduce a model with semantics, such that these

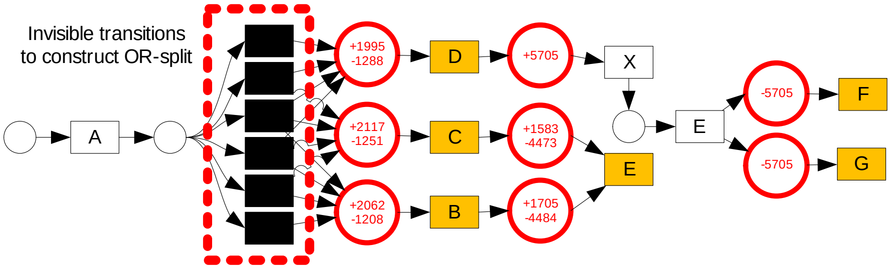


Fig. 1. False negative fitness indication in Petri-net based conformance checker

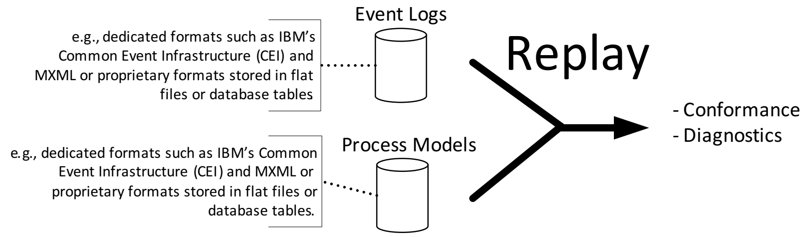

e.g., dedicated formats such as IBM's Common Event Infrastructure (CEI) and MXML or proprietary formats stored in flat files or database tables
Event Logs
Replay
- Conformance
- Diagnostics
e.g., dedicated formats such as IBM's Common Event Infrastructure (CEI) and MXML or proprietary formats stored in flat files or database tables.
Process Models

Fig. 2. Common approach to analyze conformance of process models to logs

semantics are more relaxed than Petri net semantics, but stricter than fuzzy-model semantics. Then, in Section 3, we show how several conformance metrics can be defined for these models. Section 4 shows, for one of these metrics, how to compute one of the metrics for a given log and model and in Section 5, we show some experiments. Section 6 concludes the paper.

2 Preliminaries

Conformance is measured by replaying event logs in process models (see Figure 2). With the existence of various process modeling languages, each with its own semantics, replaying event logs is a unique problem for each process modeling language. Hence, rather than developing a replay algorithm for each existing process modeling language, we use a modeling language that provides an abstraction of existing languages, while maintaining some notion of semantics.

Based on existing process modeling languages (e.g. BPMN[2], EPsC [13], YAWL [5], Heuristic nest [16], Fuzzy models [3], and Petri nets), we propose an extension of *flexible models* [9] to be a process modeling language that captures the essential aspects of existing languages in the control-flow dimension by focusing on activities and their synchronization and/or enabling alternatives.

Before introducing our flexible model, we first introduce some basic graph notation for directed graphs.

Definition 2.1. (Successor/Predecessor nodes in a directed graph) Let $G = (N, E)$ with $E \subseteq N \times N$ be a directed graph. For $n \in N$, we say *successor nodes* of node n as $n \overset{G}{\bullet} = \{n' \in N \mid (n, n') \in E\}$ and *predecessor nodes* of node n as $\overset{G}{\bullet} n = \{n' \in N \mid (n', n) \in E\}$. We omit the superscript G if the context is clear.

Definition 2.2. (Path in a directed graph) Let $G = (N, E)$ be a directed graph. For $n, n' \in N$, there exists a path from n to n' if and only if there is a sequence of edges $\langle (n_1, n_2), (n_2, n_3), ..., (n_{x-1}, n_x) \rangle$ with $x > 1$ where $n_1 = n \wedge n_x = n' \wedge \forall_{1 \leq i < x}\ (n_i, n_{i+1}) \in E$ holds. By $n \rightsquigarrow n'$ we denote that a path from n to n' exists.

---

[2] Business Process Model and Notation `http://www.bpmn.org/`

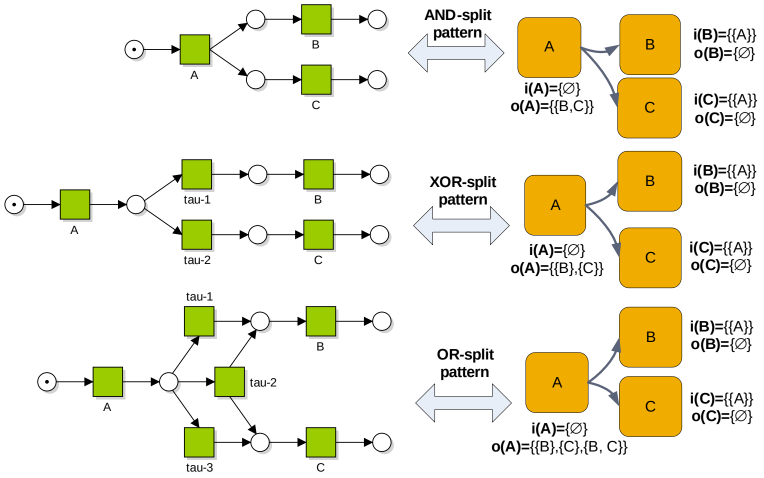


Fig. 3. Petri nets with invisible transitions labeled *tau* (left) and their possible flexible model counterparts (right)

Definition 2.3. (Acyclic graph) Let $G = (N, E)$ be a directed graph. We say that G is an acyclic graph if $\forall_{n \in N} \nexists\; n \rightsquigarrow n$ holds

2.1 Flexible Models

A flexible model is a (potentially cyclic) directed graph consisting of tasks and edges. A task represents an activity in a process. For each task, possible sets of predecessors tasks (indicated by i or ι), and sets of successors tasks (o) are enumerated. An activity in the business process may be represented by more than one task (i.e. *duplicate tasks* are permitted). Using input and output sets of tasks, flexible models can express either strict or relaxed semantics.

The idea of our work is to model processes as flexible models and measure the conformance of an event log and the flexible model. In Figure 3, we illustrate how a flexible model can express patterns that are often needed to model processes in reality, using Petri net as its counterpart. Note that the often-needed OR-split construct can be modeled using flexible model in a straightforward way.

The formal definition of Flexible Model is given as follows:

Definition 2.4. (Flexible Model)
Let A be a set of activities. A flexible model M_A over A is a tuple (T, F, ι, o, β), where:

- T is a finite set of tasks,
- $F \subseteq (T \times T)$ is a set of directed edges connecting tasks,
- $\iota : T \rightarrow \mathcal{P}(\mathcal{P}(T))$ is a function, such that for $t \in T$ and $s \in \iota(t)$, s is a synchronization alternative for t. We require that $\iota(t) \neq \emptyset$ and $\bigcup_{s \in \iota(t)} = \bullet t$.

- $o : T \rightarrow \mathcal{P}(\mathcal{P}(T))$ is a function, such that for $t \in T$ and $s \in o(t)$, s is an enabling alternative of t. We require that $o(t) \neq \emptyset$ and $\bigcup_{s \in o(t)} = t\bullet$.
- $\beta : T \rightarrow A$ is a surjective function mapping tasks to activities, i.e. each activity appears as at least one task in the model.

It is important to realize that flexible models can be obtained using several approaches, e.g. by discovering them directly from event log, by converting existing process models, or by modeling them manually. In this paper, we assume that such model already exists for a given event log.

Flexible models are intended to be models with a formal semantics. However, we do not provide execution semantics. Instead, we later provide semantics only in the context of a case, i.e. for a given sequence of task executions, we can say whether or not this sequence is a (partial) execution of a flexible model. Therefore, we formally introduce the notion of a partial and full instance of a flexible model.

Definition 2.5. (Partial instance of flexible model) Let A be a set of activities and $M_A = (T, F, \iota, o, \beta)$ be a flexible model over A. Let $I = (N, R, \lambda)$ be a tuple where N is a set of unique task instances, $R \subseteq N \times N$ is a set of edges such that (N, R) is an acyclic graph, and $\lambda : N \rightarrow T$ is a function mapping the elements of N to their corresponding tasks. We say I is a partial instance of M_A if and only if the following holds:

- $\forall_{(n,n') \in R}\ (\lambda(n), \lambda(n')) \in F$,
- $\forall_{n \in N} \forall_{n_1, n_2 \in n\bullet}\ n_1 \neq n_2 \implies \lambda(n_1) \neq \lambda(n_2)$
- $\forall_{n \in N} \forall_{n_1, n_2 \in \bullet n}\ n_1 \neq n_2 \implies \lambda(n_1) \neq \lambda(n_2)$
- $\forall_{n \in N}\ \exists_{s \in o(\lambda(n))} \lambda(n\bullet) \subseteq s \wedge \forall_{n_1, n_2 \in \bullet n}\ \lambda(n_1) = \lambda(n_2) \implies n_1 = n_2$, and
- $\forall_{n \in N}\ \exists_{s \in \iota(\lambda(n))} \lambda(\bullet n) \subseteq s \wedge \forall_{n_1, n_2 \in n\bullet}\ \lambda(n_1) = \lambda(n_2) \implies n_1 = n_2$

A partial instance of a flexible model is a partial order of task instances, such that the edges respect the existence of edges in the original flexible model. Furthermore, the input and output sets as defined in the flexible model are partly respected. Once all input and output sets are fully respected, we say that an instance is complete.

Definition 2.6. (Complete instance of flexible model) Let A be a set of activities and $M_A = (T, F, \iota, o, \beta)$ be a flexible model over A. Let $I = (N, R, \lambda)$ be a partial instance of M_A. We say I is a complete instance of M_A if and only if the following holds:

- $\forall_{n \in N}\ \exists_{s \in o(\lambda(n))} \lambda(n\bullet) = s$, and
- $\forall_{n \in N}\ \exists_{s \in \iota(\lambda(n))} \lambda(\bullet n) = s$

2.2 Event Logs

As described in Figure 2, we also need event logs in order to check for conformance. An event log records which activities have been performed in a business process. Hence, we formalize log-related terms as follows:

Definition 2.7. (Event logs)
Let A be a set of activities. An event log over A is defined as $L_A = (E, C, \alpha, \gamma, \succ)$, where:

- E is a finite set of events,
- C is a finite set of cases,
- $\alpha : E \to A$ is a function relating each event to an activity,
- $\gamma : E \to C$ is a surjective function relating each event to a case.
- $\succ \subseteq E \times E$ imposes a total ordering on the events in E. The ordering is typically based on timestamps of events.

Definition 2.8. (Case events)
Let A be a set of activities and $L_A = (E, C, \alpha, \gamma, \succ)$ be an event log over A. Let $c \in C$ be a case identifier. With E_c, we denote the events of case c, i.e. $E_c = \{e \in E \mid \gamma(e) = c\}$. As $\succ$ imposes a total ordering on E, it also imposes a total ordering on E_c.

In the following section, we show how several conformance metrics can be defined for the combination of an event log and a flexible model.

3 Conformance in Flexible Model

A flexible model as defined in Definition 2.4 is not executable. Given a task in a flexible model, we cannot provide insights into which tasks can be executed next such that in the end, a complete instance of this flexible model will be constructed. However, this is not the goal of flexible models. Instead, we aim at deciding if and to what extent a given event log can be replayed in a flexible model, i.e. for a *given* execution, we need to say whether or not this execution conforms to the flexible model.

In this paper, we focus on conformance between a model and a log that refer to the same set of activities. Through standard filtering techniques, a log can always be pre-processed to meet this requirement for a flexible model.

For a log and a flexible model, we need to define a match between a partial instance and a case, i.e. for a given case, we need to define a *class of* partial instances that this case can correspond to. At this point, we do not provide insights into constructing instances. However, in Section 4, we show how to obtain an element of the class of partial instances that matches a case and minimizes a specific conformance metric.

Definition 3.1. (Matching case and flexible model instance)
Let A be a set of activities, let $L_A = (E, C, \alpha, \gamma, \succ)$ be an event log over A and let $M_A = (T, F, \iota, o, \beta)$ be a flexible model over A. Let $c \in C$ be a case and let $I = (N, R, \lambda)$ be a partial instance of M_A.

We say that c and I match if and only if:

- $E_c = N$, i.e. each event is a node in the partial instance,

- $\forall_{e,e' \in E_C}$ $(e \succ e') \Rightarrow (e \not\rightarrowtail e')$, i.e. the ordering of events in the log is respected in the instance, and
- $\forall_{e \in E_c} \lambda(e) \in \{t \in T \mid \beta(t) = \alpha(e)\}$, i.e. each event is mapped to a task that corresponds to the activity represented by this event.

We use I_c to denote an arbitrary instance I matching c and we use $\Im_c$ to denote the (possibly infinite) set of all instances matching c.

In order to reason about matching instances for a case, we show that at least one matching instance always exists, i.e. $\Im_c \neq \emptyset$.

Lemma 3.2. (Matching partial instance exists for any case)
Let A be a set of activities, let $L_A = (E, C, \alpha, \gamma, \succ)$ be an event log over A and let $M_A = (T, F, \iota, o, \beta)$ be a flexible model over A. Let $c \in C$ be a case and let $I = (E_c, \emptyset, \lambda)$ be a partial instance of M_A. We show that I matches c (i.e. $I \in \Im_c$ for any λ that satisfies $\forall_{e \in E_c} \lambda(e) \in \{t \in T \mid \beta(t) = \alpha(e)\}$.

Proof. It is trivial to see that I follows definition 2.6. Furthermore, since there are no edges, we know that for all $e, e' \in E_c$ holds that $e \not\rightarrowtail e'$. Since $N = E_c$ and $\forall_{e \in E_c} \lambda(e) \in \{t \in T \mid \beta(t) = \alpha(e)\}$, we know that I is a matching partial flexible model instance, hence $I \in \Im_c$. $\square$

As stated before, in a partial instance of a flexible model, there can be instances of tasks for which the input conditions are not completely satisfied. If such an instance matches a case, then there are events in the log that correspond to these task instances. We call these events *unsatisfied*.

Definition 3.3. (Unsatisfied events) Let A be a set of activities, let $L_A = (E, C, \alpha, \gamma, \succ)$ be an event log over A and let $M_A = (T, F, \iota, o, \beta)$ be a flexible model over A. Let $c \in C$ be a case and let $I_c = (E_c, R, \lambda)$ be a partial instance of M_A matching c.

We say that $e \in E_c$ is an unsatisfied event if and only if $\lambda(\overset{I_c}{\bullet} e) \notin \iota(\lambda(e))$. We denote the set of unsatisfied events by $E^{us}_{I_c}$.

Similar to unsatisfied events, we define *unhandled events*.

Definition 3.4. (Unhandled events) Let A be a set of activities, let $L_A = (E, C, \alpha, \gamma, \succ)$ be an event log over A and let $M_A = (T, F, \iota, o, \beta)$ be a flexible model over A. Let $c \in C$ be a case and let $I_c = (E_c, R, \lambda)$ be a partial instance of M_A matching c.

We say that $e \in E_c$ is an unhandled event if and only if $\lambda(e \overset{I_c}{\bullet}) \notin o(\lambda(e))$. We denote the set of unhandled events by $E^{uh}_{I_c}$.

Using the notion of unhandled and unsatisfied events, we define several conformance metrics.

3.1 Conformance Metrics

Given a flexible model and a log, we can always obtain a matching instance for each case in the model. In this section, we define several metrics to express the conformance between a case and a matching instance. In Section 4, we use these metrics to construct a matching instance that maximizes conformance for each case.

Definition 3.5. (Single case fitness metrics) Let A be a set of activities, let $L_A = (E, C, \alpha, \gamma, \succ)$ be an event log over A and let $M_A = (T, F, \iota, o, \beta)$ be a flexible model over A. Let $c \in C$ be a case. We define two fitness metrics for matching instances as follow:

Case absolute fitness , $f_c^{abs} : \Im_c \to \{0, 1\}$, is a function that returns 1 only if there are no unsatisfied events in the case.

$$f_c^{abs}(I_c) = \begin{cases} 0 \text{ if } |E_{I_c}^{us}| > 0 \text{ holds, else} \\ 1 \text{ if previous condition doesn't hold} \end{cases}$$

Task ratio fitness , $f_c^{rat}(I_c) : \Im_c \to [0, 1]$, is a function that indicate the ratio between unsatisfied events and total number of events in a case.

$$f_c^{rat}(I_c) = 1 - \frac{|E_{I_c}^{us}|}{|E_c|}$$

The *absolute* fitness metric states that a case is only fitting a flexible model instance if this instance does not have unsatisfied events. On the other hand, *task ratio* fitness provides the percentage of events that are unsatisfied. We extend these two fitness metrics to the level of flexible models as follows.

Definition 3.6. (Fitness metrics) Let A be a set of activities, let $L_A = (E, C, \alpha, \gamma, \succ)$ be an event log over A and let $M_A = (T, F, \iota, o, \beta)$ be a flexible model over A.

Our fitness metrics are defined as follow:

Absolute fitness $f^{abs} \in [0, 1]$ indicates the average maximal absolute fitness.

$$f^{abs} = \frac{\sum_{c \in C} \max_{I_c \in \Im_c} f_c^{abs}(I_c)}{|C|},$$

Task ratio fitness $f^{rat} \in [0, 1]$ indicates the average maximal task ratio fitness.

$$f^{rat} = \frac{\sum_{c \in C} \max_{I_c \in \Im_c} f_c^{rat}(I_c)}{|C|},$$

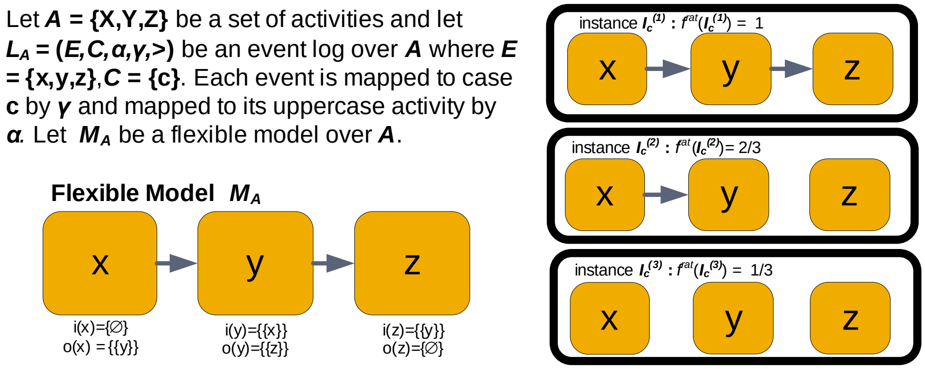
Event fitness $f^{evt} \in [0, 1]$ indicates the maximal ratio of events in the log that can be satisfied by some instance.

$$f^{evt} = \frac{\sum_{c \in C} \max_{I_c \in \Im_c} f_c^{rat}(I_c) \cdot |E_c|}{|E|}$$

So far, we defined several fitness metrics that can be computed only when for each case in the log, we can obtain a matching (partial) instance of the flexible model that maximizes any of our two case-based fitness functions. Therefore, in the following section, we present an algorithm that constructs a partial model instance that maximizes the fitness metrics we defined.

4 Constructing Matching Partial Model Instance

Given a flexible model and an event log over a set of activities, our fitness values depend on a matching (partial) model instance for each case in the log. From Definition 3.6, it is clear that for each case, we need to construct a partial model



Let $A = \{X,Y,Z\}$ be a set of activities and let $L_A = (E,C,\alpha,\gamma,>)$ be an event log over A where $E = \{x,y,z\}$, $C = \{c\}$. Each event is mapped to case **c** by γ and mapped to its uppercase activity by α. Let M_A be a flexible model over A.

Flexible Model M_A
x
y
z
i(x)={∅}
o(x) ={{y}}
i(y)={{x}}
o(y)={{z}}
i(z)={{y}}
o(z)={∅}
instance $I_c^{(1)}$: $f^{rat}(I_c^{(1)})$ = 1
instance $I_c^{(2)}$: $f^{rat}(I_c^{(2)})$= 2/3
instance $I_c^{(3)}$: $f^{rat}(I_c^{(3)})$ = 1/3
x
y
z

Fig. 4. Matching partial instances given a case and a flexible model

instance that maximizes the value of the case based fitness metrics defined in Definition 3.5. In this section, we introduce an algorithm that achieves this.

As an illustration, consider the flexible model and events of a case as shown in Figure 4. More than one matching partial model instance can be generated from the model, each of which has a different fitness value for the task ratio fitness f^{rat}. Since the model can capture the behavior of the case as shown by instance 1, the fitness value should be 1. Hence, an instance with task ration fitness 1 should be selected as the basis for fitness calculation.

From Definition 3.5, it can easily be concluded that maximum fitness will be achieved if the number of unsatisfied events of a case in an instance is minimal. According to Definition 2.5, all predecessor/successor relations between task instances should honor the predecessor/successor relation between tasks in the original model. However, from Definition 3.3, we can see that only the predecessor relation matters for the fitness metrics we defined. Therefore, the selection of predecessors of task instances is important to minimize the number of unsatisfied events.

Given a case in an event log and a flexible model, we have shown that the set of matching partial instances for that case is non-empty (Lemma 4). Furthermore it is easy to see that the number of matching partial instances is finite (in fact, it is at most exponential in the number of events in the case). Although in theory this implies that we could iterate all instances to find one maximizing fitness, this would be infeasible for real-life event logs. Therefore, we introduce a search algorithm, based on the A* algorithm [4], that guarantees us to find an instance that minimizes the number of unsatisfied events and hence maximizes the case-based fitness metric.

The A* algorithm was developed to find the shortest path from a *source* node to a *target* node in a weighted directed graph. Given a directed graph $G = (N, E)$ where N is a set of nodes and $E : N \times N$ is a set of directed arcs, A* heuristic relies on cost function $f(n) = g(n) + h(n)$, where $n \in N$ is a node in the graph. Function $g(n)$ returns the total cost so far to reach n from a source node n_{src}, and heuristic function $h(n)$ returns estimation cost from node n to target node n_{trg}. Function h should not return a value that overestimates the cost to reach the goal, and cost function f should exhibit incremental monotonicity [12]. If functions with such properties are used, the algorithm has been proven to be complete and optimal (i.e. return path from n_{src} to n_{trg} with the minimum value of $f(n)$) [1,12].

The A* algorithm can be used in the construction of matching partial instances which maximize a case's fitness value. The sketch of the approach is given as follows. We start our search from the matching partial instance that always exists, i.e. a matching partial instance that contains no edges, but only the events as nodes. The, we consider all events one by one, in the order provided by the log. For each event, we try to satisfy one of the synchronization alternatives defined in the flexible model (i.e. we need to consider all tasks in the flexible model that refer to the same activity as the event). In order to satisfy a synchronization alternative, we add edges from earlier events to the event under investigation, while maintaining the restrictions on the enabling alternatives provided by the flexible model. If no synchronization alternative can be satisfied, we do not add any edges.

Obviously the algorithm sketched above could be used to generate all matching partial instances. However, we use the A* algorithm to limit our search in the following way. First, we define the target function f as the *number of events in the case plus the number of unsatisfied events so far*. As the number of events in the case is fixed, minimizing this will also minimize the number of unsatisfied events. Furthermore, function g represents the number of events considered so-far (the depth of the search tree) plus the number of unsatisfied events so far and h provides the number of events still to consider.

Since during our search, no edges are ever added *to* an earlier event, we know that once an event was unsatisfied, it will never be satisfied later. Hence, function f is strictly increasing as the search progresses and the A* algorithm guarantees us that we find a matching partial instance with minimal number of unsatisfied events.

5 Experiments

We implemented our calculation approach with A* heuristic in the ProM framework. In addition to conformance values such as the fitness metrics presented in this paper, other useful information obtained from replaying the log in the model is projected onto the original flexible model [15].

Using our implementation, we compared the results of our approach to an existing Petri net based approach proposed in [11] that is also the basis for [7]. The goal of this experiment is to show that our approach returns the right fitness values, where Petri net based approach does not.

To perform our experiment, five event logs were generated from various Petri nets, each with OR-split or OR-join constructs, duplicate transitions, or loop constructs. For modeling the nets and generating logs, we used CPN Tools [6].

Table 1. Experiment results

Log ID	# Case	# Evts	f^{abs}	f^{rat}	f^{evt}	Petri net based.
OrSJn1	5000	18556	1	1	1	0.77
OrSJn2	10000	37153	1	1	1	0.77
OrS1	5000	26323	1	1	1	0.89
OrS2	10000	52762	1	1	1	0.89
Loop	10000	115384	1	1	1	0.89

The conformance of each log is measured against both the original Petri net and a flexible model that is the counterpart of that Petri net. Each log has a size reasonable for simulating real-life data ($\geq$ 5000 cases).

The experiments results are shown in Table 1. As shown in the table (columns 4,5 and 6), our conformance metrics return 1 for all logs and all metrics. This is expected, as the models were used to generate the log. However, when the same logs are checked against the Petri nets the fitness is less than 1, due to the inability of existing algorithms to handle the selected constructs (for example due to the detection of false negatives).

6 Conclusion and Future Work

In this paper, we provides a robust method for calculating conformance between a log and a process model. First, we introduced flexible models that provide an abstraction of many languages and allow for the modeling of complex control flow constructs, such as OR-split/joins and multiple tasks that represent the same activity. We provided semantics for these models, but without specifying how to execute them. Instead, we showed that in the context of a case that has been recorded in the log, we can construct instances of the model that maximize certain conformance metrics. Finally, using experiments on simulated data (comparable in size to real-life data sets), we have shown that our approach calculates fitness correctly in the presence of complex constructs, where existing approaches do not.

The work presented in this paper provides a solid basis for robust conformance checking. Since our flexible models do not have executable semantics, we do not rely on state-space exploration (which is required in Petri-net based conformance checking).

In the future, we plan to extend this work by defining metrics that do not only capture the unsatisfied events, but also the unhandled events. Furthermore, we aim at developing conformance metrics related to other aspects of conformance, such as appropriateness. Furthermore, there is a need to identify the “skipping” of activities in the log, i.e. by identifying which tasks were executed but not logged.

References

1. R. Dechter and J. Pearl. Generalized Best-first Search Strategies and the Optimality of A*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.
2. S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *The Journal of Machine Learning Research*, 10:1305–1340, 2009.
3. C.W. Gunther. *Process Mining in Flexible Environments*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2009.
4. P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths in Graphs. *IEEE Trans. Syst. Sci. and Cybernetics*, SSC-4(2):100–107, 1968.
5. A.H.M. Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation*. Springer-Verlag, 2010.
6. J. Kurt, L. M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(3-4):213–254, June 2007.
7. J. Muoz-Gama and J. Carmona. A Fresh Look at Precision in Process Conformances. In *Proceedings of the 8th International Conference on Business Process Management (BPM 2010)*, 2010.
8. M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Control to Users*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2008.
9. A. Rozinat. *Process Mining: Conformance and Extension*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2010.
10. A. Rozinat, I.S.M. de Jong, C.W. Gunther, and W.M.P. van der Aalst. Conformance Analysis of ASML's Test Process. In S. Sadiq, M. Indulska, M. zur Muehlen, E. Dubois, and P. Johannesson, editors, *Proceedings of the Second International Workshop on Governance, Risk and Compliance (GRCIS'09)*, volume 459, pages 1–15. CEUR-WS.org, 2009.
11. A. Rozinat, A.K. Alves de Medeiros, C.W. Gunther, 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 A. ter Hofstede, B. Benatallah, and H.Y. Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 84–89. Springer-Verlag, Berlin, 2008.
12. S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1st edition, 1995.
13. W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.
14. W.M.P. van der Aalst, K.M. van Hee, J.M. van der Werf, and M. Verdonk. Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. *Computer*, 43:90–93, March 2010.
15. B.F. van Dongen and A. Adriansyah. Process Mining: Fuzzy Clustering and Performance Visualization. In *Business Process Management Workshops*, volume 43, pages 158–169. Springer Berlin Heidelberg, 2009.
16. A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process Mining with the Heuristics Miner-algorithm. Technical report, Eindhoven University of Technology, Eindhoven, 2006. BETA Working Paper Series, WP 166.