Computing manuscript No.
(will be inserted by the editor)

Balanced Multi-Perspective Checking of Process Conformance

Felix Mannhardt · Massimiliano de Leoni · Hajo A. Reijers · Wil M.P. van der Aalst

Received: date / Accepted: date

Abstract Organizations maintain process models that describe or prescribe how cases (e.g., orders) are handled. However, reality may not agree with what is modeled. Conformance checking techniques reveal and diagnose differences between the behavior that is modeled and what is observed. Existing conformance checking approaches tend to focus on the control-flow in a process, while abstracting from data dependencies, resource assignments, and time constraints. Even in those situations when other perspectives are considered, the control-flow is aligned first, i.e., priority is given to this perspective. Data dependencies, resource assignments, and time constraints are only considered as "second-class citizens", which can lead to misleading conformance diagnostics. In this paper, a novel algorithm is proposed that balances the deviations with respect to *all* these perspectives based on a customizable cost function. Evaluations using both synthetic and real data sets show that a multi-perspective approach is indeed feasible and may help to circumvent misleading results as generated by classical single-perspective or staged approaches.

Keywords Process Mining · Data Petri Nets · Multi-Perspective Conformance Checking · Log-Process Alignment

The work reported in this paper is partly supported by the Eurostars-Eureka project PROMPT (E!6696).

F. Mannhardt · M. de Leoni · H.A. Reijers · W.M.P. van der Aalst
Technische Universiteit Eindhoven, Eindhoven, The Netherlands.
E-mail: {f.mannhardt, m.d.leoni, h.a.reijers, w.m.p.v.d.aalst}@tue.nl

F. Mannhardt · H.A. Reijers
Perceptive Software, Apeldoorn, The Netherlands.

M. de Leoni
University of Padua, Padova, Italy.

W.M.P. van der Aalst
International Laboratory of Process-Aware Information Systems, National Research University Higher School of Economics, Moscow, Russia.

Mathematics Subject Classification (2000) 68U35

1 Introduction

The practical relevance of *process mining* is on the rise as event data is readily available due to advances in data monitoring and storage. Process mining techniques aim to discover, monitor and improve real processes by extracting knowledge from event logs [1]. The two most prominent process mining tasks are: (i) *process discovery*: learning a process model from example behavior recorded in an event log, and (ii) *conformance checking*: diagnosing and quantifying discrepancies between observed behavior and modeled behavior. This paper focuses on conformance checking while considering multiple perspectives (i.e. control-flow, data, resources, time) at the same time. Deviations identified using conformance checking may, for example, point at agents in a process using undesirable work-arounds, activities that are often executed too late for a particular group of customers, or violations of the four-eyes principle for cases that follow a particular path.

Up to this point, conformance checking techniques have almost exclusively focused on the *control-flow perspective* [1,22,7]. This means that the order of steps is being analyzed to determine the conformance between prescribed and actual behavior. Of more recent date is the approach described in [13], which extends such a view with other important perspectives that may be subject to quality requirements: data, resources, and time. This paper follows the latter, multi-perspective approach. However, distinctively different from the work in [13], we do not consider the control-flow first and the other perspectives only at a later stage. As will be shown, following [13] may provide misleading results if control-flow and the other perspectives are closely inter-related.

Compared to existing approaches, this paper provides two important contributions. First of all, we address the potential concern that a multi-dimensional approach may not be feasible to apply to real-life event logs, considering the longer computations that are required compared to a single-perspective approach. Secondly, and in contrast to the multi-dimensional approach in [13] in which control-flow is considered as the most important perspective in identifying deviations, the proposed approach in this paper allows for balancing the different perspectives in a fully customizable manner. By doing so, we claim that we can provide more meaningful analysis results than by fixing one perspective as the most dominant one.

Our claims have been validated through an empirical evaluation using a real-life event log and a process model, both of which are provided by the local police of an Italian city. The log contains information about more than 140.000 road-traffic fines. Events relate to notifications, payments, and appeals. The evaluation shows that the approach described in [13] may return unlikely or even wrong explanations from a business viewpoint. By that we mean that, although formally correct, the explanations are not possible if the specific context of the business process is taken into consideration. As will be shown,

this is caused by the fact that the control-flow is considered initially without any attention for the other perspectives, which are considered separately in a second stage. Using the techniques and tools developed as part of our research, the analysis in this evaluation took roughly 20 minutes, which seems acceptable considering the large size of the event log and the complexity of the problem. To further support the feasibility of the approach, we also used a number of synthetic event logs to evaluate its performance for logs of various sizes.

Against this backdrop, the structure of this paper is as follows. We will provide a motivating example in Sect. 2 and some essential background information in Sect. 3. The main explanation of our approach can be found in Sect. 4, whereas Sect. 5 contains a brief description of the implementation. Sect. 6 presents the outcome of our real-life case study as well as the analyses of the synthetic event logs. Sect. 7 provides a review of the existing literature on conformance checking, in particular emphasizing the distinctive nature of the multi-perspective conformance checking we propose against the state of the art. Finally, our concluding remarks can be found in Sect. 8.

2 Motivating example

A process model describes the life cycle of instances of a business process (also known as *cases*). Process models consists of a number activities as well as constraints that describe which activities must be executed and in which order, depending on the characteristics of a specific case. To clarify the purpose of a process model, let us consider the the model in BPMN notation shown in Figure 1 (taken from [13]), which describes the process of handle credit requests from a credit institute. An initial *Credit Request* activity is executed for each case, which is to be followed by a *Verify* activity.

However, variables are associated to cases, which are subject to modification by the execution of activities. In Fig. 1, it can be seen that during the initial *Credit Request* the *Amount* of the loan is recorded. Also, the *Verify* activity results in a *Verification* result. Paths taken during the execution of a process are often governed by guards and conditions defined over these case variables. In Fig. 1, the choice between the execution of the activities *Renegotiate Request*, *Advanced Assessment*, and *Simple Assessment* is determined on the basis of the values for *Verification* and *Amount*. The *data perspective* of the process refers to the handling and use of such case variables.

In addition, it may also be relevant to capture the behavior of a process in terms of the resource restrictions on the execution of activities. An activity is typically associated with a particular role, i.e., a selected group of resources (also known as *actors* or *agents*). In Fig. 1, it can be seen that the *Verify* activity must be performed by a resource playing the role of *Assistant*. There may be additional assignment rules in place such as the "four-eyes principle", i.e. the same resource cannot execute two related activities for the same case. Adding this *resource perspective* to conformance checking is necessary to detect resource-related deviations.

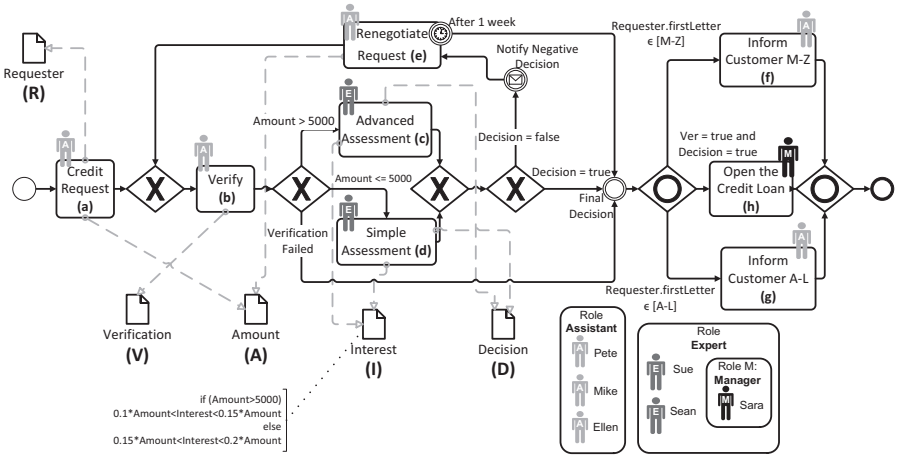


Fig. 1 BPMN diagram describing a process to handle credit requests [13]

In addition to data and resource constraints, there may be time-related constraints that are important to govern. For example, if a renegotiation activity occurs, it needs to follow an assessment within 7 days. The process model in Fig. 1 shows that a final decision is made one week after the *Renegotiate Request* activity is initiated. This adds a *time perspective* to a process model.

If conformance checking would only consider the *control-flow perspective*, the activities themselves and their ordering are the only issues of concern. However, to fully grasp whether a model conforms with reality other perspectives may be important as well. In this paper, we focus on the additional perspectives on data, resource, and time. This paper is grounded in the belief that even though resource and time are separate concerns from a business or modeling perspective, they can be encoded into the data perspective. Therefore, in the remainder of this work, we use the data perspective to capture any perspective different from control-flow. In particular, the resource and time dimensions can be handled through dedicated data elements, as will be shown.

To explain the different perspectives relevant for conformance, consider the following example trace:[1]

$$\sigma_{example} = \langle(\mathbf{a}, \{\mathbf{A} = 3000, \mathbf{R} = \text{Michael}, \mathbf{E_a} = \text{Pete}, \mathbf{T_a} = \texttt{3 Jan}\}),$$
$$(\mathbf{b}, \{\mathbf{V} = false, \mathbf{E_b} = \text{Sue}, \mathbf{T_b} = \texttt{4 Jan}\}), (\mathbf{c}, \{\mathbf{I} = 530, \mathbf{D} = true, \mathbf{E_c} = \text{Sue}, \mathbf{T_c} = \texttt{5 Jan}\}),$$
$$(\mathbf{f}, \{\mathbf{E_f} = \text{Pete}, \mathbf{T_f} = \texttt{17 Jan}\})\rangle.$$

Trace $\sigma_{example}$ consist of 4 events. Lower-case bold letters refer to activities using the mapping in Fig. 1, e.g., a=*Credit Request*. Upper-case bold letters refer to data objects. $A = 3000$ describes that the amount is 3000 (A is a

[1] Notation $(act, \{attr_1 = val_1, \ldots, attr_n = val_n\})$ is used to denote the occurrence of activity act in which variables $attr_1, \ldots, attr_n$ are assigned values $val_1, \ldots, val_n$, respectively.

shorthand for *Amount*) and R = Michael describes that credit request is initiated by Michael (R is a shorthand for *Requester*). E_x and T_x respectively denote the last *executor* of x and the *timestamp* when x was executed last.

Conformance checking techniques that only consider the control-flow perspective *cannot* find any conformance violations for this trace. After all, the trace shows the subsequent execution of activities a, b, c, and f, which is indeed permitted by the process model of Fig. 1. By considering more perspectives, however, more deviations between the process model and this trace can be identified. For example, only by explicitly considering the data perspective, one can detect that activity c is performed even though A's value of 3000 does not meet the guard for this activity (> 5000). Similarly, the value of V is 'false', which would have required activity c to be skipped altogether given the process model of Fig. 1.

The identification of non-conformance in its different forms clearly has value in itself. Nonetheless, organizations are often interested in explanations that can steer measures to improve the quality of the process. What may happen is that alternative explanations exist for a deviating trace. For the identified deviations in our example trace, the explanation may be (1) that all data values were written correctly but that activity c did not need to be performed, or (2) that activity c was performed properly but that the variables A and V were *both* not set correctly.

The approach in [13] seeks for explanations that put the control-flow first. This approach would prefer explanation (2) even when the other perspectives strongly suggest an alternative explanation with more control-flow deviations. Such explanations can be constructed quickly, at the potential expense of the inability to guarantee the optimality of the solution (e.g., the explanation may not be the simplest or most likely). Indeed, explanation (2) requires one to accept that *two* observed data values are incorrect, which in this particular case may actually be less likely than only *one* activity being executed incorrectly. Hence, explanation (1) may be more likely in certain settings. This shows that there are tradeoffs between the different perspectives.

The approach in this paper allows for balancing the control-flow, data, resources, and time perspectives in identifying explanations for deviations. If all perspectives would be equally important it identifies explanation (1) as the best explanation, since it minimizes the number of deviations to explain the erratic trace and as such delivers the simplest explanation among the two alternatives. However, the approach is customizable and one may assign different weights to the different types of deviations. Recall, that we will use data perspective to capture any perspective different from control-flow for the remainder of this paper.

3 Background

In this section, we introduce preliminaries such as the process model, event logs, and alignments.

3.1 Petri Nets with Data

Our conformance-checking technique is independent of the specific formalism used to capture processes in a model. Therefore, BPMN, EPCs, or any other formalism can be employed to represent these perspectives. However, we need a simple modeling language with clear semantics to explain our technique. We use *Petri nets* as a well-known language with clear semantics. To capture the interactions of the control-flow perspective with the other perspectives, we use *Petri nets with data*.

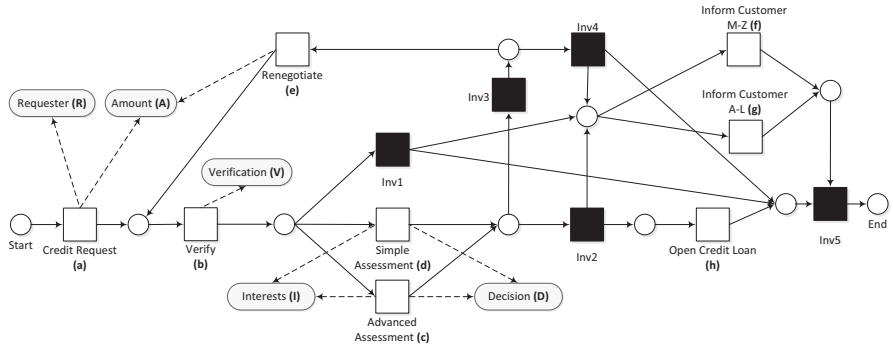
A *Petri net* is a directed bipartite graph containing places and transitions that are connected by arcs. Transitions in a Petri net represent activities that are executed in a process and places represent the states between those activities. For the complete definition and semantics of a Petri net refer to [9]. A *Petri net with data* (DPN-net) is a Petri net in which transitions can write variables [14]. A transition performs *write operations* on a given set of variables and may have a data-dependent guard. Note that, without loss of generality, we do not explicitly consider *read operations* as a source of deviations for this work. The main reason is that they hardly ever are recorded. A transition can fire only if its guard is satisfied and all input places are marked. A guard can be any formula over the process variables using relational operators ($<, >, =$) as well as logical operators such as conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). We denote with $\mathit{Formulas}(X)$ the universe of such formulas defined over a set X of variables.

Definition 1 (DPN-net) A Petri net with data (DPN-net) $N = (P, T, F, V, U, Val, W, G)$ consists of:

- a set of places P;
- a set of transitions T;
- a flow-relation $F \subseteq (P \times T) \cup (T \times P)$;
- a set V of variables;
- a set U of variable values;
- a function $\mathit{Val} : V \to 2^U$ that defines the values admissible for each variable $v \in V$, i.e. $\mathit{Val}(v)$ is the domain of variable v;
- a write function $W : T \to 2^V$ that labels each transition with a set of *write operations*, i.e. with the set of variables whose value needs to be written/updated;
- a guard function $G : T \to \mathit{Formulas}(V \cup \{v' \mid v \in V\})$ that associates each transition with a different guard.[2]

Some transitions do not correspond to actual pieces of work and are only added for routing purposes. Formally, there is no reason to distinguish such transitions from others. In practical terms, such routing transitions are characterized by not leaving any explicit trails in event logs. That is why these transitions are commonly referred to as *invisible*.

[2] If a transition t should be associated with no guard, we set $G(t) = \text{true}$.



(a) Transitions and places are represented as squares and circles, respectively. Each rounded gray rectangle identifies a different process variable and a dotted line from a transition t to a variable v denotes that $v \in W(t)$. Pictorially, black squares indicate invisible transitions.

Transition	Guard
Advanced Assessment	$Verification = \text{true} \wedge Amount > 5000 \wedge 0.1 < Interest'/Amount < 0.15$
Inv1	$Verification = \text{false}$
Inv2	$Decision = \text{true}$
Inv3	$Decision = \text{false}$
Open Credit Loan	$Verification = \text{true} \wedge Decision = \text{true}$
Inform Customer M-Z	$Requester \geq \text{“}M\text{”}$
Inform Customer A-L	$Requester \leq \text{“}L\text{”}$
Renegotiate	$Amount' \leq Amount$
Simple Assessment	$Verification = \text{true} \wedge Amount \leq 5000 \wedge 0.15 < Interest'/Amount < 0.2$

(b) The guards to encode the data-perspective constraints

Transition	Guard
Credit Request	$E'_a \in \{\text{“}Pete\text{”}, \text{“}Mike\text{”}, \text{“}Ellen\text{”}\}$
Verify	$E'_b \in \{\text{“}Pete\text{”}, \text{“}Mike\text{”}, \text{“}Ellen\text{”}\}$
Simple Assessment	$E'_c \in \{\text{“}Sue\text{”}, \text{“}Sean\text{”}, \text{“}Sara\text{”}\} \wedge E'_c \neq E_c$
Advanced Assessment	$E'_d \in \{\text{“}Sue\text{”}, \text{“}Sean\text{”}, \text{“}Sara\text{”}\} \wedge E'_d \neq E_d$
Renegotiate Request	$E'_e \in \{\text{“}Pete\text{”}, \text{“}Mike\text{”}, \text{“}Ellen\text{”}\} \wedge (T'_e \leq T_c + 7\text{days} \vee T'_e \leq T_d + 7\text{days})$
Open Credit Loan	$E'_h = \text{“}Sara\text{”}$
Inform Customer M-Z	$E'_f \in \{\text{“}Pete\text{”}, \text{“}Mike\text{”}, \text{“}Ellen\text{”}\}$
Inform Customer A-L	$E'_g \in \{\text{“}Pete\text{”}, \text{“}Mike\text{”}, \text{“}Ellen\text{”}\}$

(c) The guards to encode the constraints over resources and time. $E_i \in \{a_1, \ldots, a_n\}$ is a shortcut for expression $E_i = a_1 \vee \ldots \vee E_i = a_n$

Fig. 2 The DPN-net of the working example [13]

The preset of a transition t is the set of its input places: ${}^\bullet t = \{p \in P \mid (p,t) \in F\}$. The postset of t is the set of its output places: $t^\bullet = \{p \in P \mid (t,p) \in F\}$.

When a variable $v \in V$ appears in a guard $G(t)$, it refers to the value just before the occurrence of t. If $v \in W(t)$, it can also appear as v' (i.e., with the prime symbol). In this case, it refers to the value after the occurrence of t.

Example 1 *Fig. 2, taken from [13], shows the DPN-net that models the same process as the BPMN model in Fig. 1. In particular, Fig. 2(a) depicts the control-flow and the write operations. In addition to the variables depicted in the figure, there exists a set of variables to model the resource and time perspec-*

tive, i.e., for each transition t, there are two variables E_t and T_t. Moreover, these two variables are associated with a write operation of t. Fig. 2(b) contains the data-perspective guards $G_d(t)$ for each transition t. When defining guards, we assume that string values can be lexicographically ordered and, hence, it is also possible to use inequality operators (i.e., $<$ and $>$) for strings. To also model the resource and time perspective, a second guard $G_r(t)$ can be associated with each transition t (see Fig. 2(c)). Formally, only one guard $G(t)$ can be assigned to t and, hence, we set $G(t) = G_d(t) \wedge G_r(t)$.

Given a set X, $\mathbb{B}(X)$ denotes the set of all multi-sets over a set X. In addition, given a multiset $M \in \mathbb{B}(X)$, for each $x \in X$, we use $M(x) \in \mathbb{N}$ to indicate the number of duplicates of element x present in M.

Definition 2 (State of a DPN-net) Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net. The set of possible *states* of N is formed by all pairs (M, A) where $M \in \mathbb{B}(P)$, i.e. a multi-set of the places in P, and A is a function that associates a value with each variable, i.e. $A : V \to U \cup \{\bot\}$, with $A(v) \in Val(v) \cup \{\bot\}$.

For any state (M, A), M is known as the marking of Petri net (P, T, F) and we say the marking assigns to each place $p \in P$ a number of tokens. A special value $\bot$ is assigned to variables that have not been initialized.

Each DPN-net defines two special markings M_I, M_F: the initial and final marking. The initial state of a DPN-net is (M_I, A_I) with $A_I(v) = \bot$ undefined for each $v \in V$. A non-empty set of *final states* exists and includes every state (M, A) with $M = M_F$. In any state, zero or more transitions of a DPN-net may be able to fire.

Definition 3 (Valid and Invalid Transition Firings) Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net. A *transition firing* s is a pair $(t, w) \in T \times (V \not\to U)$.[3] Transition firing $s = (t, w)$ is *valid* in a state (M, A) of N if four conditions are satisfied:

1. each place in the preset of t contains at least one token, i.e. for each place $p \in {}^\bullet t$, $M(p) > 0$;
2. t writes new values to the defined set of variables, i.e. $\mathsf{dom}(w) = W(t)$;[4]
3. each variables takes on an admissible value, i.e. for each variable $v \in \mathsf{dom}(w)$, $w(v) \in Val(v)$;
4. guard $G(t)$ evaluates to true with respect to assignment A.

A transition firing s is *invalid* in (M, A) when, at least one of the above conditions is not satisfied.

In the remainder, the set of possible transition firings, both valid and invalid, for a DPN-net N is denoted as S_N, i.e. $S_N = T \times (V \to U)$.

We introduce the following functions to easily access the components of a transition firing $s = (t, w)$: $\#_{vars}(s) = w$ and $\#_{act}(s) = t$. Function $\#_{vars}$ is also overloaded such that $\#_{vars}(s, v) = w(v)$ if $v \in \mathsf{dom}(\#_{vars}(s))$, or

[3] We use $\not\to$ to denote partial functions, i.e. the function's domain is a subset of V.

[4] Note that w is a partial function: $\mathsf{dom}(w)$ is the domain of w.

$\#_{vars}(s, v) = \perp$ if $v \notin \mathsf{dom}(\#_{vars}(s))$. On firing of a transition the DPN-net moves from current state (M, A) to next state (M', A').

Definition 4 (Transitions Between DPN-net States) Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net and (M, A) be a state of the DPN-net. Let $s = (t, w)$ be a valid transition firing in (M, A). Firing s in state (M, A) leads to state (M', A') where:

1. for each place $p \in P$, if $p \in {}^\bullet t$, $M'(p) = M(p) - 1$ or, if $p \in t^\bullet$, $M'(p) = M(p) + 1$ or, otherwise, $M'(p) = M(p)$;
2. for each $v \in V$, $A'(v) = A(v)$ if $\#_{vars}(s, v) = \perp$, otherwise $A'(v) = \#_{vars}(s, v)$.

This is denoted as $(M, A) \xrightarrow{s} (M', A')$.

The concept of single transition firings can easily be extended to sequences $\sigma = \langle s_1, \ldots, s_n \rangle$ of valid transition firings: $(M_0, A_0) \xrightarrow{\sigma} (M_n, A_n)$ corresponds to $(M_0, A_0) \xrightarrow{s_1} (M_1, A_1) \xrightarrow{s_2} \ldots \xrightarrow{s_n} (M_n, A_n)$

In the remainder, $\mathcal{P}_{N,M_I,M_F}$ denotes the **set of valid process traces** of a DPN-net N that leads from the initial marking M_I to final marking M_F: $\mathcal{P}_{N,M_I,M_F} = \{\sigma \in S_N^* \mid \exists_{A'} \ (M_I, A_I) \xrightarrow{\sigma} (M_F, A')\}$. A valid trace σ is such that $(M_I, A_I) \xrightarrow{\sigma} (M', A')$ with (M', A') belonging to the set of final states of N.

The alignment technique described in this paper requires DPN-nets to be *relaxed data sound*: at least one sequence of transition firings exists that leads from the initial state to a final state. We use the word *relaxed* in comparison to stricter notions of data soundness, such as described in [13].

Definition 5 (Relaxed Data Soundness) Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net. Let M_I be the initial marking and let M_F be the final marking. A DPN-net N is *relaxed data sound* iff the set of valid process traces contains at least one valid trace: $\mathcal{P}_{N,M_I,M_F} \neq \varnothing$.

3.2 Alignment of Event Logs and Process Models

Event logs serve as the starting point for process mining. An event log is a multi-set of *traces*. Each trace describes the life-cycle of a particular *process instance* (i.e., a *case*) in terms of the *activities* executed.

Definition 6 (Event Log) Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net. Let S_N be the set of possible transition firings. A trace $\sigma_L \in S_N^*$ is a sequence of transition firings. An event log L over S_N is a multi-set of traces: $L \in \mathbb{B}(S_N^*)$.

Multiple instances of a process may consist of the exact same sequence of transition firings and, hence, result in the same trace. This motivates the definition of an event log as a multi-set. Transition firings in an event log are usually known as *events*. We assume that L only contains events that are part of the DPN-net N. Any event referring to a transition that is not part of the

Event-Log Trace	Process
a {**A** = 3000, **R** = Michael}	**a** {**A** = 5001, **R** = Michael}
b {**V** = *false*)	**b** {**V** = *true*}
c {**I** = 530, **D** = *true*}	**c** {**I** = 530, **D** = *false*}
≫	**Inv3**
≫	**Inv4**
f {}	**f** {}
≫	**Inv5**

(a) γ_1

Event-Log Trace	Process
a {**A** = 3000, **R** = Michael}	**a** {**A** = 3000, **R** = Michael}
b {**V** = *false*)	**b** {**V** = *false*}
≫	**Inv1**
c {**I** = 530, **D** = *true*}	≫
f {}	**f** {}
≫	**Inv5**

(b) γ_2

Fig. 3 Examples of complete alignments of $\sigma_{example}$ and N

process model is filtered out. Please note that transition firings in L are not necessarily in line with the behavior that is described by DPN-net N.

Conformance checking requires an *alignment* of event log L and process model N. The events in the event log need to be related to transitions in the model, and vice versa. Such an alignment shows how the event log can be replayed on the process model. Building this alignment is far from trivial, since the log may deviate from the model at an arbitrary number of places.

We need to relate "moves" in the log to "moves" in the model in order to establish an alignment between a process model and an event log. However, it may be that some of the moves in the log cannot be mimicked by the model and vice versa. We explicitly denote such "no moves" by $\gg$. Fig. 3(a) and Fig. 3(b) show alignments of the process model in Fig. 2 and the log trace $\sigma_{example}$ from Sect. 2. For convenience, we introduce the set $S_N^{\gg} = S_N \cup \{\gg\}$.

Definition 7 (Alignments) Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net with initial marking M_I and final marking M_F. A legal move in an *alignment* is represented by a pair $(s_L, s_M) \in (S_N^{\gg} \times S_N^{\gg}) \setminus \{(\gg, \gg)\}$ such that

- (s_L, s_M) is a *move in log* if $s_L \in S_N$ and $s_M = \gg$,
- (s_L, s_M) is a *move in model* if $s_L = \gg$ and $s_M \in S_N$,
- (s_L, s_M) is a *move in both with correct write operations* if $s_L \in S_N$, $s_M \in S_N$ and $\#_{act}(s_L) = \#_{act}(s_M)$ and $\forall v \in V\ \#_{vars}(s_L, v) = \#_{vars}(s_M, v)$,
- (s_L, s_M) is a *move in both with incorrect write operations* if $s_L \in S_N$, $s_M \in S_N$ and $\#_{act}(s_L) = \#_{act}(s_M)$ and $\exists v \in V\ \#_{vars}(s_L, v) \neq \#_{vars}(s_M, v)$,

All other moves are considered as *illegal*. $\mathcal{A}_N = \{(s_L, s_M) \in (S_N^{\gg} \times S_N^{\gg}) \setminus \{(\gg, \gg)\} \mid s_L = \gg \vee s_M = \gg \vee \#_{act}(s_L) = \#_{act}(s_M)\}$ is the set of all legal moves.

The *alignment* of two execution traces $\sigma', \sigma'' \in S_N^*$ is a sequence $\gamma \in \mathcal{A}_N^*$ such that, ignoring all occurrences of $\gg$, the projection on the first element yields σ' and the projection on the second yields σ''.

In particular, given a log trace $\sigma_L \in L$, γ is a **complete alignment** of σ_L and N if $\sigma' = \sigma_L$ and $\sigma'' \in \mathcal{P}_{N,M_I,M_F}$. The alignments in Fig. 3(a) and Fig. 3(b) are both complete alignments. In the remainder, given an alignment γ of σ' and σ'', $\gamma|_L = \sigma'$ and $\gamma|_P = \sigma''$ are referred to as the log and the process projection of γ.

Note that we do not aim to find just any complete alignment. Our goal is to find a complete alignment of σ_L and N with minimal deviation cost. In order to define the severity of a deviation, we first introduce a cost function on legal moves and, then, generalize it to alignments. The alignment with the lowest cost is called an optimal alignment.

Definition 8 (Cost Function & Optimal Alignment) Let N and σ_L be a DPN-net and a log trace, respectively. Assuming $\mathcal{A}_N$ as the set of all legal alignment moves, a *cost function* κ assigns a non-negative cost to each legal move: $\mathcal{A}_N \rightarrow \mathbb{R}_0^+$. The *cost of an alignment* γ between σ_L and N is computed as the sum of the cost of all constituent moves: $\mathcal{K}(\gamma) = \sum_{(s_L,s_M)\in\gamma} \kappa(s_L, s_M)$. Alignment γ is an **optimal alignment** if, for any complete alignment γ' of N and σ_L, $\mathcal{K}(\gamma) \leq \mathcal{K}(\gamma')$.

This cost function can be used to favor one type of explanation for deviations over the other. The cost of each legal move depends on the specific model and process domain and, hence, the cost function κ needs to be defined specifically for each setting. Note that an optimal alignment does not need to be unique, i.e. multiple complete alignments with the same minimal cost may exist.

Example 2 *We can define the following cost function for the alignment of log trace $\sigma_{example}$ to the DPN-net N. Let us shortcut $W(\#_{act}(s_M))$ as $W(s_M)$, then we choose κ to be:*[5]

$$\kappa(s_L, s_M) = \begin{cases} 1 & \text{if } (s_L, s_M) \text{ move in log} \\ 1 + |W(s_M)| & \text{if } (s_L, s_M) \text{ move in model and } \#_{act}(s_M) \notin \{\textbf{Inv1}, \ldots, \textbf{Inv5}\} \\ |\{v \in W(s_M) \mid \#_{vars}(s_L, v) \neq \#_{vars}(s_M, v)\}| & \text{if } (s_L, s_M) \text{ move in both with incorrect write operations} \\ 0 & \text{otherwise.} \end{cases}$$

Using this cost function, the cost of the alignment γ_1 in Fig. 3(a) is $\mathcal{K}(\gamma_1) = 3$ and the cost of the alignment γ_2 in Fig. 3(b) is $\mathcal{K}(\gamma_2) = 1$. For this cost function γ_2 is an optimal alignment, as no other alignment with lower cost exists.

It is worth noting that moves for transitions **Inv1**, ..., **Inv5** are always assigned a cost of 0. We previously referred to these as invisible transitions, i.e. they are never recorded in the event log. As such, there is no cost involved in not having observed them.

[5] We indicate the size of a set X as $|X|$.

When focusing on the fitness dimension of conformance, we are not only interested in finding the optimal alignment and, hence, diagnosing where a log trace does not conform with a model. Also, we wish to quantify the fitness level of traces and logs.

For this reason, we introduce a *fitness function* $\mathcal{F} : (S_N^* \times N) \to [0, 1]$. $\mathcal{F}(\sigma_L, N) = 1$ if σ_L can be replayed by the model from the beginning to the end with no discrepancies. Conversely, $\mathcal{F}(\sigma_L, N) = 0$ denotes the poorest level of conformance. $\mathcal{K}$ cannot be used as fitness function directly as we are interested in expressing the fitness level as a number between 0 and 1. Normalization can be done in multiple ways. Here, we divide the cost of the optimal alignment by a reference cost, which approximates the maximum one. Therefore, the *fitness level* of a log trace is defined with respect to some "worst case" scenario.

Definition 9 (Fitness Level) Let $\sigma_L \in S_N^*$ be a log trace and let N be a DPN-net. Let $\gamma_O \in \mathcal{A}_N^*$ be an optimal alignment of σ_L and N and $\gamma_E \in \mathcal{A}_N^*$ be an optimal alignment of the empty trace and N. The fitness level of σ_L and N is defined as follows:

$$\mathcal{F}(\sigma_L, N) = 1 - \frac{\mathcal{K}(\gamma_O)}{\mathcal{K}(\gamma_E) + \sum_{s \in \sigma_L} \kappa\big((s, \gg)\big)}$$

The denominator of the fraction is the reference cost and is computed as (1) the cost of moves in log for all events of σ_L plus (2) the cost of aligning the empty trace. Practical experiments show that the reference cost of aligning σ_L is the maximum cost in a great majority of cases. In general, it is impossible to find an alignment with the maximum cost. For example, if the model contains loops, there is no upper bound.

4 Balanced Multi-perspective Alignments

In this section, we present a technique to construct multi-perspective alignments that are *balanced*. As stated, earlier approaches focused on a single perspective or dealt with the different perspectives sequentially. To illustrate, for the trace $\sigma_{example}$, using cost function $\kappa_{example}$ defined in Example 2, the technique presented in [13] would return the alignment γ_1 in Figure 3(a). This one is sub-optimal in comparison with γ_2.

The remainder of this section requires the introduction of some additional notations. Given two sequences $x = (x_0, \ldots, x_n), y = (y_0, \ldots, y_n)$ the concatenation of both sequences is defined as $x \oplus y = (x_0, \ldots, x_n, y_0, \ldots, y_n)$. $\mathsf{prefix}(x)$ denotes the set of all prefixes of x. If $z \in \mathsf{prefix}(x)$ then a sequence w exists such that $z \oplus w = x$.

4.1 Basic Algorithm

We formulate the problem of finding such an optimal alignment as a search problem in a directed graph and employ the A* algorithm [8] to find a least expensive path through the graph. Let $Z = (Z_V, Z_E)$ be a directed graph with edges weighted based on a predefined cost structure. The A* algorithm, as initially proposed in [8], finds the path with the overall lowest cost from a given source node $v_0 \in Z_V$ to a node of a given goal set, i.e., a set of target nodes $Z_G \subseteq Z_V$. Each node $v \in Z_V$ is associated with a cost that is determined by an evaluation function $f(v) = g(v) + h(v)$, where

- $g : Z_V \to \mathbb{R}^+$ gives the smallest path cost from v_0 to v;
- $h : Z_V \to \mathbb{R}_0^+$ gives an estimate of the smallest path cost from v to any goal node $v_G \in Z_G$ from v.

Function h is admissible if it always underestimates the remaining cost to reach any goal node v_G from v: for each node $v \in Z_V$ and for each goal node $v_G \in Z_G$ that is reachable from v, $h(v) \leq g(v_G) - g(v)$ holds. If h is an admissible function, then A* always returns a path that has the lowest overall cost.

In the remainder, given an alignment $\gamma \in \mathcal{A}_N^*$ with $\gamma = \langle (s_L^1, s_M^1), \ldots, (s_L^n, s_M^n) \rangle$, we define $\mathsf{Ctrl}(\gamma)$ as returning an alignment $\gamma' \in \mathcal{A}_N^*$ with $\gamma' = \langle (p_L^1, p_M^1), \ldots, (p_L^n, p_M^n) \rangle$ that is obtained from γ by removing all write operations (i.e., only the fired transitions are retained). More precisely, for each $1 \leq i \leq n$, if $s_L^i = \gg$, then $p_L^i = \gg$, otherwise $\#_{act}(p_L^i) = \#_{act}(s_L^i)$ and $\mathsf{dom}(\#_{vars}(p_L^i)) = \varnothing$, as well as if $s_M^i = \gg$, then $p_M^i = \gg$, otherwise $\#_{act}(p_M^i) = \#_{act}(s_M^i)$ and $\mathsf{dom}(\#_{vars}(p_M^i)) = \varnothing$.

In order to use A* to find an optimal alignment, the search space needs to be defined along with the cost of search-space nodes:

Definition 10 (Search space and path costs) Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net and σ_L be a log trace. The search space to find an optimal alignment of N and σ_L is a graph $Z = (Z_V, Z_E)$. The set Z_V contains prefixes of complete alignments between σ_L and N:

$$Z_V = \{\gamma \in \mathcal{A}_N^* \mid \gamma|_L \in \mathsf{prefix}(\sigma_L) \wedge \exists \sigma_N \in \mathcal{P}_N : \gamma|_P \in \mathsf{prefix}(\sigma_N)\}.$$

The set Z_E contains all $(\gamma', \gamma'') \in Z_V \times Z_V$, where γ'' is obtained by adding one legal move to γ':

$$Z_E = \{(\gamma', \gamma'') \in Z_V \times Z_V \mid \exists (s_L, s_M) \in \mathcal{A}_N \; s.t. \; \mathsf{Ctrl}(\gamma'') = \mathsf{Ctrl}(\gamma' \oplus (s_L, s_M))\}.$$

The set of goal nodes $Z_G \subseteq Z_V$ contains all complete alignments of σ_L and N:

$$Z_G = \{\gamma_G \in \mathcal{A}_N^* \mid \gamma_G|_L = \sigma_L \wedge \gamma_G|_P \in \mathcal{P}_N\}.$$

In a refinement of the proposal in [3], we add a small negligible cost $\epsilon \in \mathbb{R}^+$ to the cost function K as to guarantee termination (see Theorem 2). Adding ϵ does not affect the optimality of the returned alignment as long as it is chosen

sufficiently small. The cost associated with a path leading to a graph node $\gamma \in Z_V$ is then defined as follows:[6]

$$g(\gamma) = \mathcal{K}(\gamma) + \epsilon \left|\gamma\right|$$

As the search space Z consists of prefixes of complete alignments, from now on we use γ to denote nodes in Z. To find an optimal alignment of N and σ_L, we search a path in Z with the lowest cost from the source node $\gamma_0 = \langle\rangle$ to a goal node $\gamma_G \in Z_G$. The A* algorithm guarantees to find such a path only if the cost is monotonically increasing while more nodes are added to the path. The following theorem proves that the cost from Def. 5 satisfies a stricter form of this property:

Theorem 1 (Cost g is strictly increasing) *Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net and σ_L be a log trace. Let $\gamma', \gamma'' \in Z_V$ be two nodes in the search space with $(\gamma', \gamma'') \in Z_E$, i.e. there is an edge from γ' to γ''. Let $g(\gamma) = \mathcal{K}(\gamma) + \epsilon \left|\gamma\right|$ be the cost associated with a path leading to a graph node $\gamma \in Z_V$. Then $\forall \gamma'' : g(\gamma') < g(\gamma'')$.*

Proof By contradiction: Assume that there is an alignment $\gamma'' \in Z_V$ with lower cost than γ': $g(\gamma'') < g(\gamma')$. As $(\gamma', \gamma'') \in Z_E$, it follows that there exists $(s_L, s_M) \in \mathcal{A}_N : \gamma'' = \gamma' \oplus (s_L, s_M)$. The cost of an alignment is defined as the sum of the cost of all moves that led to this alignment: $g(\gamma) = \mathcal{K}(\gamma) + \epsilon \left|\gamma\right| = \sum_{(s_L, s_M) \in \gamma} \kappa(s_L, s_M) + \epsilon \left|\gamma\right|$. Therefore, the cost of γ'' can be expressed as $g(\gamma'') = g(\gamma) + \kappa((s_L, s_M)) + \epsilon$. Using the assumption $g(\gamma'') = g(\gamma') + \kappa((s_L, s_M)) + \epsilon < g(\gamma') \Leftrightarrow \kappa((s_L, s_M)) + \epsilon < 0$ should hold, but $\kappa \in S_A \rightarrow \mathbb{R}_0^+$ is non-negative and $\epsilon \in \mathbb{R}^+$ is positive. □

Here, we use the heuristic function introduced in [3], which exploits the Petri-net marking equation to rule out most nodes for which all goal states have become unreachable. A formal introduction is out of scope here. Furthermore, a comprehensive explanation would require the introduction of several concepts related to marking equations. We limit ourselves to argue that this heuristic is also admissible when other perspectives are considered. To see this, imagine $g_c(\gamma)$ to be the cost of an alignment/node γ that only considers control-flow deviations. We can take an alignment prefix γ in our search space Z_V and remove all information about the other perspectives (i.e. write operations) to calculate this cost. In fact, such an alignment prefix would be part of the search space in [3] and g_c would be the same cost function that is used in [3]. In that work, $h(\gamma)$ is proven to be admissible, i.e. for each alignment γ, $g_c(\gamma) \geq h(\gamma)$. Our cost function $g(\gamma)$ as defined above only adds additional costs for deviations with respect to other perspectives. Such deviations can only be caused by incorrect write operations and we require a non-negative cost for any such deviation. Therefore, $g(\gamma) \geq g_c(\gamma) \geq h(\gamma)$ and, hence, h will remain admissible when all process perspectives are taken into consideration.

[6] We indicate the number of moves in an alignment γ with $\left|\gamma\right|$.

Algorithm 1: balancedConformance

Input: DPN-net (N), Initial and Final Marking (M_I, M_F), Log trace (σ_L), Cost Function (K)
Result: Balanced alignment (γ)

$\gamma \leftarrow \gamma_0 = \langle\rangle$
$g(\gamma) \leftarrow K(\gamma) + \epsilon\,|\gamma|$
while $\gamma|_P \notin \mathcal{P}_N \wedge \gamma|_L \neq \sigma_L$ **do**
  foreach γ'_C **in** $\texttt{ctrl\_succ}_{\sigma_L,N,M_I,M_F}(\gamma)$ **do**
    $\gamma' \leftarrow \texttt{augmentWD}_{\sigma_L,N}(\gamma_C)$
    if $\gamma' \neq \top$ **then**
      $f(\gamma') \leftarrow g(\gamma') + h(\gamma')$
      `enqueue(`Q, γ', $f(\gamma')$`)`
    end
  end
  $\gamma \leftarrow$ `pollLowestCost(`Q`)`
end

Before introducing the actual algorithm, we need to introduce the concept of control-flow successors. Let $N = (P, T, F, V, U, Val, W, G)$ be a DPN-net and M_I, M_F be the initial and final marking, respectively. The set of control-flow successors of an alignment γ of a trace σ_L and a DPN-net N with initial and final markings M_I and M_F, denoted with $\texttt{ctrl\_succ}_{\sigma_L,N,M_I,M_F}(\gamma)$, consists of every alignment that can be obtained by adding one legal move, ignoring variables, guards, and write operations.

Formally, let us introduce $N' = (P, T, F, \varnothing, U', Val', W', G')$ as a DPN-net with $\mathsf{dom}(Val') = \mathsf{dom}(U') = \mathsf{dom}(W') = \mathsf{dom}(G') = \varnothing$. Its control-flow structure is that of DPN Net N but no variables, write operations and guards are defined. Therefore, the set of control-flow successors of an alignment γ of a trace σ_L and a DPN-net N with initial and final markings M_I and M_F is :

$$\texttt{ctrl\_succ}_{\sigma_L,N,M_I,M_F}(\gamma) = \{\gamma_C \in \mathcal{A}^*_N \mid \gamma_C = \gamma \oplus \langle(s_L, s'_L)\rangle \wedge \gamma_C|_L \in \mathsf{prefix}(\sigma_L) \wedge \exists \sigma_{N',M_I,M_F} \in \mathcal{P}_{N'} : \gamma_C|_P \in \mathsf{prefix}(\sigma_{N'})\}.$$

Please note that γ_C is generally not a prefix alignment of the original DPN-net N since its process projection may not be a valid execution trace in N.

Algorithm 1 illustrates how we use the A* algorithm to search for an optimal alignment. The algorithm takes a DPN-net N and a log trace σ_L as input and returns the optimal alignment γ that is balanced according to a given cost function κ. Instead of building the graph Z beforehand – which is potentially infinite – we build up the search space incrementally. Starting with the empty alignment $\gamma_0 = \langle\rangle$ as its source node, we build the set $\texttt{ctrl\_succ}_{\sigma_L,N,M_I,M_F}$ with all successors of γ by taking only the control-flow perspective into account. As indicated, not every control-flow successor γ_C is a node of search space Z. To be part of Z, γ_C needs to be augmented with the variable's write operations. Please note that process variables can be defined on infinite domains and, as a result, γ_C may have an infinite number of successors.

Since we aim to minimize the alignment cost, we only take one of the augmentations with the lowest cost. Here, we perform the augmentation in the

same way as discussed in [13] with the notable difference that we also augment alignments of prefixes of the log trace with the process model. By contrast, in the previous work only alignments of entire log traces are augmented. At this point, we will not elaborate on this technique but provide an example.

Example 3 *Let us consider the alignment in Fig. 4(a). This alignment is not a search-space node to compute an optimal alignment of $\sigma_{example}$ and the DPN-net in Fig. 2, since its process projection is obviously not a prefix of any process trace: the write operations need to be added. Fig. 4(b) shows the skeleton of all possible augmentations where variables Amount, Requester, Verification, Interest and Decision need to be assigned values, which are represented by placeholders A_1, R_1, V_1, I_1 and D_1. These values need to be chosen so as to not violate any guard, i.e. the sequence of transition firings need to be a prefix of a valid process trace. Moreover, we aim to minimize the cost of the deviations from what observed in the corresponding events in the log. According to what proposed in [13], we need to solve a MILP problem, which is Figure 4(c) in this case. The placeholders mentioned above become MILP variables. When an optimal solution is found, the values of these MILP variables are, in fact, the values to set in the alignments. Two sets of constraints can be observed. The first set corresponds to the guards associated with the transitions, defined over these MILP variables. Moreover, for each variable, e.g. A_1, there is a constraint that says that a boolean variable, e.g. $\widehat{A_1}$, is given a value 0 if and only if A_1 is assigned the same value as observed in the corresponding event, e.g. $A_1 = 3000 \Leftrightarrow \widehat{A_1} = 0$.[7] The objective function is the cost in term of severity of the deviations, i.e. the sum of such boolean variable, e.g. $\widehat{A_1}$, weighted with coefficient corresponding to the cost for deviation of the respective process variable, which is equal to 1 for all variables for this example.*

In the remainder, the augmentation is abstracted as a function $\texttt{augmentWD}_{\sigma_L,N} : \mathcal{A}^*_N \to Z_V$, which takes a control-flow successor and returns an alignment $\gamma \in Z_V$. Since N is assumed to be only relaxed data sound, it may happen that γ_C cannot be augmented with other perspectives (i.e., the MILP problem has no solutions). In this case, the function is assumed to return the special value $\top$. Note that the augmentation γ' for γ_C needs to be computed from scratch, ignoring the predecessor γ. Indeed, the last move may refer to a transition t that is not allowed to fire in the DPN-net state reached by firing sequence $\gamma|_P$.

If an augmentation γ' exists, i.e. $\gamma' \neq \top$, the cost $f(\gamma')$ is computed and γ' is inserted into the priority queue Q using the function `enqueue`. If it does not exist, γ_C does not yield to any valid alignment to be added to Q. Once all the control flow successors are considered, a new alignment γ is picked from the head of Q using the function `pollLowestCost`, i.e. one of the alignments associated with the lowest cost. If γ is a complete alignment, it is returned as the optimal alignment. Since the heuristic function is admissible and the cost g is monotonically increasing, the application of A* guarantees that the returned

[7] Although these constraints are not expressed in a linear form, each of these can be translated into a pair of linear inequations as discussed in [13].

Event-Log Trace	Process
a {**A** = 3000, **R** = Michael}	**a** {}
b {**V** = $false$)	**b** {}
c {**I** = 530, **D** = $true$}	**c** {}

(a) Control-flow successor

Event-Log Trace	Process
a {**A** = 3000, **R** = Michael}	**a** {**A** = A_1, **R** = R_1}
b {**V** = $false$)	**b** {**V** = V_1}
c {**I** = 530, **D** = $true$}	**c** {**I** = I_1, **D** = D_1}

(b) The skeleton of all possible augmentations of the control-flow successor

$$\min \widehat{V_1} + \widehat{I_1} + \widehat{A_1} + \widehat{R_1} + \widehat{D_1}$$
$$V_1 = \text{true}$$
$$I_1 > 0.1\ A_1$$
$$I_1 < 0.15\ A_1$$
$$A_1 > 5000$$
$$A_1 = 3000 \Leftrightarrow \widehat{A_1} = 0$$
$$R_1 = \text{Michael} \Leftrightarrow \widehat{R_1} = 0$$
$$V_1 = \text{false} \Leftrightarrow \widehat{V_1} = 0$$
$$I_1 = 530 \Leftrightarrow \widehat{I_1} = 0$$
$$D_1 = \text{true} \Leftrightarrow \widehat{D_1} = 0$$

(c) The MILP problem to find the augmentations with the lowest cost in deviations

Fig. 4 Illustration of the augmentation of a control-flow successor of an alignment/node of the search space

alignment has the lowest cost of all complete alignments. Otherwise, the node is expanded and successors are added to the queue. Therefore, the entire search space is not constructed in advance but the search-space successors of a given node are only created when such a node is visited.

Example 4 *Fig. 5 shows the portion of the search space that Algorithm 1 constructs to find an optimal alignment of* $\sigma_{example}$ *and* N *when the standard cost function is used. Each node* $\gamma \in Z_V$ *is represented by a circle, which includes both the values for the actual cost* $g(\gamma)$ *and the estimated cost* $h(\gamma)$ *to extend the alignment to obtain a complete one. Nodes emphasized with a gray background are those which have been visited during the search (i.e., the nodes that are polled from the priority queue). The other nodes have been constructed and the values of functions* g *and* h *computed, because they are successors of nodes that have been visited. The gray nodes are also associated with numbers* $\#1, \ldots, \#10$*, indicating the order in which they have been visited. Goal nodes* $\gamma_G \in Z_G$ *are depicted with a double-line border. An edge between two nodes/alignments* γ' *and* γ'' *is labelled with the move* (s', s'') *with which* γ' *has been extended, i.e.* $\gamma'' = \gamma' \oplus \langle (s', s'') \rangle$*. For readability, the labels do not show the variable assignments but they only show the transition names; for the same reason, we omit some labels when they are not very significant. As a matter of fact, Example 3 refers to one of the control-flow successors of the node/alignments with label* $\#2$*, namely the successor associated with a move in both for transition c. It is easy to see that two is the cost of the optimal solution of the associated MILP problem, shown in Fig. 4(c), which is also the*

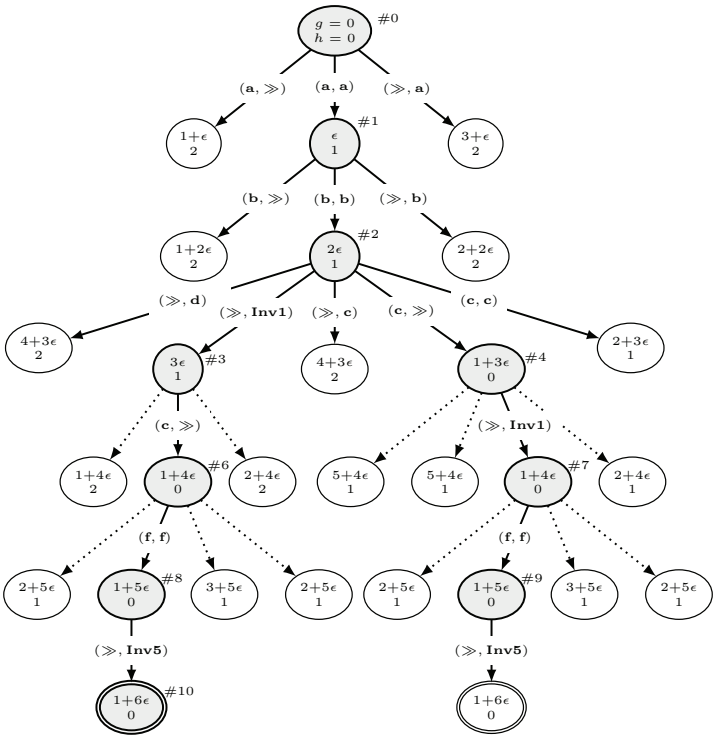


Fig. 5 Portion of the search space constructed to find an optimal alignment of $\sigma_{example}$ and the DPN-net in Fig. 2. Write operations omitted for readability

cost of such an alignment, i.e. the function K. To get the cost $g(\gamma)$, ϵ is added for each of the three steps in the alignment.

The optimal alignment is associated with the alignment/node γ_O with label #10*. Note that Fig. 5 indicates a different target node/alignment γ'_O as well. In fact, $g(\gamma'_O) = g(\gamma_O)$ and, hence, γ'_O is also an optimal alignment. The choice between γ'_O and $g(\gamma_O)$ is totally arbitrary since there is no reason to prefer one alignment over the other.*

Theorem 2 (Algorithm 1 terminates) *Let $N = (P, T, F, V, U, Val, W, G)$ be a relaxed data sound DPN-net with M_I initial marking and M_F final marking. Let $\sigma_L = \langle l_1, ..., l_n \rangle$ be a log trace. Let K be a cost function. Algorithm 1 terminates with inputs N, M_I, M_F, σ_L, K.*

Proof Since N is relaxed data sound, there exists at least one valid process trace $\sigma_M = \langle p_1, ..., p_m \rangle \in P_N$. Therefore, there exists at least one complete alignment $\gamma_O = \langle (l_1, \gg), ...(l_n, \gg), (\gg, p_1), ..., (\gg, p_m) \rangle$, which belongs to the target node of the search space used by Algorithm 1. Suppose that Algorithm 1 does not terminates with inputs N, M_I, M_F, σ_L, K. It means that for each $q \in \mathbb{N}$, there exists an alignment γ_q composed by q moves such that $f(\gamma_q) \leq f(\gamma_O)$. In particular, it holds for $q' = \lceil \frac{f(\gamma_O)}{\epsilon} + 1 \rceil$. Since each alignment move at least

adds a cost ϵ, $f(\gamma_{q'}) \geq \lceil \frac{f(\gamma_O)}{\epsilon} + 1 \rceil \cdot \epsilon \geq f(\gamma_O) + \epsilon$. This cannot be true since we assumed $f(\gamma_q) \leq f(\gamma_O)$. $\square$

So, Algorithm 1 can always terminate although, in theory, an arbitrary large number of non-complete alignments need to be visited. In practice, this number is kept reasonably small by the fact that models and cost functions are designed in a way that there is no possibility to have arbitrary long sequences in an alignment where each move takes on a zero cost.

As discussed in [8], the worst-case complexity of the A* algorithm is exponential in the length of the path that leads from the initial search-space node to the nearest goal node. Applied to the problem of finding an optimal alignment, this means that the worst-case complexity is exponential in the length of the alignment. This is of the same order of magnitude as the log-trace traces, assuming that, on average, each trace event is associated with one or two moves (e.g., a move in both or a move in log plus a move in model). For each node that is visited, a MILP problem needs to be solved. So, the number of problems to be solved is exponential in the length of log trace. The worst-case complexity of solving an MILP problem is in the number of variables and constraints, which, in our setting, is translated to the number of variables written by and guards associated to transitions. In summary, the worst-case complexity is double exponential. Fortunately, our choice of the heuristic function and the optimizations discussed in Section 4.2 allows the analysis to limit the number of nodes to be visited and the MILP problems to be solved. In fact, as discussed in Section 6, this worst case is practically almost never encountered.

4.2 Optimizations

Finding a balanced alignment comes at the price of a higher computation cost in comparison with the technique presented in [13]. Recall that in [13] one cannot balance the different perspectives. Nevertheless, efficiency is of the utmost importance. Hence, we provide a number of optimizations to speed up the computation.

Optimization #1. A first optimization is concerned with automatically preventing part of the search space from being visited. It follows from the observation that any arbitrary re-ordering between move in log $(s_L, \gg)$ and move in model $(\gg, s_M)$ steps in a sub-sequence, without a move in both, will result in the same cost. Shuffling those sub-sequences in any arbitrary order is not necessary. Therefore, instead of visiting every possible ordering, we only consider one shuffle, i.e. the one in which all moves in log come first and all moves in model follow. For instance, consider an alignment $\gamma = \gamma_1 \oplus \gamma_2 \oplus \gamma_3$ where γ_2 consists of only moves in log and γ_3 consists of only moves in model. We disallow γ to be extended with a move in log $(s_L, \gg)$, unless, before extending with such a move, γ is extended with a move in both. As a matter of fact,

$(s_L, \gg)$ could also occur before moves in γ_3 occurred. In this way, we avoid considering all alternations of moves in log and models, unless they are interleaved by a move in both. It is easy to see that this may significantly prune the search space.

Optimization #2. Most of the computations are due to the large number of MILP problems that need to be solved. In fact, one is needed for each visited node of the search space. It seems attractive, thereofore, to avoid unnecessarily solving MILP problems. Let γ be a search-space node and let (M, A) be the state reached after executing $\gamma|_P$, i.e. $(M_0, A_0) \xrightarrow{\gamma|_M} (M, A)$. Let us consider each γ control-flow successor $\gamma_C = \gamma \oplus \langle (s_L, s_M) \rangle$. If $s_M = \gg$, $\texttt{augmentWD}_{\sigma_L, N}(\gamma_C)$ is trivially $\gamma \oplus \langle (s_L, \gg) \rangle$. It is not important which kind of write operations s_L contains: s_L is added to the alignment but, in fact, it does not influence the process projection. If $s_L \neq \gg$, $s_M \neq \gg$ and, also, s_L is a valid firing in state (M, A), $\texttt{augmentWD}_{\sigma_L, N}(\gamma_C)$ is simply $\gamma \oplus \langle (s_L, s_L) \rangle$. Indeed, since (s_L, s_L) is a move in both without incorrect write operations, no additional cost is added to the inevitable cost associated with γ.

Optimization #3. In a large number of cases, the same MILP problem (i.e., with the same set of constraints and objective function) needs to be solved to augment different alignments. Therefore, we adopt an additional optimization where MILP problems are cached along with one of the optimal solutions. If the same problem is observed again, its solution can be retrieved from the cache. To build the cache we construct the MILP problems in a way that they can be associated with univocal hash codes. Consequently, the cache can be organized as a hash table where checking for the presence of a MILP problem and, possibly retrieving the associated solution, can be done in constant time.

5 Implementation

We implemented the algorithm of Section 4 as a plug-in for the open-source ProM framework.[8] The plug-in *Balanced Data Conformance Checker*, takes as inputs a process model in the form of a DPN-net and an event log. For these, it computes optimal balanced alignments for each trace based on the algorithm described in Section 4. All three optimizations, highlighted in Section 4.2, are supported by the *Balanced Data Conformance Checker* plug-in.

The output of the plug-in is a set of alignments that can be used by other plug-ins of the ProM framework. For example, all visualization plug-ins that are described in [13] require such a set of alignments as input. The basic visualization plug-in presents each alignment as color-coded trace. An example of this type of visualization is shown in Figure 6 for the alignment of $\sigma_{example}$ and the DPN-net N. Colors are used for the different types of *moves* of the alignment. For example, a *move in both with correct write operations* is colored

[8] `http://www.promtools.org`

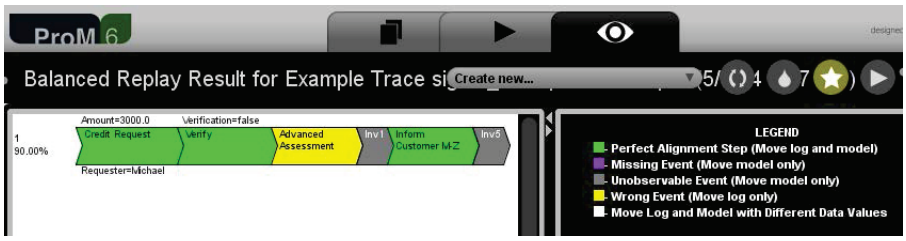


Fig. 6 Visualization of the optimal alignment for $\sigma_{example}$ that was obtained using the *Balanced Data Conformance Checker*

green and a *move in log* is colored yellow. Whereas this visualization is suited to analyze the alignment for a single trace, it does not provide an overview about the whole event log.

A *helicopter view* [13] on the whole event log is given by a second plug-in, as shown in Fig. 7. It projects the set of alignments for the entire event log onto the process model. Figure 7 depicts the output of this *Projection on net* plug-in for a set of alignments and the process model of the road traffic fines management process, which is discussed in more detail in Section 6. Transitions and variables are colored according to the number of deviations in the alignments, i.e. *move in log*, *move in model*, and *move in both with incorrect write operations*, in relation to the number of occurrences of the transition or variable. The darker the color, the higher is the percentage of deviations for a transition or variable. For example, the color of the *Send Appeal* transition encodes the fact that the highest percentage of deviations is related to *Send Appeal*.

MILP problems are solved through the open-source library *lp_solve*, which is based on the revised simplex method combined with a branch-and-bound method for the integers[9]. However, we use a standardized interface and, hence, one is free to plug in several solvers, including commercial ones.

6 Evaluation

To evaluate the usefulness and feasibility of the balanced approach we use both a real-life and a synthetic data set. To assess the usefulness, we compare results returned by the balanced approach with those returned by the non-balanced one from [13]. Regarding the feasibility of the balanced approach, we show that our implementation can handle traces with noise. Moreover, traces of considerable length can be handled in a reasonable amount of time. We use an event log from an information system of the Italian police as a real-life case. We simulate the process model as shown in Figure 2 with CPN Tools[10] to generate multiple synthetic event logs. These allow for various controlled experiments.

[9] `http://lpsolve.sourceforge.net`
[10] `http://www.cpntools.org`

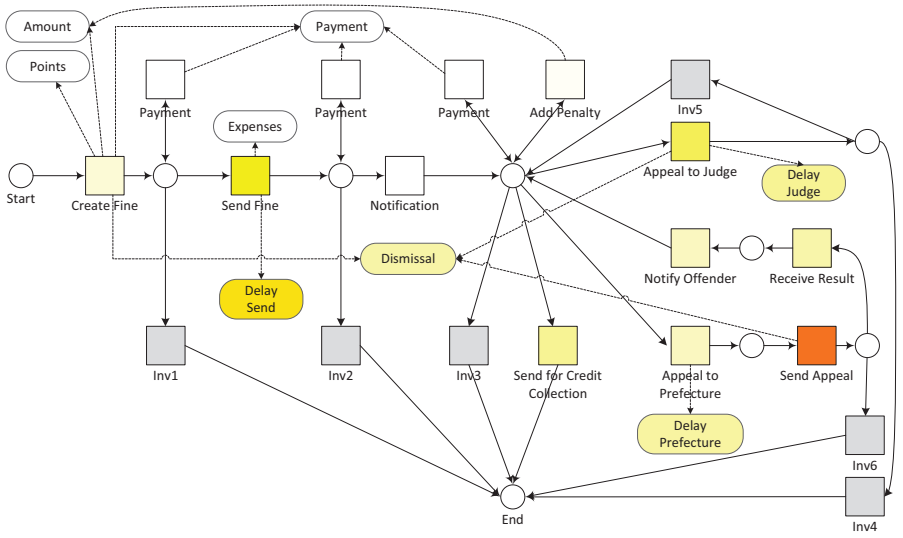


Fig. 7 Output of the *Projection on net* feature of ProM using a set of balanced alignments. The darker the color, the higher the percentage of deviations that are detected for the transition or the variable. Figure was redrawn to improve readability (ProM does not provide vector graphics)

6.1 Real-life Event Log

We applied our balanced approach to multi-perspective conformance checking to a real-life event log taken from an information system of the Italian police. The information system supports the management of road traffic fines by a local police force in Italy. The system records sufficient data to create an event log.

6.1.1 Process Model

The process model shown in Figure 8 together with the guards in Figure 9 specify the management of road traffic fines. We designed the process model manually taking domain knowledge and regulations about the management of road traffic into account. The process starts with the *Create Fine* transition that writes four variables: *Amount* (A), *Points* (PO), *Payment* (P) and *Dismissal* (D). The *Amount* variable refers to the amount that needs to be paid by the offender and the *Points* variable records the number of points that are deducted from the offender's license. *Payment* is the total amount that has been paid by the offender. It is always initialized as $P = 0.0$. *Dismissal* contains a character that encodes the diverse reasons for a possible dismissal of the fine. A value of *NIL* encodes that the fine is not dismissed (i.e. has to be paid); any other value encodes different motivations. In general, the offender can pay the fine (partly or fully) at many moments in time: Right after the creation, after a road fine notification is sent by the police to the offender's place of

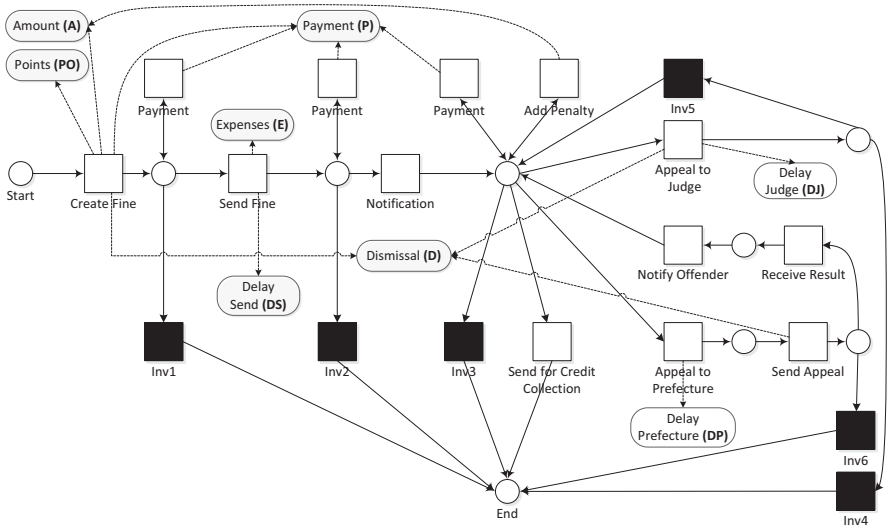


Fig. 8 DPN-net of the road traffic fine management process

Transition	Guard
Send Fine	Delay Send′ < 90 days
Appeal to Judge	Delay Judge′ < 60 days
Appeal to Prefecture	Delay Prefecture′ < 60 days
Receive Result	Dismissal = NIL
Send for Credit Collection	Payment < Amount + Expenses
Inv1	(Dismissal != NIL) ∨ (Payment ≥ Amount ∧ Points = 0)
Inv2	Payment ≥ Amount + Expenses
Inv3	Payment ≥ Amount + Expenses
Inv4	Dismissal = #
Inv5	Dismissal = NIL
Inv6	Dismissal = G

Fig. 9 Guards of the road traffic fine management DPN-net

residence, or when such a notification is received by the offender herself. If the entire amount is paid (or, even, by mistake, more than that amount), the fine management is closed. This motivates the presence of the invisible transitions $Inv1$, $Inv2$ and $Inv3$. If a notification is sent, the offender needs to also pay the postal expenses. If the offender does not pay within 180 days, a penalty is added, usually as much as the fine's amount. After being notified by post, the offender can appeal against the fine through a judge and/or the prefecture. If the appeal is successful, the variable *Dismissal* is set to value G or $\#$, respectively, and the case ends (by firing either $Inv4$ or $Inv6$). Otherwise, the case continues by firing $Inv5$ or *Receive Result*. If the offender does not pay, eventually the fine ends by handing over the case for credit collection.

The Italian laws specifies a number of time constraints. The fine notification must be sent within 90 days since its creation. After the notification, the offender may only appeal to a judge/prefecture within 60 days. To check the conformance of the fine management with respect to these laws, we have introduced three additional variables that record the various delays: *Delay Send*, *Delay Judge*, *Delay Prefecture*.

Transition	Model-Move	Log-Move
Create Fine	1	1
Send Fine	1	1
Payment	1	1
Notification	1	1
Add Penalty	1	1
Appeal to Judge	1	1
Appeal to Prefecture	1	1
Send for Credit Collection	1	1
Send Appeal	1	1
Receive Result	1	1
Notify Offender	1	1
Inv1, . . ., Inv6	0	∞

Variable	Missing	Incorrect
Delay Send	1	1
Delay Prefecture	1	1
Delay Judge	1	1
Amount	1	3
Expense	1	3
Payment	1	3
Points	1	3
Dismissal	1	1

Fig. 10 Costs function $\kappa'(s_L, s_M)$ for the road fines management process. The cost for *move in both without incorrect write operation* is 0 for all transitions. Log moves for invisible transitions take an infinite cost: this value is irrelevant since invisible transitions are never associated with any log's event. The cost for *move in both with incorrect write operation* is obtained by summing the specific costs associated with each variable for which a missing or incorrect value has been written, as per right-hand side table.

6.1.2 Event Log

The road traffic fine management process is supported by an information system that records data about its operations in a PostgresSQL database. The database snapshot at our disposal was taken in June 2013. We exported the event log to a CSV format and converted it to the XES format[11], which is the event log format supported by tools like ProM. From the analysis of the event log, we noticed that cases usually are completed within 6 months, including those cases ending with a referral to credit collection. We wish to ignore incomplete cases. As a heuristic to ensure this, we filtered out any case that started after June 2012. Since the relevant laws and procedures are rather stable over the past years, the last year of the event log should show the same behavior as in the previous years.

The resulting event log contains 145,800 event traces, which are recorded between January 2000 and June 2012. Most of the traces are short: on average, a trace consists of four events only. For 43% of the traces the process ends after two events: the fine is paid (*Payment*) before the letter with information about the fine is sent out (*Send Fine*). In contrast to this simple part of the log, 51% of the traces recorded five or more events and 62% of the traces take longer than 100 days to finish. This suggests that many offenders do not pay the fine in time or appeal against the decision.

Similar as for the log trace, the event log has been extended to incorporate variables *Delay Send*, *Delay Judge*, *Delay Prefecture* to record the delays.

6.1.3 Conformance Checking: Choice of Cost Function

The balanced approach has been used to check the conformance of the event log described in Sect. 6.1.2 against the model in Sect. 6.1.1. As described in Section 3.2, the approach requires the definition of a cost function $\kappa(s_L, s_M)$.

[11] `http://www.xes-standard.org/`

Such function may be chosen by a process analyst to fit the circumstances of the study in question. Since the fine's amount and the deducted points are defined by law and the expenses follow the Italian post tariffs, their values cannot be modified to give an explanation of deviations. In order to respect this domain characteristic, we have assigned significantly higher costs to their deviations in comparison to those for deviations of the values of *Payment* and *Dismissal* and the control flow.

The cost function is as specified by the two tables in Figure 10. The tabular representation is given to enhance the readability. However, we aim to quickly show that so-defined cost function is compliant with the cost-function definition $\kappa'(s_L, s_M) \to \mathbb{R}_0^+$. The cost for a specific legal move $(s_L, s_M) \in \mathcal{A}_N$ can be obtained by looking up the costs regarding the transition and adding the cost of missing write operations in case of a *move in model* or the cost of incorrect write operations in case of a *move in both with incorrect write operations*. The tables in Figure 10 provides a tabular representation of the cost function used in the evaluation. This definition complies with the cost-function structure as given in Definition 8. We can demonstrate this through an example. Consider activity *Create Fine*, which writes variables *Amount*, *Expense* and *Payment*. The cost function is defined as follows (for the part referring to *Create Fine*):

$$\kappa(s_L, s_M) = \begin{cases} 1 & (s_L, s_M) \text{ is a move in log for } \textbf{Create Fine} \\ 1 & (s_L, s_M) \text{ is a move in move for } \textbf{Create Fine} \\ 3 \cdot |\{v \in V : \#_{vars}(s_L, v) \neq \\ \#_{vars}(s_M, v) \wedge \#_{vars}(s_L, v) \neq \bot \\ \wedge \#_{vars}(s_M, v) \neq \bot\}| \\ + \\ |\{v \in V : \#_{vars}(s_L, v) = \bot \\ \wedge \#_{vars}(s_M, v) \neq \bot\}| & (s_L, s_M) \text{ is a move in both for } \textbf{Create Fine} \\ \ldots & \ldots \end{cases}$$

6.1.4 Conformance Checking: Analysis of Results

After applying the balanced approach for conformance checking with the cost function discussed in Sect. 6.1.3, we used the feature of the *Projection on net* to obtain an overview of the non-conformance problems. Figure 7 shows the results of the net projection.

The average fitness level was 0.96, which testifies a very good conformance of the event log with the process model. In particular, 53.1% of log traces are characterised by a fitness level of 1. However, as the net projection indicates, a number of deviations are still present. For example, the transition *Send Appeal* is colored with the darkest color: In 79.1% of the alignments that contain *Send Appeal* there is a deviation from the process model concerning this transition. Particularly, *Send Appeal* is executed 2895 times with an incorrect value for *Dismissal*, 112 times as *move in log*, 3 times as *move in model* and the remaining 795 times without deviations. In particular, often the dismissal value is # instead of *G*. This suggests that process participants should pay attention not to confuse the dismissal code corresponding to a successful appeal to a judge with one referring to a successful appeal to the prefecture.

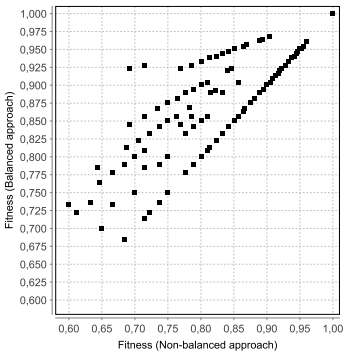


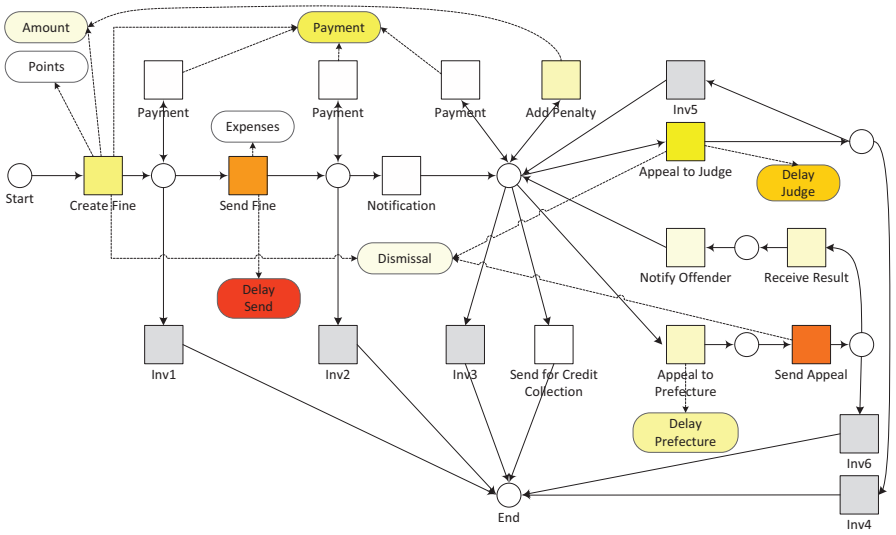
Fig. 11 Comparison of the fitness level returned by non-balanced and balanced approach

Moreover, Figure 7 shows that the time constraints regarding *Send Fine*, *Appeal to Judge* and *Appeal to Prefecture* are often not adhered to. The value of the *Delay Send* variable is incorrect in 46.8% of the traces, the value of *Delay Judge* in 18% of the traces, and the value of *Delay Prefecture* in 15.4% of the traces. This suggests that the available process resources are incapable to manage every road fine on time. Therefore, to remedy this situation more resources, i.e. police officers, should be assigned by the municipality. Alternatively, some parts of the management should be outsourced, e.g. the steps necessary to print fine notifications, put these in envelops, and send them by post. Indeed, these are manual steps that require a lot of time from the involved police officers.

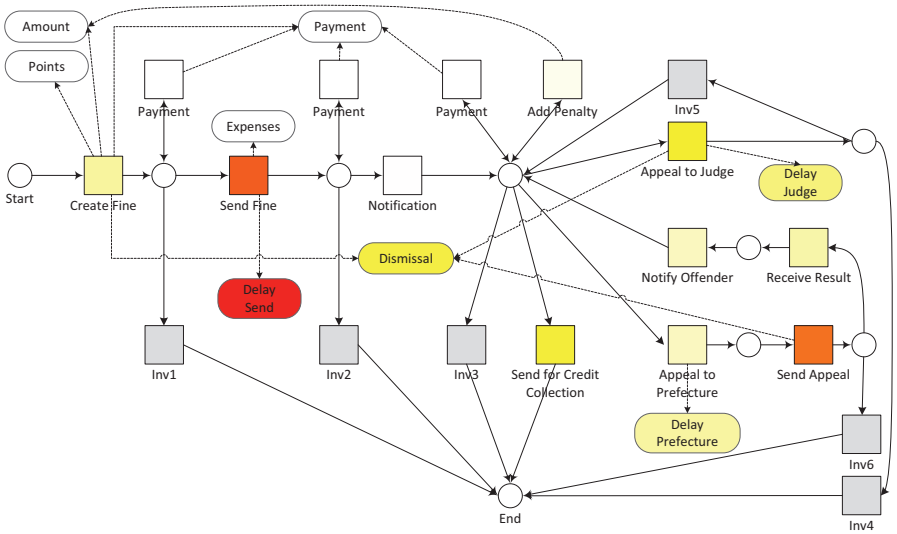
A valuable insight is that there are deviations recorded for the *Send for Credit Collection* transition. In 8.2% of all traces (i.e. 11,945 times) the transition appears as a *move in model* in the alignment. This implies that after at least one year for 11,945 traces the fee has neither been paid in full nor was the fine forwarded to credit collection. This insight indicates that there may be an issue with the tracking of unpaid fines.

In order to compare the results of our balanced approach with those returned by a non-balanced approach [13], we also applied the latter. Figure 11 shows a scattered plot in which each black box represents one trace. The x-axis shows the fitness level of non-balanced alignment and the y-axis shows the fitness level of the balanced alignment. For all traces that are left from the main diagonal, which amounts to 21.7% of all traces, the balanced approach improved the alignment. For all traces on the main diagonal the fitness level remains unchanged. The balanced approach improved the fitness level of the alignment for 46.4% of the traces, excluding the perfectly-fitting traces, for which this is obviously impossible. In particular, 68,330 traces contain at least one deviation.

After removing every perfectly-fitting trace from the event log, we again applied both the balanced and non-balanced approach. Figure 12(a) and Figure 12(b) show the output of the *Projection on net* plug-in of ProM for the



(a) Projection of the non-balanced alignments on the DPN-net



(b) Projection of the balanced alignments on the DPN-net

Fig. 12 Comparison of the *Projection on net* output returned for the non-balanced and balanced approach. The darker the color, the higher the percentage of detected deviations for the transition or the variable. The figure clearly shows that the balanced approach provides different results suggesting that the approach to first construct control-flow alignments may provide misleading results

68,330 non-perfectly fitting traces for both the balanced and the non-balanced approach. The comparison of both figures shows that there are significant differences in the identification of the root-causes of the deviations. In particular, applying the non-balanced approach, the net projection highlights that many traces are deviant because of wrong values of *Amount* and *Payment*. Indeed, Figure 12(a) colors these variables yellow and dark yellow. Vice versa, if one looks at Figure 12(b), the corresponding variables are white-colored.

In order to understand the reason for such a significant difference in the identification of root causes, we have inspected the alignments returned by the two approaches. We found out that there are alignments for hundreds of log traces of the following form:

$$\sigma_A = \langle(\text{Create Fine}, \{\mathbf{A} = 131.0, \mathbf{D} = \text{NIL}, \mathbf{PO} = 0, \mathbf{P} = 0\}), (\text{Send Fine}, \{\mathbf{DS} = 1152, \mathbf{E} = 10.0\}\})\rangle$$

and a smaller number of traces of the following form:

$$\sigma_B = \langle(\text{Create Fine}, \{\mathbf{A} = 138.0, \mathbf{D} = \text{NIL}, \mathbf{PO} = 6, \mathbf{P} = 0\}), (\text{Send Fine}, \{\mathbf{DS} = 3409, \mathbf{E} = 11.0\}\}), (\text{Notification}), (\text{Appeal to Judge}, \{\mathbf{DS} = 840, \mathbf{D} = \text{NIL}\}\}), (\text{Add Penalty}, \{\mathbf{A} = 275.0\}\}) (\text{Payment}, \{\mathbf{P} = 149.0\}\})\rangle.$$

Figure 13 compares the alignments returned by the balanced and non-balanced approach for the trace σ_A. The non-balanced approach highlights that the fine at creation time should have already been associated with a payment of 49 Euros. By contrast, the balanced approach suggests that the fine should have been dismissed, e.g. with code Q, and never sent out. It is easy to see that the alignment returned by the non-balanced approach is not plausible, since it is impossible to create a fine that already has a full payment associated to it. The payment by necessity can only be made at a later stage.

Regarding *Amount*, any root cause that consists of changing the assignment of such a data variable (i.e. an incorrect write operation) is not acceptable. After all, this amount is defined by the Italian law and police officers use road-fine forms in which the amount is predetermined. For instance, let us consider the trace σ_B. Clearly, this trace has problems. First, the fine was sent too late, since the delay is larger than what law permits. Second, the fine has been closed with a payment of 149 Euros, which corresponds to the initial amount plus the postal expenses. Unfortunately, a penalty was also included, which the offender did not pay. As shown in Figure 14, both of the approaches highlight the problem of a sending delay. For the second source of mis-conformance, the non-balanced approach suggests that, after applying the penalty, the due amount does not change. This is definitely not plausible since adding a penalty needs to result in a larger amount to pay. As a matter of fact, the Italian law states that, besides very few exceptions, the due amount should even be doubled, excluding expenses. By contrast, the balanced approach returns a meaningful result: The fine was not paid in full and, hence, needs to be sent for credit collection.

The reason for these differences is related to the fact that the non-balanced approach attempts to stick to the same control-flow alignment while varying the data assignments. As we argue, this may lead to implausible explanations.

Event-Log Trace	**Process**
Create Fine $\{\mathbf{A} = 131.0, \mathbf{D} = \text{NIL}, \mathbf{PO} = 0, \mathbf{P} = \underline{0}\}$	Create Fine $\{\mathbf{A} = 131.0, \mathbf{D} = \text{NIL}, \mathbf{PO} = 0, \mathbf{P} = 141.0\}$
Send Fine $\{\mathbf{DS} = 1152, \mathbf{E} = 10.0\}$	Send Fine $\{\mathbf{DS} = 1152, \mathbf{E} = 10.0\}$
$\gg$	Inv2

(a) Non-balanced alignment ($\mathcal{F}(\sigma_A, N) = 0.77$)

Event-Log Trace	**Process**
Create Fine $\{\mathbf{A} = 131.0, \mathbf{D} = \underline{\text{NIL}}, \mathbf{PO} = 0, \mathbf{P} = 0\}$	Create Fine $\{\mathbf{A} = 131.0, \mathbf{D} = \text{Q}, \mathbf{PO} = 0, \mathbf{P} = 0\}$
Send Fine $\{\mathbf{DS} = 1152, \mathbf{E} = 10.0\}$	$\gg$
$\gg$	Inv1

(b) Balanced alignment ($\mathcal{F}(\sigma_A, N) = 0.85$)

Fig. 13 Comparison between balanced and non-balanced alignments of a trace σ_A taken from the real-life event log. The non-balanced alignment contains an incorrect variable assignment for the initial payment P, whereas the balanced alignment contains an incorrect variables assignment for dismissal D

We conclude this section by briefly reporting on the execution time. Finding the alignments took on average 2.3 milliseconds for the balanced approach with a standard deviation of 3.5 milliseconds, versus 1.2 milliseconds for the non-balanced one. The balanced approach required slightly more time, which still appears reasonable and certainly justified by obtaining more meaningful explanations for the deviations.

6.2 Synthetic Event Log

We also conducted experiments with synthetic event logs in order to show that the balanced approach is still feasible when dealing with loops in the process model, event logs containing longer traces, and higher levels of noise. We constructed 3 event logs that contain traces of considerable length (between 3 and 35 events per trace) and we introduced different levels of noise in such logs L_1, L_2, L_3, namely 5, 10 and 15% of noise, respectively. Introducing x% of noise in a perfectly fitting log means that we manipulated the event logs by swapping x% of the events in each trace and by changing the attribute values associated with transition firings such that x% of firings are invalid because the respective guards are not satisfied. Fig. 15 illustrates the execution time with 3 scattered plots (one for each event log). A dot located at position (x, y) identifies the fact that a trace of length x required y milliseconds to be aligned. Two series of dots are displayed: black dots refer to the execution time of the balanced algorithm described in Section 3.2, also featuring the optimizations, whereas gray dots refer to the execution time of the non-balanced approach that is described in [13]. The balanced approach is computationally more expensive than the unbalanced approach: The execution time grows exponentially with

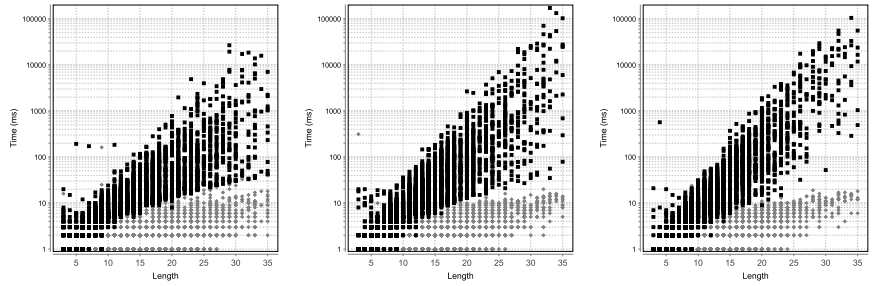
Event-Log Trace	**Process**
Create Fine $\{\mathbf{A} = 138.0, \mathbf{D} = \text{NIL}, \mathbf{PO} = 6, \mathbf{P} = 0\}$	Create Fine $\{\mathbf{A} = 138.0, \mathbf{D} = \text{NIL}, \mathbf{PO} = 6, \mathbf{P} = 0\}$
Send Fine $\{\mathbf{DS} = \underline{3409}, \mathbf{E} = 11.0\}$	Send Fine $\{\mathbf{DS} = 2159, \mathbf{E} = 11.0\}$
Notification	Notification
Appeal to Judge; $\{\mathbf{DS} = 840, \mathbf{D} = \text{NIL}\}$	Appeal to Judge; $\{\mathbf{DS} = 840, \mathbf{D} = \text{NIL}\}$
$\gg$	Inv5
Add Penalty $\{\mathbf{A} = \underline{275.0}\}$	Add Penalty $\{\mathbf{A} = 138.0\}$
Payment $\{\mathbf{P} = 149.0\}$	Payment $\{\mathbf{P} = 149.0\}$
$\gg$	Inv3

(a) Non-balanced alignment ($\mathcal{F}(\sigma_B, N) = 0.76$)

Event-Log Trace	**Process**
Create Fine $\{\mathbf{A} = 138.0, \mathbf{D} = \text{NIL}, \mathbf{PO} = 6, \mathbf{P} = 0\}$	Create Fine $\{\mathbf{A} = 138.0, \mathbf{D} = \text{NIL}, \mathbf{PO} = 6, \mathbf{P} = 0\}$
Send Fine $\{\mathbf{DS} = \underline{3409}, \mathbf{E} = 11.0\}$	Send Fine $\{\mathbf{DS} = 2159, \mathbf{E} = 11.0\}$
Notification	Notification
Appeal to Judge; $\{\mathbf{DS} = 840, \mathbf{D} = \text{NIL}\}$	Appeal to Judge; $\{\mathbf{DS} = 840, \mathbf{D} = \text{NIL}\}$
$\gg$	Inv5
Add Penalty $\{\mathbf{A} = 275.0\}$	Add Penalty $\{\mathbf{P} = 275.0\}$
Payment $\{\mathbf{P} = 149.0\}$	Payment $\{\mathbf{P} = 149.0\}$
$\gg$	Send for Credit Collection

(b) Balanced alignment ($\mathcal{F}(\sigma_B, N) = 0.88$)

Fig. 14 Comparison between balanced and non-balanced alignments of a trace σ_B taken from the real-life event log. In the non-balanced alignment the amount *A* after adding a penalty and the delay send *DS* variables are marked as incorrect (underlined). In the balanced alignment the amount *A* is considered as correct and instead the transition *Send to Credit Collection* is inserted as *model move*



(a) L_1: 5% Noise (b) L_2: 10% Noise (c) L_3: 15% Noise

Fig. 15 Run-time per trace for different trace length and noise level. Non-balanced approach gray diamonds, balanced approach black dots

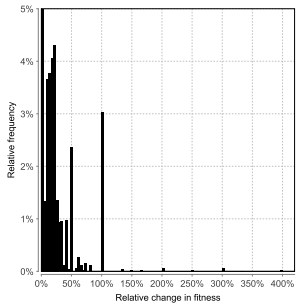


Fig. 16 Improvement in fitness compared to non-balanced approach for log L_3 (Cut-off at 6%)

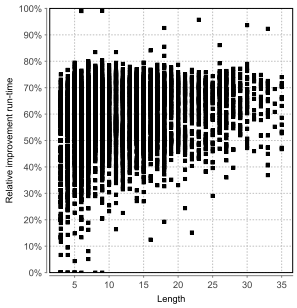


Fig. 17 Improvement in run-time by optimization #2 and #3 for L_1

log traces of increasing size. It is also easy to see that the trend does not significantly change in the three scattered plots.

An obvious next question is: Are better alignments justified by the increase of execution time? Excluding the traces whose fitness is simply 1.0, *for 35% of traces, the alignment returned as optimal by the non-balanced approach [13] was not actually so.* Fig. 16 shows the distribution of the fitness level improvement. Among these traces, the balanced approach returns optimal alignments with a fitness that is 33% higher, on average. For 5% of all traces, the fitness improvements even amounted to 50%, with peaks of nearly 400%. This means that, for 5% of traces, the non-balanced approach returned supposedly optimal alignments with deviations one and a half times as severe. To illustrate why one can obtain a quite high improvement in fitness, let us consider again trace $\sigma_{example}$, which is included in log L_3. The fitness level of the alignment in Table 3(a), containing three wrong write operations, is $\mathcal{F}(\sigma_{example}, N) = 1 - \frac{3}{10} = 0.7$ and the fitness level of the optimal alignment in Table 3(b), with one move in model, is $\mathcal{F}(\sigma_{example}, N) = 1 - \frac{1}{10} = 0.9$, i.e., the fitness level can be improved by 28% using the balanced approach.

Last but not least, we conducted experiments to verify the effectiveness of the optimizations introduced in Sect. 4.2 by using event log L_3. Without Optimization #1, it was impossible to construct the alignments for all traces: The machine employed ran out of memory. Enabling Optimization #1 and disabling Optimization #2 and #3, the experiments could be carried out: Fig. 17 illustrates how much execution time was saved when all optimizations were enabled. This slightly depended on the length of the trace. For example, for traces of length 30 this amounted to an average time saving of 65% .

7 Related Work

As we explained in the introduction, the data, resource and time perspectives are often neglected when considering conformance checking of an event log to a process model. However, there are many papers on conformance checking that

only consider the control-flow perspective. Therefore, we first discuss some conformance approaches that abstract from data-flow, resources, time, etc.

One of the earlier works in this context is [7]. In [7], the log is considered as a stream of events, which is matched to the model which is also considered as a stream of events. In contrast to our approach, no guarantees are made about the optimality of the result and in special cases, the approach may not even terminate. In [22] and [21], techniques are presented that compare an abstraction of a process model with a log. In both cases, the process model is required to exhibit finite behavior. Furthermore, no alignment is provided. Instead, only a number that quantifies the conformance is returned.

Token-based replay techniques [20,23,17] can handle infinite behavior but need to resort to heuristics to deal with silent/duplicate activities. In some cases, the use of heuristics may lead to false negatives (i.e. perfectly fitting process executions are evaluated as non-fitting executions), as shown in [2]. Moreover, the user cannot set the severity of different deviations (i.e., non-conformance is measure in terms of missing and remaining tokens).

To overcome the limitations of earlier approaches (no guarantees for correctness and the inability to handle silent/duplicate activities), alignment-based techniques [3] were proposed by Adriansyah et al. These are tailored towards aligning the control flow of a procedural process model with a trace. They show that the A* algorithm can provide an efficient solution to the problem. Unfortunately, the approach cannot be straightforwardly extended to account, e.g., for the data perspective. Variables are generally defined over infinite domains; as a consequence, a (non-complete) alignment can be extended with an infinite number of moves. Each different value's assignment would create a different alignment. As result, the successors of a given search space node (i.e. the alignments obtained by adding a legal move to a non-complete alignment) are infinite in number. Therefore, the A* algorithm is not directly applicable because it requires the successors of a search space node to be finite. This paper uses a technique that limits the number of successors to three, two of which are obtained by solving two MILP problems.

This paper reports on a technique that is significantly different from what proposed in [13]. The approach presented in [13] performs the alignment computation in two steps. For each trace, a control-flow alignment is built leveraging on work [3]; then, the alignment is augmented with the write operations by solving a MILP problem. This approach is certainly faster since the A* algorithm only considers the control flow and one MILP problem needs to be solved in total. Unfortunately, in certain situations, the alignment is not optimal and, thus, can even return unlikely or wrong explanations of deviations from a domain perspective. This has also been confirmed by the real-life experiments reported on in Section 6.1. Since in this real-life case study the alignment is firstly computed only considering the control flow, there is no way to balance the costs related to data and control-flow and thus avoiding such wrong explanations. The technique proposed in our work is guaranteed to return optimal solutions and, hence, more likely explanations of diagnosed

deviations. This is due to the fact that the different perspectives are considered all together rather than sequentially.

Our approach notably advances beyond existing techniques for data-aware behavioral compliance checking [16,6]. There is more work related to compliance checking of business processes with regard to norms and regulations [10, 15,5,11]. In contrast to our work, these approaches focus on checking whether a process model can exhibit non-compliant behavior at design-time.

Some research works focus on verifying the compliance of process models with respect to a set of formulas, which are mostly intended to encode business rules of which one wants to verify the compliance (e.g. [16,6,18]) A log trace can possibly be represented by a set of formulas (e.g., an event for activity A is followed by an event for activity B) and, hence, its compliance can be checked by applying these works. Unfortunately, their diagnostics is limited to highlighting *which formulas* are not satisfied. We aim to pinpoint *where in the process* deviations occur, such as the case that an activity has not been executed or has written a wrong value for a variable. It is far from easy to derive the same insights on the basis of not-satisfied formulas. This is due to the fact that the same log trace can be "repaired" in multiple ways to satisfy one formula. When multiple, non-satisfied formulas come in to play, we would be interested in finding the least expensive changes that are needed to ensure all formulas are satisfied. In fact, this is again the problem of finding the least expensive solution in a certain search space, which is exactly what our application of the A* algorithm aims to be. To our knowledge, the same limitation is also shared by approaches that use alternative languages to handle verification with data variables in processes (e.g. [4]), as well as by techniques to debug the execution of distributed systems (e.g. [19,24]).

Efficient algorithms also exist to perform sequence alignments (e.g., the algorithms of Needleman-Wunsch and Smith-Waterman). Similarly, in process mining, Bose et al. [12] have proposed techniques to efficiently align pairs of log traces. Unfortunately, they cannot be applied to find an alignment between a log trace and a process model. In our setting, we do not know a priori the process trace to align with the log trace; conversely, the process trace needs to be chosen, thus minimizing the severity of the deviations. Moreover, sequence and trace alignments only focus on the activity names, i.e. the control-flow perspective, ignoring the other perspectives.

8 Conclusion

In recent years, many techniques have been proposed to evaluate a model's conformance with respect to given logs. As mentioned in Section 1, these techniques can only evaluate the conformance with respect to control-flow considerations and, hence, are unable to check, for example, the correctness of routing decisions, activities performed by unqualified resources, and activities that happen outside the correct time frame.

In [13], a technique was presented to align log traces and processes for conformance checking that takes all perspectives into account. The main drawback of that approach is that in certain situations the returned alignments are not optimal and, hence, more deviations are highlighted than necessary. As discussed, a sub-optimal solution may be returned if there are trade-offs between the different perspectives. As a consequence, the explanation of the deviations may be unlikely or, even, wrong from a business viewpoint. As an example, consider the conformance-checking analysis on the event log about road-fine management, which has been reported on in Sect. 6.1.4. We have shown that, for this case study, the approach discussed in [13] would return futile results in a number of cases, such as diagnosing non-plausible deviations on *Amount*. This experiments show that the balanced approach is also feasible in practice with event logs of a considerable, industry-strength size. Along with the evaluation of a real-life event log, we also tested the approach on several synthetic event logs to evaluate how the approach scales with logs of increasing size. While the problem is intrinsically NP-hard, the experiments show that solutions can be found in a reasonable amount of time.

In contrast to [13], our approach allows for arbitrary cost functions covering all perspectives. This allows us to express statements such as "Skipping activity a is more severe than executing activity a at too late a time" and "Executing activity a by a person not having role r is less severe than entering the wrong amount". For future work, we aim to investigate the nature and effects of different cost functions.

The application of the techniques described in different case studies demonstrated the importance of providing a helicopter view that summarizes where deviations occur most commonly. At the same time, one may be interested in looking at the specific alignments to dig into specific deviations at the case level. Therefore, as another part of our future work we plan to improve the visualization such that it is easier to explore a large set of alignments. It would also be interesting to not align traces in isolation. In several scenarios, e.g. in process security checking, the conformance of a case depends on the behavior observed in other cases that are being executed.

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards Robust Conformance Checking. In: Proceedings of the 6th Workshop on Business Process Intelligence (BPI 2010), *Lecture Notes in Business Information Processing*, vol. 66, pp. 122–133. Springer (2011)
3. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Memory-efficient alignment of observed and modeled behavior. Tech. rep., BPMcenter.org (2013). BPM Center Report BPM-13-03
4. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interactions using social integrity constraints. Electronic Notes in Theoretical Computer Science **85**(2) (2004)

5. Awad, A., Weidlich, M., Weske, M.: Specification, verification and explanation of violation for data aware compliance rules. In: L. Baresi, C.H. Chi, J. Suzuki (eds.) Service-Oriented Computing, *Lecture Notes in Computer Science*, vol. 5900, pp. 500–515. Springer (2009)
6. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of GSM-Based Artifact-Centric Systems through Finite Abstraction. In: Proceedings of the 10th International Conference on Service-Oriented Computing (ICSOC'12), *Lecture Notes in Computer Science*, vol. 7636, pp. 17–31. Springer (2012)
7. Cook, J.E., Wolf, A.L.: Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. ACM Transactions on Software Engineering and Methodology (TOSEM) **8**, 147–176 (1999)
8. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. Journal of the ACM (JACM) **32**, 505–536 (1985)
9. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press (1995)
10. Governatori, G., Milosevic, Z., Sadiq, S.W.: Compliance checking between business processes and business contracts. In: Proceedings of the 10th International Enterprise Distributed Object Computing Conference (EDOC 2006), pp. 221–232. IEEE Computer Society (2006)
11. Hoffmann, J., Weber, I., Governatori, G.: On compliance checking for clausal constraints in annotated process models. Information Systems Frontiers **14**(2), 155–177 (2012)
12. Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.P.: Process Diagnostics Using Trace Alignment: Opportunities, Issues, and Challenges. Information Systems **37**(2) (2012)
13. de Leoni, M., van der Aalst, W.M.P.: Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In: The 11th International Conference on Business Process Management (BPM'13), *Lecture Notes in Computer Science*, vol. 8094, pp. 113–129. Springer (2013)
14. de Leoni, M., van der Aalst, W.M.P.: Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments. In: Proc. of the 28th ACM symposium on Applied Computing (SAC'13). ACM (2013)
15. Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Syst. J. **46**(2), 335–361 (2007)
16. Ly, L., Rinderle-Ma, S., Knuplesch, D., Dadam, P.: Monitoring business process compliance using compliance rule graphs. In: Proceedings of OnTheMove Federated Conferences & Workshops (OTM 2011), *Lecture Notes in Computer Science*, vol. 7044, pp. 82–99. Springer (2011)
17. Alves de Medeiros, A.K., Weijters, A.J., van der Aalst, W.M.P.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery **14**, 245–304 (2007)
18. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, *Lecture Notes in Business Information Processing*, vol. 56. Springer (2010)
19. Reynolds, P., Killian, C., Wiener, J.L., Mogul, J.C., Shah, M.A., Vahdat, A.: Pip: detecting the unexpected in distributed systems. In: Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3, pp. 115–128. USENIX Association (2006)
20. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems **33**, 64–95 (2008)
21. Rozinat, A., Veloso, M., van der Aalst, W.M.P.: Using hidden markov models to evaluate the quality of discovered process models (2008). BPM Center Report BPM-08-10
22. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M.: Process compliance analysis based on behavioural profiles. Inf. Syst. **36**(7), 1009–1025 (2011)
23. Weijters, A.J.M.M., van der Aalst, W.M.P., Alves de Medeiros, A.K.: Process Mining with the Heuristics Miner-algorithm. Tech. rep., Eindhoven University of Technology, Eindhoven (2006). BETA Working Paper Series, WP 166
24. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I.: Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP'09), pp. 117–132. ACM (2009)