Information Systems 117 (2023) 102230

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Data-aware conformance checking with SMT[☆]

Paolo Felli [a], Alessandro Gianola [b], Marco Montali [b], Andrey Rivkin [b,c], Sarah Winkler [b,*]

[a] Faculty of Computer Science, University of Bologna, Bologna, Italy
[b] Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
[c] Technical University of Denmark, Kongens Lyngby, Denmark

A R T I C L E I N F O

Article history:
Received 22 March 2022
Received in revised form 27 February 2023
Accepted 17 May 2023
Available online 26 May 2023
Recommended by Gottfried Vossen

Keywords:
Conformance checking
Data Petri nets
SMT

A B S T R A C T

Conformance checking is a key process mining task to confront the normative behavior imposed by a process model with the actual behavior recorded in a log. While this problem has been extensively studied for pure control-flow processes, data-aware conformance checking has received comparatively little attention. In this paper, we tackle the conformance checking problem for the challenging scenario of processes that combine data and control-flow dimensions. Concretely, we adopt the formalism of data Petri nets (DPNs) and show how solid, well-established automated reasoning techniques from the area of Satisfiability Modulo Theories (SMT) can be effectively harnessed to compute conformance metrics and optimal data-aware alignments. To this end, we introduce the CoCoMoT (Computing Conformance Modulo Theories) framework, with a fourfold contribution. First, we show how SMT allows to leverage SAT-based encodings for the pure control-flow setting to the data-aware case. Second, we introduce a novel preprocessing technique based on a notion of property-preserving clustering, to speed up the computation of conformance checking outputs. Third, we show how our approach extends seamlessly to the more comprehensive conformance checking artifacts of multi- and anti-alignments. Fourth, we describe a proof-of-concept implementation based on state-of-the-art SMT solvers, and report on experiments. Finally, we discuss how CoCoMoT directly lends itself to further process mining tasks like log analysis by clustering and model repair, and the use of SMT facilitates the support of even richer multi-perspective models, where, for example, more expressive DPN guards languages are considered or generic datatypes (other than integers or reals) are employed.

1. Introduction

In process mining, the task of conformance checking is crucial to test the expected behavior described by a process model against the actual action sequences documented in a log [1]. While the problem has been thoroughly studied for pure control-flow processes such as classical Petri nets [1,2], the situation changes for process models equipped with additional perspectives beyond the control-flow, such as for example the data perspective. Notice that, while there are various works that primarily focus on the formalization and analysis of data or object-aware extensions of Petri nets (e.g., [3–6]), attacking the conformance checking problem in the non-classical setting is a very challenging task. Indeed, this problem requires to simultaneously consider, in a combined way, both the control-flow of the process and the data that the process manipulates. These two components are inter-dependent, with the control flow generating data constrained by expressive logics, and the data in turn influencing the control flow. Existing approaches almost exclusively focused on control-flow alignments and can therefore not be applied off-the-shelf. To the best of our knowledge, there are in fact very few existing approaches dealing with the aforementioned problem, and they concentrate on declarative [7] and procedural [8,9] multi-perspective process models with rather restrictive assumptions on the data dimension.

In this paper, we provide a new stepping stone in the line of research focused on conformance checking of multi-perspective procedural, Petri net-based process models. Specifically, we consider data Petri nets (DPNs), an extensively studied formalism within BPM [10–12] and process mining [8,9,13], which allows for a succinct but flexible and expressive model presentation. For this setting, we introduce a novel conformance checking framework called CoCoMoT (Computing Conformance Modulo Theories). The main feature of CoCoMoT is that, instead of providing ad-hoc algorithmic techniques for checking conformance, it

☆ This work was partially supported by the UNIBZ, Italy projects ADAPTERS, SMART-APP, REKAP, VERBA and DUB, and the PRIN MIUR project PINPOINT Prot. 2020FNEB27.

* Corresponding author.
E-mail addresses: paolo.felli@unibo.it (P. Felli), gianola@inf.unibz.it (A. Gianola), montali@inf.unibz.it (M. Montali), ariv@dtu.dk (A. Rivkin), winkler@inf.unibz.it (S. Winkler).

https://doi.org/10.1016/j.is.2023.102230

provides an overarching approach based on the theory and practice of Satisfiability Modulo Theories (SMT) [14]: SMT provides a very expressive and flexible framework supporting efficient algorithmic techniques that are exploited by state-of-the-art and highly performing solvers, and that can deal with a wide range of different datatypes, making them particularly suitable for being applied in a data-aware context. Respective SMT solvers [15,16] are mature, efficient, and highly optimized tools that we use in our implementation. By relying on an SMT backend, we exploit well-established automated reasoning techniques that can support data and operations from a variety of theories, restricting the data dimension as little as possible.

On top of this basis, we provide a fourfold contribution. First, we show that conformance checking of DPNs with arithmetic constraints can be reduced to satisfiability of SMT formulas over the theory of linear integer and rational arithmetic. While our approach is inspired by the use of SAT solvers for a similar purpose [17,18], the use of SMT not only allows us to account for data manipulating guards, but also capture unbounded nets. Our CoCoMoT approach results in a conformance checking procedure running in NP, which we prove optimal for the problem, in contrast to earlier conformance checking approaches for DPNs running in exponential time [8,9]. We also formally demonstrate the correctness of our approach and provide an extensive discussion on how various SMT theories can be used in order to extend the theoretical setting of our conformance checking framework.

Second, we show how to simplify and optimize conformance checking by introducing a preprocessing, trace clustering technique for DPNs that groups together traces that have the same minimal alignment cost. Clustering allows one to compute conformance metrics by just computing alignments for one representative per cluster, and to obtain alignments for other members of the same cluster from a simple adjustment of the alignment computed for the representative trace. Besides the general notion of clustering, we then propose a concrete clustering strategy grounded in data abstraction for variable-to-constant constraints, which are often present in automatically mined nets, and show how this strategy leads to a significant speedup in our experiments.

Third, we show how due to its flexibility, our approach can be seamlessly extended to further conformance checking artifacts, namely multi- and anti-alignments.

Fourth, we report on a proof-of-concept implementation of CoCoMoT that does not only support plain conformance checking, but also lends itself to performing search of multi- and anti-alignments. We discuss optimizations and show the feasibility of the approach with an experimental evaluation on three different benchmark sets available in the literature.

This article extends the conference version of this paper [19] in several ways:

(i) we provide complete proofs of all formal results, including the correctness of the decoding, the complexity result, and correctness of clustering;
(ii) we lay out the extension of our approach to multi- and anti-alignments, which are now also covered by our implementation;
(iii) we extended the experimental section, to evaluate in detail the performance of our implementation on different data sets, assess the scalability of our approach, and compare with the other conformance checking tool for DPNs proposed in [8,9];
(iv) we devise a concrete way to compute an upper bound to the length of the alignment – a crucial ingredient to our method which was so far determined by heuristics in related SAT-based approaches – and formally demonstrate that this computation scheme comes with a correctness guarantee;
(v) we elaborate the decoding phase of our approach, specifying how to construct an optimal alignment from the output of the SMT solver; and
(vi) we include a detailed section about related work.

The remainder of the paper is structured as follows. In Section 2 we recall the relevant basics about data Petri nets, alignments, cost functions and distances, and define the conformance checking problem. This paves the way to present our SMT encoding (together with related theoretical results) in Section 3. Our clustering technique that serves as a preprocessor for conformance checking is the topic of Section 4. The treatment of multi- and anti-alignments is discussed in Section 5. In Section 6 we describe our implementation and the conducted experiments. Related work is discussed in Section 7. In Section 8 we conclude.

2. Data-aware conformance checking

In this section, we present formal foundations of the data-aware conformance checking framework used in this paper. First, in Section 2.1 we provide preliminaries on data Petri nets (DPNs) and their execution semantics. Then we introduce the conformance checking problem and discuss a distance-based measure we use for computing optimal alignments. Finally, we briefly introduce satisfiability modulo theories (SMT) — the main machinery behind our approach.

2.1. Data Petri nets

We use Data Petri nets (DPNs) for modeling multi-perspective processes, adopting a formalization as in [8,9].

To capture the data types of variables that are manipulated by a process, we fix a set of *(process variable) types* $\Sigma = \{\texttt{bool}, \texttt{int}, \texttt{rat}, \texttt{string}\}$ with associated domains: $\mathcal{D}(\texttt{bool}) = \mathbb{B}$, the booleans; $\mathcal{D}(\texttt{int}) = \mathbb{Z}$, the integers; $\mathcal{D}(\texttt{rat}) = \mathbb{Q}$, the rational numbers; and $\mathcal{D}(\texttt{string}) = \mathbb{S}$, the strings over some fixed alphabet. A set of *process variables* V is *typed* if there is a function $type\colon V \to \Sigma$ assigning a type to each variable in V. For a set of variables V, we consider two disjoint sets of annotated variables $V^r = \{v^r \mid v \in V\}$ and $V^w = \{v^w \mid v \in V\}$ to be respectively read and written by process activities, as explained below, and we assume $type(v^r) = type(v^w) = type(v)$ for every $v \in V$. Given V and a type $\sigma \in \Sigma$ (for example, σ can be the integer type), V_σ denotes the subset of annotated variables in $V^r \cup V^w$ of type σ. To manipulate typed variables, we consider the following expressions:

Definition 1. For a set of variables V, *constraint* c and expressions s, n, and r of types `string`, `int`, and `rat` are defined by the following grammar:

$$c ::= v_b \mid b \mid n \geq n \mid r \geq r \mid r > r \mid s = s \mid c \wedge c \mid \neg c \qquad s ::= v_s \mid t$$

$$n ::= v_z \mid z \mid n + n \mid -n \qquad r ::= v_r \mid q \mid r + r \mid -r$$

where $v_b \in V_{\texttt{bool}}$, $b \in \mathbb{B}$, $v_s \in V_{\texttt{string}}$, $t \in \mathbb{S}$, $v_z \in V_{\texttt{int}}$, $z \in \mathbb{Z}$, $v_r \in V_{\texttt{rat}}$, and $q \in \mathbb{Q}$.

We denote the set of all constraints over variables V by $\mathcal{C}(V)$, and the set of variables that appear in a constraint c is denoted by $\mathcal{V}ar(c)$, hence $\mathcal{V}ar(c) \subseteq V^w \cup V^r$. Standard equivalences apply, hence disjunction (i.e., $\vee$) of constraints can be used, as well as comparisons $\neq$, $<$, $\leq$ on integer and rational expressions, though expressions of types `bool` and `string` only support (in)equality comparisons. The constraints defined in Definition 1 serve to express conditions on the values of variables that are read and written during the execution of activities in the process. Intuitively, a constraint $(x^r > y^r)$ for $x, y \in V$ states that the current value of variable x is greater than the current value of y. Similarly,

$(x^w > y^r + 1) \wedge (x^w < z^r)$ requires that the new value given to x (i.e., assigned to x as a result of executing the activity to which this constraint is attached) is greater than the current value of y plus 1, and smaller than z. In general, given a constraint c as above, we refer to the annotated variables in V^r and V^w that appear in c as the *read* and *written variables*, respectively.

Definition 2 (*DPN*). A *Petri net with data* (DPN) is given by a tuple $\mathcal{N} = (P, T, F, \ell, A, V, guard)$, where:

- (P, T, F, ℓ) is a Petri net with two non-empty disjoint sets of places P and transitions T, a flow relation $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ and a labeling function $\ell : T \to A \cup \{\tau\}$, where A is a finite set of activity labels and τ is a special symbol for silent transitions;
- V is a typed set of process variables; and
- $guard : T \to \mathcal{C}(V)$ assigns executability constraints.

As customary, given $x \in P \cup T$, we use ${}^\bullet x := \{y \mid F(y, x) > 0\}$ to denote the *preset* of x and $x^\bullet := \{y \mid F(x, y) > 0\}$ to denote the *postset* of x. In order to refer to the variables read and written by a transition t, we use the notations $read(t) = \{v \mid v^r \in \mathcal{V}ar(guard(t))\}$ and $write(t) = \{v \mid v^w \in \mathcal{V}ar(guard(t))\}$. Finally, $G_{\mathcal{N}}$ is the set of all the guards appearing in $\mathcal{N}$.

Hereinafter, we follow an important assumption that *forbids silent transitions to modify process variables*. Formally, for every $t \in T$ s.t. $\ell(t) = \tau$, it holds that $write(t) = \emptyset$. This assumption is very natural as it prohibits process models to perform any adversarial, unobservable changes on the process data that also cannot be aligned with data in logs.

From now on, we assume that V is the set of process variables. To assign values to variables, we use assignments. A *state variable assignment* is a function $\alpha : V \to \bigcup_{v \in V} \mathcal{D}(type(v))$ such that $\alpha(v) \in \mathcal{D}(type(v))$ for all $v \in V$: notice that $\textsc{dom}(\alpha) = V$ where $\textsc{dom}$ denotes the domain of functions. State variable assignments are used to specify the current value of process variables. Similarly, a *transition variable assignment* is a partial function β with $\textsc{dom}(\beta) \subseteq V^r \cup V^w$ that assigns a value to annotated variables, namely $\beta(x) \in \mathcal{D}(type(x))$, with $x \in V^r \cup V^w$. Transition variable assignments are used to specify how variables change as the result of activity executions (cf. Definition 3).

A *state* in a DPN $\mathcal{N}$ is a pair (M, α) constituted by a marking $M : P \to \mathbb{N}$ for the underlying Petri net (P, T, F, ℓ), plus a state variable assignment α. Therefore, a state simultaneously accounts for the control flow progress and for the current values of all variables in V, as specified by α. For ease of notation, we denote with $[p_1^{i_1}, \ldots, p_n^{i_n}]$ a concrete multiset representing a marking in which each place p_k contains i_k tokens.

We now define when a Petri net transition may fire from a given state.

Definition 3 (*Transition firing*). A transition $t \in T$ is *enabled* in state (M, α) if there exists a transition variable assignment β such that:

- $\textsc{dom}(\beta) = \mathcal{V}ar(guard(t))$: β is defined for the variables in the guard;
- $\beta(v^r) = \alpha(v)$ for every $v \in read(t)$, i.e., β is as α for read variables;
- $\beta \models guard(t)$, i.e., β satisfies the guard; and
- $M(p) \geq F(p, t)$ for every $p \in {}^\bullet t$.

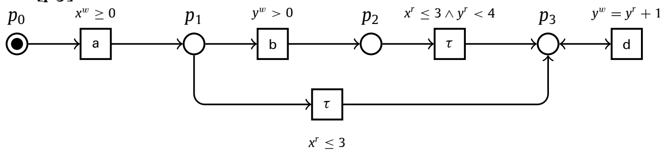
An enabled transition may *fire*, producing a new state (M', α'), s.t. $M'(p) = M(p) - F(p, t) + F(t, p)$ for every $p \in P$, and $\alpha'(v) = \beta(v^w)$ for every $v \in write(t)$, and $\alpha'(v) = \alpha(v)$ for every $v \notin write(t)$. A pair (t, β) as above is called (valid) *transition firing*, and we denote its firing by $(M, \alpha) \xrightarrow{(t,\beta)} (M', \alpha')$.

Given $\mathcal{N}$, we fix one state (M_I, α_0) as *initial*, where M_I is the initial marking of the underlying Petri net (P, T, F, ℓ) and α_0 specifies the initial value of all variables in V. Similarly, we denote the final marking as M_F, and call a state of the form (M_F, α_F) for some α_F *final*.

We say that a state (M, α) is *reachable* in a DPN iff there exists a sequence of transition firings $\mathbf{f} = \langle f_1, \ldots, f_n \rangle = \langle (t_1, \beta_1), \ldots, (t_n, \beta_n) \rangle$, s.t. $(M_I, \alpha_0) \xrightarrow{(t_1,\beta_1)} (M_1, \alpha_1) \xrightarrow{(t_2,\beta_2)} \ldots \xrightarrow{(t_n,\beta_n)} (M_n, \alpha_n) = (M, \alpha)$, denoted as $(M_I, \alpha_0) \xrightarrow{\mathbf{f}} (M, \alpha)$. Moreover, $\mathbf{f}$ is called a (valid) *process run* of $\mathcal{N}$ if $(M_I, \alpha_0) \xrightarrow{\mathbf{f}} (M_F, \alpha_F)$ for some α_F, so that the run starts in the initial state and ends in a final state. Similar to [8], we restrict our setting to DPNs that are *relaxed data sound*, i.e., where at least one final state is reachable.

We denote the set of valid transition firings of a DPN $\mathcal{N}$ as $\mathcal{F}(\mathcal{N})$, and the set of process runs as $Runs(\mathcal{N})$.

Example 1. Let $\mathcal{N}$ be as shown where the marking $[p_0]$ is initial and $[p_3]$ final:



For the initial assignment $\alpha_0 = \{x \mapsto 0, y \mapsto 0\}$, the set $Runs(\mathcal{N})$ contains e.g. $\mathbf{f}_1 = \langle (\mathsf{a}, \{x^w \mapsto 2\}), (\mathsf{b}, \{y^w \mapsto 1\}), (\tau, \{x^r \mapsto 2, y^r \mapsto 1\}) \rangle$ as well as $\mathbf{f}_2 = \langle (\mathsf{a}, \{x^w \mapsto 1\}), (\tau, \{x^r \mapsto 1\}), (\mathsf{d}, \{y^w \mapsto 1\}) \rangle$.

2.2. Event logs, alignments and distance-based cost functions

Given an arbitrary set A of activity labels and a set V of variables, an *event* is a pair (b, α), where $b \in A$ and α is a so-called *event variable assignment*, that is, a function that associates values to variables in V. Differently from state variable assignments, an event variable assignment can be a partial function.

Definition 4 (*Log trace, event log*). Given a set $\mathcal{E}$ of events, a *log trace* $\mathbf{e} \in \mathcal{E}^*$ is a sequence of events in $\mathcal{E}$ and an *event log* $L \in \mathcal{M}(\mathcal{E}^*)$ is a multiset of log traces from $\mathcal{E}$, where $\mathcal{M}(\mathcal{E}^*)$ denotes the set of all multisets over $\mathcal{E}^*$.

We focus on a conformance checking procedure that aims at constructing an *alignment* of a given log trace $\mathbf{e}$ w.r.t. a process model (i.e., a DPN $\mathcal{N}$), by matching events in the log trace against transition firings in the process runs of $\mathcal{N}$. However, when constructing an alignment, not every event can always be put in correspondence with a transition firing, and vice versa. Therefore, we introduce a special "skip" symbol $\gg$ and the extended set of events $\mathcal{E}^{\gg} = \mathcal{E} \cup \{\gg\}$ and, given $\mathcal{N}$, the extended set of transition firings $\mathcal{F}^{\gg} = \mathcal{F}(\mathcal{N}) \cup \{\gg\}$.

Definition 5 (*Move*). Given a DPN $\mathcal{N}$ and a set of events $\mathcal{E}$, a pair $(e, f) \in \mathcal{E}^{\gg} \times \mathcal{F}^{\gg} \setminus \{(\gg, \gg)\}$ is called *move*.[1] A move (e, f) is called:

- *log move* if $e \in \mathcal{E}$ and $f = \gg$;
- *model move* if $e = \gg$ and $f \in \mathcal{F}(\mathcal{N})$;
- *synchronous move* if $(e, f) \in \mathcal{E} \times \mathcal{F}(\mathcal{N})$.

Let $Moves_{\mathcal{N}}$ be the set of all such moves. We now show how moves can be used to define alignments of log traces.

[1] In contrast to [8], we do not distinguish between synchronous moves with correct and incorrect write operations, and defer this differentiation to the cost function.

For a sequence of moves $\gamma = \langle (e_1, f_1), \dots, (e_n, f_n)\rangle$, the *log projection* $\gamma|_L$ of γ is the longest subsequence $\mathbf{e}' = \langle e'_1, \dots, e'_i\rangle$ of $\mathbf{e} = \langle e_1, \dots, e_n\rangle$ such that $\mathbf{e}' \in \mathcal{E}^*$, i.e., $\mathbf{e}'$ is obtained from $\mathbf{e}$ by omitting all $\gg$ symbols. Similarly, the *model projection* $\gamma|_M$ of γ is the longest subsequence $\mathbf{f}' = \langle f'_1, \dots, f'_j\rangle$ of $\mathbf{f} = \langle f_1, \dots, f_n\rangle$ such that $\mathbf{f}' \in \mathcal{F}(\mathcal{N})^*$, i.e., $\mathbf{f}'$ is obtained from $\mathbf{f}$ by projecting away all $\gg$ symbols. We also write $\mathbf{x}|_j$, $0 \le j \le n$, for a prefix of length j of a sequence $\mathbf{x} = \langle x_1, \dots, x_n\rangle$, and denote an empty prefix (i.e., for $j = 0$) with ϵ.

Definition 6 (*Alignment*)**.** Given a DPN $\mathcal{N}$, a sequence of moves γ is an *alignment* of a log trace $\mathbf{e}$ if $\gamma|_L = \mathbf{e}$, and it is *complete* if $\gamma|_M \in Runs(\mathcal{N})$.

Notice that in the case of complete alignments, it is implied that the final marking of $\mathcal{N}$ is reached.

Example 2. The sequences γ_1, γ_2 and γ_3 below are complete alignments of the log trace $\mathbf{e} = \langle (\mathsf{a}, \{x \mapsto 2\}), (\mathsf{b}, \{y \mapsto 1\})\rangle$ w.r.t. the DPN from Example 1:

γ_1:

a $\{x \mapsto 2\}$	b $\{y \mapsto 1\}$	$\gg$
a $\{x^w \mapsto 2\}$	b $\{y^w \mapsto 1\}$	τ

γ_2:

a $\{x \mapsto 2\}$	$\gg$	b $\{y \mapsto 1\}$
a $\{x^w \mapsto 3\}$	τ	$\gg$

γ_3:

a $\{x \mapsto 2\}$	b $\{y \mapsto 1\}$	$\gg$	$\gg$
$\gg$	$\gg$	a $\{x^w \mapsto 3\}$	τ

Here, alignments are depicted as tables where the cells in the top row correspond to trace events, and cells in the bottom row correspond to transitions in the model run. Cells of trace events contain activity labels and event variable assignments; in cells for model transitions we list the written variables (instead of the entire transition variable assignment, for the sake of readability).

For a process run $\mathbf{f}$, we denote by $Align(\mathbf{f}, \mathbf{e})$ the set of alignments γ such that $\gamma|_L = \mathbf{e}$ and $\gamma|_M = \mathbf{f}$. Moreover, $Align(\mathcal{N}, \mathbf{e})$ is the set of complete alignments for a log trace $\mathbf{e}$ w.r.t. a DPN $\mathcal{N}$. A *cost function* is a mapping $\kappa : Moves_{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ that assigns a cost to every move. It is naturally extended to alignments as follows.

Definition 7 (*Alignment cost*)**.** Let $\mathcal{N}$ be a DPN and γ a complete alignment of a log trace $\mathbf{e}$ with respect to $\mathcal{N}$, i.e., $\gamma \in Align(\mathcal{N}, \mathbf{e})$. For $\gamma = \langle (e_1, f_1), \dots, (e_n, f_n)\rangle$, the *cost* of γ is given by $\kappa(\gamma) = \sum_{i=1}^{n} \kappa(e_i, f_i)$.

Thus, the cost of an alignment is given by the sum of the costs of its moves. This allows us to define the notion of an optimal alignment for a log trace with respect to a DPN and a given cost function:

Definition 8 (*Optimal alignment*)**.** A complete alignment $\gamma \in Align(\mathcal{N}, \mathbf{e})$ for a log trace $\mathbf{e}$ with respect to a DPN $\mathcal{N}$ is *optimal* if $\kappa(\gamma)$ is minimal among all complete alignments for $\mathbf{e}$, i.e., there is no $\gamma' \in Align(\mathcal{N}, \mathbf{e})$ s.t. $\kappa(\gamma') < \kappa(\gamma)$.

The cost of an optimal alignment for $\mathbf{e}$ with respect to $\mathcal{N}$ is denoted by $\kappa^{opt}_{\mathcal{N}}(\mathbf{e})$. Given $\mathcal{N}$, we denote the set of optimal alignments for $\mathbf{e}$ as $Align^{opt}(\mathcal{N}, \mathbf{e})$. Indeed, optimal alignments need not be unique, as illustrated in Example 3 below after defining specific cost functions.

Various metrics to evaluate process models against observed behaviors in the log are conceivable. In this work, we use a distance metric that guides the alignment procedure so that the latter finds a model run that is as close as possible to the observed trace. To this end, we formalize the distance between a log trace and a process run as follows:

Definition 9 (*Distance*)**.** Given a log trace $\mathbf{e} \in \mathcal{E}^*$ and a process run $\mathbf{f} \in Runs(\mathcal{N})$, the *distance* between $\mathbf{e}$ and $\mathbf{f}$ is defined as $dist_\kappa(\mathbf{e}, \mathbf{f}) = \min_{\gamma \in Align(\mathbf{f}, \mathbf{e})} \kappa(\gamma)$.

In other words, the distance between $\mathbf{e}$ and $\mathbf{f}$ is the minimal number of edits needed to align the process model behavior with the one observed in the log. It is easy to see that optimal alignments can be formalized using the notion of distance: indeed, the optimal alignment cost for $\mathbf{e}$ is given by the minimum distance $dist(\mathbf{e}, \mathbf{f})$ between $\mathbf{e}$ and $\mathbf{f}$, for all $\mathbf{f} \in Runs(\mathcal{N})$.

In order to effectively compute an optimal alignment, we will use a variant of the well-known notion of *edit distance* (also known as *Levenshtein distance*), as is standard in conformance checking [2]. This allows us in particular to adopt a similar SAT-based encoding as used in the literature [20]. However, we adopt a more general version of the edit distance that can be used not only to represent the usual Levenshtein distance, but also its variants such as the *discounted cost function* discussed in [17] and distance functions previously used for multi-perspective conformance checking [8,9] (the latter we will call *standard cost function*, as discussed below).

Our cost measure is parameterized by three penalty functions:

$$P_L : \mathcal{E} \to \mathbb{N} \qquad P_M : \mathcal{F}(\mathcal{N}) \to \mathbb{N} \qquad P_= : \mathcal{E} \times \mathcal{F}(\mathcal{N}) \to \mathbb{N}$$

respectively called the *log move penalty*, *model move penalty*, and *synchronous move penalty* functions (cf. Section 2.2). These functions naturally give rise to a cost function $\kappa_{dist} : Moves_{\mathcal{N}} \to \mathbb{R}_{\geq 0}$, by using P_L for log moves, P_M for model moves, and $P_=$ for synchronous moves. In this case we say that κ_{dist} is a cost function that *has parameters* P_L, P_M, and $P_=$.

Definition 10. A cost function κ is *distance-based* if it is defined as $\kappa(\gamma) = \sum_{i=1}^{n} \kappa_{dist}(e_i, f_i)$ for some κ_{dist} having parameters P_L, P_M, and $P_=$, for all $\gamma = \langle (e_1, f_1), \dots, (e_n, f_n)\rangle$.

In the sequel, we consider such *distance-based* cost functions.

Definition 11 (*Generalized edit distance*)**.** Given a DPN $\mathcal{N}$, let $\mathbf{e} = \langle e_1, \dots, e_m\rangle$ be a log trace and $\mathbf{f} = \langle f_1, \dots, f_n\rangle$ a process run of $\mathcal{N}$. For all i and j, $0 \le i \le m$ and $0 \le j \le n$, the *edit distance* $\delta(\mathbf{e}|_i, \mathbf{f}|_j)$ is recursively defined as follows:

$$\delta(\epsilon, \epsilon) = 0$$
$$\delta(\mathbf{e}|_{i+1}, \epsilon) = P_L(e_{i+1}) + \delta(\mathbf{e}|_i, \epsilon)$$
$$\delta(\epsilon, \mathbf{f}|_{j+1}) = P_M(f_{j+1}) + \delta(\epsilon, \mathbf{f}|_j)$$
$$\delta(\mathbf{e}|_{i+1}, \mathbf{f}|_{j+1}) = \min \begin{cases} P_=(e_{i+1}, f_{j+1}) + \delta(\mathbf{e}|_i, \mathbf{f}|_j) \\ P_L(e_{i+1}) + \delta(\mathbf{e}|_i, \mathbf{f}|_{j+1}) \\ P_M(f_{j+1}) + \delta(\mathbf{e}|_{i+1}, \mathbf{f}|_j) \end{cases}$$

In fact, given a log trace $\mathbf{e}$ and a process run $\mathbf{f}$, the edit distance $\delta(\mathbf{e}, \mathbf{f})$ is exactly the distance $dist(\mathbf{e}, \mathbf{f})$ from Definition 9, taking κ_{dist} as cost function, as stated next:

Lemma 1. *A distance-based cost function κ satisfies $dist_\kappa(\mathbf{e}, \mathbf{f}) = \delta(\mathbf{e}, \mathbf{f})$.*

This can be shown using known properties of the edit distance [17,21], and for our setting we will later on show optimality directly.

We next demonstrate how by fixing the functional parameters $P_=$, P_L, and P_M, one can obtain concrete, known distance-based cost functions are obtained.

Standard cost function. As defined in [8, Ex. 2] and [9, Def. 4.5], the following standard cost function is a "default" choice that can be used, for example, for providing initial assessments on the log conformance. To this end, we instantiate the distance function from Definition 10 as follows:

$$P_L(b, \alpha) = 1$$
$$P_M(t, \beta) = \begin{cases} 0, & \text{if } \ell(t) = \tau \\ |write(t)| + 1, & \text{otherwise} \end{cases}$$

$$P_=((b,\alpha),(t,\beta)) = \begin{cases} |\{v \in \text{DOM}(\alpha) \mid \alpha(v) \neq \beta(v^w)\}|, & \text{if } \ell(t) = b \\ \infty, & \text{otherwise} \end{cases}$$

Example 3. Consider alignments γ_1, γ_2, and γ_3 from Example 2. Using the standard cost function κ, we get the following:

- $\kappa(\gamma_1) = 0$, since P_M does not penalize moves with silent transitions;
- $\kappa(\gamma_2) = 2$, because $P_=$ penalizes with 1 synchronous moves with incorrect write operations and P_L always returns 1 for log moves (as in the previous case, model moves with silent transitions are not penalized);
- $\kappa(\gamma_3) = 4$, since, as in the previous case, we get 1 for each of the log moves and no penalty for the silent model move, but we get 2 for a non-silent model move that writes one variable.

However, optimal alignments need not be unique. The trace $\mathbf{e} = \langle(\mathsf{b}, \{y \mapsto 1\})\rangle$ admits the two alignments

$\gg$	b $\{y \mapsto 1\}$	$\gg$
a $\{x^w \mapsto 1\}$	b $\{y^w \mapsto 1\}$	τ

and

$\gg$	b $\{y \mapsto 1\}$	$\gg$
a $\{x^w \mapsto 3\}$	b $\{y^w \mapsto 1\}$	τ

which are both optimal with cost 2.

Levenshtein distance. Another natural way of estimating how "far" two sequences are from each other uses the plain Levenshtein distance, computing the minimum number of edits needed to transform one sequence into another. This measure can be obtained by the following edit distance instantiation:

$$P_L(b,\alpha) = 1$$
$$P_M(t,\beta) = 1$$
$$P_=((b,\alpha),(t,\beta)) = \begin{cases} 0, & \text{if } \ell(t) = b \\ \infty, & \text{otherwise} \end{cases}$$

Note that this measure is "data-agnostic" as it ignores transition variable assignments.

Discounted cost function. Following the recent proposal in [17], we slightly modify the P_L and P_M functions, and extend the above Levenshtein distance with an ability to penalize more those log and model moves that occur at early stages of the alignment construction:

$$P_L(b,\alpha,\mathbf{e}) = 1 + \theta^{-\pi_L(\mathbf{e},b)}$$
$$P_M(t,\beta,\mathbf{f}) = 1 + \theta^{-\pi_M(\mathbf{f},t)}$$

Here, $\theta \in \mathbb{R}_{\geq 1}$ is a discount parameter, and $\pi_L(\mathbf{e}, b)$ and $\pi_M(\mathbf{f}, t)$ functions that provide positions at which b and t respectively occur in $\mathbf{e}$ and $\mathbf{f}$. In [17] it is suggested that with $\theta = 2$ the discount becomes severe enough, as the edit at a given position can cost more than the sum of all the following edits.

2.3. Satisfiability Modulo Theories (SMT)

Here, we introduce the main technical machinery exploited in this paper: SMT solving. In order to describe it formally, we assume the usual syntactic (e.g., signature, variable, term, atom, literal, and formula) and semantic (e.g., structure, truth, satisfiability, and validity) notions of first-order logic. The equality symbol = is always included in all signatures.

Formally, according to the current practice in the SMT literature [14,22], a theory $\mathcal{T}$ is a pair (Σ, Z), where Σ is a signature and Z is a class of Σ-structures; the structures in Z are the models of T. A Σ-formula ϕ is $\mathcal{T}$-satisfiable if there exists a Σ-structure $\mathcal{M}$ in Z such that ϕ is true in $\mathcal{M}$ under a suitable assignment a to the free variables of ϕ (in symbols, $(\mathcal{M}, \mathsf{a}) \models \phi$); it is $\mathcal{T}$-valid (in symbols, $\mathcal{T} \vdash \phi$) if its negation is $\mathcal{T}$-unsatisfiable. Two formulae ϕ_1 and ϕ_2 are $\mathcal{T}$-equivalent if $\phi_1 \leftrightarrow \phi_2$ is $\mathcal{T}$-valid. The problem of (quantifier-free) *satisfiability modulo the theory* $\mathcal{T}$ (*SMT*($\mathcal{T}$)) amounts to establishing the $\mathcal{T}$-satisfiability of quantifier-free Σ-formulae.

Intuitively, the Satisfiability Modulo Theories (SMT) problem [14] is an extension of the classic propositional satisfiability (SAT) problem. The SAT problem asks, given a propositional formula φ, to either find a satisfying assignment ν under which φ evaluates to true, or detect that φ is unsatisfiable. For instance, given the formula $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg r \vee \neg q)$, a satisfying assignment is $\nu(p) = \nu(r) = \top$, $\nu(q) = \bot$, where, as customary in first-order logic, $\top$ stands for the truth value 'true' and $\bot$ stands for the truth value 'false'. The SMT problem extends SAT by asking to decide satisfiability of a formula φ whose language extends propositional formulas by constants and operators from one or more first-order theories $\mathcal{T}$. There exists a plethora of solvers, called *SMT solvers*, able to solve the SMT problem: they extend SAT solvers with specific decision procedures customized for the specific theories involved. SMT solvers are useful both for computer-aided verification, to prove the correctness of software programs against some property of interest, and for synthesis, to generate candidate program fragments. Examples of well-studied SMT theories are the theory of uninterpreted functions $\mathcal{EUF}$, the theory of bitvectors $\mathcal{BV}$ and the theory of arrays $\mathcal{AX}$ [14,22]. All these theories are usually employed in applications to program verification. For this paper, only the theories of linear integer and rational arithmetic ($\mathcal{LIA}$ and $\mathcal{LQA}$) are relevant. For instance, the SMT formula $a > 1 \wedge (a + b = 10 \vee a - b = 20) \wedge p$, where a, b are integer and p is a propositional variable, is satisfiable by the assignment ν such that $\nu(a) = \nu(b) = 5$ and $\nu(p) = \top$.

Another important problem studied in the area of SMT and relevant to this paper is the one of Optimization Modulo Theories (OMT) [23]. The OMT problem asks, given a formula φ, to find a satisfying assignment of φ that minimizes or maximizes a given objective expression. SMT-LIB [22] is an international initiative aiming at providing an extensive on-line library of benchmarks and promoting the adoption of common languages and interfaces for SMT solvers. In this paper, we make use of the SMT solvers Yices 2 [16] and Z3 [15].

3. Conformance checking via SMT

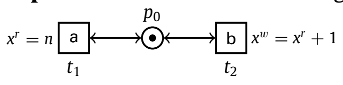
This section describes the core of our approach, by presenting the encoding building blocks used in the CoCoMoT framework. We first explain how a bound on the length of a run in an optimal alignment can be computed (Section 3.1). This bound is then used as the basis SMT-based encoding that we introduce in Section 3.2 to solve the problem of finding optimal alignments. In Section 3.3 we prove correctness, and in Section 3.4 we analyze the computational complexity.

3.1. Length bound on optimal alignment

Given a DPN $\mathcal{N}$ and a log trace $\mathbf{e}$, our approach aims to find an optimal alignment by constructing a symbolic representation both of a process run and an alignment, and subsequently "grounding" that representation to a concrete process run $\mathbf{f}$ and an alignment γ, using an SMT solver. Since the symbolic representation is encoded using a finite set of variables, we need to fix upfront an upper bound on the optimal alignment size. This, in turn, amounts to fixing an upper bound $n_{\mathbf{f}}$ on the symbolic representation of the process run (that is, its length), so that there is some $\gamma \in Align^{opt}(\mathcal{N}, \mathbf{e})$ for which $n_{\mathbf{f}} \geq \gamma|_M$. Notice that such an upper bound was already required in earlier SAT-based approaches [17,20].

In general, the above upper bound strictly depends on the cost function of choice. Hereinafter, for simplicity, we consider the standard cost function from Section 2.2. But even for this cost function, computation of a suitable upper bound is considerably more intricate in the presence of guards that manipulate data. We elaborate on this in the next example.

Example 4. Consider the following DPN $\mathcal{N}$:



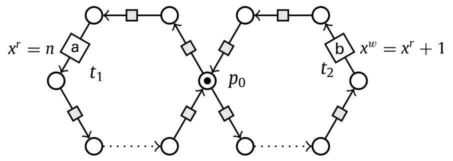
The initial and final markings are such that $M_I = M_F = [p_0]$, and $V = \{x\}$, with $\alpha_0(x) = 0$ being the initial assignment. Let $\mathbf{e} = \langle e, \dots, e\rangle$, with $e = (\mathsf{a}, \emptyset)$ and $|\mathbf{e}| = m$. Consider the following two alignments:[2]

$$\gamma_1 = \boxed{\begin{array}{c}\mathsf{a}\\ \gg\end{array}}\cdots\boxed{\begin{array}{c}\mathsf{a}\\ \gg\end{array}} \qquad \gamma_2 = \boxed{\begin{array}{c}\gg\\ \mathsf{b}\ \{x^w \mapsto 1\}\end{array}}\cdots\boxed{\begin{array}{c|c}\gg & \mathsf{a}\\ \mathsf{b}\ \{x^w \mapsto n\} & \mathsf{a}\end{array}}\cdots\boxed{\begin{array}{c}\mathsf{a}\\ \mathsf{a}\end{array}}$$

For γ_1, we have that $\kappa(\gamma_1) = |\mathbf{e}| = m$, that is, the cost amounts to the length of the log trace. In γ_2, every b-model move has cost 2 as only one variable is written, while the synchronous moves have cost 0. Thus, $\kappa(\gamma_2) = 2n$. From that we can deduce that, if $m = |\mathbf{e}| < 2n$, then γ_1 is optimal, while otherwise γ_2 will be the optimal alignment (as its cost will remain unchanged due to the guard of t_1). In fact, in the latter case, the length of the process run associated with the optimal alignment is $n + m$.

In the following example we demonstrate how silent transitions can further complicate the computation of the upper bound.

Example 5. Consider the following net $\mathcal{N}'$, a variant of the net from Example 4:



Here, both the left and right cycles consist of k-silent transitions (depicted in gray). For $\mathbf{e}$ as in Example 4, γ_1 is still a valid alignment. Now consider a variant of alignment γ_2, denoted γ_2', in which every model move and synchronous move in γ_2 is combined with k silent transitions. As silent transitions are not penalized by the standard cost function, $\kappa(\gamma_2') = 2n$. Thus, if γ_2' is optimal, the length of the respective process run $\gamma_2'|_M$ is $(n+m)\cdot(k+1)$. Note that even without data manipulating guards, the length of the process run would be $m \cdot (k + 1)$.

These examples show that the length of a process run $\mathbf{f}$ associated with an optimal alignment may not only depend on the length of the log trace and the length of paths on the net graph, but also the data manipulating guards. However, we next show that an upper bound for the length of $\mathbf{f}$ exists whenever the standard cost function is used. Notice that a similar reasoning applies when using the Levenshtein distance.

Lemma 2. *Let $\mathcal{N}$ be a DPN, $\langle f_1, \dots, f_n\rangle$ be a process run of $\mathcal{N}$ such that $c = \sum_{j=1}^{n} P_M(f_j)$ is minimal, and k the length of the longest acyclic path of silent transitions in $\mathcal{N}$. Then there is some $\gamma \in Align^{opt}(\mathcal{N}, \mathbf{e})$ for $\mathbf{e}$ w.r.t. the standard cost function such that $|\gamma|_M| \leq (2|\mathbf{e}| + c + 1) \cdot k$.*

Proof. Consider $\gamma_0 = \langle(e_1, \gg), \dots, (e_m, \gg), (\gg, f_1), \dots (\gg, f_n)\rangle$, which is a valid alignment for $\mathbf{e} = \langle e_1, \dots, e_m\rangle$ that satisfies $\kappa(\gamma_0) = m + c$. By optimality, $\kappa(\gamma) \leq \kappa(\gamma_0)$ must hold. Observe that γ can have at most $|\mathbf{e}|$ synchronous moves, with the best case cost of 0 (when all write operations match), so that $\gamma|_M$ has the same number of corresponding transitions. Since every model move with a non-silent transition has at least cost 1, $\gamma|_M$ can have at most $m + c$ non-silent transitions (otherwise, we would have $\kappa(\gamma) > \kappa(\gamma_0)$). Thus $\gamma|_M$ has at most $2m + c$ non-silent transitions used in synchronous and model moves of γ. However, there may exist a process run composed of all the transitions from $\langle f_1, \dots, f_n\rangle$ and silent transitions appearing between every one of these, as well as before and afterwards such that: (1) the order of firing of non-silent transition is the same as in the run from above; (2) the cost of the eventual alignment is not affected by the silent transitions. This is similar to the case illustrated in Example 5. Thus, since silent transitions do not write variables, the number of such silent transitions can be bound by the length of the longest acyclic path constructed of silent transitions in $\mathcal{N}$. Like that, we obtain that the length of $\gamma|_M$ is at most $(2m+c+1)\cdot k$.

For safe nets, shortest paths are known to be computable in polynomial time [24]. The quantity c in Lemma 2 corresponds to a weighted shortest path problem, and is hence computable accordingly. In fact the bound proven in Lemma 2 is conservative and in practice often unnecessarily high. E.g., optimal alignments need not contain silent transitions at all, but they *may* be required to get an optimal cost alignment (cf. Example 5). However, the crucial point here is that a computable upper bound exists. Moreover, the bound of Lemma 2 will exclude optimal alignments with loops composed of silent transitions only. This is not a problem since another optimal alignment without such loops exists whose length is within the bounds.

3.2. Encoding

We now describe the building blocks of our encoding. It is inspired by the SAT-based approach presented in [17,20], but significantly differs from it as it works also for nets with arc multiplicities and unbounded nets, beyond the safe case considered in [17].

Given a log trace $\mathbf{e} = \langle e_1, \dots, e_m\rangle$ and a DPN $\mathcal{N}$ with initial marking M_I, initial state variable assignment α_0, and final marking M_F, we want to construct an optimal alignment $\gamma \in Align^{opt}(\mathcal{N}, \mathbf{e})$. To that end, we assume throughout this section that the length of the process run associated with γ is bounded by some number n. In the interest of a more succinct presentation, we also assume that the given net $\mathcal{N}$ has a silent *wait* transition in the final marking, as also done in [20,25]. However, we comment on the end of this section that our approach can also be adapted to the case where this is not possible because the final marking is empty (cf. Remark 1).

Our approach comprises the following four steps: (1) represent the alignment symbolically by a set of SMT variables, (2) set up constraints Φ that symbolically express optimality of this alignment, (3) solve the constraints Φ to obtain a satisfying assignment ν, and (4) decode an optimal alignment γ from ν. We next elaborate these steps in detail. An overview over the SMT variables and other notational shorthands related to the encoding are provided in Table 1.

(1) Alignment representation. We use the following SMT variables:

(a) transition step variables $\mathtt{S}_i$ for $1 \leq i \leq n$ of type integer; if $T = \{t_1, \dots, t_{|T|}\}$ then it is ensured that $1 \leq \mathtt{S}_i \leq |T|$, with the semantics that $\mathtt{S}_i$ is assigned j iff the ith transition in the process run is t_j;

(b) marking variables $\mathtt{M}_{i,p}$ of type integer for all i, p with $0 \leq i \leq n$ and $p \in P$, where $\mathtt{M}_{i,p}$ is assigned k iff there are k tokens in place p at instant i;

(c) data variables $\mathtt{X}_{i,v}$ for all $v \in V$ and i, $0 \leq i \leq n$; the type of these variables depends on v, with the semantics that $\mathtt{X}_{i,v}$

[2] Both in γ_1 and γ_2 moves containing label a appear m times.

Table 1
Notation used to describe the SMT encoding. Here, m is a log trace length and n is the (maximal) length of the process run associated with a computed alignment γ.

symbol	description
$\mathtt{S}_i$	transition step variables encoding the process run ($1 \le i \le n$)
$\mathtt{M}_{i,p}$	marking variables ($1 \le i \le n$ and $p \in P$)
$\mathtt{X}_{i,v}$	data variables encoding process variable assignments ($0 \le i \le n$ and $v \in V$)
$\mathtt{d}_{i,j}$	distance variables encoding the alignment cost of the prefixes $\mathbf{e}\|_i$ and $\mathbf{f}\|_j$ ($0 \le i \le m$ and $0 \le j \le n$)
$tids(a)$	the set of transition indices corresponding to label a
$lab(\mathtt{S}, a)$	a formula encoding that the value assigned to S corresponds to a transition labeled with a
ν	satisfying variable assignment to all the variables in the SMT encoding
$\mathbf{f}_\nu$	process run decoding for a satisfying assignment ν

is assigned r iff the value of v at instant i is r; we also write $\mathtt{X}_i$ for $(\mathtt{X}_{i,v_1}, \dots, \mathtt{X}_{i,v_k})$;

(d) distance variables $\mathtt{d}_{i,j}$ of type integer for $0 \le i \le m$ and $0 \le j \le n$, where $\mathtt{d}_{i,j}$ is to represent the alignment cost of the prefix $\mathbf{e}|_i$ of the log trace $\mathbf{e}$, and prefix $\mathbf{f}|_j$ of the (yet to be determined) run $\mathbf{f}$.

Note that variables (a)–(c) comprise all information required to capture a process run with n steps, which will make up the model projection of the alignment γ, while the distance variables (d) will be used to encode the alignment.

(2) Encoding. To ensure that the values of variables correspond to a valid run, we assert the following constraints:

- The initial marking M_I and the initial assignment α_0 are respected:

$$\bigwedge_{p\in P} \mathtt{M}_{0,p} = M_I(p) \wedge \bigwedge_{v\in V} \mathtt{X}_{0,v} = \alpha_0(v) \qquad (\varphi_{init})$$

- The final marking M_F is respected:

$$\bigwedge_{p\in P} \mathtt{M}_{n,p} = M_F(p) \qquad (\varphi_{final})$$

- Transitions correspond to transition firings in the DPN:

$$\bigwedge_{1\le i\le n} 1 \le \mathtt{S}_i \le |T| \qquad (\varphi_{trans})$$

- Transitions are enabled when they fire:

$$\bigwedge_{1\le i\le n}\ \bigwedge_{1\le j\le |T|} (\mathtt{S}_i = j) \rightarrow \bigwedge_{p\in {}^\bullet t_j} \mathtt{M}_{i-1,p} \ge |{}^\bullet t_j|_p \qquad (\varphi_{enabled})$$

where $|{}^\bullet t_j|_p$ denotes the multiplicity of p in the multiset ${}^\bullet t_j$.

- We encode the token game:

$$\bigwedge_{1\le i\le n}\ \bigwedge_{1\le j\le |T|} (\mathtt{S}_i = j) \rightarrow \bigwedge_{p\in P} \mathtt{M}_{i,p} - \mathtt{M}_{i-1,p} = |t_j{}^\bullet|_p - |{}^\bullet t_j|_p \qquad (\varphi_{mark})$$

where $|t_j{}^\bullet|_p$ is the multiplicity of p in the multiset $t_j{}^\bullet$.

- The transitions satisfy the constraints on data:

$$\bigwedge_{1\le i< n}\ \bigwedge_{1\le j\le |T|} (\mathtt{S}_i = j) \rightarrow guard(t_j)\chi \wedge \bigwedge_{v\notin write(t_j)} \mathtt{X}_{i-1,v} = \mathtt{X}_{i,v} \qquad (\varphi_{data})$$

where the substitution χ uniformly replaces V^r by $\mathtt{X}_{i-1}$ and V^w by $\mathtt{X}_i$.

- The encoding of the edit distance depends on the penalty functions $P_=$, P_M, and P_L. We illustrate here the formulae obtained for the standard cost function in Section 2.2. First, for $T = \{t_1, \dots, t_{|T|}\}$ and $a \in A$, we write $tids(a) = \{i \mid \ell(t_i) = a\}$ for the set of transition indices corresponding to label a; and for a transition variable S, $lab(\mathtt{S}, a) := \bigvee_{k\in tids(a)} \mathtt{S} = k$ expresses that the value assigned to S corresponds to a transition labeled a. Given a log trace $\mathbf{e} = (b_1, \alpha_1), \dots, (b_m, \alpha_m)$, let the expressions $[P_L]$, $[P_M]_j$, and $[P_=]_{i,j}$ be defined as follows, for all i and j:

$$[P_L] = 1$$
$$[P_M]_j = ite(\mathtt{S}_j = 1, c_w(t_1), \dots ite(\mathtt{S}_j = |T|-1, c_w(t_{|T|-1}), c_w(t_{|T|}))\dots)$$
$$[P_=]_{i,j} = ite(lab(\mathtt{S}_j, b_i), \sum_{v\in write(b_i)} ite(\alpha_i(v) = \mathtt{X}_{i,v}, 0, 1), \infty)$$

where the *write cost* $c_w(t)$ of transition $t \in T$ is 0 if $\ell(t) = \tau$, or $|write(t)| + 1$ otherwise, and *ite* is the if-then-else operator. It is then straightforward to encode the data edit distance by combining all equations in Definition 9:

$$\mathtt{d}_{0,0} = 0 \qquad \mathtt{d}_{i+1,0} = [P_L] + \mathtt{d}_{i,0} \qquad \mathtt{d}_{0,j+1} = [P_M]_{j+1} + \mathtt{d}_{0,j}$$
$$\mathtt{d}_{i+1,j+1} = \min([P_=]_{i+1,j+1} + \mathtt{d}_{i,j},\ [P_L] + \mathtt{d}_{i,j+1},\ [P_M]_{j+1} + \mathtt{d}_{i+1,j}) \qquad (\varphi_\delta)$$

In this way, we will see that $\mathtt{d}_{m,n}$ gets assigned the cost of an optimal alignment for the given trace.

(3) Solving. We use an SMT solver to obtain a satisfying assignment ν for the following constrained optimization problem:

$$\varphi_{init} \wedge \varphi_{final} \wedge \varphi_{trans} \wedge \varphi_{enabled} \wedge \varphi_{mark} \wedge \varphi_{data} \wedge \varphi_\delta \quad \text{minimizing} \quad \mathtt{d}_{m,n} \qquad (\Phi)$$

Next, we illustrate the encoding on our running example.

Example 6. We illustrate the encoding on the DPN from Example 1, and the log trace $\mathbf{e} = \langle(\mathsf{a}, \{x \mapsto 2\}), (\mathsf{b}, \{y \mapsto 1\})\rangle$ from Example 2. For simplicity, we assume that the maximal length of the process run in the optimal alignment is $n = 3$. We add an additional silent transition from p_3 to itself (cf. the remarks at the beginning of Section 3.2), called f. For readability, the silent transition from p_2 to p_3 is called c, and the transition from p_1 to p_3 is called e; and we write $\mathsf{a}, \mathsf{b}, \dots, \mathsf{f}$ for the integers $1, 2, \dots, 6$ when referring to transition numbers. First, we get the following constraints for the process run:

$$\mathtt{M}_{0,0} = 1 \wedge \mathtt{M}_{0,1} = \mathtt{M}_{0,2} = \mathtt{M}_{0,3} = 0 \wedge \mathtt{X}_{0,x} = \mathtt{X}_{0,y} = 0 \qquad (\varphi_{init})$$
$$\mathtt{M}_{3,0} = \mathtt{M}_{3,1} = \mathtt{M}_{3,2} = 0 \wedge \mathtt{M}_{3,3} = 1 \qquad (\varphi_{final})$$
$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{a} \vee \mathtt{S}_i = \mathsf{b} \vee \mathtt{S}_i = \mathsf{c} \vee \mathtt{S}_i = \mathsf{d} \vee \mathtt{S}_i = \mathsf{e} \vee \mathtt{S}_i = \mathsf{f}) \qquad (\varphi_{trans})$$
$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{a} \rightarrow \mathtt{M}_{i-1,0} \ge 1) \wedge (\mathtt{S}_i = \mathsf{b} \rightarrow \mathtt{M}_{i-1,1} \ge 1)$$
$$\wedge (\mathtt{S}_i = \mathsf{c} \rightarrow \mathtt{M}_{i-1,2} \ge 1) \wedge$$
$$(\mathtt{S}_i = \mathsf{d} \rightarrow \mathtt{M}_{i-1,3} \ge 1) \wedge (\mathtt{S}_i = \mathsf{e} \rightarrow \mathtt{M}_{i-1,1} \ge 1)$$
$$\wedge (\mathtt{S}_i = \mathsf{f} \rightarrow \mathtt{M}_{i-1,3} \ge 1) \qquad (\varphi_{enabled})$$

$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{a} \rightarrow (\mathtt{M}_{i-1,0} - \mathtt{M}_{i,0} = 1 \wedge \mathtt{M}_{i,1} - \mathtt{M}_{i-1,1} = 1$$

$$\begin{aligned} \mathtt{d}_{1,1} &= min(\mathtt{d}_{0,0} + [P_=]_{1,1}, \mathtt{d}_{0,1}+1, \mathtt{d}_{1,0} + [P_M]_1) & \mathtt{d}_{2,1} &= min(\mathtt{d}_{1,0} + [P_=]_{2,1}, \mathtt{d}_{1,1}+1, \mathtt{d}_{2,0} + [P_M]_1)\\ \mathtt{d}_{1,2} &= min(\mathtt{d}_{0,1} + [P_=]_{1,2}, \mathtt{d}_{0,2}+1, \mathtt{d}_{1,1} + [P_M]_2) & \mathtt{d}_{2,2} &= min(\mathtt{d}_{1,1} + [P_=]_{2,2}, \mathtt{d}_{1,2}+1, \mathtt{d}_{2,1} + [P_M]_2)\\ \mathtt{d}_{1,3} &= min(\mathtt{d}_{0,2} + [P_=]_{1,3}, \mathtt{d}_{0,3}+1, \mathtt{d}_{1,2} + [P_M]_3) & \mathtt{d}_{2,3} &= min(\mathtt{d}_{1,2} + [P_=]_{2,3}, \mathtt{d}_{1,3}+1, \mathtt{d}_{2,2} + [P_M]_3) \end{aligned}$$

Box I.

$$\wedge\, \mathtt{M}_{i,2} = \mathtt{M}_{i-1,2} \wedge \mathtt{M}_{i,3} = \mathtt{M}_{i-1,3}))$$

$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{b} \to (\mathtt{M}_{i-1,1} - \mathtt{M}_{i,1} = 1 \wedge \mathtt{M}_{i,2} - \mathtt{M}_{i-1,2} = 1$$
$$\wedge\, \mathtt{M}_{i,0} = \mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,3} = \mathtt{M}_{i-1,3}))$$

$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{c} \to (\mathtt{M}_{i-1,2} - \mathtt{M}_{i,2} = 1 \wedge \mathtt{M}_{i,3} - \mathtt{M}_{i-1,3} = 1$$
$$\wedge\, \mathtt{M}_{i,0} = \mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,1} = \mathtt{M}_{i-1,1}))$$

$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{d} \to (\mathtt{M}_{i,0} = \mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,1} = \mathtt{M}_{i-1,1}$$
$$\wedge\, \mathtt{M}_{i,2} = \mathtt{M}_{i-1,2} \wedge \mathtt{M}_{i,3} = \mathtt{M}_{i-1,3}))$$

$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{e} \to (\mathtt{M}_{i-1,1} - \mathtt{M}_{i,1} = 1 \wedge \mathtt{M}_{i,3} - \mathtt{M}_{i-1,3} = 1$$
$$\wedge\, \mathtt{M}_{i,0} = \mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,2} = \mathtt{M}_{i-1,2}))$$

$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{f} \to (\mathtt{M}_{i,0} = \mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,1} = \mathtt{M}_{i-1,1}$$
$$\wedge \mathtt{M}_{i,2} = \mathtt{M}_{i-1,2} \wedge \mathtt{M}_{1,3} = \mathtt{M}_{0,3})) \qquad (\varphi_{mark})$$

$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{a} \to (\mathtt{X}_{i,x} \ge 0 \wedge \mathtt{X}_{i,y} = \mathtt{X}_{i-1,y}))$$
$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{b} \to (\mathtt{X}_{i,y} > 0 \wedge \mathtt{X}_{i,x} = \mathtt{X}_{i-1,x}))$$
$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{c} \to (\mathtt{X}_{i-1,x} \le 3 \wedge \mathtt{X}_{i-1,y} < 4$$
$$\wedge\, \mathtt{X}_{i,x} = \mathtt{X}_{i-1,x} \wedge \mathtt{X}_{i,y} = \mathtt{X}_{i-1,y}))$$
$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{d} \to (\mathtt{X}_{i,y} = \mathtt{X}_{i-1,y} + 1 \wedge \mathtt{X}_{i,x} = \mathtt{X}_{i-1,x}))$$
$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{e} \to (\mathtt{X}_{i-1,x} \le 3 \wedge \mathtt{X}_{i,y} = \mathtt{X}_{i-1,y} \wedge \mathtt{X}_{i,x} = \mathtt{X}_{i-1,x}))$$
$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i = \mathsf{f} \to (\mathtt{X}_{i,x} = \mathtt{X}_{i-1,x} \wedge \mathtt{X}_{i,y} = \mathtt{X}_{i-1,y})) \qquad (\varphi_{data})$$

It remains to add constraints for the alignment cost. Using the standard cost function, we set $[P_L]_i = 1$ for all i; $[P_M]_j = ite(\mathtt{S}_j = \mathsf{c} \vee \mathtt{S}_j = \mathsf{e}, 0, 1)$ for all j to give model moves cost 0 if they use a silent transition and cost 1 otherwise; and

$$[P_=]_{1,j} = ite(\mathtt{S}_j = \mathsf{a}, ite(\mathtt{X}_{j,x} = 2, 0, 1), \infty)$$
$$[P_=]_{2,j} = ite(\mathtt{S}_j = \mathsf{b}, ite(\mathtt{X}_{j,y} = 1, 0, 1), \infty).$$

Then, we get from (φ_δ) the equations $\mathtt{d}_{i,0} = i$ for all $0 \le i \le 2$, $\mathtt{d}_{0,j} = 1 + \mathtt{d}_{0,j-1}$ for all $0 < j \le 3$, and the equations in Box I.

One possible solution to the constrained optimization problem is the assignment ν such that $\nu(\mathtt{M}_{0,0}) = \nu(\mathtt{M}_{1,1}) = \nu(\mathtt{M}_{2,2}) = \nu(\mathtt{M}_{3,3}) = 1$ and all other $\mathtt{M}_{i,j}$ are assigned 0; $\nu(\mathtt{S}_1) = \mathsf{a}$, $\nu(\mathtt{S}_2) = \mathsf{b}$, and $\nu(\mathtt{S}_3) = \mathsf{c}$; $\nu(\mathtt{X}_{0,x}) = \nu(\mathtt{X}_{0,y}) = \nu(\mathtt{X}_{1,y}) = 0$, $\nu(\mathtt{X}_{1,x}) = \nu(\mathtt{X}_{2,x}) = \nu(\mathtt{X}_{3,x}) = 2$, and $\nu(\mathtt{X}_{2,y}) = \nu(\mathtt{X}_{3,y}) = 1$. The relevant distance variable assignments are $\nu(\mathtt{d}_{0,0}) = \nu(\mathtt{d}_{1,1}) = \nu(\mathtt{d}_{2,2}) = 0$, so the cost of the alignment is 0. This corresponds to the optimal alignment γ_1 shown in Example 2.

(4) Decoding. Assuming that the set of transitions $T = \{t_1, \dots, t_{|T|}\}$ is ordered following the order already used for the encoding, for a satisfying assignment ν for (Φ), we can obtain a *process run decoding* $\mathbf{f}_\nu = \langle f_1, \dots, f_n\rangle$, where $f_i = (t_{\nu(\mathtt{S}_i)}, \beta_i)$ and the transition variable assignment β_i is procured as follows: Let the state variable assignments α_j, $0 \le j \le n$, be given by $\alpha_j(v) = \nu(\mathtt{X}_{j,v})$, for all $v \in V$. Then, $\beta_i(v^r) = \alpha_{i-1}(v)$ and $\beta_i(v^w) = \alpha_i(v)$, for all $v \in V$. Next, we use properties of the edit distance [21] to decode an alignment $\gamma = \gamma_{m,n}$ of $\mathbf{e}$ and $\mathbf{f}_\nu = \langle f_1, \dots, f_n\rangle$. To that end, consider the (partial) alignments $\gamma_{i,j}$ recursively defined as follows:[3]

$$\gamma_{0,0} = \epsilon \qquad \gamma_{i+1,0} = \gamma_{i,0} \cdot (e_{i+1}, \gg) \qquad \gamma_{0,j+1} = \gamma_{0,j} \cdot (\gg, f_{j+1})$$

$$\gamma_{i+1,j+1} = \begin{cases} \gamma_{i,j+1} \cdot (e_{i+1}, \gg) & \text{if } \nu(\mathtt{d}_{i+1,j+1}) = \nu([P_L] + \mathtt{d}_{i,j+1}) \\ \gamma_{i+1,j} \cdot (\gg, f_{j+1}) & \text{if otherwise } \nu(\mathtt{d}_{i+1,j+1}) \\ & \quad = \nu([P_M]_{j+1} + \mathtt{d}_{i+1,j}) \\ \gamma_{i,j} \cdot (e_{i+1}, f_{j+1}) & \text{otherwise} \end{cases}$$

We will show in Section 3.3 that the alignment constructed in this way is indeed optimal.

As remarked at the beginning of this section, our approach so far assumed that the final marking of the DPN has a silent *wait* transition. While this is not possible if the final marking is empty, we next explain that our encoding can be adapted to accommodate this case:

Remark 1. To avoid fixing the length of the process run in the alignment to *exactly* n, and allow instead a process run that has *at most* length n without adding artificial *wait* transitions, the encoding can be adapted as follows: we use an additional SMT variable `len` of type integer to represent the length of the process run, add a constraint $0 \le \mathtt{len} \le n$, and modify φ_{final} to state that the run is final at instant `len`:

$$\bigwedge_{j=0}^{n} \mathtt{len} = j \to \bigwedge_{p\in P} \mathtt{M}_{j,p} = M_F(p)$$

and the minimization objective is modified to perform a case distinction on `len`:

$$ite(\mathtt{len} = 0, \mathtt{d}_{m,0}, \dots ite(\mathtt{len} = n-1, \mathtt{d}_{m,n-1}, \mathtt{d}_{m,n}) \dots)$$

While this modified encoding is always sound, it is in general preferable to add silent transitions on final states if possible, as nested *ite* expressions may negatively impact solver performance.

3.3. Correctness

In this section we prove key formal properties of the decoding defined above. First, we show that the process run decoding is indeed a valid run.

Lemma 3. *Let $\mathcal{N}$ be a DPN, $\mathbf{e}$ a log trace and ν a solution to (Φ). Then the process run decoding satisfies $\mathbf{f}_\nu \in Runs(\mathcal{N})$.*

[3] Here, given $\mathbf{x} = \langle x_1, \dots, x_n\rangle$, its concatenation with an element x' is defined as $\mathbf{x} \cdot x' = \langle x_1, \dots, x_n, x'\rangle$.

Proof. Let M_i be the marking such that $M_i(p) = \nu(\mathtt{M}_{i,p})$, for all $p \in P$, and α_i the state variable assignment such that $\alpha_i(v) = \nu(\mathtt{X}_{i,v})$, for all $v \in V$ and i, $0 \le i \le n$. First, we show by induction on n that for the process run $\mathbf{f}_\nu = \langle f_1, \dots, f_n\rangle$ it holds that $(M_I, \alpha_0)\xrightarrow{\mathbf{f}_\nu}(M_n, \alpha_n)$.

Base case. If $n = 0$, then $\mathbf{f}_\nu$ is empty. As ν satisfies φ_{init}, it must be that $M_0 = M_I$ and α_0 is the initial assignment, so the claim trivially holds.

Inductive step. Assume that $\mathbf{f}_\nu = \langle f_1, \dots, f_{n+1}\rangle$ and for its prefix $\mathbf{f}' = \langle f_1, \dots, f_n\rangle$ it holds that $(M_I, \alpha_0)\xrightarrow{\mathbf{f}'}(M_n, \alpha_n)$. If $f_{n+1} = (t_j, \beta)$ for some j s.t. $1 \le j \le |T|$, then, by construction of $\mathbf{f}_\nu$, we have that $\nu(\mathtt{S}_n) = j$. Since ν is a solution to (Φ), it satisfies $\varphi_{enabled}$ so that t_j is enabled in M_n. Moreover, as ν satisfies φ_{mark} and φ_{data}, it holds that $(M_n, \alpha_n)\xrightarrow{f_{n+1}}(M_{n+1}, \alpha_{n+1})$. This concludes the induction proof.

Finally, given that ν also satisfies φ_{final}, the last marking must be final and hence $\mathbf{f}_\nu \in Runs(\mathcal{N})$.

Next, we demonstrate that the above decoding yields an optimal alignment.

Theorem 1. *Let $\mathcal{N}$ be a DPN, $\mathbf{e}$ a log trace and ν a solution to (Φ). Then $\gamma_{m,n} \in Align^{opt}(\mathcal{N}, \mathbf{e})$.*

Proof. By Lemma 3, $\mathbf{f}_\nu \in Runs(\mathcal{N})$. We first note that $[P_=]$, $[P_L]$, and $[P_M]$ are correct encodings of $P_=$, P_L, and P_M from Section 2.2, respectively. For P_L this is trivial. For P_M, the case distinction encodes the number of written variables depending on the value of $\mathtt{S}_j$. For $P_=$, $lab(\mathtt{S}_j, b_i)$ is true iff the value of $\mathtt{S}_j$ corresponds to a transition that is labeled b_i. If the labels match, the expression $\sum_{v\in write(b_i)} ite(\alpha_i(v) = \mathtt{X}_{i,v}, 0, 1)$ encodes the number of mismatching data values, as stipulated by the standard cost function.

Let $d_{i,j} = \nu(\mathtt{d}_{i,j})$, for all i, j such that $0 \le i \le m$ and $0 \le j \le n$. We show the stronger statement that $\gamma_{i,j}$ is an optimal alignment for $\mathbf{e}|_i$ and $\mathbf{f}_\nu|_j$ with cost $d_{i,j}$, by induction on (i, j).

Base case. If $i = j = 0$, then $\gamma_{i,j}$ is the trivial, empty alignment of an empty log trace and an empty process run, which is clearly optimal with cost $d_{i,j} = 0$, as defined in (φ_δ).

Step case. If $i = 0$ and $j > 0$, then the optimal alignment is a sequence of model moves with cost $[P_M]_1 + \cdots + [P_M]_j$, as stipulated in ($\varphi_\delta$), and the optimal alignment is defined as $\gamma_{0,j} = \langle(\gg, f_1), \dots, (\gg, f_j)\rangle$.

Step case. If $j = 0$ and $i > 0$, then the optimal alignment is a sequence of log moves with cost $[P_L] + \cdots + [P_L]$, again as stated in (φ_δ), and the optimal alignment is defined as $\gamma_{i,0} = \langle(e_1, \gg), \dots, (e_i, \gg)\rangle$.

Step case. If $i, j > 0$, then, since ν satisfies (φ_δ), $d_{i,j}$ is the minimum of $\nu([P_=]_{i,j}) + d_{i-1,j-1}$, $\nu([P_L]) + d_{i-1,j}$, and $\nu([P_M]_j) + d_{i,j-1}$. By the induction hypothesis, $d_{i-1,j-1}$, $d_{i-1,j}$, and $d_{i-1,j}$ are the costs of the optimal alignments for $\mathbf{e}|_{i-1}$ and $\mathbf{f}_\nu|_{j-1}$; $\mathbf{e}|_{i-1}$ and $\mathbf{f}_\nu|_j$; and $\mathbf{e}|_i$ and $\mathbf{f}_\nu|_{j-1}$, respectively. As $[P_=]$, $[P_L]$, and $[P_M]$ are correct encodings of $P_=$, P_L, and P_M, respectively, by definition of a distance-based cost function (Definition 9), $d_{i,j}$ is thus the optimal alignment cost of $\mathbf{e}|_i$ and $\mathbf{f}_\nu|_j$, and it is clear from the construction of $\gamma_{i,j}$ is an alignment that has exactly this cost.

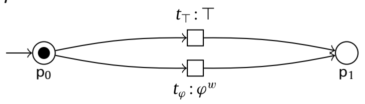
3.4. Complexity

In this section we show that the decision problem associated with the optimal alignment problem is NP-complete for relaxed data-sound DPNs with linear arithmetic, so that the complexity of the approach presented in Section 3.2 is optimal. Let a cost function κ be *admissible* if it is distance-based, its parameter functions $P_=$, P_M, and P_L are effectively computable, and they can be defined by linear arithmetic expressions and case distinctions. For $c \in \mathbb{N}$ and an admissible cost function κ, let $\textsc{Align}_c$ be the problem to decide whether a relaxed data-sound DPN and a log trace admit an alignment of cost c.

Lemma 4. *For an admissible cost function, the problem $\textsc{Align}_c$ is NP-complete.*

Proof. Given a DPN $\mathcal{N}$, a log trace $\mathbf{e}$, and some $c \in \mathbb{N}$, the encoding from Section 3.2 yields an SMT problem over linear arithmetic that is satisfiable if and only if an alignment of cost c exists, by adding the constraint $\mathtt{d}_{m,n} = c$ instead of minimizing $\mathtt{d}_{m,n}$ (cf. Theorem 1). The size of the encoding is polynomial in the size of $\mathcal{N}$ and the length of $\mathbf{e}$. Thus, since satisfiability of the relevant class of SMT problems is in NP [26], our approach to decide $\textsc{Align}_c$ is in NP.

Next, $\textsc{Align}_c$ is NP-hard since it is easy to reduce satisfiability of a boolean formula (SAT) to $\textsc{Align}_0$: For a formula φ with variables V, let $\mathcal{N}_\varphi$ be the DPN where V is the set of variables in φ and the control structure is as follows:



Here φ^w is the formula obtained from φ by replacing all variables $v \in V$ by v^w. The initial marking is $\{\mathsf{p}_0\}$, and the final marking is $\{\mathsf{p}_1\}$. The DPN $\mathcal{N}_\varphi$ is relaxed data-sound due to the transition $t_\top$. Let $\mathbf{e}$ be the log trace consisting of the single event $(t_\varphi, \emptyset)$, and κ the standard (distance-based) cost function κ_{dist} (as defined in Section 2.2). We have $\mathbf{f}_0 = (t_\top, \varnothing) \in Runs(\mathcal{N}_\varphi)$ and $\kappa(\mathbf{e}, \mathbf{f}_0) = \infty$. If φ is satisfiable by some assignment α, we also have $\mathbf{f}_1 = \langle(t_\varphi, \beta_w)\rangle \in Runs(\mathcal{N}_\varphi)$, where β_w is the assignment such that $\beta_w(v^w) = \alpha(v)$ for all $v \in V$, and $\kappa(\mathbf{e}, \mathbf{f}_1) = 0$. Thus, $\mathbf{e}$ admits an alignment of cost 0 if and only if φ is satisfiable.

The proof shows that our approach to decide $\textsc{Align}_c$ is in NP, whereas the method presented in [8,9] is exponential in the length of the log trace.

3.5. Possible extensions

In this section, we discuss how our SMT-based approach can be extended towards the support of numerous *background theories* not mentioned in this paper so as to constrain the data objects manipulated by the DPN as well as to express sophisticated cost functions. Indeed, in the related literature [8,9], the guards of DPNs are 'linear boolean' expressions, i.e., they combine arithmetical operations with boolean operators; in our approach, we can enrich the language of guards by employing *every* function and relation symbol used in *every* first-order theory supported by SMT solvers (e.g., arrays, lists, and sets). For example, the use of relational predicates in DPN guards opens the possibility of natively extending our framework so as to model structured background information: this additional structure allows us, e.g., to incorporate in the formal model full-fledged relational databases from which data injected in the net are taken, following the SMT-based approaches as in [27,28]: theoretically, this would require to exploit, as background first-order theory, the theory of uninterpreted symbols $\mathcal{EUF}$ built on top of an arbitrary first-order signature Σ. If, additionally, one wants to admit the user to inject external (possibly fresh) value data into the relational database, such as integers or reals, then one can rely on the SMT theories for constraining arithmetic datatypes like $\mathcal{LIA}$ and/or $\mathcal{LQA}$. The integration of such datatypes with the predicates from the relational database is guaranteed by the native flexibility of SMT solvers to reason modulo the *combination* of different theories: specifically, in this context, the underlying combined theory of interest would be $\mathcal{EUF} \cup \mathcal{LIA}$, $\mathcal{EUF} \cup \mathcal{LQA}$ or $\mathcal{EUF} \cup \mathcal{LIA} \cup \mathcal{LQA}$.

Considering more complicated background theories that can model different kinds of datatypes opens several directions towards possible extensions of the theoretical setting where to perform conformance checking. Indeed, more sophisticated frameworks for modeling data-aware processes that extend DPNs in several ways have been studied recently: a notable one is given by COA-nets [6,29], where, as it happens for DPNs, the process component is formalized using a Petri Net-based model, but where places contain tokens carrying unboundedly many 'data' tuples retrieved from a persistent storage, called catalog. SMT-based techniques have been proved to be successful to verify safety of COA-nets, which strongly suggests that the approach via SMT presented in this paper can be effectively lifted to perform conformance-checking tasks where the underlying process is modeled using COA-nets instead of DPNs.

Finally, the background theory can not only constrain the data object manipulated by the net, but also increase the expressiveness of the kind of cost functions that could be defined. In fact, since sophisticated predicates are available in our language, one can employ them, e.g., in the parallel move penalty function, so as to model different examples of edit distance as the following one (inspired by setting such as the one is [28,29]):

Example 7. Suppose that the background information of the process is stored in a typed relational DB. In this DB, relations are typed in the sense that each attribute has a specific type. Types for relation attributes are usually of two different kinds: 'id' types and 'value' types. The first ones account for identifiers of objects contained in the DB relations, whereas the second ones are used for injecting specific datatype values into the DBs, such strings or integers. Let us assume that our DPNs interact with such a DB, in the sense that variables can take values from the attributes of its relations. Accordingly, variables are typed as well: for this reason we assume that $V := V_{id} \uplus V_{val}$, i.e., every variable is either of the 'id-type' or of the 'value-type'.

In such a context, we define the synchronous move penalty as a combination of two different contributions, depending on whether the variables has id or value type. Indeed, a natural definition of $P_=$ would be:

$$P_=((b,\alpha),(t,\beta)) = |\{v \in \textsc{dom}(\alpha) \text{ and } v \in V_{id} \mid \neg R(\alpha(v)), R(\beta(v^w))\}| + \Sigma_i c_i,$$

if $\ell(t) = b$, where R is some relation from the database DB and c_i are integer numbers such that $\alpha(v_i) := \beta(v_i^w) + c_i$, for every $v_i \in V_{val}$. This means that $P_=$ is the sum of two contributions: the first one applies to 'id-type' variables and counts the number of write variables such that their values in the model run is stored in the relation R whereas their values in the log trace are not; the second one sums, for every 'value-type' variable, the discrete distance between the values taken in the model run and in the log trace. Notice that reasoning with such kinds of penalties requires that the underlying SMT solver exploits combinations of theories: specifically, for this example, we are leveraging the combined theory $\mathcal{EUF} \cup \mathcal{LIA}$, where $\mathcal{EUF}$ is employed for the DB relation symbols, while $\mathcal{LIA}$ allows us to use sums and integers.

4. Trace clustering

Clustering techniques are used to group together multiple traces in a process log so as to simplify and optimize several forms of analysis [30], including conformance checking [18,31]. In this section we introduce a novel form of clustering that is instrumental to simplify our multi-perspective conformance checking technique. The idea is to partition the log into *clusters*, where all traces within the same cluster are guaranteed to share the same alignment cost. That is, the alignment cost has to be computed only once for one (non-deterministically selected) representative trace of the cluster, which can provide substantial computational benefits for clusters with considerably many traces. We present our clustering idea in two steps. We first introduce a general equivalence relation on log traces, which thus identifies clusters as equivalence classes. We then provide an instantiation of such a relation that compares traces in the log by considering the satisfaction of guards of the DPN, thus providing a sort of 'data abstraction-based' clustering.

Definition 12 (*Cost-based clustering*)**.** Given a DPN $\mathcal{N}$, a log L, and a cost function κ, a *cost-based clustering* is an equivalence relation $\equiv_{\kappa_{\mathcal{N}}^{opt}}$ over L, where, for all traces $\mathbf{e}, \mathbf{e}' \in L$ s.t. $\mathbf{e} \equiv_{\kappa_{\mathcal{N}}^{opt}} \mathbf{e}'$ we have that $\kappa_{\mathcal{N}}^{opt}(\mathbf{e}) = \kappa_{\mathcal{N}}^{opt}(\mathbf{e}')$.

Notice that, being linked by an equivalence relation from above, two traces are simply meant to belong to the same cluster. With the above definition we add a condition that if two traces are in the same cluster, they must have the same optimal alignment cost. This, however, does not mean that different clusters necessarily correspond to different optimal alignment costs as the clustering need not group *all* traces with the same optimal alignment cost into the same equivalence class.

We now introduce one specific equivalence relation that focuses on DPN guards performing *variable-to-constant* comparisons, and then show that this equivalence relation is a *cost-based clustering*. By focusing on guards, one can in fact improve performance of alignment-based analytic tasks. Indeed, variable-to-constant guards, although simple, are extensively used in practice, and they have been subject to an extensive body of research [10]. Moreover, this class of guards is common in benchmarks from the literature, is the one required to model decisions based on the DMN S-FEEL standard, and is the target of guard discovery techniques based on decision trees [13]. Note, however, that we do not at all require that the considered DPNs use *only* such guards — richer guards are simply not exploited in the clustering.

Recalling that constraints are used in DPNs as guards associated to transitions, and that a constraint is in general a boolean expression whose atoms are comparisons (cf. Section 2.1), we write $Atoms(c)$ to refer to the set of all atoms in a guard $c \in G_{\mathcal{N}}$. Given a DPN $\mathcal{N}$, a *variable-to-constant* atom is an expression of the form $x \odot k$, where $x \in V^r \cup V^w$, $\odot \in \{>, \geq, =\}$, and k is a constant in $\mathbb{Z}$ or $\mathbb{Q}$. We say that a variable $v \in V$ is *restricted to constant comparison* if all atoms in the guards of $\mathcal{N}$ that involve v^r or v^w are variable-to-constant atoms. For such variables, we also introduce the set $ats_v = \{v \odot k \mid x \odot k \in Atoms(c), \text{ for some } c \in G_{\mathcal{N}}, x \in \{v^r, v^w\}\}$, i.e., the set of comparison atoms $v \odot k$ as above, this time expressed with non-annotated variables. The set ats_v can be seen as a set of predicates with free variable v.

Intuitively, given a cost function as in Section 2.2, the optimal alignment of a log trace does not depend on the actual variable values specified in the events in the log trace, but only on whether the atoms in ats_v are satisfied. In this sense, our approach can be considered as a special form of *predicate abstraction*. Based on this idea, trace equivalence is defined as follows.

Definition 13. For a variable v that is restricted to constant comparison and two values u_1, u_2, let $u_1 \sim_{cc}^{v} u_2$ if for all $v \odot k \in ats_v$, $u_1 \odot k$ holds iff $u_2 \odot k$ holds. Two event variable assignments α

and α' are *equivalent up to constant comparison*, denoted $\alpha \sim_{cc} \alpha'$, if $\text{DOM}(\alpha) = \text{DOM}(\alpha')$ and for all variables $v \in \text{DOM}(\alpha)$, either (i) $\alpha(v) = \alpha'(v)$, or (ii) v is restricted to constant comparison and $\alpha(v) \sim^{v}_{cc} \alpha'(v)$.

This definition intuitively guarantees that α and α' "agree on satisfying" the same atomic constraints in the process. For example, if $\alpha(x) = 4$ and $\alpha'(x) = 5$, then, given two constraints $x > 3$ and $x < 2$, we will get that $\alpha \models x > 3$ and $\alpha' \models x > 3$, whereas $\alpha \not\models x < 2$ as well as $\alpha' \not\models x < 2$.

Definition 14 (*Equivalence up to constant comparison*)**.** Two events $e = (b, \alpha)$ and $e' = (b', \alpha')$ are *equivalent up to constant comparison*, denoted $e \sim_{cc} e'$, if $b = b'$ and $\alpha \sim_{cc} \alpha'$. Two log traces $\mathbf{e}$, $\mathbf{e}'$ are *equivalent up to constant comparison*, denoted $\mathbf{e} \sim_{cc} \mathbf{e}'$, iff their events are pairwise equivalent up to constant comparison. That is, $\mathbf{e} = \langle e_1, \dots, e_n\rangle$, $\mathbf{e}' = \langle e'_1, \dots, e'_n\rangle$, and $e_i \sim_{cc} e'_i$ for all i, $1 \le i \le n$.

Example 8. In Example 1, variable x is restricted to constant comparison, while y is not. Since $ats_x = \{x \geq 0, x \leq 3\}$, the log traces $\mathbf{e}_1 = \langle(\mathsf{a}, \{x \mapsto 2\}), (\mathsf{b}, \{y \mapsto 1\})\rangle$ and $\mathbf{e}_2 = \langle(\mathsf{a}, \{x \mapsto 3\}), (\mathsf{b}, \{y \mapsto 1\})\rangle$, satisfy $\mathbf{e}_1 \sim_{cc} \mathbf{e}_2$, but for $\mathbf{e}_3 = \langle(\mathsf{a}, \{x \mapsto 4\}), (\mathsf{b}, \{y \mapsto 1\})\rangle$ we have $\mathbf{e}_1 \not\sim_{cc} \mathbf{e}_3$ because $3 \not\sim^{x}_{cc} 4$, and $\mathbf{e}_4 = \langle(\mathsf{a}, \{x \mapsto 3\}), (\mathsf{b}, \{y \mapsto 2\})\rangle$ satisfies $\mathbf{e}_1 \not\sim_{cc} \mathbf{e}_4$ since the values for y differ. The equivalent traces $\mathbf{e}_1$ and $\mathbf{e}_2$ have the same optimal cost with respect to the standard cost function from Section 2.2: for the alignments

$$\gamma_1:\ \begin{array}{|l|l|c|}\hline \mathsf{a}\ x \mapsto 2 & \mathsf{b}\ y \mapsto 1 & \gg \\ \hline \mathsf{a}\ \{x^w \mapsto 2\} & \mathsf{b}\ \{y^w \mapsto 1\} & \tau \\ \hline \end{array} \qquad \gamma_2:\ \begin{array}{|l|l|c|}\hline \mathsf{a}\ x \mapsto 3 & \mathsf{b}\ y \mapsto 1 & \gg \\ \hline \mathsf{a}\ \{x^w \mapsto 3\} & \mathsf{b}\ \{y^w \mapsto 1\} & \tau \\ \hline \end{array}$$

$$\gamma_3:\ \begin{array}{|l|l|c|}\hline \mathsf{a}\ x \mapsto 4 & \mathsf{b}\ y \mapsto 1 & \gg \\ \hline \mathsf{a}\ \{x^w \mapsto 3\} & \mathsf{b}\ \{y^w \mapsto 1\} & \tau \\ \hline \end{array}$$

we have $\kappa^{opt}_{\mathcal{N}}(\mathbf{e}_1) = \kappa(\gamma_1) = 0$ and $\kappa^{opt}_{\mathcal{N}}(\mathbf{e}_2) = \kappa(\gamma_2) = 0$. Note, however, that the respective process runs $\gamma_1|_M$ and $\gamma_2|_M$ differ. On the other hand, γ_3 is an optimal alignment for $\mathbf{e}_3$ but $\kappa(\gamma_3) = \kappa^{opt}_{\mathcal{N}}(\mathbf{e}_3) = 1$.

Moreover, $\mathbf{e}_1$ and $\mathbf{e}_3$ show that for trace equivalence it does not suffice to consider model transitions with activity labels that occur in the traces: all events in $\mathbf{e}_1$ and $\mathbf{e}_3$ correctly correspond to transitions with the same labels in $\mathcal{N}$, but for a *later* transition the value of x makes a difference. This motivates the requirement that in equivalent traces (Definition 14) the values of a variable v that is restricted to constant comparison satisfies the same subset of ats_v.

We next show that equivalence up to constant comparison is a cost-based clustering, provided that the cost function satisfies certain mild requirements. To that end, we consider a distance-based cost function κ from Definition 9 and call it *comparison-based* if the following conditions hold: 1. $P_L(b, \alpha)$ does not depend on the values assigned by α, and $P_M(t, \beta)$ does not depend on the values assigned by β; 2. the value of $P_=((b, \alpha), (t, \beta))$ depends only on whether conditions $b = \ell(t)$ and $\alpha(v) = \beta(v^w)$ are satisfied or not.

Note that this requirement is satisfied by the cost functions in Section 2.2. Indeed, in the standard cost function, $P_L(b, \alpha) = 1$ and thus it does not depend on α. Moreover, the second condition is clearly satisfied, as in $P_=((b, \alpha), (t, \beta)) = |\{v \in \text{DOM}(\alpha) \mid \alpha(v) \neq \beta(v^w)\}|$, for $b = \ell(t)$, we only need to check whether $\alpha(v) \neq \beta(v^w)$.

Theorem 2. *Equivalence up to constant comparison is a cost-based clustering with respect to any comparison-based cost function.*

Proof. We need to show that for any two traces $\mathbf{e}_1$ and $\mathbf{e}_2$ such that $\mathbf{e}_1 \sim_{cc} \mathbf{e}_2$ and a comparison-based cost function κ, it holds that $\mathbf{e}_1 \equiv_{\kappa^{opt}_{\mathcal{N}}} \mathbf{e}_1$. For a partial process run σ, let $\alpha_{sv}(\sigma)$ be the state variable assignment after the last transition firing of the partial process run σ. Note that since $\mathbf{e}_1 \sim_{cc} \mathbf{e}_2$, the lengths of the two traces as well as their sequences of executed activities coincide. To prove the claim, we verify that if $\mathbf{e}_1$ has an alignment γ_1 with cost $\kappa(\gamma_1) = \delta(\mathbf{e}_1, \mathbf{f}_1)$ for some process run $\mathbf{f}_1 = \gamma_1|_M$, then there is a process run $\mathbf{f}_2$ such that $\delta(\mathbf{e}_2, \mathbf{f}_2) = \kappa(\gamma_1)$, and hence there is an alignment γ_2 with $\gamma_2|_L = \mathbf{e}_2$, $\gamma_2|_M = \mathbf{f}_2$ and $\kappa(\gamma_2) = \delta(\mathbf{e}_2, \mathbf{f}_2)$. More precisely, let $|\mathbf{e}_1| = |\mathbf{e}_2| = m$, $\mathbf{f}_1 = \gamma_1|_M$ and $|\mathbf{f}_1| = n$. Then, we show by induction on $m+n$ that there exists a process run $\mathbf{f}_2$ such that $|\mathbf{f}_2| = n$, $\delta(\mathbf{e}_1, \mathbf{f}_1) = \delta(\mathbf{e}_2, \mathbf{f}_2)$, and $\alpha_{sv}(\mathbf{f}_1) \sim_{cc} \alpha_{sv}(\mathbf{f}_2)$.

Base case ($m = n = 0$)**.** In this case all of $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{f}_1$ are empty. By taking the empty run also for $\mathbf{f}_2$, the claim is trivially satisfied as $\delta(\epsilon, \epsilon) = 0$.

Step case ($m > 0$, $n = 0$)**.** By definition, $\delta(\mathbf{e}_1, \epsilon) = P_L((\mathbf{e}_1)_m) + \delta(\mathbf{e}_1|_{m-1}, \epsilon)$. As $\mathbf{e}_1 \sim_{cc} \mathbf{e}_2$ implies $\mathbf{e}_1|_{m-1} \sim_{cc} \mathbf{e}_2|_{m-1}$, we can apply the induction hypothesis to obtain $\delta(\mathbf{e}_1|_{m-1}, \epsilon) = \delta(\mathbf{e}_2|_m, \epsilon)$. By the assumption κ is comparison-based, and activities in $\mathbf{e}_1$ and $\mathbf{e}_2$ coincide, so $P_L((\mathbf{e}_1)_m) = P_L((\mathbf{e}_2)_m)$. It follows that $\delta(\mathbf{e}_2, \epsilon) = P_L((\mathbf{e}_2)_m) + \delta(\mathbf{e}_2|_{m-1}, \epsilon) = P_L((\mathbf{e}_1)_m) + \delta(\mathbf{e}_1|_{m-1}, \epsilon) = \delta(\mathbf{e}_1, \epsilon)$.

Step case ($m = 0$, $n > 0$)**.** Similar as the previous case, using the fact that $P_M((\mathbf{f}_1)_n) = P_M((\mathbf{f}_2)_n)$ because κ is comparison-based.

Step case ($m > 0$, $n > 0$)**.** Let $e_1 = (b, \alpha_1) = (\mathbf{e}_1)_m$ (resp. $e_2 = (b, \alpha_2) = (\mathbf{e}_2)_m$) be the last event in $\mathbf{e}_1$ (resp. $\mathbf{e}_2$), and $f = (t, \beta_1)$ the last transition firing in $\mathbf{f}_1$. According to Definition 9, $\delta(\mathbf{e}_1, \mathbf{f}_1)$ is defined as a minimum of three expressions. Reasoning as in the previous two cases shows that there are process runs $\hat{\mathbf{f}}_2$, $\bar{\mathbf{f}}_2$ such that $P_L(e_1) + \delta(\mathbf{e}_1|_{m-1}, \mathbf{f}_1) = P_L(e_2) + \delta(\mathbf{e}_2|_{m-1}, \hat{\mathbf{f}}_2)$ and $P_M(f) + \delta(\mathbf{e}_1, \mathbf{f}_1|_{n-1}) = P_M((\bar{\mathbf{f}}_2)_n) + \delta(\mathbf{e}_2, \bar{\mathbf{f}}_2|_{n-1})$. We now show that there is also a process run $\mathbf{f}_2$ such that

$$P_=(e_1, f) + \delta(\mathbf{e}_1|_{m-1}, \mathbf{f}_1|_{n-1}) = P_=(e_2, (\mathbf{f}_2)_n) + \delta(\mathbf{e}_2|_{m-1}, \mathbf{f}_2|_{n-1}) \tag{1}$$

so $\delta(\mathbf{e}_1, \mathbf{f}_1) = \delta(\mathbf{e}_2, \mathbf{f}_2)$ follows. As $\mathbf{e}_1 \sim_{cc} \mathbf{e}_2$ implies $\mathbf{e}_1|_{m-1} \sim_{cc} \mathbf{e}_2|_{m-1}$, by the induction hypothesis there exists a process run $\mathbf{f}'_2$ such that $|\mathbf{f}'_2| = n-1$, $\delta(\mathbf{e}_1|_{m-1}, \mathbf{f}_1|_{n-1}) = \delta(\mathbf{e}_2|_{m-1}, \mathbf{f}'_2)$, and $\alpha_{sv}(\mathbf{f}_1|_{n-1}) \sim_{cc} \alpha_{sv}(\mathbf{f}'_2)$.

We set $\mathbf{f}_2 = \mathbf{f}'_2 \cdot (t, \beta_2)$, where β_2 is defined as follows: for all $v \in V$, $\beta_2(v^r) = \alpha_{sv}(\mathbf{f}'_2)(v)$, and $\beta_2(v^w)$ is defined as either $\beta_2(v^w) = \beta_1(v^w)$ if v is not restricted to constant comparison, or otherwise

$$\beta_2(v^w) = \begin{cases} \alpha_2(v) & \text{if } \beta_1(v^w) = \alpha_1(v) \\ \alpha_1(v) & \text{if } \beta_1(v^w) \neq \alpha_1(v) \text{ and } \beta_1(v^w) = \alpha_2(v) \\ \beta_1(v^w) & \text{otherwise} \end{cases} \tag{2}$$

We now show that (i) β_2 satisfies *guard*(t), (ii) $\alpha_{sv}(\mathbf{f}_1) \sim_{cc} \alpha_{sv}(\mathbf{f}_2)$, and (iii) $P_=((b, \alpha_1), (t, \beta_1)) = P_=((b, \alpha_2), (t, \beta_2))$.

For *(i)*, note that $\alpha_{sv}(\mathbf{f}_1|_{n-1}) \sim_{cc} \alpha_{sv}(\mathbf{f}'_2)$ implies that for all $v \in V$, either $\beta_1(v^r) = \beta_2(v^r)$, or v is restricted to constant comparison and $\beta_1(v^r) \sim^{v}_{cc} \beta_2(v^r)$. Moreover, by definition of β_2 we have for all $v \in V$, either $\beta_1(v^w) = \beta_2(v^w)$, or v is restricted to constant comparison and by Eq. (2) one of the following holds: $\beta_2(v^w) = \alpha_2(v) \sim^{v}_{cc} \alpha_1(v) = \beta_1(v^w)$, or $\beta_2(v^w) = \alpha_1(v) \sim^{v}_{cc} \alpha_2(v) = \beta_1(v^w)$, or $\beta_1(v^w) = \beta_2(v^w)$; where we use $\alpha_1(v) \sim^{v}_{cc} \alpha_2(v)$, which follows from $\mathbf{e}_1 \sim_{cc} \mathbf{e}_2$. Thus, we have the following ($\star$): β_1 and β_2 coincide on all variables that are not restricted to constant comparison, and satisfy $\beta_2(v^w) \sim_{cc}^{v} \beta_1(v^w)$ otherwise. It follows that since $\sim_{cc}$-equivalent assignments satisfy the same constraints, and $\beta_1 \models guard(t)$, also $\beta_2 \models guard(t)$. Item *(ii)* then follows from $(\star)$ and the construction of a state variable assignment after a transition firing.

For *(iii)*, we observe that for all variables v such that $\beta_1(v^w) \neq \beta_2(v^w)$, i.e., $\beta_2(v^w)$ is defined by one of the three cases in Eq. (2), one can check that $\beta_1(x^w) = \alpha_1(x)$ if and only if $\beta_2(x^w) = \alpha_2(x)$. As κ is a comparison-based cost function, it follows that $P_=((b, \alpha_1), (t, \beta_1)) = P_=((b, \alpha_2), (t, \beta_2))$.

From *(i)* we obtain that $\mathbf{f}_2$ is indeed a (partial) process run in $\mathcal{N}$, and *(iii)* implies Eq. (1). □

An interesting byproduct of the constructive proof of Theorem 2 is that given $\gamma \in Align^{opt}(\mathcal{N}, \mathbf{e})$, an optimal alignment γ for a log trace $\mathbf{e}$, for every trace $\mathbf{e}'$ in the same cluster (i.e., $\mathbf{e} \sim_{cc} \mathbf{e}'$) an optimal alignment is easily computed from γ, $\mathbf{e}$, and $\mathbf{e}'$ in linear time.

All in all, we thus get that our clustering technique allows us to compute faithful conformance metrics on logs by calculating alignment costs only on a single representative trace per cluster.

5. Extended conformance checking artifacts

We next demonstrate how the CoCoMoT framework accommodates further conformance checking tasks studied in the process mining literature, notably multi- and anti-alignments. We again assume a distance-based cost function κ.

5.1. Multi-alignments

Multi-alignments leverage the concept of alignments to a set of traces, by asking for one process run that minimizes the distance to all traces in the set. They are for instance of interest to find a representative full run of a model for a given portion of a log [18]. The following definition is based on [25].

Definition 15 (*Multi-alignment*)**.** For a DPN $\mathcal{N}$ and a finite set of log traces S, an *optimal multi-alignment* is given by a process run $\mathbf{f} \in Runs(\mathcal{N})$ that minimizes the quantity $\max_{\mathbf{e}\in S} dist(\mathbf{e}, \mathbf{f})$ representing the maximal distance to any log trace in S, that is, $\mathbf{f}$ is a solution to $\mathrm{argmin}_{\mathbf{u}\in Runs(\mathcal{N})} \max_{\mathbf{e}\in S} dist(\mathbf{e}, \mathbf{u})$.

Note that, instead of the maximum, one can use different aggregation functions such as the sum, cf. [18, Def. 4]. The next example illustrates the problem:

Example 9. Consider the DPN from Ex. 1 and the log traces $\mathbf{e}_1 = \langle(\mathsf{a}, \{x \mapsto 2\}), (\mathsf{b}, \{y \mapsto 1\})\rangle$ and $\mathbf{e}_2 = \langle(\mathsf{a}, \{x \mapsto 2\}), (\mathsf{d}, \{y \mapsto 2\})\rangle$. On their own, both traces can be aligned with cost 0. The situation changes if one considers instead multi-alignments, for instance the following ones:

γ_1:

a $\{x \mapsto 2\}$	b $\{y \mapsto 1\}$	$\gg$	$\gg$
a $\{x \mapsto 2\}$	$\gg$	$\gg$	d $\{y \mapsto 2\}$
a $\{x^w \mapsto 2\}$	b $\{y^w \mapsto 1\}$	τ	d $\{y^w \mapsto 2\}$

γ_2:

a $\{x \mapsto 2\}$	b $\{y \mapsto 1\}$	$\gg$	$\gg$
a $\{x \mapsto 2\}$	$\gg$	$\gg$	d $\{y \mapsto 2\}$
a $\{x^w \mapsto 2\}$	b $\{y^w \mapsto 1\}$	τ	$\gg$

In γ_1, the cost of the alignment for both $\mathbf{e}_1$ and $\mathbf{e}_2$ is 2 according to the standard cost function, as both contain a model move that writes one variable. On the other hand, in γ_2, the cost of the alignment for $\mathbf{e}_1$ is 0 and for $\mathbf{e}_2$ it is 1. In fact, the latter is an optimal solution to the multi-alignment problem, with multi-alignment cost 1.

Encoding. Suppose one aims to solve the multi-alignment problem for a DPN $\mathcal{N}$ and a finite set of log traces $S = \{\mathbf{e}_1, \dots, \mathbf{e}_k\}$. First of all, in a similar way as described in Section 3.1, we can obtain an upper bound on the length of the process run. Let n be this bound. For each i, let (φ_δ^i) be the instantiation of the formula (φ_δ) in Section 3 with respect to $\mathbf{e}_i$. Let moreover $m_i = |\mathbf{e}_i|$ and $\mathrm{d}^i_{m_i,n}$ be as defined in the equations (φ_δ^i). We abbreviate by $\varphi_{run} = \varphi_{init} \wedge \varphi_{final} \wedge \varphi_{trans} \wedge \varphi_{enabled} \wedge \varphi_{mark} \wedge \varphi_{data}$ the part of the encoding that represents the process run. To solve the multi-alignment problem for $\mathcal{N}$ and S, we consider the following optimization problem:

$$\varphi_{run} \wedge \bigwedge_{i=1}^{n} \varphi_\delta^i \quad \text{minimizing} \quad \max_{i=1}^{k} \mathrm{d}^i_{m_i,n} \tag{Φ_{multi}}$$

Here, the maximum over a finite set can be encoded in SMT by nested if-then-else-expressions. Given a satisfying assignment for (Φ_{multi}), we can apply the decoding from Section 3 to obtain a process run $\mathbf{f}$ and k alignments $\gamma_1, \dots, \gamma_k$, that solve the multi-alignment problem. Note that it is also easy to adopt a definition of multi-alignments that uses the sum as aggregation function (cf. [18, Def. 4]), by changing the optimization objective in (Φ_{multi}) to $\sum_{i=1}^{k} \delta^i_{m_i,n}$.

Note that the cost-based clustering technique as presented in Section 4 is compatible with multi-alignments, in the following sense: suppose that a set of log traces S contains different traces $\mathbf{e}$ and $\mathbf{e}'$ such that $\mathbf{e} \sim_{cc} \mathbf{e}'$, and let $S' = S \setminus \{\mathbf{e}'\}$. Then the costs of an optimal multi-alignment for S and an optimal multi-alignment for S' coincide, i.e., $\max_{\mathbf{e}\in S} dist_\kappa(\mathbf{e}, \mathbf{f}) = \max_{\mathbf{e}\in S'} dist_\kappa(\mathbf{e}, \mathbf{f})$: Indeed, the proof of Theorem 2 shows that every alignment γ of cost k for $\mathbf{e}$ corresponds to an alignment γ' of cost k for $\mathbf{e}'$.

5.2. Anti-alignments

The conformance checking artifact of anti-alignments was introduced to measure the precision of a model, by quantifying how much the model's behavior may differ from the behavior observed in a log. To that end, one seeks a model run that deviates as much as possible from all traces in a given log [32,33].

We first give the definition of anti-alignments from [25], which can be seen as an "inversion" of multi-alignments in that minimum and maximum are swapped:

Definition 16 (*Anti-alignment*)**.** For a DPN $\mathcal{N}$ and a finite set of log traces $S = \{\mathbf{e}_1, \dots, \mathbf{e}_n\}$, an *optimal anti-alignment of* S is given by a process run $\mathbf{f} \in \mathcal{P}_\mathcal{N}$ that maximizes the quantity $\min_{\mathbf{e}\in S} dist_\kappa(\mathbf{e}, \mathbf{f})$ (which represents the minimal distance to any log trace in S), i.e., $\mathbf{f}$ is a solution to $\mathrm{argmax}_{\mathbf{u}\in\mathcal{P}_\mathcal{N}} \min_{\mathbf{e}\in S} dist_\kappa(\mathbf{e}, \mathbf{u})$.

However, as also stated in [25], anti-alignments according to Definition 16 are in general not defined in processes with loops. We therefore consider the following variation, which restricts to runs of bounded length, and assume the standard cost function from Section 2.2.

Definition 17 (*n-anti-alignment*)**.** For a DPN $\mathcal{N}$, a finite set of log traces S, and $n \in \mathbb{N}$, an *n-anti-alignment* is given by a process run $\mathbf{f} \in \mathcal{P}_\mathcal{N}$ of length at most n such that $\min_{\mathbf{e}\in S} dist_\kappa(\mathbf{e}, \mathbf{f})$ is maximal, that is, $\mathbf{f}$ is a solution to $\mathrm{argmax}_{\mathbf{u}\in\mathcal{P}_\mathcal{N}, |\mathbf{u}|\leq n} \min_{\mathbf{e}\in S} dist_\kappa(\mathbf{e}, \mathbf{u})$.

Example 10. Consider the DPN from Ex. 1 and the log trace $\mathbf{e} = \langle(\mathsf{a}, \{x \mapsto 2\}), (\mathsf{b}, \{y \mapsto 1\})\rangle$. A 3-anti-alignment for $S = \{\mathbf{e}\}$ is e.g. the following one, with cost 4 (cost 1 for a synchronous move with incorrect write operation, cost 1 for a log move, and cost 2 for a model move that writes one variable):

a $\{x \mapsto 2\}$	b $\{y \mapsto 1\}$	$\gg$	$\gg$
a $\{x^w \mapsto 1\}$	$\gg$	τ	d $\{y^w \mapsto 1\}$

Note that the solution is not unique, as x could be assigned a different value.

Encoding. We again abbreviate by $\varphi_{run} = \varphi_{init} \wedge \varphi_{final} \wedge \varphi_{trans} \wedge \varphi_{enabled} \wedge \varphi_{mark} \wedge \varphi_{data}$ the encoding of the process run. To solve the n-anti-alignment problem for $\mathcal{N}$ and S, we consider the following optimization problem:

$$\varphi_{run} \wedge \bigwedge_{i=1}^{n} \varphi_\delta^i \quad \text{maximizing} \quad \min_{i=1}^{k} \mathtt{d}^i_{m_i,n} \qquad (\Phi_{anti})$$

The minimum over a finite set can be encoded by nested if-then-else-expressions. Given a satisfying assignment for (Φ_{multi}), we can apply the decoding from Section 3 to obtain the desired process run $\mathbf{f}$ and k alignments $\gamma_1, \dots, \gamma_k$.

In the literature, also the notion of a (d, n)-anti-alignment for a set of traces S is used, which asks to find a run of length exactly n that has distance at least d to all traces in S [32]. CoCoMoT can also accommodate this task, by replacing the maximization objective in (Φ_{anti}) by hard constraints $\bigwedge_{i=1}^{k} (\mathtt{d}^i_{m_i,n} = d)$. The task thus becomes a satisfaction rather than an optimization problem.

Finally, we note that just like the cost-based clustering technique from Section 4 is compatible with multi-alignments, it is also compatible with anti-alignments, for the same reasons as given in Section 5.1,.

6. Implementation

In this section we report on the DPN conformance checking tool `cocomot`, a proof-of-concept implementation based on the encodings in Section 3.2 and Section 5. We focus on the implementation, optimizations, and experiments on benchmarks from the literature. Source code and data sets are publicly available.[4]

6.1. Implementation

The tool `cocomot` is a Python command line script: it takes as input a DPN and a log, and computes an optimal alignment, using the standard cost function from Section 2.2. To this end, the following standard input formats are supported:

- For process models the PNML[5] format is used, which supports DPNs by allowing to specify the data variables along with their types, as well as transition guards involving linear arithmetic expressions. E.g., the PNML representation of the DPN from Example 1 can be found in the tool repository.[6]
- The log is expected to be in the XES format.[7] E.g., the trace from Example 2 can be found in the tool repository.[8]

Alternatively, instead of computing the optimal alignment for every trace in the log, `cocomot` can also produce multi- or anti-alignments for a sublog. For efficiency, `cocomot` first preprocesses the log to a sublog of unique traces, and second applies trace clustering as described in Section 4 to further partition the sublog into equivalent traces. The conformance check is then run for one representative from every equivalence class.

The tool uses `pm4py` [34][9] to parse traces, and employs the SMT solvers Yices 2 [16] and Z3 [15] as backends, using the bindings provided by the respective Python interfaces. Since Yices 2 has no built-in optimization, we implemented a minimization scheme using multiple satisfiability checks. Every check is run with a timeout, to avoid divergence on large problems.

Encoding optimizations. To facilitate faster solving, and prune the search space, we modified the encoding presented in Section 3.2 in several ways. We report here on the most effective changes.

(1) We perform a reachability analysis in a preprocessing step. This allows us to restrict the range of transition variables $\mathtt{S}_i$ in (φ_{trans}), as well as the cases $\mathtt{S}_i = j$ in ($\varphi_{enabled}$) and (φ_{mark}) to those transitions that are actually reachable. Moreover, if a data variable $v \in V$ cannot be written in step i because no respective transition is reachable, we set $\mathtt{X}_{i,v}$ identical to $\mathtt{X}_{i-1,v}$ to reduce the number of variables that represent data values, for $i > 0$.
(2) If the net is 1-bounded, the marking variables $\mathtt{M}_{i,p}$ are boolean rather than integer, as done in [20].
(3) Recall that the optimization objective $\mathtt{d}_{m,n}$ in (Φ) is to be minimized. Therefore, the equations of the form $\mathtt{d}_{i+1,j+1} = min(e_1, e_2, e_3)$ in (φ_δ) can be replaced by inequalities $\mathtt{d}_{i+1,j+1} \geq min(e_1, e_2, e_3)$. The latter is equivalent to $\mathtt{d}_{i+1,j+1} \geq e_1 \vee \mathtt{d}_{i+1,j+1} \geq e_2 \vee \mathtt{d}_{i+1,j+1} \geq e_3$, which is processed by the solver much more efficiently since it avoids an if-then-else construct.
(4) Numerous subexpressions are replaced by fresh variables together with defining equations (in particular when occurring repeatedly).

We conclude this section by illustrating CoCoMoT on a real-world example.

Example 11. Fig. 1 shows a DPN for a road fine management process by the Italian police [8]. It was generated by automatic mining techniques along with expert domain knowledge. Here, guards $v^w = ?$ express that variable v is written by this transition to an arbitrary value. For the example trace

$\langle$(create fine, $\{a \mapsto 36, t \mapsto 0, p \mapsto 0, d \mapsto \text{nil}\}$),
(send fine, $\{ds \mapsto 2879, e \mapsto 13\}$), (insert fine, $\emptyset$),
(appeal to prefecture, $\{dp \mapsto 192\}$), (add penalty, $\{a \mapsto 74\}$),
(send appeal, $\{d \mapsto \#\}$)$\rangle$

the following optimal alignment is generated by `cocomot`:

create fine	send fine	insert fine	appeal to prefecture	add penalty	send appeal	$\gg$
$a \mapsto 36$ $t \mapsto 0$ $p \mapsto 0$ $d \mapsto$ nil	$ds \mapsto 2879$ $e \mapsto 13$		$dp \mapsto 192$	$a \mapsto 74$	$d \mapsto \#$	
create fine	send fine	insert fine	appeal to prefecture	$\gg$	send appeal	τ_6
$a^w \mapsto 36$ $t^w \mapsto 0$ $p^w \mapsto 0$ $d^w \mapsto$ nil	$ds^w \mapsto 0$ $e^w \mapsto 13$		$dp^w \mapsto 192$		$d^w \mapsto$ G	

This alignment has cost 3 according to the standard cost function: 1 for the log move, and 1 for each of the two mismatching data values in send fine and send appeal. Alignments identify steps where cases do not follow the normative model, thus hinting at aspects where the business process could be improved: in this case, first, add penalty should have happened before send appeal; second, the too high value for ds (a time interval) in send fine indicates that this event occurred too late; and third, while the assignment $d \mapsto \#$ is valid in send appeal, this choice causes the

[4] https://github.com/bytekid/cocomot
[5] https://www.pnml.org/
[6] https://github.com/bytekid/cocomot/blob/main/tests/test9/net.pnml
[7] https://xes-standard.org/
[8] https://github.com/bytekid/cocomot/blob/main/tests/test9/trace1_0.xes
[9] https://pm4py.fit.fraunhofer.de/

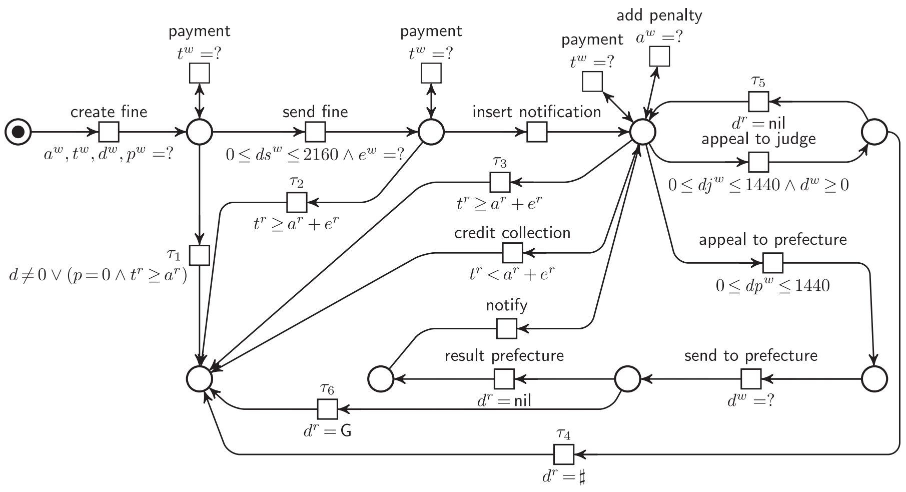


Fig. 1. DPN for road fine data set.

Table 2
Characteristics of data sets used for the experiments.

	# traces	# unique	# non-eq	avg	med	max length
(A) road fines	150370	35681	4290	5.6	6	20
(B) hospital billing	100000	4047	4039	8.9	8	217
(C) sepsis	1050	846	846	16.3	14	185

Table 3
Characteristics of DPNs used for the experiments.

	$\|P\|$	$\|T\|$	$\|T_g\|$	$\|L\|$	$\|V\|$	Data variable types
(A) road fines	9	19	11	13	8	4 `int`, 3 `rat`, 1 `str`
(B) hospital billing	34	36	16	29	4	1 `bool`, 3 `str`
(C) sepsis	48	36	2	3	2	3 `rat`

Table 4
Results of experiments: overview.

	# traces	∞	enc	solve	κ_{avg}	κ_{med}	κ_{max}	ProM
(A) road fines	4290	0	75	293	1.8	2	7	107
(B) hospital billing	4039	182	197	31083	2.9	3	51	267
(C) sepsis	846	653	192	471446	3.2	3	5	16

process to be in a deadlock afterwards, which witnesses the fact that this process model is not data-aware sound [11].

6.2. Experiments

In this section, we describe first the benchmarks used, and then different experiments performed with these along with our conclusions. Unless stated otherwise, tests were run single-threaded on a 12-core Intel i7-5930K 3.50 GHz machine with 32 GB of main memory; we used Yices 2; and a timeout of two minutes per trace.

Benchmarks. We tested `cocomot` on three data sets used in earlier work [8,9], which are publicly available:

- Dataset (A) is the log of road traffic fines [35] that was presented together with the DPN from Example 11 shown in Fig. 1.
- Dataset (B) is a hospital billing log [36], matched against the DPN [9, Fig. 15.3], a normative model created by domain experts.
- Dataset (C) is a log of a triage process for sepsis patients in a hospital [37], matched against the normative model [9, Fig. 13.3].

Table 2 gives an overview over these datasets, listing the number of traces, the unique number of traces, the equivalence classes after applying the clustering technique from Section 4, and the average/median/maximum trace length. Some of these data sets use string variables. In `cocomot`, they are represented by integer variables, encoding the string literals in the model as distinct natural numbers (cf. [9, p. 87]). Table 3 collects some data about the respective DPNs, namely the number of places $|P|$ and transitions $|T|$, the number of transitions with guard $|T_g|$, the total number of guard literals $|L|$, and the number of data variables $|V|$ with their type. All DPNs are one-bounded, and (A) is even a transition system. The table shows that in data set (A) the control-flow perspective is small but the data perspective is comparatively rich, whereas data set (C) has contrastive characteristics, and (B) is somewhere in between.

Feasibility experiment. To evaluate feasibility, we ran `cocomot` on datasets (A), (B), and (C). Table 4 shows the results, listing the number of processed traces (i.e., the equivalence classes after clustering), the number of timeouts (120 s), the total encoding time in seconds, the total solving time in seconds (including timeouts), and the average/median/maximal cost of an optimal alignment. We also compare our results to those provided by the other existing alignment-based conformance checking tool for DPNs based on [8,9] and implemented as a plugin in ProM.[10] We ran this plugin under the default settings over the datasets from above and included its execution time in the last column of Table 4.

Fig. 2 shows two diagrams for each data set: the left scatter plots shows the computation time in seconds (y-axis) against the trace length (x-axis); the right plots show how many traces

[10] https://promtools.org/

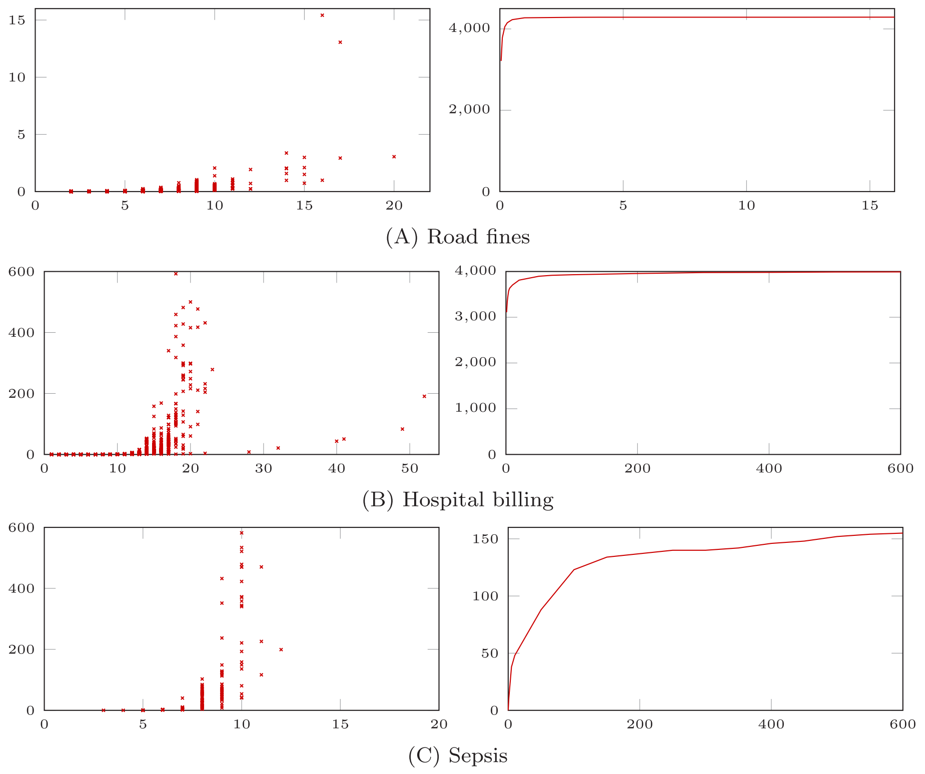


Fig. 2. Trace length vs. solving time, and number of problems solved within time limit.

(y-axis) were solved within different time limits in seconds (x-axis).

We briefly comment on the results for each data set:

(A) The great majority of traces in this data set is relatively short, with a median length of 6, and accordingly the median optimal alignment cost is as small as 2. All optimal alignments are discovered within at most 16 s, and in fact, 99.6% of the traces are handled within 1s. The encoding time accounts for slightly more than 20% of the total time, which seems comparatively high, but this is due to the small computation time overall. Trace clustering was particularly effective for this data set, using the predicates $delaySend \geq 2160$, $delayJudge \geq 1440$, $delayPrefecture \geq 1440$, $points < 0$, and $points > 0$. In fact, trace clustering cuts down the time to process the alignment by a factor of 8.

(B) While the median trace length is only 8, there are some long outliers (up to length 217). Accordingly, also the median cost of the optimal alignment is 3, thus higher than for data set (A), and the maximal cost encountered was 51. In fact, `cocomot` times out on 56 long traces, and for some others the optimal alignment computation required almost the given timeout. However, 78% of the traces were handled within 1s, and 93% within 10 s. For this data set, the encoding time was negligible in comparison to the solving time, and comprised less than 1% of the overall computation time. The benefit of clustering was small.

(C) This log has longer traces than the previous two data sets, and the control structure of the respective DPN is far more complex. `cocomot` here suffers from many timeouts. However, when increasing the timeout, we could find optimal alignments for all except for 37 examples. For long traces, the computation required up to two hours, with optimal alignment costs of up to 120. Also for this data set the encoding time was negligible compared to the solving time; clustering was not applicable.

Overall, we see that `cocomot` performs reasonably well if traces are not too long and the control flow structure is moderately complex, even with rich data perspectives as in data sets (A) and (B). In contrast, complex traces and control flow cause a blowup in the search space and performance deteriorates, as for data set (C). When comparing with ProM, this becomes even more obvious. While ProM is faster than `cocomot` on all data sets, for (A) the conformance checking times are roughly in the same magnitude, whereas ProM outperforms `cocomot` by far on data set (C). We attribute the relatively good performance of `cocomot` on data set (A) to the rich data perspective, which requires the approach implemented in ProM comparatively often to backtrack and update assignments. On the other hand, the data perspective in (B) and (C) are rather plain (transition guards are few and simple), while the control flow is much more complex, involving multiple tokens. In this case, the encoding-based approach does not seem to be an advantage, as was also noted in earlier works on conformance checking via SAT-based encodings [38]. However, we emphasize again the flexibility and modularity of the CoCoMoT framework: it can easily be adapted to check e.g. anti- and multi-alignments; whereas it is not clear how the approach from [8,9] can be adopted for these tasks.

Scalability. To evaluate scalability of `cocomot` with respect to model characteristics, we ran experiments with subsequently enriched versions of the benchmarks (A)–(C), as follows. For each of the three data sets, we selected a subset of traces for which the conformance checking task can be accomplished by `cocomot` in a couple of minutes. Precisely, for dataset (A) a representative for every trace cluster in the log was taken (3856 traces); for (B) all traces up to length 9 (2160 traces); and for (C) all traces up to

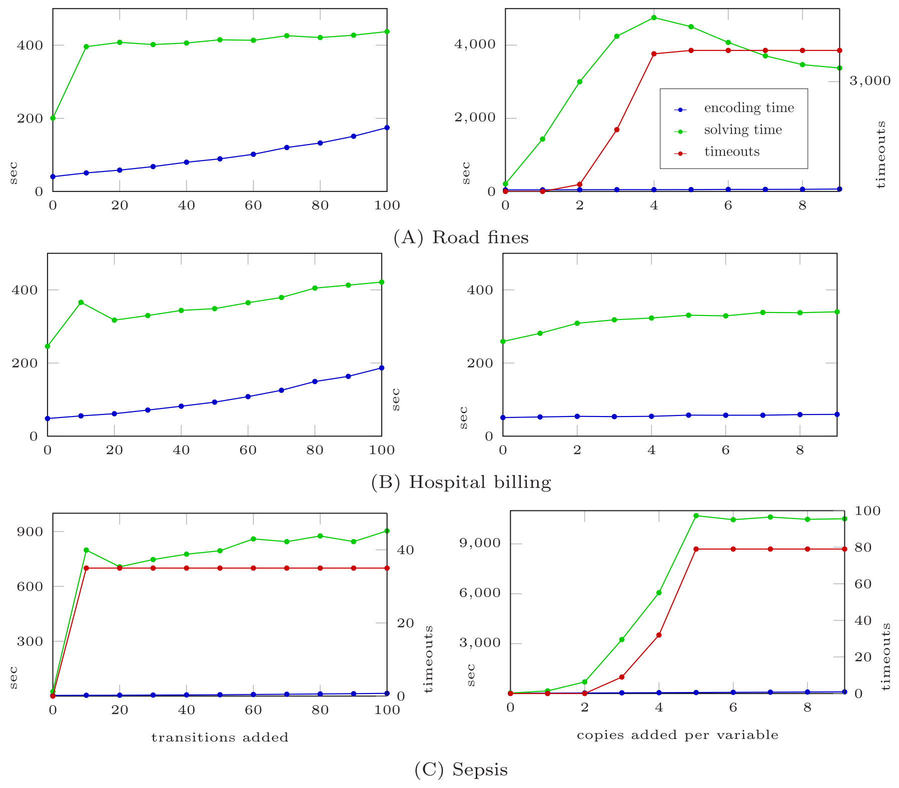


Fig. 3. Scalability with respect to the control flow (left) and data perspectives (right).

length 7 (128 traces). Below, we refer to these sublogs by L_A, L_B, and L_C.

First, we focused on scalability with respect to the control flow: Given DPN $\mathcal{N}$ and some $k \in \mathbb{N}$, we generated an "obfuscated" DPN $\mathcal{N}_k$ by adding k silent transitions without guards, in a way such that the set of valid runs is not modified as far as visible transitions are concerned. The results for $k = 0, 10, \dots, 100$ are shown in the left column of Fig. 3, with the x-axes listing the number of added transitions, and the y-axes on the left the encoding (blue) and solving time (green) in seconds. For dataset (C), the y-axis on the right shows in addition the number of timeouts (red); for datasets (A) and (B) no timeouts occurred. These results show that for datasets (A) and (B), the additional states have only moderate impact on the performance as the runtime increases linearly. Data set (C) is in general difficult for `cocomot` as shown above, and due to the complex control flow additional states cause a severe deterioration.

Second, we evaluated scalability with respect to the data perspective. Given a DPN $\mathcal{N}$ for one of the three data sets and some $k \in \mathbb{N}$, we generated a modified DPN $\mathcal{N}_k$ as follows: for every data variable v in $\mathcal{N}$, copies $v_0, \dots, v_{k-1}$ were added; all v_i are written by all transitions that write v; every guard of the form $v^r \odot e$ for some operator $\odot$ and expression e was expanded to $v^r = v_0^r \wedge v_0^r = v_1^r \wedge \dots \wedge v_{k-1}^r \odot e$, and similar for constraints $v^w \odot e$. Given log L and k, a modified log L_k was constructed by creating for every assignment $v \mapsto n$, for some n, copies $v_i \mapsto n$, for all $0 \leq i < k$. The left column of Fig. 3 shows the results for $k = 0, 1, 2, \dots, 9$, with the x-axes showing k and the y-axes as before. On data set (B), `cocomot` scales very well, likely because there are only four variables and few guards. On data set (A), the situation is different since the data perspective is rich; e.g., for $k = 4$ the modified DPN has 32 variables and eleven guards of size at least 19. However, for up to 20 variables the performance seems reasonable. For data set (C), we attribute the moderate scalability again to the fact that `cocomot` has in general difficulty with the complex control flow.

7. Related work

To compare with the relevant literature, we single out three kinds of related approaches:

- conformance checking techniques dealing with multiple perspectives, paying particular attention to those dealing with processes and data;
- conformance checking methods that, instead of using ad-hoc algorithmic techniques, rely on encodings into general-purpose symbolic AI technology, to the conformance checking task is reduced; and
- work related to trace clustering.

Multi-perspective conformance checking. [39] proposed an extension over the traditional alignment approach in which first the process perspective was aligned, and afterwards additional case attributes of the process were taken into account. This idea was further developed in [40] into a framework using data Petri nets as process models and solving the conformance checking problem by, again, first aligning the process perspective only, using off-the-shelf conformance checking techniques, and afterwards addressing the data perspective by augmenting the already computed alignment with write operations over process variables by solving a mixed integer linear programming problem. Noticeably, the number of such problems to be solved is proportional to the number of traces in the log. A further improvement of the framework proposed in [40] was presented in [13]. There, the authors proposed a faster A*-based technique which would consider both process and data perspectives at once thanks to a customizable cost function. The latter can be tweaked so as to take into consideration concrete scenarios and, for example, assign higher penalties to particular activities that enforce certain data modifications. The setting considered in our work is very similar to the one proposed in [13], as we do not only use DPNs as the process formalism, but also keep our cost function as abstract as possible, leaving its instantiation to the end user. We also discuss caveats that one must take into account when designing a cost function within the CoCoMoT framework. Notice also that our approach is in principle more expressive than the one in [13], thanks to the flexibility provided by SMT-based techniques: as discussed in detail in Section 3.5, one can enrich DPN guards with more sophisticated languages and datatypes (in the SMT spirit) than arithmetical ones. Indeed, any datatype supported by SMT solvers can be exploited for our reasoning task and, hence, incorporated into the language of DPN guards. Moreover, our approach naturally extends to multi- and anti-alignments as discussed in Section 5, while these tasks were not considered in [13].

In [41], the authors suggested a more fine-grained taxonomy for addressing data and process interactions so as to identify a large range of deviations. By introducing so-called composite moves, that pairwise match activities in a log and activities in a process model together with the required operations on the data, they devise a set of criteria for assessing the degree of non-conformity. The approach is similar to the one in [39] in that it first aligns the control flow, and only then addresses the data dimension. However, the main difference (also with [13] and our work) lies in the ability to discover links between data manipulations and process activities, and can account for, for example, data access operations executed outside log events.

A parallel line of research focuses on conformance checking of declarative, multi-perspective processes enriched with data. This traces back to seminal works like [42], which in case of non-conformance only returns a coarse-grained output indicating which constraints are violated, and by which events. The approach in [43] improves on such coarse-grained output, employing ad-hoc algorithms tailored to Declare patterns extended with scalar data and metric temporal conditions. These algorithms do not rely on alignments, but are instead based on the notions of constraint activation and fulfillment, to compute counting metrics on how many times a trace “interacts” with a constraint by satisfying or violating it [44,45].

It is an interesting open question how the SMT-based approach presented here can be tuned to handle declarative, data-aware constraints. This would pave the way towards conformance checking for data-aware mixed-paradigm process models containing both procedural control-flow patterns and declarative constraints, which constitutes an compelling, recent line of investigation that was so far tackled only for a pure control-flow setting [46].

Lastly, it is important to mention another work dealing with multi-perspective conformance checking that does not use alignments and instead relies on a replay-based technique. In [47] the authors propose a conformance checking technique for (a variant of) colored Petri nets that does not only capture the control flow dimension, but also considers objects with unique identity that dynamically evolve throughout the process execution. The proposed replay-based technique allows to detect deviations that happen at the control flow level due to missing object identifiers, violations of user-defined first-order rules (that can be evaluated only “locally”, that is, given an event and a current marking), and differences between modeled and observed object identifiers in the log. While the class of Petri nets considered in [47] is more expressive than DPNs (for example, a DPN can account only for a predefined finite number of simultaneously evolving objects), the replay-based conformance technique is different from the one based on alignments. Specifically, the former only searches for deviations and terminates whenever the first one has been detected, whereas the latter tries to find the “best” process model run closest to a given trace by accommodating encountered deviations in the computed alignment.

Symbolic AI techniques for conformance checking. Alternatives to more traditional, A*-based approaches are various works that exploit formal methods for checking conformance between a process model and a log trace. The first family of approaches relies on automated planning techniques. In [48], the authors showed how, given a k-bounded Petri net and a log trace, one can check conformance by encoding the entire problem into PDDL. The encoded problem can then be solved by state-of-the-art planners, with the resulting plan corresponding to an optimal alignment. A planning-based approach was also recently used for aligning data-aware declarative processes in [49]. There, the main idea lies in transforming both declarative models and log traces into finite constraint automata (for the declarative models this is obtained using a known propositionalization technique), and then encoding everything into PDDL. There are more works that address the alignment-based conformance checking problem for Petri nets without data guards using automata-based techniques (for example, [50,51]).

In our approach, we build upon a series of works [20,52] using SAT-based reasoning for conformance checking via alignments. There, the whole conformance checking problem is encoded into a set of clauses, with minimization objectives and weighted variables. Specifically, the clauses represent the underlying Petri net together with its execution semantics, and encode a distance function of choice, that embeds the alignment computation. Our encoding is virtually done in the same vein and extended so as to account for data manipulating guards. At the same time, as opposed to the SAT encoding, our approach allows to account for potentially unbounded nets and consider other types of data guards as well as more advanced variants of cost functions. [38] proposed a new partial MaxSAT encoding for computing alignments (using distance functions) as well as multi- and anti-alignments. It is reported that the new encoding reduces the memory footprint requirements and outperforms in solving time the existing SAT encoding. It is subject to future work if and to what extent such an optimized encoding could be lifted to SMT and the data-aware setting studied here.

Trace clustering. Trace clustering is a wide area of investigation in process mining. Traces are typically clustered to detect outlier vs. common behaviors [53] or to improve the feasibility of process discovery techniques, in terms of understandability [54] and/or performance [2,18].

Alignment-based techniques (exploiting in particular the notion of anti-alignment) have been used to cluster traces depending on their degree of conformance to a reference process model, focusing on pure control-flow specifications without data [18,20].

The conceptual idea behind our clustering approach is to single out traces that represent a set of traces that differ only in their data values. Each such trace acts as a representative for a cluster of multiple traces, all yielding the same alignment cost and alignments that are easily reducible to each other (in the precise sense detailed in Section 4). Differently from the approaches mentioned above, we employ clustering here to tackle a radically different problem, namely that of speeding up the computation of conformance checking. This idea is closely related to the notion of sampling introduced in [55], where conformance checking of an entire log is simplified to conformance checking of a selected subset of traces. In [55], this selection is based on statistical relevance, with the goal of speeding up the conformance checking computation by conducting it only on a portion of the entire log, while obtaining a good approximation of the exact result that would be obtained by exhaustively checking all the traces in the log. Our approach, instead, is exact: clusters are used to group together traces with the same alignment cost, and alignments of traces from the same cluster can be computed from one another in linear time. We also provide (formal) guarantees that the conformance checking result computed over the representative traces faithfully reconstructs the one obtained for the whole log.

8. Conclusions

We have introduced CoCoMoT, a foundational framework equipped with a proof-of-concept, feasible implementation for alignment-based conformance checking of multi-perspective processes. Besides the several technical results provided in the paper, the core contribution provided by CoCoMoT is to connect the area of (data-aware) conformance checking with that of declarative problem solving via SMT. This comes with a great potential for homogeneously tackling a plethora of related problems in a single framework with a solid theoretical basis and several state-of-the-art algorithmic techniques.

The CoCoMoT approach, due to its modularity, readily lends itself to further tasks related to the analysis of data-aware processes. First, following the idea of trace clustering based on multi-alignments from [18,20], one can use CoCoMoT to partition a log of DPN traces, as an alternative to the already employed clustering technique. Then, CoCoMoT can be also used in model repair tasks: given a set of traces, multi-alignments can be leveraged so as to minimize the sum of the trace distances, while replacing some parameter of the DPN by a variable (e.g., the threshold value in a guard). After that, from the satisfying assignment returned by CoCoMoT we obtain the value for this parameter that fits the observed behavior best.

As constraints (φ_{init})–(φ_{data}) symbolically describe a process run of bounded length, our encoding supports bounded model checking. Thus one could also implement *scenario-based* conformance checking, that, for a given trace, finds the best-matching process run that satisfies additional constraints, such as that certain data values are not exceeded.

While in this paper we consider simple linear arithmetics for encoding cost functions in SMT, more complex theories, as well as their combinations, can be considered, thanks to the generality offered by SMT techniques. As discussed in Section 3.5, one can capture more sophisticated cost functions involving background knowledge coming from additional data sources or correctly addressing privacy related aspects (where one typically needs to employ uninterpreted functions). Thanks to the aforementioned modularity, our encoding could be readily extended with such features. All this provides additional motivation for the use of SMT as the operational counterpart of the framework and will be studied in more detail in future work.

Data availability

Data will be made available on request

Acknowledgments

This research has been partially supported by the UNIBZ projects *ADAPTERS*, *SMART-APP*, *REKAP*, *VERBA* and *DUB*, as well as the PRIN MIUR project PINPOINT Prot. 2020FNEB27.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

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

[2] W.M.P. van der Aalst, Process Mining – Discovery, Conformance and Enhancement of Business Processes, Springer, 2011, http://dx.doi.org/10.1007/978-3-642-19345-3.

[3] D. Fahland, Describing behavior of processes with many-to-many interactions, in: Proc. 40th Petri Nets, in: LNCS, vol. 11522, 2019, pp. 3–24, http://dx.doi.org/10.1007/978-3-030-21571-2_1.

[4] M. Montali, A. Rivkin, DB-Nets: On the marriage of colored Petri nets and relational databases, Trans. Petri Nets Other Model. Concurr. 12 (2017) 91–118, http://dx.doi.org/10.1007/978-3-662-55862-1_5.

[5] A. Polyvyanyy, J.M.E.M. van der Werf, S. Overbeek, R. Brouwers, Information systems modeling: Language, verification, and tool support, in: Proc. CAiSE 2019, in: LNCS, vol. 11483, 2019, pp. 194–212, http://dx.doi.org/10.1007/978-3-030-21290-2_13.

[6] S. Ghilardi, A. Gianola, M. Montali, A. Rivkin, Petri nets with parameterised data - modelling and verification, in: Proc. BPM 2020, in: LNCS, vol. 12168, 2020, pp. 55–74, http://dx.doi.org/10.1007/978-3-030-58666-9_4.

[7] A. Burattin, F.M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, Expert Syst. Appl. 65 (2016) 194–211, http://dx.doi.org/10.1016/j.eswa.2016.08.040.

[8] F. Mannhardt, M. de Leoni, H. Reijers, W. van der Aalst, Balanced multi-perspective checking of process conformance, Computing 98 (4) (2016) 407–437, http://dx.doi.org/10.1007/s00607-015-0441-1.

[9] F. Mannhardt, Multi-perspective Process Mining (Ph.D. thesis), Technical University of Eindhoven, 2018.

[10] M. de Leoni, P. Felli, M. Montali, A holistic approach for soundness verification of decision-aware process models, in: Proc. 37th International Conference on Conceptual Modeling, in: LNCS, vol. 11157, 2018, pp. 219–235, http://dx.doi.org/10.1007/978-3-030-00847-5_17.

[11] P. Felli, M. Montali, S. Winkler, Soundness of data-aware processes with arithmetic conditions, in: Proc. 34th CAiSE, in: LNCS, vol. 13295, Springer, 2022, pp. 389–406, http://dx.doi.org/10.1007/978-3-031-07472-1_23.

[12] M. de Leoni, P. Felli, M. Montali, Strategy synthesis for data-aware dynamic systems with multiple actors, in: Proc. 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), 2020, pp. 315–325, http://dx.doi.org/10.24963/kr.2020/32.

[13] F. Mannhardt, M. de Leoni, H.A. Reijers, W.M.P. van der Aalst, Decision mining revisited - discovering overlapping rules, in: Proc. 28th CAiSE, in: LNCS, vol. 9694, 2016, pp. 377–392, http://dx.doi.org/10.1007/978-3-319-39696-5_23.

[14] C.W. Barrett, C. Tinelli, Satisfiability modulo theories, in: Handbook of Model Checking, Springer, 2018, pp. 305–343, http://dx.doi.org/10.1007/978-3-319-10575-8_11.

[15] L. de Moura, N.B. rner, Z3: an efficient SMT solver, in: Proc. 14th TACAS, in: LNCS, vol. 4963, 2008, pp. 337–340, http://dx.doi.org/10.1007/978-3-540-78800-3_24.

[16] B. Dutertre, Yices 2.2, in: Proc. 26th CAV, in: LNCS, vol. 8559, 2014, pp. 737–744, http://dx.doi.org/10.1007/978-3-319-08867-9_49.

[17] M. Boltenhagen, T. Chatain, J. Carmona, A discounted cost function for fast alignments of business processes, in: A. Polyvyanyy, M.T. Wynn, A.V. Looy, M. Reichert (Eds.), Proc. BPM 2021, in: Lecture Notes in Computer Science, vol. 12875, Springer, 2021, pp. 252–269, http://dx.doi.org/10.1007/978-3-030-85469-0_17.

[18] T. Chatain, J. Carmona, B. van Dongen, Alignment-based trace clustering, in: Proc. 36th International Conference on Conceptual Modeling, in: LNCS, vol. 10650, 2017, pp. 295–308, http://dx.doi.org/10.1007/978-3-319-69904-2_24.

[19] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, CoCoMoT: Conformance checking of multi-perspective processes via SMT, in: A. Polyvyanyy, M.T. Wynn, A.V. Looy, M. Reichert (Eds.), Proc. 19th BPM, in: LNCS, vol. 12875, Springer, 2021, pp. 217–234, http://dx.doi.org/10.1007/978-3-030-85469-0_15.

[20] M. Boltenhagen, T. Chatain, J. Carmona, Encoding conformance checking artefacts in SAT, in: Proc. Business Process Management Workshops 2019, in: LNCS, vol. 362, 2019, pp. 160–171, http://dx.doi.org/10.1007/978-3-030-37453-2_14.

[21] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (3) (1970) 443–453, http://dx.doi.org/10.1016/0022-2836(70)90057-4.

[22] C. Barrett, P. Fontaine, C. Tinelli, The SMT-LIB Standard: Version 2.6, Tech. rep., Available at: , 2018, http://smtlib.cs.uiowa.edu/language.shtml.

[23] R. Sebastiani, S. Tomasi, Optimization modulo theories with linear rational costs, ACM Trans. Comput. Log. 16 (2) (2015) 12:1–12:43, http://dx.doi.org/10.1145/2699915.

[24] J. Desel, J. Esparza, Shortest paths in reachability graphs, J. Comput. System Sci. 51 (2) (1995) 314–323, http://dx.doi.org/10.1006/jcss.1995.1070.

[25] M. Boltenhagen, Process Instance Clustering Based on Conformance Checking Artefacts (Ph.D. thesis), University of Paris-Saclay, France, 2021, URL https://tel.archives-ouvertes.fr/tel-03461959.

[26] A.R. Bradley, Z. Manna, The Calculus of Computation – Decision Procedures with Applications to Verification, Springer, 2007, http://dx.doi.org/10.1007/978-3-540-74113-8.
[27] D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, A. Rivkin, SMT-based verification of data-aware processes: a model-theoretic approach, Math. Struct. Comput. Sci. 30 (3) (2020) 271–313, http://dx.doi.org/10.1017/S0960129520000067.
[28] D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, A. Rivkin, Formal modeling and SMT-based parameterized verification of data-aware BPMN, in: Proc. BPM 2019, in: LNCS, 11675, 2019, pp. 157–175, http://dx.doi.org/10.1007/978-3-030-26619-6_12.
[29] S. Ghilardi, A. Gianola, M. Montali, A. Rivkin, Petri net-based object-centric processes with read-only data, Inf. Syst. (2022) http://dx.doi.org/10.1016/j.is.2022.102011.
[30] F. Zandkarimi, J. Rehse, P. Soudmand, H. Hoehle, A generic framework for trace clustering in process mining, in: Proc. 2nd ICPM, IEEE, 2020, pp. 177–184, http://dx.doi.org/10.1109/ICPM49681.2020.00034.
[31] M. Boltenhagen, T. Chatain, J. Carmona, Generalized alignment-based trace clustering of process behavior, in: Proc. 40th Petri Nets, in: LNCS, vol. 11522, 2019, pp. 237–257, http://dx.doi.org/10.1007/978-3-030-21571-2_14.
[32] T. Chatain, J. Carmona, Anti-alignments in conformance checking – the dark side of process models, in: Proc. 37th Petri Nets, in: LNCS, vol. 9698, 2016, pp. 240–258, http://dx.doi.org/10.1007/978-3-319-39086-4_15.
[33] B.F. van Dongen, J. Carmona, T. Chatain, A unified approach for measuring precision and generalization based on anti-alignments, in: M.L. Rosa, P. Loos, O. Pastor (Eds.), Proc. BPM 2016, in: Lecture Notes in Computer Science, vol. 9850, Springer, 2016, pp. 39–56, http://dx.doi.org/10.1007/978-3-319-45348-4_3.
[34] A. Berti, S.J. van Zelst, W.M.P. van der Aalst, Process mining for Python (PM4Py): Bridging the gap between process- and data science, 2019, CoRR abs/1905.06169 arXiv:1905.06169.
[35] M. de Leoni, F. Mannhardt, Road traffic fine management process, 2015, http://dx.doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.
[36] F. Mannhardt, Hospital billing – event log, 2017, http://dx.doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741.
[37] F. Mannhardt, Sepsis data – event log, 2017, http://dx.doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.
[38] J. Ojeda, Partial MaxSAT computation of conformance checking artefacts, in: C. Di Ciccio, C. Di Francescomarino, P. Soffer (Eds.), Proc. ICPM 2021, IEEE, 2021, pp. 17–24, http://dx.doi.org/10.1109/ICPM53251.2021.9576889.
[39] M. de Leoni, W.M.P. van der Aalst, B.F. van Dongen, Data- and resource-aware conformance checking of business processes, in: W. Abramowicz, D. Kriksciuniene, V. Sakalauskas (Eds.), Proc. BIS 2012, in: Lecture Notes in Business Information Processing, vol. 117, Springer, 2012, pp. 48–59, http://dx.doi.org/10.1007/978-3-642-30359-3_5.
[40] M. de Leoni, W.M.P. van der Aalst, Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming, in: Proc. 11th BPM, in: LNCS, vol. 8094, 2013, pp. 113–129, http://dx.doi.org/10.1007/978-3-642-40176-3_10.
[41] M. Alizadeh, X. Lu, D. Fahland, N. Zannone, W.M.P. van der Aalst, Linking data and process perspectives for conformance analysis, Comput. Secur. 73 (2018) 172–193, http://dx.doi.org/10.1016/j.cose.2017.10.010.
[42] F. Chesani, P. Mello, M. Montali, S. Storari, Testing careflow process execution conformance by translating a graphical language to computational logic, in: R. Bellazzi, A. Abu-Hanna, J. Hunter (Eds.), Artificial Intelligence in Medicine, 11th Conference on Artificial Intelligence in Medicine, AIME 2007, Amsterdam, the Netherlands, July 7-11, 2007, Proceedings, in: Lecture Notes in Computer Science, vol. 4594, Springer, 2007, pp. 479–488, http://dx.doi.org/10.1007/978-3-540-73599-1_64.
[43] A. Burattin, F.M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, Expert Syst. Appl. 65 (2016) 194–211, http://dx.doi.org/10.1016/j.eswa.2016.08.040.
[44] M. Montali, F.M. Maggi, F. Chesani, P. Mello, W.M.P. van der Aalst, Monitoring business constraints with the event calculus, ACM Trans. Intell. Syst. Technol. 5 (1) (2013) 17:1–17:30, http://dx.doi.org/10.1145/2542182.2542199.
[45] A. Cecconi, C. Di Ciccio, G. De Giacomo, J. Mendling, Interestingness of traces in declarative process mining: The janus ltlp _f approach, in: M. Weske, M. Montali, I. Weber, J. vom Brocke (Eds.), Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings, in: Lecture Notes in Computer Science, vol. 11080, Springer, 2018, pp. 121–138, http://dx.doi.org/10.1007/978-3-319-98648-7_8.
[46] B.F. van Dongen, J.D. Smedt, C. Di Ciccio, J. Mendling, Conformance checking of mixed-paradigm process models, Inf. Syst. 102 (2021) 101685, http://dx.doi.org/10.1016/j.is.2020.101685.
[47] J.C. Carrasquel, K. Mecheraoui, I.A. Lomazova, Checking conformance between colored Petri nets and event logs, in: W.M.P. van der Aalst, V. Batagelj, D.I. Ignatov, M.Y. Khachay, O. Koltsova, A. Kutuzov, S.O. Kuznetsov, I.A. Lomazova, N.V. Loukachevitch, A. Napoli, A. Panchenko, P.M. Pardalos, M. Pelillo, A.V. Savchenko, E. Tutubalina (Eds.), Proc. AIST 2020, in: Lecture Notes in Computer Science, vol. 12602, Springer, 2020, pp. 435–452, http://dx.doi.org/10.1007/978-3-030-72610-2_33.
[48] M. de Leoni, A. Marrella, Aligning real process executions and prescriptive process models through automated planning, Expert Syst. Appl. 82 (2017) 162–183, http://dx.doi.org/10.1016/j.eswa.2017.03.047.
[49] G. Bergami, F.M. Maggi, A. Marrella, M. Montali, Aligning data-aware declarative process models and event logs, in: A. Polyvyanyy, M.T. Wynn, A.V. Looy, M. Reichert (Eds.), Proc. BPM 2021, in: Lecture Notes in Computer Science, vol. 12875, Springer, 2021, pp. 235–251, http://dx.doi.org/10.1007/978-3-030-85469-0_16.
[50] D. Reißner, R. Conforti, M. Dumas, M. La Rosa, A. Armas-Cervantes, Scalable conformance checking of business processes, in: H. Panetto, C. Debruyne, W. Gaaloul, M.P. Papazoglou, A. Paschke, C.A. Ardagna, R. Meersman (Eds.), On the Move To Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I, in: Lecture Notes in Computer Science, 10573, Springer, 2017, pp. 607–627, http://dx.doi.org/10.1007/978-3-319-69462-7_38.
[51] S.J.J. Leemans, D. Fahland, W.M.P. van der Aalst, Scalable process discovery and conformance checking, Softw. Syst. Model. 17 (2) (2018) 599–631, http://dx.doi.org/10.1007/s10270-016-0545-x.
[52] M. Boltenhagen, T. Chatain, J. Carmona, Optimized SAT encoding of conformance checking artefacts, Computing 103 (1) (2021) 29–50, http://dx.doi.org/10.1007/s00607-020-00831-8.
[53] L. Ghionna, G. Greco, A. Guzzo, L. Pontieri, Outlier detection techniques for process mining applications, in: A. An, S. Matwin, Z.W. Ras, D. Slezak (Eds.), Foundations of Intelligent Systems, 17th International Symposium, ISMIS 2008, Toronto, Canada, May 20-23, 2008, Proceedings, in: Lecture Notes in Computer Science, vol. 4994, Springer, 2008, pp. 150–159, http://dx.doi.org/10.1007/978-3-540-68123-6_17.
[54] F. Zandkarimi, J. Rehse, P. Soudmand, H. Hoehle, A generic framework for trace clustering in process mining, in: B.F. van Dongen, M. Montali, M.T. Wynn (Eds.), 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020, IEEE, 2020, pp. 177–184, http://dx.doi.org/10.1109/ICPM49681.2020.00034.
[55] M. Bauer, H. van der Aa, M. Weidlich, Sampling and approximation techniques for efficient process conformance checking, Inf. Syst. 104 (2022) 101666, http://dx.doi.org/10.1016/j.is.2020.101666.