CoCoMoT: Conformance Checking of Multi-Perspective Processes via SMT

Paolo Felli[1], Alessandro Gianola[1], Marco Montali[1],
Andrey Rivkin[1], and Sarah Winkler[1]

Free University of Bozen-Bolzano, Bolzano, Italy
{pfelli,gianola,montali,rivkin,winkler}@inf.unibz.it

Abstract. Conformance checking is a key process mining task for comparing the expected behavior captured in a process model and the actual behavior recorded in a log. While this problem has been extensively studied for pure control-flow processes, conformance checking with multi-perspective processes is still at its infancy. In this paper, we attack this challenging problem by considering processes that combine the data and control-flow dimensions. In particular, we adopt data Petri nets (DPNs) as the underlying reference formalism, and show how solid, well-established automated reasoning techniques can be effectively employed for computing conformance metrics and data-aware alignments. We do so by introducing the CoCoMoT (Computing Conformance Modulo Theories) framework, with a fourfold contribution. First, we show how SAT-based encodings studied in the pure control-flow setting can be lifted to our data-aware case, using SMT as the underlying formal and algorithmic framework. 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 provide a proof-of-concept implementation that uses a state-of-the-art SMT solver and report on preliminary experiments. Finally, we discuss how CoCoMoT directly lends itself to a number of further tasks, like multi- and anti-alignments, log analysis by clustering, and model repair.

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 [?]. While the problem has been thoroughly studied for pure control-flow processes such as classical Petri nets [?,?], 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., [?,?,?,?]), attacking the conformance checking problem in the non-classical setting is a very challenging task. This problem indeed requires to simultaneously consider, in a combined way, both the control-flow of the process and the data that the process manipulates. 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 [?] and procedural [?,?] 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 introduce a novel general framework, called CoCoMoT, to tackle conformance checking of data Petri nets (DPNs), an extensively studied formalism within BPM [?,?,?] and process mining [?,?,?]. The main feature of CoCoMoT is that, instead of providing ad-hoc algorithmic techniques for checking conformance, it provides an overarching approach based on the theory and practice of Satisfiability Modulo Theories (SMT) [?]. By relying on an SMT backend, we employ 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 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 [?,?], the use of SMT not only allows us to support data, but also capture unbounded nets. Our CoCoMoT approach results in a conformance checking procedure running in NP, which is optimal for the problem, in contrast to earlier approaches running in exponential time [?,?].

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 of 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, and show how this strategy leads to a significant speedup in our experiments.

Third, we report on a proof-of-concept implementation of CoCoMoT, discussing optimization techniques and showing the feasibility of the approach with an experimental evaluation on three different benchmark sets.

Finally, we discuss how our approach, due to its modularity, directly lends itself to a number of further process analysis tasks such as computing *multi-* and *anti-alignments*, using CoCoMoT as a *log clustering* method in the spirit of earlier work for Petri nets without data [?,?], doing *model repair*, and handling more sophisticated data such as persistent, relational data.

The remainder of the paper is structured as follows. In Section ?? we recall the relevant basics about data Petri nets and alignments. This paves the way to present our SMT encoding in Section ??. Our clustering technique that serves as a preprocessor for conformance checking is the topic of Section ??. In Section ?? we describe our prototype implementation and the conducted experiments. Afterwards, we discuss perspectives and potential of our approach in Section **??**.

2 Preliminaries

In this section we provide the required preliminaries. We first recall data Petri nets (DPNs) and their execution semantics, then delve into event logs and conformance checking alignments, and finally discuss the main machinery behind our approach for satisfiability modulo theories (SMT).

2.1 Data Petri Nets

We use Data Petri nets (DPNs) for modelling multi-perspective processes, adopting a formalization as in [**?**,**?**].

We start by introducing sorts – data types of variables manipulated by a process. We fix a set of *(process variable) sorts* $\Sigma = \{\texttt{bool}, \texttt{int}, \texttt{rat}, \texttt{string}\}$ with associated domains of booleans $\mathcal{D}(\texttt{bool}) = \mathbb{B}$, integers $\mathcal{D}(\texttt{int}) = \mathbb{Z}$, rationals $\mathcal{D}(\texttt{rat}) = \mathbb{Q}$, and strings $\mathcal{D}(\texttt{string}) = \mathbb{S}$. A set of *process variables* V is *sorted* if there is a function $sort\colon V \to \Sigma$ assigning a sort 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 $sort(v^r) = sort(v^w) = sort(v)$ for every $v \in V$. For a sort $\sigma \in \Sigma$, V_σ denotes the subset of $V^r \cup V^w$ of annotated variables of sort σ. To manipulate sorted variables, we consider expressions c with the following grammar:

$$
\begin{array}{ll}
c = V_{\texttt{bool}} \mid \mathbb{B} \mid n \geq n \mid r \geq r \mid r > r \mid s = s \mid b \wedge b \mid \neg b & s = V_{\texttt{string}} \mid \mathbb{S} \\
n = V_{\texttt{int}} \mid \mathbb{Z} \mid n + n \mid -n & r = V_{\texttt{rat}} \mid \mathbb{Q} \mid r + r \mid -r
\end{array}
$$

Standard equivalences apply, hence disjunction (i.e., $\vee$) and comparisons $\neq$, $<$, $\leq$ can be used as well (`bool` and `string` only support (in)equality). These expressions form the basis for capturing conditions on the values of variables that are read and written during the execution of activities in the process. For this reason, we call them *constraints*. Intuitively, a constraint $(v_1^r > v_2^r)$ dictates that the current value of variable v_1 is greater than the current value of v_2. Similarly, $(v_1^w > v_2^r + 1) \wedge (v_1^w < v_3^r)$ requires that the new value given to v_1 (i.e., assigned to v_1 as a result of the execution of the activity to which this constraint is attached) is greater than the current value of v_2 plus 1, and smaller than v_3. More 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. The set of read and written variables that appear in a constraint c is denoted by $\mathcal{V}ar(c)$, hence $\mathcal{V}ar(c) \subseteq V^w \cup V^r$. We denote the set of all constraints by $\mathcal{C}(V)$.

Definition 1 (DPN). *A* Petri net with data *(DPN) is given by a tuple $\mathcal{N} = (P, T, F, \ell, A, V, guard)$, where (1) (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 denoting silent transitions; (2) V is a sorted set of process variables; and (3) guard: $T \to \mathcal{C}(V)$ is a guard assignment.*

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 $\mathit{read}(t) = \{v \mid v^r \in \mathcal{V}ar(\mathit{guard}(t))\}$ and $\mathit{write}(t) = \{v \mid v^w \in \mathcal{V}ar(\mathit{guard}(t))\}$. Finally, $G_{\mathcal{N}}$ is the set of all the guards appearing in $\mathcal{N}$.

To assign values to variables, we use variable assignments. A *state variable assignment* is a total function α that assigns a value to each variable in V, such that $\alpha(v) \in \mathcal{D}(\mathit{sort}(v))$ for all $v \in V$. These assignments are used to specify the current value of all variables. Similarly, a *transition variable assignment* is a partial function β that assigns a value to annotated variables, namely $\beta(x) \in \mathcal{D}(\mathit{sort}(x))$, with $x \in V^r \cup V^w$. These are used to specify how variables change as the result of activity executions (cf. Def. **??**).

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.

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

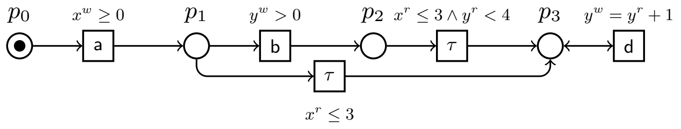
Definition 2 (Transition firing). *A transition $t \in T$ is* enabled *in state (M, α) if a transition variable assignment β exists such that:*
(i) $\beta(v^r) = \alpha(v)$ for every $v \in \mathit{read}(t)$, i.e., β is as α for read variables;
(ii) $\beta \models \mathit{guard}(t)$, i.e., β satisfies the guard; and
(iii) $M(p) \geq F(p, t)$ for every $p \in {}^\bullet t$;
(iv) $\textsc{dom}(\beta) = \mathcal{V}ar(\mathit{guard}(t))$, where DOM *denotes the domain of functions: β is defined for the annotated variables in the guard.*
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 $v \in \mathit{write}(t)$, and $\alpha'(v) = \alpha(v)$ for every $v \notin \mathit{write}(t)$. 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 *final* any state of $\mathcal{N}$ of the form (M_F, α_F) for some α_F.

We say that (M', α') is *reachable* in a DPN iff there exists a sequence of transition firings $\mathbf{f} = (t_1, \beta_1), \dots, (t_n, \beta_n)$, s.t. $(M_I, \alpha_0) \xrightarrow{(t_1,\beta_1)} \dots \xrightarrow{(t_n,\beta_n)} (M', \alpha')$, denoted as $(M_I, \alpha_0) \xrightarrow{\mathbf{f}} (M_n, \alpha_n)$. 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, that is, if the run leads to a final state from the initial state (M_I, α_0). Similar to [**?**], we restrict to DPNs that are *relaxed data sound*, that is, 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 $\mathit{Runs}(\mathcal{N})$.

Example 1. Let $\mathcal{N}$ be as shown (with initial marking $[p_0]$ and final marking $[p_3]$):



The set $\mathit{Runs}(\mathcal{N})$ contains, e.g., $\langle(\mathsf{a}, \{x^w \mapsto 2\}), (\mathsf{b}, \{y^w \mapsto 1\}), (\tau, \{x^r \mapsto 2, y^r \mapsto 1\})\rangle$ and $\langle(\mathsf{a}, \{x^w \mapsto 1\}), (\tau, \{x^r \mapsto 1\}), (\mathsf{d}, \{y^w \mapsto 1\})\rangle$, for $\alpha_0 = \{x \mapsto 0, y \mapsto 0\}$.

2.2 Event Logs and Alignments

Given an arbitrary set A of activity labels, 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 3 (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 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. the process model (i.e., the DPN $\mathcal{N}$), by matching events in the log trace against transitions 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 $\mathcal{F}^{\gg} = \mathcal{F}(\mathcal{N}) \cup \{\gg\}$.

Given a DPN $\mathcal{N}$ and a set $\mathcal{E}$ of events as above, 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: (i) *log move* if $e \in \mathcal{E}$ and $f = \gg$; (ii) *model move* if $e = \gg$ and $f \in \mathcal{F}(\mathcal{N})$; (iii) *synchronous move* if $(e, f) \in \mathcal{E} \times \mathcal{F}(\mathcal{N})$. Let $\mathit{Moves}_{\mathcal{N}}$ be the set of all such moves. We now show how moves can be used to define alignments of log traces.

For a sequence of moves $\gamma = (e_1, f_1), \dots, (e_n, f_n)$, the *log projection* $\gamma|_L$ of γ is the subsequence $e'_1, \dots, e'_i$ of $e_1, \dots, e_n$ such that $e'_1, \dots, e'_i \in \mathcal{E}^*$ is obtained by projecting away from γ all $\gg$ symbols. Similarly, the *model projection* $\gamma|_M$ of γ is the subsequence $f'_1, \dots, f'_j$ of $f_1, \dots, f_n$ such that $f'_1, \dots, f'_j \in \mathcal{F}(\mathcal{N})^*$.

Definition 4 (Alignment). *Given $\mathcal{N}$, a sequence of legal moves γ is an* alignment *of a log trace $\mathbf{e}$ if $\gamma|_L = \mathbf{e}$, and it is* complete *if $\gamma|_M \in \mathit{Runs}(\mathcal{N})$.*

Example 2. The sequences γ_1, γ_2 and γ_3 below are possible 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 Ex. **??**:

$$\gamma_1 = \begin{array}{|l|l|c|}\hline \mathsf{a}\quad x \mapsto 2 & \mathsf{b}\quad y \mapsto 1 & \gg \\ \hline \mathsf{a}\quad x^w \mapsto 2 & \mathsf{b}\quad y^w \mapsto 1 & \tau\ \dots \\ \hline \end{array} \quad \gamma_2 = \begin{array}{|l|c|l|}\hline \mathsf{a}\quad x \mapsto 2 & \gg & \mathsf{b}\quad y \mapsto 1 \\ \hline \mathsf{a}\quad x^w \mapsto 3 & \tau\ \dots & \gg \\ \hline \end{array} \quad \gamma_3 = \begin{array}{|l|l|c|c|}\hline \mathsf{a}\quad x \mapsto 2 & \mathsf{b}\quad y \mapsto 1 & \gg & \gg \\ \hline \gg & \gg & \mathsf{a}\quad x^w \mapsto 3 & \tau\ \dots \\ \hline \end{array}$$

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

We denote by $\mathit{Align}(\mathcal{N}, \mathbf{e})$ the set of complete alignments for a log trace $\mathbf{e}$ w.r.t. $\mathcal{N}$. A *cost function* is a mapping $\kappa \colon \mathit{Moves}_{\mathcal{N}} \to \mathbb{R}^+$ that assigns a cost to every move. It is naturally extended to alignments as follows.

Definition 5 (Cost). *Given $\mathcal{N}$, $\mathbf{e}$ and $\gamma = (e_1, f_1), \dots, (e_n, f_n) \in \mathit{Align}(\mathcal{N}, \mathbf{e})$, the* cost *of γ is obtained by summing up the costs of its moves, that is, $\kappa(\gamma) = \sum_{i=1}^{n} \kappa(e_i, f_i)$. Moreover, γ is* optimal *for $\mathbf{e}$ if $\kappa(\gamma)$ is minimal among all complete alignments for $\mathbf{e}$, namely there is no $\gamma' \in \mathit{Align}(\mathcal{N}, \mathbf{e})$ with $\kappa(\gamma') < \kappa(\gamma)$.*

We denote the cost of an optimal alignment for $\mathbf{e}$ with respect to $\mathcal{N}$ by $\kappa_{\mathcal{N}}^{opt}(\mathbf{e})$. Given $\mathcal{N}$, the set of optimal alignments for $\mathbf{e}$ is denoted by $\mathit{Align}^{opt}(\mathcal{N}, \mathbf{e})$.

2.3 Satisfiability Modulo Theories (SMT)

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 $\nu(p) = \nu(r) = \top$, $\nu(q) = \bot$. The Satisfiability Modulo Theories (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 theories $\mathcal{T}$ (e.g., arithmetics, bit-vectors, arrays, uninterpreted functions). 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) [?]. 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 [?] 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 [?] and Z3 [?].

3 Conformance Checking via SMT

In this section we illustrate our approach. We first describe in Section **??** a generic distance measure to be used as cost function. Then, in Section **??** we detail our encoding of the problem of finding optimal alignments in SMT. Notably, this technique works also for nets with arc multiplicities and unbounded nets, beyond the safe case considered in [?]. Finally, in Section **??** we analyze the computational complexity. Full proofs and details can be found in [?].

3.1 Distance-based Cost Function

We present here a function used to measure the distance between a log trace and a process run. The recursive definition has the same structure as that of the standard edit distance, which allows us to adopt a similar encoding as used in the literature [**?**]. However, it generalizes both the standard edit distance and distance functions previously used for multi-perspective conformance checking [**?**,**?**], and admits also other measures that are specific to the model and the SMT theory used. Our measure is parameterized by three functions:

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

respectively called the *log move penalty*, *model move penalty*, and *synchronous move penalty* functions (cf. Section **??**). We use these functions to assign penalties to log moves, model moves, or synchronous moves. In what follows, we denote prefixes of length j of a log trace $\mathbf{e} \in \mathcal{E}^*$ of length m as $\mathbf{e}|_j$, provided $0 \leq j \leq m$, and analogously for a process run $\mathbf{f} \in \mathit{Runs}(\mathcal{N})$ (recall that these are sequences of transition firings in $\mathcal{F}(\mathcal{N})$).

Definition 6 (Edit distance). *Given a DPN $\mathcal{N}$, let $\mathbf{e} = e_1, \dots, e_m$ be a log trace and $\mathbf{f} = f_1, \dots, f_n$ a process run. For all i and j, $0 \leq i \leq m$ and $0 \leq j \leq n$, the* edit distance $\delta(\mathbf{e}|_i, \mathbf{f}|_j)$ *is recursively defined as follows:*

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

Def. **??** can be used to define a cost function by setting $\kappa(\gamma) = \delta(\gamma|_L, \gamma|_M)$, for any alignment γ. In the sequel, we call such a cost function *distance-based.* Moreover, it is known that for any trace $\mathbf{e}$ and process run $\mathbf{f}$ with $|\mathbf{e}| = m$ and $|\mathbf{f}| = n$, given the $(n+1) \times (m+1)$-matrix D such that $D_{ij} = \delta(\mathbf{e}|_i, \mathbf{f}|_j)$, one can reconstruct an alignment of $\mathbf{e}$ and $\mathbf{f}$ that is optimal with respect to κ [**?**,**?**].

Remark 1. By fixing the parameters $P_=$, P_L, and P_M of Def. **??**, one obtains concrete, known distance-based cost functions, such as the following:
Standard cost function. Def. **??** can be instantiated to the measure in [**?**, Ex. 2], [**?**, Def. 4.5]. To that end, we set $P_L(b, \alpha) = 1$; $P_M(t, \beta) = 0$ if t is silent (i.e., $\ell(t) = \tau$) and $P_M(t, \beta) = |\mathit{write}(t)| + 1$ otherwise; and $P_=((b, \alpha), (t, \beta)) = |\{v \in \textsc{dom}(\alpha) \mid \alpha(v) \neq \beta(v^w)\}|$ if $b = \ell(t)$ and $P_=((b, \alpha), (t, \beta)) = \infty$ otherwise.
Levenshtein distance. The standard edit distance is obtained with $P_L(b, \alpha) = P_M(t, \beta) = 1$, and $P_=((b, \alpha), (t, \beta)) = 0$ if $b = \ell(t)$ and $P_=((b, \alpha), (t, \beta)) = \infty$ otherwise. Note that this measure ignores transition variable assignments β.

For instance, for the alignments γ_1, γ_2, and γ_3 from Ex. **??**, the standard cost function yields $\kappa(\gamma_1) = 0$; $\kappa(\gamma_2) = 2$ (because we get penalty 1 for a synchronous move with incorrect write operation, no penalty for the silent model move, and penalty 1 for the log move); and $\kappa(\gamma_3) = 4$ (because we get penalty 1 for each of the log moves, penalty 2 for a non-silent model move that writes one variable, and no penalty for the silent model move).

3.2 Encoding

Our approach relies on the fact that the optimal alignment for a given log trace is upper-bounded in length. To this end, we use the following observation.

Remark 2. Given a DPN $\mathcal{N}$ and a log trace $\mathbf{e} = e_1, \dots, e_m$, let $\mathbf{f} = f_1, \dots, f_n$ be a valid process run such that $\sum_{j=1}^{n} P_M(f_j)$ is minimal. Then an optimal alignment γ for $\mathbf{e}$ and $\mathcal{N}$ satisfies $\kappa(\gamma) \leq \kappa(\gamma_{max})$, and hence $|\gamma| \leq |\gamma_{max}|$, where γ_{max} is the alignment $(e_1, \gg), \dots, (e_m, \gg), (\gg, f_1), \dots (\gg, f_n)$.

Given a log trace $\mathbf{e} = e_1, \dots, e_m$ and a DPN $\mathcal{N}$ with initial marking M_I, initial state variable assignment α_0, final marking M_F, we want to construct an optimal alignment $\gamma \in \mathit{Align}^{opt}(\mathcal{N}, \mathbf{e})$. To that end, we assume throughout this section that the number of non-empty model steps in γ is bounded by some fixed number n (cf. Rem. **??**). 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.

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

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

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

(c) data variables $X_{i,v}$ for all $v \in V$ and i, $0 \leq i \leq n$; the type of these variables depends on v, with the semantics that $X_{i,v}$ is assigned r iff the value of v at instant i is r; we also write X_i for $(X_{i,v_1}, \dots, X_{i,v_k})$;

(d) distance variables $\delta_{i,j}$ of type integer for $0 \leq i \leq m$ and $0 \leq j \leq n$, where $\delta_{i,j} = d$ if d is 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 $\mathbf{f}$, i.e., $d = \delta(\mathbf{e}|_i, \mathbf{f}_j)$ by Def. **??**.

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:

$$\textstyle\bigwedge_{p \in P} M_{0,p} = M_I(p) \wedge \bigwedge_{v \in V} X_{0,v} = \alpha_0(v) \qquad (\varphi_{init})$$

- The final marking M_F is respected:

$$\textstyle\bigwedge_{p \in P} M_{n,p} = M_F(p) \qquad (\varphi_{final})$$

- Transitions correspond to transition firings in the DPN:

$$\textstyle\bigwedge_{1 \leq i \leq n} 1 \leq S_i \leq |T| \qquad (\varphi_{trans})$$

  In contrast to [**?**], no constraints are needed to express that at every instant exactly one transition occurs, since the value of S_i is unique.

- Transitions are enabled when they fire:

$$\textstyle\bigwedge_{1\leq i\leq n} \bigwedge_{1\leq j\leq |T|} (S_i = j) \to \bigwedge_{p\,\in\, {}^\bullet t_j} M_{i-1,p} \geq |{}^\bullet t_j|_p \qquad (\varphi_{enabled})$$

  where $|{}^\bullet t_j|_p$ denotes the multiplicity of p in the multiset ${}^\bullet t_j$.
- We encode the token game:

$$\bigwedge_{1\leq i\leq n} \bigwedge_{1\leq j\leq |T|} (S_i = j) \to \bigwedge_{p\,\in\, P} M_{i,p} - M_{i-1,p} = |t_j{}^\bullet|_p - |{}^\bullet t_j|_p \qquad (\varphi_{mark})$$

  where $|t_j{}^\bullet|_p$ is the multiplicity of p in the multiset $t_j{}^\bullet$.
- The transitions satisfy the constraints on data:

$$\bigwedge_{1\leq i< n} \bigwedge_{1\leq j\leq |T|} (S_i = j) \to \mathit{guard}(t_j)\chi \wedge \bigwedge_{v\notin \mathit{write}(t_j)} X_{i-1,v} = X_{i,v} \qquad (\varphi_{data})$$

  where the substitution χ uniformly replaces V^r by X_{i-1} and V^w by X_i.
- The encoding of the data 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 Remark **??**. 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:

$$\begin{aligned}
[P_L] &= 1\\
[P_M]_j &= \mathit{ite}(S_j = 1, c_w(t_1), \dots \mathit{ite}(S_j = |T|-1, c_w(t_{|T|-1}), c_w(t_{|T|}))\dots)\\
[P_=]_{i,j} &= \mathit{ite}(S_j = b_i, \sum_{v\in \mathit{write}(b_i)} \mathit{ite}(\alpha_i(v) = X_{i,v}, 0, 1), \infty)
\end{aligned}$$

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

$$\begin{gathered}
\delta_{0,0} = 0 \qquad \delta_{i+1,0} = [P_L] + \delta_{i,0} \qquad \delta_{0,j+1} = [P_M]_{j+1} + \delta_{0,j}\\
\delta_{i+1,j+1} = \min([P_=]_{i+1,j+1} + \delta_{i,j},\ [P_L] + \delta_{i,j+1},\ [P_M]_{j+1} + \delta_{i+1,j})
\end{gathered} \qquad (\varphi_\delta)$$

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

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

(4) Decoding. We obtain a valid process run $\mathbf{f} = f_1, \dots, f_n$ by decoding with respect to ν the variable sets S_i (to get the transitions taken), $M_{i,p}$ (to get the markings), and $X_{i,v}$ (to get the state variable assignments) for every instant i, as described in Step (1). Moreover, we use the known correspondence between edit distance and alignments [**?**] to reconstruct an alignment $\gamma = \gamma_{m,n}$ of $\mathbf{e}$ and $\mathbf{f}$. To that end, consider the (partial) alignments $\gamma_{i,j}$ recursively defined as follows:

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

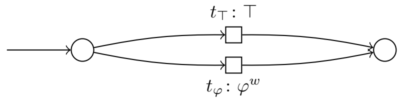
$$\gamma_{i+1,j+1} = \begin{cases} \gamma_{i,j+1} \cdot (e_{i+1}, \gg) & \text{if } \nu(\delta_{i+1,j+1}) = \nu([P_L] + \delta_{i,j+1})\\ \gamma_{i+1,j} \cdot (\gg, f_{j+1}) & \text{if otherwise } \nu(\delta_{i+1,j+1}) = \nu([P_M]_{j+1} + \delta_{i+1,j})\\ \gamma_{i,j} \cdot (e_{i+1}, f_{j+1}) & \text{otherwise}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

To obtain an optimal alignment, we use the following result:

Theorem 1. *Let $\mathcal{N}$ be a DPN,* $\mathbf{e}$ *a log trace and* ν *a solution to* (**??**)*. Then* $\gamma_{m,n}$ *is an optimal alignment for* $\mathbf{e}$*, i.e.,* $\gamma_{m,n} \in \mathit{Align}^{opt}(\mathcal{N}, \mathbf{e})$*.*

3.3 Complexity

In this section we briefly comment on the computational complexity of our approach and the (decision problem version of the) optimal alignment problem. To that end, let a cost function κ be *well-behaved* if it is distance-based and its parameter functions $P_=$, P_M, and P_L are effectively computable and can be defined by linear arithmetic expressions and case distinctions. For $c \in \mathbb{N}$ and a well-behaved cost function κ, let $\textsc{Align}_c$ be the problem that, given a relaxed data-sound DPN and a log trace, checks whether an alignment of cost c with respect to κ exists. For any given DPN $\mathcal{N}$, log trace $\mathbf{e}$ and cost c, the encoding presented in Sec. **??** is used to construct an SMT problem over linear integer/rational arithmetic that is satisfiable if and only if an alignment of cost c exists. The size of such an encoding is polynomial in the size of the DPN and the length of the log trace. Thus, since satisfiability of the relevant class of SMT problems is in NP [**?**], our approach to decide $\textsc{Align}_c$ is in NP. In contrast, the approach presented in [**?**,**?**] is exponential in the length of the log trace. Moreover, $\textsc{Align}_c$ is NP-hard since it is easy to reduce satisfiability of a boolean formula (SAT) to $\textsc{Align}_0$. Hence, all in all $\textsc{Align}_c$ is NP-complete. Given a boolean formula φ with variables V, let $\mathcal{N}_\varphi$ be the following DPN:



where φ^w is the formula obtained from φ by replacing all variables $v \in V$ by v^w. 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 edit distance (cf. Remark **??**). Note that $\mathit{Runs}(\mathcal{N}_\varphi)$ contains at most two valid process runs: we have $\mathbf{f}_0 = (t_\top, \varnothing) \in \mathit{Runs}(\mathcal{N}_\varphi)$ and $\kappa(\mathbf{e}, \mathbf{f}_0) = \infty$. If φ is satisfiable by some transition variable assignment β, we also have $\mathbf{f}_1 = (t_\varphi, \beta_w) \in \mathit{Runs}(\mathcal{N}_\varphi)$, where β_w is the assignment such that $\alpha(v) = \beta_w(v^w)$ 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.

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 [**?**], including conformance checking [**?**,**?**]. 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 share the same optimal alignment cost. We do so 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 7 (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, according to the definition, different clusters do not necessarily correspond to different optimal alignment costs. 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 such guards, one can 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 [**?**]. 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 [**?**]. Note, however, that we do not at all restrict the DPNs we consider to 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 **??**), we use $Atoms(c)$ to define 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 $\odot \in \{>, \geq, =\}$, $x \in V^r \cup V^w$ 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 $ats_v = \{v \odot k \mid x \odot k \in Atoms(c)$, 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 Remark **??**, 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 8. *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 $\textsc{dom}(\alpha) = \textsc{dom}(\alpha')$ and for all variables $v \in \textsc{dom}(\alpha)$, either (i) $\alpha(v) = \alpha'(v)$, or (ii) v is restricted to constant comparison and $\alpha(v) \sim_{cc}^{v} \alpha'(v)$.*

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 9 (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,* $\mathbf{e} = e_1, \dots, e_n$*,* $\mathbf{e}' = e'_1, \dots, e'_n$*, and* $e_i \sim_{cc} e'_i$ *for all* i*,* $1 \leq i \leq n$*.*

Example 3. In Ex. **??**, 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 $3 \not\sim_{cc}^{x} 4$, and $\mathbf{e}_4 = \langle(\mathsf{a}, \{x \mapsto 3\}), (\mathsf{b}, \{y \mapsto 2\})\rangle$ satisfies $\mathbf{e}_1 \not\sim_{cc} \mathbf{e}_4$ because 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 Remark **??**: for the alignments

$$\gamma_1 = \begin{array}{|l|l|l|}\hline \mathsf{a} \quad x \mapsto 2 & \mathsf{b} \quad y \mapsto 1 & \quad \gg \\ \hline \mathsf{a} \quad x^w \mapsto 2 & \mathsf{b} \quad y^w \mapsto 1 & \tau \; \dots \\ \hline \end{array} \qquad \gamma_2 = \begin{array}{|l|l|l|}\hline \mathsf{a} \quad x \mapsto 3 & \mathsf{b} \quad y \mapsto 1 & \quad \gg \\ \hline \mathsf{a} \quad x^w \mapsto 3 & \mathsf{b} \quad y^w \mapsto 1 & \tau \; \dots \\ \hline \end{array} \qquad \gamma_3 = \begin{array}{|l|l|l|}\hline \mathsf{a} \quad x \mapsto 4 & \mathsf{b} \quad y \mapsto 1 & \quad \gg \\ \hline \mathsf{a} \quad x^w \mapsto 3 & \mathsf{b} \quad y^w \mapsto 1 & \tau \; \dots \\ \hline \end{array}$$

we have $\kappa_{\mathcal{N}}^{opt}(\mathbf{e}_1) = \kappa(\gamma_1) = 0$ and $\kappa_{\mathcal{N}}^{opt}(\mathbf{e}_2) = \kappa(\gamma_2) = 0$. 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 $\kappa(\gamma_3) = \kappa_{\mathcal{N}}^{opt}(\mathbf{e}_3) = 1$.

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 (Def. **??** and Def. **??**) 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 is of a certain format. To that end, we consider a distance-based cost function κ from Def. **??** and call it *comparison-based* when 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 distance-based cost function in Remark **??**. 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 $P_=((b, \alpha), (t, \beta)) = |\{v \in \textsc{dom}(\alpha) \mid \alpha(v) \neq \beta(v^w)\}|$, 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 (sketch). We prove that if $\mathbf{e}_1$ has an alignment γ_1 with cost $\kappa(\gamma_1) = \delta(\mathbf{e}_1, \mathbf{f}_1)$, where $\mathbf{f}_1 = \gamma_1|_M$, 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, if $|\mathbf{e}_1| = |\mathbf{e}_2| = m$ and $|\mathbf{f}_1| = n$, we show by induction on $m + n$ that there is some $\mathbf{f}_2$ such that $|\mathbf{f}_2| = n$, $\delta(\mathbf{e}_1, \mathbf{f}_1) = \delta(\mathbf{e}_2, \mathbf{f}_2)$, and $\mathbf{f}_1$ and $\mathbf{f}_2$ result in state variable assignments that are equivalent up to $\sim_{cc}$. The inductive step works by a case distinction on the cases in Def. **??**, exploiting the properties of a comparison-based cost function. The full proof is in [**?**].

An interesting byproduct of the constructive proof of Theorem **??** (see [**?**]) is that given $\gamma \in \mathit{Align}^{opt}(\mathcal{N}, \mathbf{e})$, 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 Implementation and Experiments

We now report on the DPN conformance checking tool `cocomot`, a proof-of-concept implementation based on the encoding in Sec. **??**. We focus on the implementation, some optimizations, and experiments on benchmarks from the literature. The source code together with related datasets are publicly available on the tool webpage: https://github.com/bytekid/cocomot.

Implementation. Our `cocomot` prototype is a Python command line script: it takes as input a DPN (as `.pnml` file) and a log (as `.xes`) and computes the optimal alignment distance, using the standard cost function from Remark **??**, for every trace in the log. In verbose mode, it additionally prints an optimal alignment. To reduce effort, `cocomot` first preprocesses the log to a sublog of unique traces, and second applies trace clustering as described in Sec. **??** to further partition the sublog into equivalent traces. The conformance check is then run for one representative from every equivalence class.

`cocomot` uses `pm4py` (https://pm4py.fit.fraunhofer.de/) to parse traces, and employs the SMT solver Yices 2 [**?**] , or alternatively Z3 [**?**], as backend solver. Instead of writing the formulas to files, we use the bindings provided by the respective Python interfaces. Since Yices 2 has no optimization built-in, 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 prune the search space, we modified the encoding presented in Sec. **??**. We report here on the four most effective changes. (1) We perform a reachability analysis in a preprocessing step. This allows us to restrict the range of transition variables S_i in (**??**), as well as the cases $S_i = j$ in (**??**) and (**??**) to those that are actually reachable. Moreover, if a data variable

$v \in V$ will never be written in some step i, $1 \leq i \leq n$, because no respective transition is reachable, we set $X_{i,v}$ identical to $X_{i-1,v}$ to reduce the number of variables. (2) If the net is 1-bounded, the marking variables $M_{i,p}$ are typed as boolean rather than integer, similar to [**?**]. (3) As $\delta_{m,n}$ is minimized, the equation of the form $\delta_{i+1,j+1} = min(e_1, e_2, e_3)$ in (**??**) can be replaced by inequalities $\delta_{i+1,j+1} \geq min(e_1, e_2, e_3)$. The latter is equivalent to $\delta_{i+1,j+1} \geq e_1 \vee \delta_{i+1,j+1} \geq e_2 \vee \delta_{i+1,j+1} \geq e_3$, which is processed by the solver much more efficiently since it avoids an if-then-else construct. (4) Several subexpressions are replaced by fresh variables (in particular when occurring repeatedly) – this is empirically known to positively affect performance.

Some of the data sets described below contain data variables of non-numeric types, namely boolean and string variables. The encoding represents the former by boolean SMT variables; and the latter by integer variables, encoding the string literals in the model as distinct natural numbers (cf. [**?**, p. 87]).

Experiments. We tested `cocomot` on three data sets used in earlier work [**?**,**?**], which have also been made publicly available on the tool's webpage. All experiments were run single-threaded on a 12-core Intel i7-5930K 3.50GHz machine with 32GB of main memory.

The first data set contains 150370 traces (35681 unique) of *road fines* issued by the Italian police. By trace clustering, the log reduces to 4290 non-equivalent traces. In 268 seconds, `cocomot` computes optimal alignments for all traces in this set, spending 13% of the computation time on parsing the log, 1.5% on trace clustering, 13% on the generation of the encoding, and the rest on SMT solving.[2] When omitting the clustering preprocessor, `cocomot` requires about 30 minutes to process all 35681 traces. We note some data about the model and log. The maximal length of a trace is 20, and its average alignment cost is 1.5. The average time spent on a trace is 0.1 seconds. The process model has less than 20 transitions, and at most one token around at any point in time.

The second data set contains 100000 traces (4047 unique) of a *hospital billing* process. Trace clustering slightly reduces the number of non-equivalent traces to 4039. For 3392 traces `cocomot` finds an optimal alignment, while SMT timeouts occur for the remaining, very long traces (the maximal trace length is 217).

The third data set is about *sepsis*, and contains 1050 unique (and non-equivalent) traces. For 1006 traces `cocomot` finds an optimal alignment, while it times out for the remaining, very long traces (the maximal trace length is 185).

For the experiments described above we used Yices (with an SMT timeout of 10 minutes) as Z3 turned out to be considerably slower: checking conformance of the road fine log using Z3 (with its built-in minimization routine) takes more than two hours. Across all data sets, very little time is spent on generating the encoding, while the vast majority of time is used for SMT solving. Indeed the trace clustering approach suggested in this paper considerably improves the performance over the presented data sets, as already pointed out for the one of road fines.

[2] Notice that in this paper we do not fix the exact algorithm for trace clustering, which in our implementation is achieved by exhaustively comparing log traces.

6 Discussion

In this section we outline how the CoCoMoT approach, due to its modularity, readily lends itself to further tasks related to the analysis of data-aware processes. The **multi-alignment** problem asks, given a DPN $\mathcal{N}$ and a set of log traces $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$, to find a process run $\mathbf{f} \in \mathcal{P}_\mathcal{N}$ such that $\sum_{i=1}^{n} \kappa(\gamma_i)$ is minimal, where γ_i is a minimal-cost alignment of $\mathbf{e}_i$ and $\mathbf{f}$ for all i, $1 \leq i \leq n$ [**?**].[3] Our encoding can solve such problems by combining n copies of the distance variables and their defining equations (**??**) with (**??**)–(**??**), and minimizing the above objective. Generalizing alignments, multi-alignments are of interest for their own sake, but also useful for further tasks, described next.
Anti-alignments were introduced to find model runs that deviate as much as possible from a log, e.g., for precision checking [**?**]. For a set of traces $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$, the aim is to find $\mathbf{f} \in \mathcal{P}_\mathcal{N}$ of bounded length such that $\sum_{i=1}^{n} \kappa(\gamma_i)$ is *maximal*, with γ_i as before. Using our encoding, this can be done as in the multi-alignment case, replacing minimization by maximization.
Trace clustering was studied as a method to partition event logs into more homogeneous sub-logs, with the hope that process discovery techniques will perform better on the sub-logs than if applied to the original log [**?**,**?**]. Chatain *et al* [**?**,**?**] propose trace clustering based on multi-alignments. In the same fashion, our approach can be used to partition a log of DPN traces.
Our approach can also be used for **model repair** tasks: given a set of traces, we can use multi-alignments to minimize the sum of the trace distances, while replacing a parameter of the DPN by a variable (e.g., the threshold value in a guard). From the satisfying assignment we obtain the value for this parameter that fits the observed behavior best. As constraints (**??**)–(**??**) symbolically describe a process run of bounded length, our encoding supports **bounded model checking**. Thus we could also implement *scenario-based* conformance checking, to find for a given trace the best-matching process run that satisfies additional constraints, such as that certain data values are not exceeded.
Finally, but crucially, the **main advantage of SMT** is that it offers a multitude of *background theories* to capture the data manipulated by the DPN, and to express sophisticated cost functions. The approach by Mannhardt *et al* [**?**,**?**] needs to restrict guards of DPNs to linear arithmetic expressions in order to use the MILP backend. In our approach, the language of guards may employ *arbitrary* functions and predicates from first-order theories supported by SMT solvers (e.g., uninterpreted functions, arrays, lists, and sets). For example, the use of (relational) predicates would allow to model structured background information, and possibly even refer to full-fledged relational databases from which data injected in the net are taken, following the SMT-based approaches as in [**?**,**?**]. Moreover, the background theory allows to express sophisticated cost functions as in Def. **??** with the following parameters (inspired by [**?**]): $P_=((b,\alpha),(t,\beta)) = |\{v \in \mathit{write}(t) \mid \neg R(\alpha(v)), R(\beta(v^w))\}|$ if $b = \ell(t)$, for some relation R from a database DB: in this way, $P_=$ counts the number of written

[3] Instead of the sum, also other aggregation functions can be used, e.g., maximum.

variables whose values in the model run are stored in the relation R from DB whereas their values in the log trace are not.

7 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 (multi-perspective) 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, as shown in Sec. **??**. 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. For example, one can capture more sophisticated cost functions involving background knowledge coming from additional data sources (in line of [**?**]) or correctly addressing privacy related aspects (where one typically needs to employ uninterpreted functions). All this is left for future work, but motivates once again the use of SMT.

Acknowledgments This research has been partially supported by the UNIBZ projects *SMART-APP*, *REKAP*, *VERBA* and *DUB*.