Approximating Multi-Perspective Trace Alignment Using Trace Encodings[⋆]

Alessandro Gianola, Jonghyeon Ko, Fabrizio Maria Maggi, Marco Montali, and Sarah Winkler

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

Abstract. Alignments provide sophisticated diagnostics that pinpoint deviations in a trace with respect to a process model. One crucial aspect is to consider, in the alignment task, not only the control flow perspective but also other sources of information available in event logs like data payloads. However, the combination of these dimensions makes the problem of multi-perspective trace alignment highly challenging since the number of traces accepted by the model is typically infinite. In this paper, we address this problem by proposing an approximate approach to alignment computation: instead of computing the optimal alignments based on the complete knowledge about a process trace available in the log, we perform approximate alignments based on lossy trace encodings that only consider certain information about the trace. The advantage of this approach is twofold. First, the trace alignment task is much faster. Second, the analyst can choose what type of information is relevant for computing the alignments by selecting the encodings that represent a trace based on that information. Our experiments show that the approximate approach is faster than the optimal one and, for encodings sufficiently rich, able to provide accurate results.

Keywords: Conformance checking · Trace Encoding · Multi-Perspective process mining · SMT

1 Introduction

Conformance checking is one of the central tasks of process mining [2]. Its main goal is to compare a reference process model with an event log containing actual process executions to understand whether such concrete executions deviate from the model. Within the family of conformance checking techniques, a prominent approach is to measure and explain deviations through *alignments*.

An alignment is intuitively a sequence of pairs, called moves, consisting of an event from the log and a transition in the process model. Given a suitable function that assigns cost to moves, an optimal alignment is an alignment whose

[⋆] This research has been partially supported by the Italian Ministry of University and Research (MUR) under the PRIN project PINPOINT Prot. 2020FNEB27, and by the Free University of Bozen-Bolzano with the ADAPTERS and CAT projects.

overall cost is minimal. This is notoriously challenging to compute, as it requires to solve an optimization problem over a finite portion of the space of model traces, where the portion to be considered depends on the length of the trace under scrutiny, and comes with the additional computational burden of computing trace distances. A plethora of techniques have been therefore defined to tackle the problem in an optimal [2] or approximate [1] way.

In the alignment task, however, not only the control flow perspective is crucial, but also other sources of information from event logs like data payloads. This has led to a recent series of approaches to tackle data-aware conformance checking [3, 16, 17, 4]. There, Data Petri Nets (DPNs) [16, 10] are the reference model to represent a process that accounts for control-flow and data, with process variables that can carry data values of different types.

The standard way for measuring the distance between a log trace and a DPN is to compute *optimal* alignments, based on a notion of distance that tackles at once the events, their orderings, and their data payloads. However, in the presence of models with rich data and control flow perspectives, computing optimal alignments can be extremely costly in terms of performance. This is also due to the fact that even by bounding the maximum length of model traces, the number of them is usually infinite, because of data. This calls for sophisticated techniques to handle the data component.

In this paper, we propose an alternative approach for data-aware conformance checking, which *approximates* optimal alignments based on machine learning techniques, in particular lossy trace encodings [12, 14]. To this end, we do not work directly on models, but on sets of abstract traces. Roughly, our approach proceeds in three stages:

(1) We build a set $\mathcal{T}$ of *abstract traces*, i.e., classes of traces representative for all possible behaviors of the process. For this set, we propose two possibilities: (1a) For the class of DPNs whose transition guards are only variable-to-constant comparisons, we show how all possible behaviors up to a bounded length can be succinctly represented by a *finite* set of abstract traces. (1b) For DPNs with numeric variables but more complex guards, such a representation is in general not possible. In this case, our approach can be applied by taking as $\mathcal{T}$ simply a set of "happy paths", i.e., traces that are considered representative of the process behavior (e.g., obtained by collecting sufficiently many cases).

(2) We use machine learning techniques known as *encodings* to represent $\mathcal{T}$ as a set of vectors in a vector space. Here, different encodings can be employed to preserve from the abstract traces the information that is deemed most relevant for the conformance checking task. The result, called *behavior encoding space* is a compact numeric representation of all relevant behaviors.

(3) In order to check the conformance of a concrete trace, we apply the encoding from the previous stage to it, obtaining a vector $\mathbf{X}$, and subsequently compute the k vectors from the behavior encoding space that are closest to $\mathbf{X}$, using a kNN-based method. From these vectors, we can then get back the abstract traces that are considered closest to the input trace.

Note that the class of DPNs in (1a) has been found expressive and useful in practice, and is amenable to automatic discovery techniques [11, 8]. Moreover, it is known that the process run in an optimal alignment can be upper-bounded in length in terms of the given trace [3], and $\mathcal{T}$ is a complete set of representatives. Therefore, the conformance checking task can be reformulated as the task to select a suitable abstract trace from $\mathcal{T}$, without loss of precision, which justifies the subsequent approximation approach in stages (2) and (3).

We experimentally validate our approach for both settings (1a) and (1b), comparing the results with the conformance checker CoCoMoT [3]. These experiments show that abstract traces (1a) together with smart trace encodings and vector space distance measures allow for a good approximation of the optimal alignments, in terms of precision and similarity. Moreover, we show that even when using a plain trace set as a representation of the process behaviors (1b), the encoding-based approach approximates the optimal one with high precision.

The remainder of this paper is structured as follows: We first recall background about DPNs and alignments (Section 2). Then, we present our notions of trace-based distance function and abstract traces (Section 3), and subsequently, trace encodings (Section 4). We evaluate our approach in Section 5. Finally, we discuss related work (Section 6) and conclude (Section 7).

2 Background and Preliminaries

We use a restricted but significant class of Data Petri nets (DPNs) for modeling multi-perspective processes, adopting the same formalization as in [3, 16].

Let V be a set of *process variables*, each with a type and an associated domain: integers (`int`), or rationals (`rat`).[1] 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 read and written by process activities, as explained below. Based on these, we define constraints according to the grammar for c:

$$c ::= v_z \odot z \mid v_r \odot q \mid c \wedge c$$

where $v_z \in V_{\texttt{int}}$, $z \in \mathbb{Z}$, $v_q \in V_{\texttt{rat}}$, and $q \in \mathbb{Q}$, and $\odot$ is in $\{\geq, \leq, >, <, =\}$. Our constraints are thus more restrictive than in other sources [3], permitting only variable-to-constant comparisons, but this will allow us to define precise abstract traces. The set of constraints over variables V is denoted $\mathcal{C}(V)$; they are used for read and write operations in process activities.

Definition 1 (DPN). *A tuple $\mathcal{N} = (P, T, F, \ell, A, V, guard)$ is a* Petri net with data *(DPN), 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 denoting silent transitions;*

[1] Booleans and strings can be encoded as integers, as commonly done [3, 17].

- *V is a set of typed process variables; and*
- *$guard\colon T \to \mathcal{C}(V^r \cup V^w)$ is a guard assignment; for $t \in T$ with $\ell(t) = \tau$ we assume that $guard(t)$ does not use variables in V^w.*

Transition guards serve to simultaneously read and write variables. For instance, a transition with guard $(x^r > 3)$ can only be taken if the current value of variable x is greater than 3 (the superscript r indicates that the guard is on the current, or *read*, variable). On the other hand, a guard $(x^w > 1) \wedge (x^r < 4)$ requires that the current value of x is smaller than 4 and, at the same time, it non-deterministically *writes* to x a new value that is greater than 1 (superscripts w refer to *written* values). Note that transition guards with disjunctions can be simulated by employing multiple transitions between the same places.

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 $write(t) = \{v \mid v^w \in \mathcal{V}ar(guard(t))\}$.

To represent the current values of variables, we consider a *state variable assignment*, i.e., a (possibly partial) function α that assigns a value (of the right type) to each variable in V. We denote by $\textsc{dom}(\alpha)$ the domain of α. A *state* in a DPN $\mathcal{N}$ is a pair (M, α) constituted by a marking $M\colon P \to \mathbb{N}$ for the underlying Petri net (P, T, F, ℓ), plus a total 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 fix one state (M_I, α_0) as *initial*, where M_I is the initial marking of the underlying Petri net 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 the form (M_F, α_F) for some α_F.

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. Def. 2).

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 there exists a transition variable assignment β such that:*

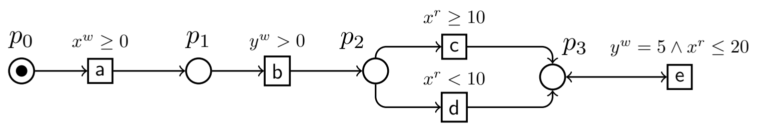
- *$\textsc{dom}(\beta) = \mathcal{V}ar(guard(t))$: β is defined for the variables in the guard;*
- *$\beta(v^r) = \alpha(v)$ for every $v \in read(t)$, i.e., β is as α for read variables;*
- *$\beta \models guard(t)$, i.e., β satisfies the guard; and*
- *$M(p) \geq F(p,t)$ for every $p \in {}^\bullet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

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 $(M, \alpha) \xrightarrow{(t,\beta)} (M', \alpha')$.*

Informally, a transition firing between the current state (M, α) and the next state (M', α') is a couple (t, β) where: *i)* $t \in T$ is a transition that is enabled in the 'token game' sense of standard Petri nets; *ii)* β is a function connecting the values of the read variables (matching the values assigned by α in the current state) to the values of the write variables (matching the values assigned by α' in the next state); *iii)* β satisfies the guard associated to t.

Based on this single-step transition firing, we say that a state (M', α') is *reachable* in a DPN with initial state (M_I, α_0) iff there exists a sequence of valid transition firings of the form $\mathbf{f} = \langle (t_1, \beta_1), \ldots, (t_n, \beta_n) \rangle$ such that $(M_I, \alpha_0) \xrightarrow{(t_1,\beta_1)} \ldots \xrightarrow{(t_n,\beta_n)} (M', \alpha')$. Moreover, such a sequence $\mathbf{f}$ is called a *process run* of $\mathcal{N}$ if $(M_I, \alpha_0) \xrightarrow{\mathbf{f}} (M_F, \alpha_F)$ for some α_F, i.e., if the run leads to a final state. As in [3, 17], we restrict to DPNs where a final state is reachable. We denote the set of transition firings of $\mathcal{N}$ by $\mathcal{F}(\mathcal{N})$, and the set of process runs by $\mathit{Runs}(\mathcal{N})$.

Example 1. Let $\mathcal{N}$ be as shown below (with initial marking $[p_0]$ and final marking $[p_3]$). $\mathit{Runs}(\mathcal{N})$ contains, e.g., $\langle (\mathsf{a}, \{x^w \mapsto 12\}), (\mathsf{b}, \{y^w \mapsto 1\}), (\mathsf{c}, \{x^r \mapsto 12\}) \rangle$ and $\langle (\mathsf{a}, \{x^w \mapsto 1\}), (\mathsf{b}, \{y^w \mapsto 1\}), (\mathsf{d}, \{x^r \mapsto 1\}) \rangle$, for $\alpha_0 = \{x, y \mapsto 0\}$.



Given a set S, we denote S^* as the set of sequences of elements from S, and $\mathcal{M}(S)$ as the set of multisets over S. For a set A of activity labels, an *event* is a pair (b, α) for $b \in A$ and α a (typically partial) state variable assignment, associating values to variables in V.

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 $\mathcal{E}$.*

Conformance checking aims at constructing an *alignment* of a given log trace $\mathbf{e}$ wrt the DPN $\mathcal{N}$, by matching events in the log trace against transitions firings in a process run. Since not every event can typically be put in correspondence with a transition firing, a "skip" symbol $\gg$ is used. Let $\mathcal{E}^{\gg} = \mathcal{E} \cup \{\gg\}$ 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*. A move (e, f) is a *log move* if $e \in \mathcal{E}$ and $f = \gg$; a *model move* if $e = \gg$ and $f \in \mathcal{F}(\mathcal{N})$; and *synchronous move* if $(e, f) \in \mathcal{E} \times \mathcal{F}(\mathcal{N})$.

For a sequence of moves $\gamma = (e_1, f_1), \ldots, (e_n, f_n)$, the *log projection* $\gamma|_L$ of γ is the maximal subsequence of $e_1, \ldots, e_n$ in $\mathcal{E}^*$, and the *model projection* $\gamma|_M$ of γ is the maximal subsequence of $f_1, \ldots, f_n$ in $\mathcal{F}(\mathcal{N})^*$ (i.e., without $\gg$ symbols).

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 and γ_2 below are possible complete alignments of the log trace $\mathbf{e} = \langle (\mathsf{a}, \{x \mapsto 2\}), (\mathsf{b}, \{y \mapsto 1\}), (\mathsf{d}, \emptyset) \rangle$ wrt the DPN from Ex. 1:

$\gamma_1 =$

a	$x \mapsto 2$	b	$y \mapsto 1$	d
a	$x^w \mapsto 2$	b	$y^w \mapsto 1$	d

$\gamma_2 =$

a	$x \mapsto 2$	b	$y \mapsto 1$	d	$\gg$
a	$x^w \mapsto 12$	b	$y^w \mapsto 1$	$\gg$	c

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

Definition 5 (Cost). *Given $\mathcal{N}$, $\mathbf{e}$ and $\gamma = (e_1, f_1), \dots, (e_n, f_n) \in Align(\mathcal{N}, \mathbf{e})$, 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 $\gamma' \in Align(\mathcal{N}, \mathbf{e})$ with $\kappa(\gamma') < \kappa(\gamma)$.*

For instance, using the standard cost function from [3, Def. 6] and the alignments in Example 2, we would have $\kappa(\gamma_1) = 0$ and $\kappa(\gamma_2) = 3$. We denote the cost of an optimal alignment for $\mathbf{e}$ wrt $\mathcal{N}$ by $\kappa^{opt}_{\mathcal{N}}(\mathbf{e})$.

3 Trace-Based Conformance Checking

In this section, we develop notions to perform (approximated) conformance checking on the basis of trace classes rather than the model itself.

Abstract Trace. In order to simulate the conformance checking procedure, we first extract a set of abstract traces that are representative for the given DPN. To that end, we use the following definitions, for a DPN with data variables V. A *variable range assignment* ι is a (possibly partial) function from the set of data variables V to intervals, such that for all $v \in V$, $\iota(v)$ is of the form $[l, u]$, $]l, u]$, $[l, u[$, or $]l, u[$, for l, u numeric values in $\textsc{dom}(v)$. Then, given the set T of transitions, an *abstract event* is a pair (t, ι), where $t \in T$ and ι is a variable range assignment, and an *abstract trace* is a sequence of abstract events.

A trace $\mathbf{e} = \langle e_1, \dots, e_n \rangle$ *matches* an abstract trace $\mathbf{f} = \langle f_1, \dots, f_m \rangle$ if $m = n$ (same length); and for all $1 \leq i \leq n$, if $e_i = (l, \alpha)$ with corresponding $f_i = (t, \iota)$, it holds that $l = \ell(t)$, i.e., they have the same label; and $\textsc{dom}(\alpha) = \textsc{dom}(\iota)$, and for all $v \in \textsc{dom}(\alpha)$, the value $\alpha(v)$ is in the interval $\iota(v)$. Finally, a finite set of abstract traces $\mathcal{T}$ is *representative* for a DPN $\mathcal{N}$ *up to length* k if for every trace $\mathbf{e}$ with $|\mathbf{e}| \leq k$ and $\kappa^{opt}_{\mathcal{N}}(\mathbf{e}) = 0$, the trace $\mathbf{e}$ matches some $\mathbf{e}_a \in \mathcal{T}$.

Our approach exploits that for a given log trace and DPN, the length of a process run in an optimal alignment can be bounded upfront. More precisely:

Lemma 1 ([5, Lem. 2]). *Given a log trace $\mathbf{e}$ of length n and a DPN $\mathcal{N}$, there is a computable function $maxlen(\mathcal{N}, n)$ s.t. $\mathbf{e}$ has an optimal alignment γ wrt the standard cost function s.t. $\gamma|_M$ has length at most $maxlen(\mathcal{N}, n)$.*

Let a trace $\mathbf{e}$ *correspond* to a process run $\mathbf{f}$ if, for $\langle f_1, \dots, f_n \rangle$ the subsequence of non-silent transitions in $\mathbf{f}$, $\mathbf{e} = \langle e_1, \dots, e_n \rangle$, and for all i, $1 \leq i \leq n$, if $f_i = (t, \beta)$ then $e_i = (l, \alpha)$ such that $\ell(t) = l$ and $\alpha(v) = \beta(v^w)$ for all $v \in \textsc{dom}(\beta)$.

Using this notion, we get the following useful corollary of Lemma 1:

Theorem 1. *Let a set of abstract traces T be representative for a DPN $\mathcal{N}$ up to $maxlen(\mathcal{N}, n)$. Then, for every trace $\mathbf{e}$ with $|\mathbf{e}| \leq n$, there is an optimal alignment γ such that $\mathcal{T}$ has an abstract trace $\mathbf{e}_a$ that corresponds to $\gamma|_M$.*

This means that, in order to find the process run associated with an optimal alignment for a given log trace, it suffices to consider abstract traces in a set of representative abstract traces $\mathcal{T}$.

Computing representative sets of abstract traces. We now show one concrete method to compute a representative set $\mathcal{T}$ for a DPN $\mathcal{N}$.

1. For a given k, we enumerate all transition sequences of $\mathcal{N}$ from the initial to a final marking that have length at most k, and select from these the subset $\mathcal{T}'$ of sequences which correspond to actual process runs. This filtering can be done, e.g., by checking with an SMT encoding (as done in CoCoMoT) whether the sequence of transitions is satisfiable.
2. For every sequence $\langle t_1, \dots, t_n\rangle$ in $\mathcal{T}'$ and $1 \leq i \leq n$, we define a trace range substitution ι_i as follows. First, a variable $v \in V$ is in $dom(\iota_i)$ iff $v \in write(t_i)$. For such v, let j (s.t. $i < j \leq n$) be the smallest number such that either $j = n$, or $v \in write(t_{j+1})$. Thus, the value of v written in t_i persists until instant j. All guards in $t_i, \dots, t_j$ are, by construction, conjunctions of variable-to-constant comparisons. Let L be the greatest lower bound set for v, and U the smallest upper bound set for v in $t_i, \dots, t_j$; if no respective bound occurs, $L = -\infty$ or $U = \infty$. We fix $\iota_i(v)$ to either $[L, U]$, $[L, U[$, $]L, U]$ or $]L, U[$, depending on whether L and U are included or not.

   Finally, $\mathcal{T}$ consists of all $\langle(t_1, \iota_1)\dots,(t_n, \iota_n)\rangle$ such that $\langle t_1, \dots, t_n\rangle$ is in $\mathcal{T}'$.

It can be checked that the set $\mathcal{T}$ constructed in this way is indeed a representative set of abstract traces.

Example 3. For $\mathcal{N}$ as in Example 1, a representative set of abstract traces up to length 4 consists of $\langle(\mathsf{a}, x \mapsto [0, 10[), (\mathsf{b}, y \mapsto]0, \infty[), (\mathsf{d}, \emptyset), (\mathsf{e}, y \mapsto [5, 5])\rangle$, $\langle(\mathsf{a}, x \mapsto [10, \infty[), (\mathsf{b}, y \mapsto]0, \infty[), (\mathsf{c}, \emptyset)\rangle$, $\langle(\mathsf{a}, x \mapsto [0, 10[), (\mathsf{b}, y \mapsto]0, \infty[), (\mathsf{d}, \emptyset)\rangle$, and $\langle(\mathsf{a}, x \mapsto [10, 20]), (\mathsf{b}, y \mapsto]0, \infty[), (\mathsf{c}, \emptyset), (\mathsf{e}, y \mapsto [5, 5])\rangle$.

Measuring the Distance between Two Traces. In conformance checking, one usually measures the distance between a trace and a model run. Here, we approximate such a cost by taking the distance between two traces:

Definition 6. *For log traces* $\mathbf{e} = \langle e_1, \dots, e_m\rangle$ *and* $\mathbf{e}' = \langle e'_1, \dots, e'_n\rangle$*, the* trace distance $\delta(\mathbf{e}|_i, \mathbf{e}'|_j)$ *is recursively defined for all* $0 \leq i \leq m$ *and* $0 \leq j \leq n$*:*

$$\delta(\epsilon, \epsilon) = 0 \quad \delta(\mathbf{e}|_{i+1}, \epsilon) = Q_L(e_{i+1}) + \delta(\mathbf{e}|_i, \epsilon) \quad \delta(\epsilon, \mathbf{e}'|_{j+1}) = Q_L(e'_{j+1}) + \delta(\epsilon, \mathbf{e}'|_j)$$

$$\delta(\mathbf{e}|_{i+1}, \mathbf{e}'|_{j+1}) = \min \begin{cases} Q_=(e_{i+1}, e'_{j+1}) + \delta(\mathbf{e}|_i, \mathbf{e}'|_j) \\ Q_L(e_{i+1}) + \delta(\mathbf{e}|_i, \mathbf{e}'|_{j+1}) \\ Q_L(e'_{j+1}) + \delta(\mathbf{e}|_{i+1}, \mathbf{e}'|_j) \end{cases}$$

Here $Q_=$ and Q_L are two *penalty functions*, the former for synchronous moves and the latter for asynchronous moves in one of the logs. These penalties can be instantiated in different ways. We adapt the *standard cost function* [17, 3] to two traces and set

$$Q_L(b, \alpha) = 1 \quad Q_=((b, \alpha), (b', \alpha')) = \begin{cases} |\{v \in \textsc{dom}(\alpha) \mid \alpha(v) \neq \alpha'(v)\}| \text{ if } b = b' \\ \infty, \text{ otherwise} \end{cases}$$

For instance, for the log trace $\mathbf{e} = \langle(\mathsf{a}, \{x \mapsto 2\}), (\mathsf{b}, \{y \mapsto 1\}), (\mathsf{d}, \emptyset)\rangle$ from Example 2 and $\mathbf{e}' = \langle(\mathsf{a}, \{x \mapsto 12\}), (\mathsf{b}, \{y \mapsto 1\}), (\mathsf{c}, \emptyset)\rangle$ (matching the process run of γ_2), we have $\delta(\mathbf{e}, \mathbf{e}) = 0$ and $\delta(\mathbf{e}, \mathbf{e}') = 3$.

4 Approximating Alignments with Trace Encodings

In this section, we introduce an encoding approach for abstract traces (Section 4.1) and then a kNN-based method to obtain an approximate solution of the trace alignment problem (Section 4.2).

4.1 Encodings for Abstract Traces

To have a lossy representation of abstract traces, we use an encoding $\mathcal{E} : \mathcal{T} \to \mathbb{R}^n$ with $n \in \mathbb{N}$ that transforms each abstract trace into a vector of the n-dimensional Euclidean space $\mathbb{R}^n$. We call the resulting set of vectors $\mathcal{E}(\mathcal{T})$ *behavior encoding space*. The literature provides encoding functions to represent strings [6], which we can directly employ for representing the control-flow dimension of the abstract traces. For example, the *boolean* encoding represents a trace through a vector of boolean values each indicating if a specific activity label is present or not in the trace. The *frequency-based* encoding, instead of boolean values, represents the control flow in a trace with the frequency of each activity label in the trace. Another way of encoding a trace is by taking into account also information about the order in which events occur in it, as in the *simple index* encoding. Here, each dimension corresponds to a position in the trace and its value is a numeric code representing the activity label occurring in that position.

A more complex control-flow encoding is obtained by associating each dimension in $\mathbb{R}^n$ to a different sub-trace of size p (i.e., p-grams). Each feature of this encoding represents how frequently and "compactly" a sub-trace appears in the trace of interest. For simplicity, we consider 2-grams, but the following can be easily generalized to p-grams. Given an abstract trace $\mathbf{e}_a$, we employ a simplified version of the encoding from [14] to transform $\mathbf{e}_a$ into a vector in $\mathbb{R}^n$ in two steps. First, we identify all 2-grams occurring in all the abstract traces in $\mathcal{T}$. Then, we construct a vector in $\mathbb{R}^n$ where each dimension of the vector is a real number representing the frequency and the compactness of a specific 2-gram. E.g., for the 2-gram ab, this value is given by $\mathcal{E}_{ab}(\mathbf{e}_a) = \sum_{1 \leq i \leq |\mathbf{e}_a|-1} \lambda^i [\Lambda^i]_{ab}$, where $[\Lambda^i]_{ab} \to 0, 1$ indicates the occurrences of ab at distance i in $\mathbf{e}_a$, and $\lambda \in]0, 1]$ is a parameter that represents the penalty provided for less compact 2-grams. Lower values of λ correspond to a higher distance between the numeric representation of more compact 2-grams wrt less compact ones (if λ is equal to 1 the compactness has no influence on the feature values of this encoding).

Example 4. Table 1 shows the 2-gram encodings of some traces over the activity labels $A = \{a, b, c\}$. Trace cb has only one non-zero dimension $\mathcal{E}_{cb}(cb) = \lambda$; trace caa has two non-zero dimensions: $\mathcal{E}_{ca}(caa) = \lambda + \lambda^2$ (ca occurs once with c and a at distance 1, i.e., $\underline{ca}a$, and once with c and a at distance 2, i.e., $\underline{c}a\underline{a}$), and $\mathcal{E}_{aa}(caa) = \lambda$ (a is repeated after a single step in the trace only once, i.e., $c\underline{aa}$).

	aa	*ab*	*ac*	*ba*	*bb*	*bc*	*ca*	*cb*	*cc*
caba	λ^2	λ	0	λ	0	0	$\lambda+\lambda^3$	λ^2	0
caa	λ	0	0	0	0	0	$\lambda+\lambda^2$	0	0
cb	0	0	0	0	0	0	0	λ	0

Table 1. encoding of traces *caba*, *caa* and *cb*.

In addition to control-flow, abstract traces include variable range assignments from the set of data variables V linked to each activity. For instance, if variable $\mathit{Amount} \in [10, 20[$ triggers activity b after a in $\mathbf{e}_a$, then, the abstract event corresponding to b contains interval $[10, 20[$ for variable *Amount*. All the possible intervals for a variable (derived from all the abstract traces for a given DPN) have to be transformed into specific values to apply existing methods for trace encoding like the ones in [12]. To do so, we encode each variable v of each abstract event e_i using a feature space of *interval features* that are boolean features composed of v and a possible interval I. In this way, for each abstract event e_i, we have a set of interval features with values $\mathbb{D}_{e_i,v,I}$ given by:

$$\mathbb{D}_{e_i,v,I} = \begin{cases} 1 & \text{if } \iota(v) \subseteq I, \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\iota(v)$ is the variable range assignment in e_i. As an example, given a set of possible intervals for variable *Amount* $\{[0, 10[,\ [10, 20[,\ [10, 30[\}$, an abstract event $e_i = (l, \mathit{Amount} = [16, 24])$ is encoded by three boolean features: $\mathbb{D}_{e_i,\mathit{Amount}:[0,10[} = 0$, $\mathbb{D}_{e_i,\mathit{Amount}:[10,20[} = 0$, and $\mathbb{D}_{e_i,\mathit{Amount}:[10,30[} = 1$.

We use these boolean features as "event attributes" of each abstract event in an abstract trace. In this way, we can directly apply existing trace encodings [12] to abstract traces. These encodings can include control flow features (that can range from a simple boolean encoding to more complex encodings like the one based on p-grams), and data-flow features (derived from the interval features introduced above). We point out here that the choice of the encoding is part of the analysis. The analyst can select the information that is more relevant for computing the alignments depending on the specific process and the specific context in which the alignments are computed. The encoded abstract traces in the behavior encoding space $\mathcal{E}(\mathcal{T})$ are used to identify the top-k alignments of a (concrete) log trace $\mathbf{e}$ as described in the next section.

4.2 Approximating Alignment Computation using kNN

In the alignment problem, we assume to have a set of (abstract) traces $\mathbf{e}_a \in \mathcal{T}$ and a set of non-conforming (concrete) log traces $\mathbf{e}$. The trace alignment task consists in searching the model trace $\mathbf{e}_a^* \in \mathcal{T}$ that is the closest to each log trace according to a given distance/cost function $\delta_{\mathcal{E}} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.

When in the alignment task, together with the control-flow also other perspectives available in event logs like timestamps and data payloads are taken into consideration, this multi-perspective trace alignment becomes a challenging problem. However, if we use encodings to represent model and log traces, we can select the most relevant information needed to compute the alignments and, at the same time, reduce the time needed to perform the task. The log traces and the data space including the possible model traces (i.e., the possible alignments) can be explored, once a log trace to be aligned is given, by using k-nearest neighbors (kNN) algorithms. By using trace encoding, we can compute the encodings of the log trace $\mathcal{E}(\mathbf{e})$ and of all the possible model traces $\mathcal{E}(\mathbf{e}_a)$, and compute their distance using a distance function $\delta_{\mathcal{E}}(\mathbf{e}_a, \mathbf{e}) := \langle \mathcal{E}(\mathbf{e}_a), \mathcal{E}(\mathbf{e}) \rangle$. Then, the trace alignment problem is solved by computing the approximate alignment for an observed log trace $\mathbf{e}$ as $\min \arg_{\mathbf{e}_a} \delta_{\mathcal{E}}(\mathbf{e}_a, \mathbf{e})$. In particular, the approximate alignment(s) can be computed using kNN algorithms that find the k nearest data points to a *query* x from a set $\mathcal{X}$ of *data points* according to a distance function. By casting the trace alignment problem to a kNN problem, we can find the best k alignments of a log trace $\mathbf{e}$ in the space of the model traces. This can be done by using ad-hoc data structures like Ball-Tree and KD-Tree [13] to retrieve the k-neighborhood of $\mathbf{e}$ by pre-ordering (*indexing*) the space of the (embedded) model traces wrt a distance function $\delta_{\mathcal{E}}$.

kNN algorithms, being unsupervised, give to all features in $\mathcal{E}(\mathbf{e}_a)$ the same weight. However, since control-flow is, in general, represented by a lower number of features wrt the data flow, an equal distribution of the weights would penalize the control-flow, which is, instead, crucial for the alignment task. Moreover, since each variable is divided into interval features to represent the data-flow of the abstract trace, based on the possible intervals for that variable, a variable having a higher number of possible intervals would have a higher weight.

To overcome this problem, it is possible to use weighted kNN algorithms and force a different distribution of weights. This can be done in two steps. First, we separate $\mathcal{E}(\mathbf{e}_a)$ into $\mathcal{E}_A(\mathbf{e}_a)$ and $\mathcal{E}_V(\mathbf{e}_a)$ containing the control-flow and the data-flow features for trace $\mathbf{e}_a$ respectively. Then, we fix a parameter $s \in [0, 1]$ and assign weights s and $1 - s$, to $\mathcal{E}_A(\mathbf{e}_a)$ and $\mathcal{E}_V(\mathbf{e}_a)$. In this way, if, for instance, $s = 0.4$, we can assign weight 0.4 to the entire set of control flow features and weight 0.6 to the entire set of data features. Secondly, to avoid that a data variable having more possible intervals in $\mathcal{E}_V(\mathbf{e}_a)$ gets a higher weight, we uniformly distribute the weight $1 - s$ over the data variables and not over the interval features. For instance, if we have two data variables *Amount* and *Point* with possible intervals $Amount \in \{[0, 10[, [10, 20[, [10, 30[\}$ and $Point \in \{[0, 5[, [5, 10[\}$, respectively, weight $1 - s = 0.6$ is uniformly distributed over the two data variables and, then, the resulting weights (0.3 for each data variable) are distributed over the corresponding interval features, leading the weight distribution $(0.1, 0.1, 0.1, 0.15, 0.15)$ over the interval features ($Amount : [0, 10[$, $Amount : [10, 20[$, $Amount : [10, 30[$, $Point : [0, 5[$, and $Point : [5, 10[$). Therefore, each interval feature for a data variable v_1 having more possible intervals gets a lower weight, but the sum of the weights of all the interval features of each data variable is the same. In this way, considering all the features $l_1, ..., l_p$ from $\mathcal{E}_A(\mathbf{e}_a)$ and $v_1, ..., v_q$ from $\mathcal{E}_V(\mathbf{e}_a)$, we define a variable weight $w = (w_{l_1}, ..., w_{l_p}, w_{v_1}, ..., w_{v_q})$, where $\sum w = 1$, $\sum(w_{l_1}, ..., w_{l_p}) = s$, and $\sum(w_{v_1}, ..., w_{v_q}) = 1 - s$. In the kNN algorithm, we multiply the features in $\mathcal{E}(\mathbf{e}_a)$ by w to normalize them.

The computation of approximate alignments is more efficient than the computation of optimal alignments. However, this computational gain comes with a loss in precision. It is well-known that the generation of precise encodings for graph data with loops is NP-complete [6]. To keep the information preserved at most, we investigate different encodings recently provided by the process mining community, as well as the proposed simplified p-grams encoding, in next section.

5 Experimentation

In this section, we report on experiments that contrast our approximate, encoding-based approach with precise conformance checking techniques. As outlined in the introduction, we performed two experiments to that end. (a) First, we compare our encoding-based approach with the state-of-the-art data-aware conformance checker CoCoMoT [3]: here, for a given trace, we compare the best-matching model run as computed by CoCoMoT, with the best-matching abstract trace as estimated by our approximate approach (recall that every model run corresponds to a unique abstract trace). (b) Second, we consider the situation where the process behaviors are specified by a set of traces that plays the role of a reference log. Here, for a given trace, we compare the best-matching trace from the reference log according to the distance function from [17], with the best-matching trace as estimated by our approximate approach. This second setting can be of interest if no DPN is available; but the experiment also helps to study specifically how well encodings can emulate the distance function.

Consequently, for stage (1) of our approach, process behaviors were represented as follows: (1a) We represented all behaviors of a DPN with variable-to-constant comparisons by a complete set $\mathcal{T}$ of abstract traces, as described in Section 3. (1b) We took as $\mathcal{T}$ a plain set of traces. Then, we (2) applied the trace encodings discussed in Section 4.1 to obtain the behavior encoding space, and (3) compared the returned approximate alignments with the optimal ones. The implementation we used for the experiments is publicly available at https://github.com/jonghyeonk/Multi-Trace-Alignment.

Datasets. For (1a), we used a DPN modeling a road fine management process [10, Fig. 13] that was mined automatically and where all guards are variable-to-constant comparisons. With the proposed abstract trace, we generated a representative set $\mathcal{T}$ with 639 abstract traces, as well as a test set L of 1,885 log traces with random values which comply with the DPN to a varying degree.

For (1b), we considered the *Sepsis* [15] event log, which represents the pathway of patients with symptoms of sepsis in a Dutch hospital. Here, we took $\mathcal{T}$ as all traces in the above dataset. As test set, we generated a log L consisting of 30 non-compliant log traces by modifying 10 traces in $\mathcal{T}$ based on three types of deviations: (i) modification of an activity label, (ii) modification of a categorical feature ('Diagnosis'), and (iii) modification of a numerical feature ('CRP') obtained by multiplying the original value by 10. Statistics of the trace representations $\mathcal{T}$ are summarized in Table 2.

Road Fines				*Sepsis*			
# of abstract traces	639	# of events	3,422	# of traces	1,079	# of events	15,214
# of activities	9	trace length	1∼6	# of activities	16	trace length	3∼185
# of data variables	5			# of data variables	2		

Table 2. Descriptive statistics of the trace representations $\mathcal{T}$ for the datasets.

Experiment Setup. For (1a) the *Road Fines* experiment, for each trace $\mathbf{e}$ in the test set L, we computed the optimal alignment γ and the associated process run $\gamma|_M$ using CoCoMoT, as well as its (unique) matching abstract trace $\mathbf{e}_a$. Then, we compared $\mathbf{e}_a$ with the result of the encoding-based approach. To compute $\gamma|_M$, we used the default settings of CoCoMoT, including its heuristic to determine $maxlen(\mathcal{N}, |\mathbf{e}|)$, which was obtained as $|\mathbf{e}| + m$, where m is the length of the shortest trace accepted by $\mathcal{N}$.

For (1b) the *Sepsis* experiment, for each trace $\mathbf{e}$ in the test set L, we computed the trace $\mathbf{e}'$ in $\mathcal{T}$ such that $\delta(\mathbf{e}, \mathbf{e}')$ is minimal according to the trace distance (Definition 6), and compared it with the result of the encoding-based approach.

Experimental Settings. For both experiments, we used the same settings: we set the split parameter s for feature weights to 0.5, selected the top k alignments with $k = \{k_{(10\%)}, k_{(20\%)}, k_{(30\%)}\}$ (k is the percentage of abstract traces returned) using a kNN algorithm,[2] and used three standard distance metrics (*Cosine*, *Manhattan*, *Euclidean*) with the following five encodings [12, 14]:

- *aggregate*: the control-flow is represented using numerical features indicating the frequency of each activity label. Similarly, for categorical data variables, we use numerical features indicating the frequency of each possible categorical value. For numerical data variables, instead, we use the average, standard deviation, max, min, and sum of the values;
- *boolean*: the numerical data variables are represented as in the aggregate encoding. The control-flow and the categorical data variables are, instead, represented through boolean features (true/false) indicating whether a certain activity label or a certain categorical value is present in the trace;
- *complexindex*: this is the complex-index encoding introduced in [12]. The control flow is represented using the simple-index encoding, i.e., each control-flow feature corresponds to a position in the trace and the value of the feature is the activity occurring in that position of the trace. Similarly, for each data variable, we have different features representing that data variable in different positions of the trace and the value of the feature is the value of the data variable if the variable is numeric and, if the variable is instead categorical, a code representing its categorical value;
- *laststate*: this encoding represents the control-flow with the simple-index encoding and the data variables using the latest payload of a trace, i.e., data

[2] We used a function provided by the *sklearn* Python library, using the parameter *auto* for the selection of the algorithm, which makes the function able to select the most appropriate algorithm based on the input.

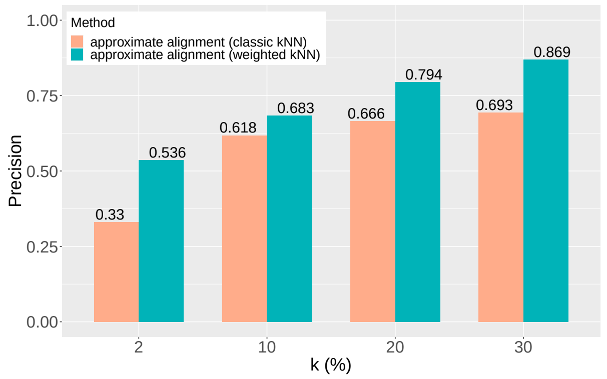


Fig. 1. Performance improvement on the *Road Fines* experiment after integrating the variable weight function (for *complexindex* encoding and *Euclidean* distance)

variables are treated as static features without taking into consideration their evolution over time;

– *p-gram+aggregate*: we used the encoding based on p-grams introduced in Section 4.1 with $p = 2$ and $\lambda = 0.7$. We integrated in this encoding the data perspective in the same way as done in the aggregate encoding.

In the evaluation, we measure (i) whether among the top-k alignments returned by the kNN algorithm, there is the optimal alignment returned by CoCoMoT (*precision*), (ii) how similar the top-k alignments returned by the approximate method are wrt the optimal alignment returned by CoCoMoT (*similarity*), and (iii) the execution times (*time*). For computing precision, we count the number of true positives TP, i.e., how many times the top-k alignments include the optimal alignment returned by CoCoMoT, and the number of false positives FP, i.e., how many times the top-k alignments do not include the optimal alignment returned by CoCoMoT. Then, the precision is computed as: $Precision = TP/(TP + FP)$. For similarity, we calculate the average Euclidean distance *dist* between the top-k alignments and the optimal alignment returned by CoCoMoT and we compute the similarity score as: $similarity = 1 - dist$. We computed the execution times (in seconds) by running the alignment tools on an Intel Core i9-12900H CPU with 2.5 GHz, 40GB RAM, with MS Windows 11, and measuring the total time needed to align all log traces.

Experimental Results. Tables 3 and 4 show the precision of the approximate method achieved by varying three parameters (encoding method, distance metric, and k) for the *Road Fines* experiment and the *Sepsis* experiment. The results show that, as expected, when k increases, the top-k alignments are more likely to include the optimal trace or abstract trace. In the cases in which the precision is higher for lower values of k, the approximate approach results to be more effective. Regarding the encoding methods, *complexindex* and *p-grams+aggregate* have a higher precision overall wrt the other encodings. This was expected since the *aggregate* and *boolean* encodings are less rich in the representation of the

Encoding method	Distance metric	k (%)	Precision (ref = CoCoMot)	Time (sec)
aggregate	Cosine	(10% / 20% / 30%)	(0.613 / 0.740 / 0.803)	(0.08 / 0.09 / 0.10)
aggregate	Euclidean	(10% / 20% / 30%)	(0.195 / 0.501 / 0.551)	(0.19 / 0.13 / 0.17)
aggregate	Manhattan	(10% / 20% / 30%)	(0.195 / 0.518 / 0.555)	(0.14 / 0.21 / 0.16)
boolean	Cosine	(10% / 20% / 30%)	(0.580 / 0.712 / 0.756)	(0.08 / 0.17 / 0.10)
boolean	Euclidean	(10% / 20% / 30%)	(0.597 / 0.725 / 0.808)	(0.13 / 0.19 / 0.13)
boolean	Manhattan	(10% / 20% / 30%)	(0.596 / 0.729 / 0.828)	(0.16 / 0.15 / 0.16)
complexindex	Cosine	(10% / 20% / 30%)	(0.683 / 0.775 / 0.838)	(0.28 / 0.24 / 0.27)
complexindex	Euclidean	(10% / 20% / 30%)	(0.683 / 0.794 / 0.869)	(0.36 / 0.33 / 0.37)
complexindex	Manhattan	(10% / 20% / 30%)	(0.481 / 0.790 / 0.862)	(0.70 / 0.73 / 0.67)
laststate	Cosine	(10% / 20% / 30%)	(0.420 / 0.688 / 0.800)	(0.08 / 0.08 / 0.17)
laststate	Euclidean	(10% / 20% / 30%)	(0.494 / 0.712 / 0.845)	(0.12 / 0.13 / 0.14)
laststate	Manhattan	(10% / 20% / 30%)	(0.510 / 0.734 / 0.882)	(0.16 / 0.18 / 0.16)
p-gram+aggregate	Cosine	(10% / 20% / 30%)	(0.705 / 0.776 / 0.817)	(0.07 / 0.08 / 0.08)
p-gram+aggregate	Euclidean	(10% / 20% / 30%)	(0.715 / 0.776 / 0.853)	(0.11 / 0.12 / 0.12)
p-gram+aggregate	Manhattan	(10% / 20% / 30%)	(**0.719** / **0.798** / **0.898**)	(0.12 / 0.14 / 0.18)

Table 3. Precision of approximate trace alignment with different parameters in the *Road Fines* experiment. For each k, the best precision is highlighted in bold.

Encoding method	Distance metric	k (%)	Precision (ref = CoCoMot)	Time (sec)
aggregate	Cosine	(10% / 20% / 30%)	(0.813 / 0.833 / 0.841)	(0.01 / 0.01 / 0.01)
aggregate	Euclidean	(10% / 20% / 30%)	(0.888 / 0.898 / 0.906)	(0.02 / 0.02 / 0.02)
aggregate	Manhattan	(10% / 20% / 30%)	(0.888 / 0.898 / 0.906)	(0.02 / 0.03 / 0.02)
boolean	Cosine	(10% / 20% / 30%)	(0.776 / 0.800 / 0.808)	(0.01 / 0.01 / 0.02)
boolean	Euclidean	(10% / 20% / 30%)	(0.854 / 0.864 / 0.873)	(0.02 / 0.02 / 0.02)
boolean	Manhattan	(10% / 20% / 30%)	(0.852 / 0.864 / 0.873)	(0.02 / 0.02 / 0.03)
complexindex	Cosine	(10% / 20% / 30%)	(0.816 / 0.816 / 0.888)	(0.04 / 0.04 / 0.04)
complexindex	Euclidean	(10% / 20% / 30%)	(0.891 / 0.931 / 0.949)	(0.05 / 0.04 / 0.05)
complexindex	Manhattan	(10% / 20% / 30%)	(**0.924** / **0.938** / **0.951**)	(0.08 / 0.07 / 0.07)
laststate	Cosine	(10% / 20% / 30%)	(0.822 / 0.822 / 0.822)	(0.01 / 0.01 / 0.01)
laststate	Euclidean	(10% / 20% / 30%)	(0.857 / 0.923 / 0.924)	(0.01 / 0.01 / 0.02)
laststate	Manhattan	(10% / 20% / 30%)	(0.855 / 0.923 / 0.924)	(0.02 / 0.02 / 0.02)
p-gram+aggregate	Cosine	(10% / 20% / 30%)	(0.864 / 0.881 / 0.891)	(0.01 / 0.01 / 0.01)
p-gram+aggregate	Euclidean	(10% / 20% / 30%)	(0.914 / 0.926 / 0.939)	(0.02 / 0.02 / 0.02)
p-gram+aggregate	Manhattan	(10% / 20% / 30%)	(0.914 / 0.926 / 0.939)	(0.04 / 0.03 / 0.03)

Table 4. Precision of approximate trace alignment with different parameters in *Sepsis* experiment. For each k, the best precision is highlighted in bold.

control flow information and the *laststate* reflects less accurately the data flow. Figure 1 highlights the effectiveness of the variable weight function we have introduced to guide the kNN algorithms in finding the top k alignments.

Concerning the similarity measure, Tables 5 and 6 report the average similarity between the top k alignments returned by the approximate approach with $k \in \{1, 3, 5, 10\}$ and the optimal abstract trace returned by CoCoMoT (for *Road Fines*), or the closest trace (for *Sepsis*). As the number of traces is much lower wrt the number of traces used in Tables 3 and 4 ($k_{(10\%)} = 64$ and $k_{(10\%)} = 108$ for *Road Fines* and *Sepsis*, respectively), the precision becomes much lower. However, even if the optimal alignment could not be easily identified, the high similarity values shown in Table 5 indicate that the approximate approach returns alignments that are very close to the optimal one. For *Sepsis*, where we do not perform trace alignment wrt a DPN, the similarity is lower. This is due to the fact that, in this case, the alignment task is more challenging since the returned alignments are concrete traces whereas, in the *Road Fines* case, the alignment returned is an abstract trace, i.e., a class of traces. However, as shown in Table 4, when a larger number of possible alignments are returned, this issue does not affect the precision of the approximate approach.

Encoding method	Distance metric	k	Similarity (ref = CoCoMot)	Time (sec)
aggregate	Cosine	(1 / 3 / 5 / 10)	(0.971 / 0.971 / 0.971 / 0.973)	(0.07 / 0.07 / 0.07 / 0.07)
aggregate	Euclidean	(1 / 3 / 5 / 10)	(0.963 / 0.963 / 0.963 / 0.960)	(0.11 / 0.12 / 0.12 / 0.12)
aggregate	Manhattan	(1 / 3 / 5 / 10)	(0.933 / 0.938 / 0.942 / 0.948)	(0.18 / 0.13 / 0.16 / 0.14)
boolean	Cosine	(1 / 3 / 5 / 10)	(0.978 / 0.978 / 0.978 / 0.977)	(0.06 / 0.07 / 0.08 / 0.08)
boolean	Euclidean	(1 / 3 / 5 / 10)	(0.978 / 0.978 / 0.978 / 0.977)	(0.16 / 0.12 / 0.12 / 0.17)
boolean	Manhattan	(1 / 3 / 5 / 10)	(0.978 / 0.978 / 0.978 / 0.977)	(0.13 / 0.13 / 0.14 / 0.14)
complexindex	Cosine	(1 / 3 / 5 / 10)	(0.991 / 0.991 / 0.991 / 0.990)	(0.22 / 0.25 / 0.25 / 0.25)
complexindex	Euclidean	(1 / 3 / 5 / 10)	(0.991 / 0.991 / 0.991 / 0.990)	(0.37 / 0.31 / 0.35 / 0.35)
complexindex	Manhattan	(1 / 3 / 5 / 10)	(0.978 / 0.979 / 0.980 / 0.981)	(0.68 / 0.68 / 0.70 / 0.64)
laststate	Cosine	(1 / 3 / 5 / 10)	(0.985 / 0.985 / 0.984 / 0.984)	(0.07 / 0.07 / 0.07 / 0.07)
laststate	Euclidean	(1 / 3 / 5 / 10)	(0.985 / 0.985 / 0.984 / 0.984)	(0.17 / 0.12 / 0.12 / 0.17)
laststate	Manhattan	(1 / 3 / 5 / 10)	(0.985 / 0.985 / 0.984 / 0.984)	(0.13 / 0.18 / 0.13 / 0.14)
p-gram+aggregate	Cosine	(1 / 3 / 5 / 10)	(**0.996** / **0.996** / **0.996** / **0.995**)	(0.06 / 0.07 / 0.06 / 0.07)
p-gram+aggregate	Euclidean	(1 / 3 / 5 / 10)	(**0.996** / **0.996** / **0.996** / **0.995**)	(0.10 / 0.10 / 0.15 / 0.11)
p-gram+aggregate	Manhattan	(1 / 3 / 5 / 10)	(**0.996** / **0.996** / **0.996** / **0.995**)	(0.15 / 0.12 / 0.13 / 0.12)

Table 5. Similarity between the approximate trace alignment and the CoCoMoT alignment with different parameters in the *Road Fines* experiment.

Encoding method	Distance metric	k	Similarity (ref = CoCoMot)	Time (sec)
aggregate	Cosine	(1 / 3 / 5 / 10)	(0.609 / 0.610 / 0.609 / 0.608)	(0.01 / 0.01 / 0.01 / 0.01)
aggregate	Euclidean	(1 / 3 / 5 / 10)	(0.608 / 0.608 / 0.608 / 0.608)	(0.02 / 0.02 / 0.02 / 0.02)
aggregate	Manhattan	(1 / 3 / 5 / 10)	(0.608 / 0.608 / 0.608 / 0.608)	(0.02 / 0.02 / 0.02 / 0.02)
boolean	Cosine	(1 / 3 / 5 / 10)	(0.606 / 0.606 / 0.607 / 0.606)	(0.01 / 0.01 / 0.01 / 0.01)
boolean	Euclidean	(1 / 3 / 5 / 10)	(0.606 / 0.606 / 0.606 / 0.607)	(0.02 / 0.02 / 0.01 / 0.02)
boolean	Manhattan	(1 / 3 / 5 / 10)	(0.606 / 0.606 / 0.606 / 0.606)	(0.02 / 0.02 / 0.02 / 0.02)
complexindex	Cosine	(1 / 3 / 5 / 10)	(**0.620** / **0.620** / **0.620** / 0.620)	(0.04 / 0.04 / 0.04 / 0.04)
complexindex	Euclidean	(1 / 3 / 5 / 10)	(0.619 / **0.620** / **0.620** / 0.620)	(0.04 / 0.04 / 0.04 / 0.04)
complexindex	Manhattan	(1 / 3 / 5 / 10)	(0.619 / **0.620** / **0.620** / **0.621**)	(0.05 / 0.05 / 0.05 / 0.05)
laststate	Cosine	(1 / 3 / 5 / 10)	(0.603 / 0.602 / 0.601 / 0.602)	(0.01 / 0.01 / 0.01 / 0.01)
laststate	Euclidean	(1 / 3 / 5 / 10)	(0.603 / 0.601 / 0.600 / 0.600)	(0.02 / 0.02 / 0.02 / 0.02)
laststate	Manhattan	(1 / 3 / 5 / 10)	(0.603 / 0.601 / 0.600 / 0.601)	(0.01 / 0.02 / 0.02 / 0.02)
p-gram+aggregate	Cosine	(1 / 3 / 5 / 10)	(0.604 / 0.607 / 0.607 / 0.607)	(0.01 / 0.01 / 0.01 / 0.01)
p-gram+aggregate	Euclidean	(1 / 3 / 5 / 10)	(0.605 / 0.607 / 0.607 / 0.607)	(0.03 / 0.02 / 0.02 / 0.02)
p-gram+aggregate	Manhattan	(1 / 3 / 5 / 10)	(0.605 / 0.607 / 0.606 / 0.607)	(0.02 / 0.02 / 0.02 / 0.02)

Table 6. Similarity of approximate trace alignment in respect to the CoCoMoT alignment with different parameters on the *Sepsis* event log.

For what concerns the execution times, the approximate approach is, on average, 100 times faster than CoCoMoT while producing alignments that are very similar to the optimal one. For instance, the alignment task for the *Road Fines* experiment has been completed in 22.02 seconds using CoCoMot, but it has been completed in 0.18 seconds using the approximate approach with the *p-grams + aggregate* encoding, *Manhattan* distance and $k(\%) = 30\%$. We also highlight that, although the precision is high only when k is sufficiently large, the fact that the approximate approach can return a fraction of the input log that likely contains the optimal alignment can render this approach useful as a sampling method, in combination with an optimal approach: the approximation technique can be applied as a preprocessor to find a set of candidates C for optimal alignments. Afterwards, less performant but optimal tools like CoCoMoT [3] can be applied only to this limited set of candidates (either using trace distances or incorporating the information from C into the DPN).

6 Related Work

A few papers provide extensions over the computation of control-flow alignments so that also the data dimension is considered. The approach in [9] first takes into account the control-flow and, after it has been aligned, aligns case attributes. The one in [8] is similar, but makes use of DPNs as process models, where the data-perspective is taken into consideration by augmenting the computed alignment with write operations over process variables via MILP solving. An improvement of this is obtained in [16, 17], where in a faster A*-based technique the process and data dimensions are considered at the same time. Another recent notable approach is the one we use as baseline in this paper, i.e., the one from [3], where a very general multi-perspective conformance checking problem based on an abstract notion of cost function is solved via state-of-the-art SMT solving. Differently from our approach, all the above techniques compute optimal alignments.

As the current approaches for alignment computation have the main problem of the complexity both in space and time, various approximate solutions for control-flow alignment have been proposed. In [1], the authors provide a statistical approach to conformance checking that employs trace sampling from the event log and result approximation in order to derive conformance results in an efficient manner. This solution is orthogonal to our technique since, instead of sampling the log, we take abstractions from the model. Moreover, general approximation schemes for alignment have been proposed in [19]. The work casts a recursive strategy to solve the alignment problem by splitting ILP models into small pieces. The same authors present in [20] a technique to decrease both in execution time and memory the computation of alignments via the reduction of the given process model and the event log. A decomposition-based method is proposed in [7] for an approximation of the alignments with good precision and low execution time. Recently, an approximate alignment approach based on process trees has been proposed [18]. The approach splits the problem of alignments into smaller sub-problems along the tree hierarchy and solves them individually and in parallel. The approximate approaches presented so far consider the alignment problem at the control-flow level, while there is no existing work for handling the multi-perspective approximate alignment problem.

7 Conclusion

In this paper, we showed how trace encoding methods can be used to compute multi-perspective trace alignments. By opportunely selecting the encoding methods, the analyst can choose the relevant information for computing the alignments. This also makes the alignment task faster. Our experiments show that the approximate approach is 100 times faster than CoCoMoT. The results are accurate in terms of precision (identification of the optimal alignment) and similarity between the approximate and the optimal alignments. In future work, we want to extend this approach to probabilistic trace alignment both in the standard conformance checking scenario, in which a log trace is aligned with a DPN, and in the case in which a trace is aligned wrt a log of "happy paths". In the latter, the probability of a path can be computed using clustering methods, more precisely by taking the density of the cluster to which the path belongs.

References

1. Bauer, M., van der Aa, H., Weidlich, M.: Sampling and approximation techniques for efficient process conformance checking. Inf. Syst. **104**, 101666 (2022)
2. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)
3. Felli, P., Gianola, A., Montali, M., Rivkin, A., Winkler, S.: Cocomot: conformance checking of multi-perspective processes via SMT. In: Proc. of BPM 2021 (2021)
4. Felli, P., Gianola, A., Montali, M., Rivkin, A., Winkler, S.: Conformance checking with uncertainty via SMT. In: Proc. of BPM 2022 (2022)
5. Felli, P., Gianola, A., Montali, M., Rivkin, A., Winkler, S.: Data-aware conformance checking with SMT. Information Systems **117** (2023). https://doi.org/10.1016/j.is.2023.102230
6. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: Hardness results and efficient alternatives. In: Proc. of COLT 2003 (2003)
7. Lee, W.L.J., Verbeek, H., Munoz-Gama, J., van der Aalst, W.M., Sepúlveda, M.: Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. Information Sciences **466**, 55–91 (2018)
8. de Leoni, M., van der Aalst, W.M.P.: Data-aware process mining: discovering decisions in processes using alignments. In: Proc. 13th SAC. ACM (2013)
9. de Leoni, M., van der Aalst, W.M.P., van Dongen, B.F.: Data- and resource-aware conformance checking of business processes. In: Proc. BIS 2012 (2012)
10. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: Proc. 37th ER (2018)
11. de Leoni, M., Felli, P., Montali, M.: Integrating BPMN and DMN: modeling and analysis. J. Data Semant. **10**(1), 165–188 (2021)
12. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Proc. BPM 2015 (2015)
13. Liu, T., Moore, A.W., Gray, A., Cardie, C.: New algorithms for efficient high-dimensional nonparametric classification. J. Mach. Learn. Res. **7**(6) (2006)
14. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.J.C.H.: Text classification using string kernels. J. Mach. Learn. Res. **2** (2002)
15. Mannhardt, F.: Sepsis cases - event log (2016), https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639/1
16. Mannhardt, F.: Multi-perspective Process Mining. Ph.D. thesis, Technical University of Eindhoven (2018)
17. Mannhardt, F., de Leoni, M., Reijers, H., van der Aalst, W.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
18. Schuster, D., van Zelst, S., van der Aalst, W.M.: Alignment approximation for process trees. In: Proc. ICPM 2020. pp. 247–259. Springer (2021)
19. Taymouri, F., Carmona, J.: A recursive paradigm for aligning observed behavior of large structured process models. In: Proc. BPM 2016 (2016)
20. Taymouri, F., Carmona, J.: Model and event log reductions to boost the computation of alignments. In: Proc. Data-Driven Process Discovery and Analysis: 6th IFIP WG 2.6 International Symposium. pp. 1–21. Springer (2018)