

On-the-fly Event Disambiguation via Alignments

Marco Franceschetti¹, Abel Armas-Cervantes², Astrid Rivera-Partida³,
Luciano Garcia-Bañuelos³, and Barbara Weber¹

¹ University of St.Gallen, St.Gallen, Switzerland

² The University of Melbourne, Australia

³ Tecnológico de Monterrey, Mexico

{marco.franceschetti@unisg.ch}

Abstract. In Internet of Things (IoT)-based process monitoring, the abstraction of sensor data into process events is often ambiguous. This ambiguity results in multiple plausible interpretations of the resulting event log, hindering process monitoring. We propose a novel algorithm that leverages alignment-based conformance checking to disambiguate process events in an online monitoring setting. For each ambiguous event, the algorithm explores the state space up to a k -bounded depth to identify the interpretation with the local minimum number of mismatches as the most plausible event interpretation. To make the approach feasible at run time, we employ an FM-index-based technique that enables fast computation of local all optimal alignments. A prototypical implementation demonstrates suitability for online disambiguation in IoT scenarios.

Keywords: Alignments · Ambiguity · IoT-based process monitoring

1 Introduction

In IoT-based scenarios, process events are derived by abstracting sensor data rather than being generated directly by a process engine [15]. A critical challenge in such scenarios is event ambiguity, i.e., multiple plausible interpretations about which specific process event has occurred [14]. Ambiguity in these scenarios typically arises because the same pattern of IoT data can be mapped to more than one process event [18]. This makes IoT-based process monitoring unreliable. Although ambiguity is a recognized issue, existing monitoring techniques assume unambiguous events and do not address event disambiguation.

We frame event disambiguation as a conformance checking problem: given a trace with ambiguous events, the goal is to identify the closest valid execution, according to a normative process model, that eliminates the ambiguity. Conformance checking is a key technique to determine whether the observed process behavior aligns with a normative model [8]. A central mechanism in this comparison is the use of alignments [4], which map events in a trace to activities in a model. An alignment indicates which events can be matched between trace and model (synchronous move), and identifies deviations as log moves or model

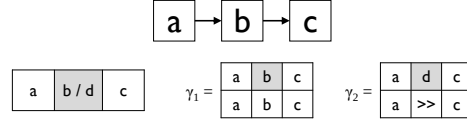


Fig. 1: A trace $a[b/d]c$ with event ambiguity, where the second event is either b or d . Interpreting the event as b yields a synchronous move (alignment γ_1); interpreting it as d yields a model move (γ_2).

moves. While event ambiguity entails multiple plausible interpretations of IoT data as process events, only one interpretation—the correct one corresponding to the actual activity execution—is meaningful. Other interpretations lead to incorrect conformance conclusions. Therefore, it is critical to identify the correct interpretation among multiple candidates. We argue that the normative behavior encoded in a process model provides the contextual knowledge to disambiguate events. This underlying assumption—that the correct interpretation is the one that best aligns with the normative behavior—has been supported in prior work in the process mining literature, for instance in model repair [24].

Based on this rationale, we address event ambiguity by leveraging alignment-based conformance checking. Specifically, we use alignments to actively attempt event disambiguation by comparing all interpretations of a trace and selecting the one that exhibits the minimal deviation, rather than to check conformance. To illustrate this, Figure 1 presents an example trace $a[b/d]c$ where event ambiguity leads to multiple possible trace interpretations. In the example, the same pattern of sensed IoT data could plausibly be abstracted to either b or d . The top of Fig. 1 shows the reference process model, while alignments γ_1 and γ_2 illustrate that this ambiguity manifests as two alternative traces (abc , adc) in the log. While both are plausible, we notice that γ_1 yields a synchronous move, while γ_2 yields a model move (denoted as “>>” in Fig. 1). This indicates that b is the correct interpretation under the alignment-based criterion, while d is not. This example highlights how ambiguous logs result in suboptimal alignments— γ_2 in Fig. 1. By identifying the interpretation with the minimal deviation from the model, alignment-based conformance checking serves as a principled method for event disambiguation. Note that the example shows the alignment of a complete trace; however, online process monitoring requires on-the-fly disambiguation as new events are observed, without future knowledge of the trace continuation.

To guide this work, we formulate the following research question: *How can process events be disambiguated in an online process monitoring setting?*

In addressing this question, we acknowledge the algorithm proposed in [23] as a highly efficient method to compute all optimal k -bounded alignments for complete traces in offline settings by using the FM-index [11]. Here, we leverage and significantly extend this algorithm in two ways: (1) shifting to an *on-the-fly* locally-optimal computation of alignments for partial traces, (2) to disambiguate events in online settings. Our algorithm enables the analysis of complete traces, prefixes, and suffixes, allowing incremental event disambiguation. To the best of

our knowledge, it is the first to: (1) apply the FM-index to online alignment computation, and (2) leverage alignment-based conformance checking as a *decision mechanism* to directly attempt event disambiguation in online process monitoring. This shifts the role of alignments from (post-hoc) conformance analysis to an active decision mechanism for disambiguating events during process execution.

Our evaluation confirms that the proposed algorithm is both accurate and performant, hence suitable for online monitoring. Moreover, it is domain-independent. Traditional techniques cannot achieve online conformance checking over ambiguous logs without generating a potentially large number of (complete) traces by enumerating all combinations of interpretations. Therefore, this work represents a foundational step toward ambiguity-aware process mining in dynamic, data-rich environments such as those enabled by IoT technologies.

The remainder of the paper is structured as follows: Section 2 illustrates a motivating scenario; Sect. 3 introduces preliminary concepts; Sect. 4 describes the proposed algorithm; Sect. 5 discusses implementation and evaluation; Sect. 6 reviews related work; Sect. 7 concludes the paper.

2 Motivating Scenario

As a motivating scenario, we consider the IoT-based monitoring of healthcare worker trainees (henceforth shortened as *HCW*) practicing phlebotomy—the process of conducting blood extractions, which we have validated with domain experts and described in our prior work [13].

The process starts with the blood donor’s check-in, followed by the HCW’s check-in. The HCW performs a hand hygiene, then checks the blood drawing machine. Next, the HCW disinfects the injection site, performs a hand hygiene, applies the tourniquet on the donor’s arm, inserts the needle, removes the tourniquet from the donor’s arm, and starts the blood drawing machine. While the machine draws blood, the HCW monitors the patient. Once enough blood is drawn, the HCW stops the machine, removes the needle, and performs a hand hygiene. Next, the donor checks-out, the HCW extracts the blood samples from the machine to store them, and ends the process by checking out. At any time, the HCW may perform additional hand hygiene steps in response to hand contamination events such as signing a document or answering a phone call. Due to their broad variety and unpredictability, hand contamination events are not part of the process and are left unmonitored.

Figure 2 shows our IoT-augmented setup for executing and monitoring the process. It includes two donor stations, which share a central hand hygiene station where the HCW performs the *Hand hygiene* activity throughout the process. Each station is enriched with IoT sensors that produce IoT data describing, e.g., the weight applied on a scale over which the hand sanitizer dispenser is positioned, someone’s proximity from the dispenser, or the motion at a donor station. At process run time, these IoT events are abstracted into *high-level* process events. This event abstraction can be realized by different means. One option is to employ unsupervised machine learning techniques that associate labels of



Fig. 2: IoT-augmented phlebotomy lab setup

process events with multivariate time series of IoT data [17]. Another option is to employ supervised activity detection services that emit process events by matching IoT event patterns defined by a domain expert [25]. These approaches enable monitoring processes executed without the support of a process engine, such as those in healthcare. For example, a hand hygiene detection service, upon matching an IoT event pattern describing a weight $> 1000g$ on the scale along with a proximity $< 30cm$ from the dispenser, emits a *Hand hygiene* process event. Together with similar activity-specific services, this service contributes to realizing the monitoring of the blood donation process, which in turn enables process mining analyses such as online conformance checking [6].

Each donor station in the setup is equipped with a motion, a proximity, and an ambient luminance sensor. Among other process events, the following events take place at each donor station: *Disinfect injection site*, *Apply tourniquet*, *Insert needle*, *Remove tourniquet*, and *Remove needle*. Whenever one of these five process events occurs, the sensors at the donor station emit an identical pattern of IoT events. The unavailability of additional sensors allowing for a finer-grained sensing makes it impossible to distinguish which of these process events actually occurs based on the IoT event pattern alone. As a consequence, event abstraction from this IoT event pattern results in the recording in the log of five candidate process events—one per interpretation—thereby introducing event ambiguity.

3 Preliminaries

This section introduces preliminary concepts and definitions that provide the foundation for our contribution.

3.1 Events and Traces

Typically, process executions are recorded in event logs in the form of traces, i.e., sequences of events that represent activity instances and are ordered by their time of occurrence. Each event is recorded with a set of attributes, including at least a case id referring to the process instance the event belongs to, the

label of the activity it represents, and the timestamp of its occurrence. Since we focus on ambiguity involving activity labels, we do not make any assumption on additional attributes that may be recorded with an event.

Let us recall the formalization of event and trace.

Definition 1 (Event). *Let $\mathcal{A}$ be the universe of activity labels; let $\mathcal{C}$ be the universe of case ids. An event is represented as a tuple $e = (t, a, c)$, where t is the timestamp, $a \in \mathcal{A}$ is the activity label, and $c \in \mathcal{C}$ is the case id. The timestamp, activity label, and case id of e can be accessed as $e.t$, $e.a$, $e.c$, respectively.*

In an IoT-based monitoring setting, process events are derived by abstracting IoT data while monitoring process executions through sensors. Here, akin to our prior work in [12], we assume that events affected by event ambiguity are generated incorporating ambiguity *explicitly* by recording all possible interpretations. Extending Definition 1 of event, such ambiguous events are defined as follows:

Definition 2 (Ambiguous event). *An event $e = (t, a, c)$ is affected by event ambiguity, i.e., is ambiguous, if $e.a = A \subseteq \mathcal{A}$, with $|A| > 1$.*

If event abstraction is not ambiguity-aware and produces single-label events, ambiguity can be made explicit in a preprocessing step. This can be achieved, for instance, by deriving alternative interpretations from the event abstraction mechanism (e.g., when multiple activities are associated with the same IoT event patterns), or by adopting windowing strategies that group temporally co-occurring observations and treat their corresponding labels as competing interpretations of the same event.

A sequence of events from the same case forms a trace:

Definition 3 (Trace). *A trace is a non-empty ordered event sequence $\sigma = \langle e_1, \dots, e_n \rangle$ such that $\forall i \in [1, \dots, n-1] \ e_i.c = e_{i+1}.c \wedge e_i.t \leq e_{i+1}.t$.*

Informally, we define a *complete trace* as a sequence of events representing a completed case from start to end. A *partial trace*, instead, does not contain all events from start to end. Given a trace $\sigma = \langle e_1, \dots, e_n \rangle$, we refer to a partial trace $\sigma' = \langle e_1, \dots, e_k \rangle$ with $k < n$ as a *prefix* of σ , and to a partial trace $\sigma'' = \langle e_j, \dots, e_n \rangle$ with $j > 1$ as a *suffix* of σ . Moreover, we refer to a trace that contains at least one ambiguous event as an *ambiguous trace*.

3.2 Alignments

Alignments [4] describe the commonalities and differences between two sequences of events. In conformance checking, alignments are computed between a trace from an event log and a process model run. While several alignments exist between traces and runs, commonly only those that are optimal (minimum number of differences) are considered. This allows detecting deviations in the executed cases from the behavior prescribed by the model. Below we present the definitions from [23] for pair-wise alignment, and synchronous and sequence moves.

Definition 4 (Alignment between two traces). An alignment between two traces σ_1 and σ_2 consists of a sequence of pairs of events (e, e') called moves. Moves can be either synchronous moves or sequence moves.

Definition 5 (Synchronous move). A synchronous move is a pair of events (e, e') such that $e.a = e'.a$ and $e \in \sigma_1$ and $e' \in \sigma_2$. A synchronous move denotes a pair of matching events representing the same activity.

Definition 6 (Sequence move). A sequence move (e, e') is used to make a move in either of the compared sequences, such that either:

- $e = >> \wedge e' \in \sigma_2$, or
- $e \in \sigma_1 \wedge e' = >>$,

where $>>$ represents no counterpart. A sequence move can be interpreted as an edit (insertion or deletion) operation, which in traditional model-based alignments defines a model- or a log-move [8].

An *optimal alignment* is an alignment with the least number of sequence moves [23]. For example, let us consider two traces $\sigma_1 = \langle a, b, c \rangle$ and $\sigma_2 = \langle a, c \rangle$. An optimal alignment between σ_1 and σ_2 is $[(a, a), (b, >>), (c, c)]$, which includes one sequence move. By contrast, $[(a, a), (>>, c), (b, >>), (c, >>)]$ is non-optimal, as it includes three sequence moves. Note that, in general, multiple optimal alignments may exist; however, in this example, the optimal alignment is unique.

3.3 All Optimal k -Bounded Alignments Using the FM-index

In [23], the authors present an approach to compute *all* optimal k -bounded alignments, i.e., all the optimal alignments with the same number of sequence moves—up to k —using the FM-index [11]. The FM-index is a text index built using the Burrows and Wheeler transformation [7] that enables an efficient sequence alignment technique, allowing for fast alignment computation. Due to its efficiency, we build on this approach to disambiguate events in an online setting. Here, we briefly summarize the approach and refer to [23] for further details.

The approach requires a reference behavior, which can either be a family of traces or an event log, a set of traces for which all optimal alignments are needed, and a value k . The algorithm starts by defining an alphabet from the set of unique labels in the reference behavior, including two special characters to identify the end of a trace and the end of the input. Both the reference behavior and the traces to be aligned are then relabeled. The reference behavior is formatted to construct the FM-index. The FM-index is finally used specifically to compute all k -bounded optimal alignments between the traces and the reference behavior.

The approach exploits backward and forward search over the FM-index to explore the space of candidate alignments. In backward search, a query pattern is matched from right to left by iteratively narrowing a character range in the FM-index. This is done using a procedure called *LF mapping* (for details see [23]), avoiding the need for explicit substring comparisons. To enable left-to-right alignments, the approach does a forward search by computing the FM-index

over the reverse of the reference behavior and applying backward search over the reversed trace. This enables the alignment of a complete trace in either direction.

While the algorithm targets the post-mortem analysis of complete traces, in the next section, we show that it can be adjusted to be applicable to online settings with partial traces. In such settings, only forward search is applicable, since events are observed incrementally and future behavior is unknown.

4 Alignment-based Event Disambiguation

We now illustrate the proposed algorithm to disambiguate events. While the core idea builds on the FM-index-based approach [23], our algorithm is a significant extension able to handle ambiguous events in an online setting, thereby supporting partial traces. Our algorithm uses alignments to serve disambiguation, not conformance checking; the cost model guides event label selection rather than deviation scoring. Here, we break down the algorithm into its major constituents.

The algorithm operates over three inputs: (i) a set of reference traces—encoded as sequences of characters over an alphabet—that represent the normative behavior for computing all optimal alignments; (ii) a stream of process events that may carry ambiguous labels, each represented by a character (*candidates*); and (iii) an alignment cost bound $maxK$ to limit the allowed deviation during alignment. We assume the reference traces to be precomputed, e.g., via normative model simulation [5]. While in principle infinite in length, these traces are bounded in practice, reflecting realistic execution limits and ensuring computational tractability.

Initialization. The initialization stage (cf. Alg. 1) prepares the index and the first search state used during alignment. Procedure `Init` (line 1) is executed once before alignment-based disambiguation to carry out this initialization. It constructs an FM-index over the reference traces (passed with parameter `Runs`) so that later we can quickly test whether extending a prefix by a character keeps us on a substring that actually occurs in any reference trace. The FM-index is constructed by calling a respective function `BuildFM` (line 2), in analogy with [23]. Procedure `CCHandler` (line 4) is a conformance checker constructor, which is instantiated for each new case in input. It receives as input $maxK$, the bound to the allowed deviations during alignment. It creates an initial row for a dynamic-programming (DP) matrix representing zero work done and zero cost, sets the FM-index range to span the whole alphabet, and pushes this single starting state onto an *open frontier* that represents the current search state. From this point onward, every advancement in the search will extend that state.

Function AddRow. Function `AddRow` (cf. Alg. 2) advances the DP matrix by one column whenever we hypothesize appending a character to the matched prefix. It computes the next cost vector using the standard Levenshtein recurrence: matching the same character adds zero, insertions and deletions add one, and substitutions add two (line 4). The result is packaged as a new row with a back-pointer to its predecessor (line 6; note that the prefix “&” denotes a pointer). Conceptually, each call encodes the cost of saying “the next character is c .”

Algorithm 1 Initialization

```

1: procedure INIT(Runs)
2:   Global FMIndex  $\leftarrow$  BUILDFM(Runs)
3: end procedure
4: procedure CCHANDLER(MaxK)
5:    $k \leftarrow \text{MaxK}$ 
6:    $R \leftarrow \text{ROW}(0, [0], \perp)$ 
7:    $l, r = 0, |\text{FMIndex}|$ 
8:    $\text{open} \leftarrow \emptyset$ 
9:    $\text{PUSH}(\text{open}, [R, l, r, 1, \text{""}])$ 
10: end procedure

```

Algorithm 2 AddRow

```

1: function ADDROW(R, Q', c, cnum)
2:    $\text{row} \leftarrow [R.v[0] + 1]$ 
3:   for  $\text{col} \in [1, \text{cnum}]$  do
4:      $\text{row}[\text{col}] \leftarrow \min \left( \begin{array}{l} R.v[\text{col} - 1] + (0 \text{ if } Q'[\text{col}] = c \text{ else } 2) \\ R.v[\text{col}] + 1 \\ \text{row}[\text{col} - 1] + 1 \end{array} \right)$ 
5:   end for
6:    $\text{nR} \leftarrow \text{ROW}(R.\text{num} + 1, \text{row}, \&R)$ 
7:   return nR
8: end function

```

Function Step. Function Step (cf. Alg. 3) is the core of the algorithm. It incrementally advances all partial alignments by processing one ambiguous event at a time, exploring all possible label interpretations and retaining those that minimize cumulative alignment cost. It receives a batch of candidate characters representing an (ambiguous) event. For each open search state (line 4), it iterates over every character to determine whether it can continue a valid substring of the reference traces (lines 5–14). This is determined via the call to function ForwardSearch (line 6), which performs a forward search for the current character over the FM-index (cf. Sect. 3.3). For every feasible extension, function Step updates the DP matrix via AddRow (line 10). Next, it checks whether at least one character produced a match:

- If at least one candidate character in the batch can be consumed immediately while staying on a reference substring, the procedure selects the next state with the lowest DP cost and returns it (lines 57–64). This case is illustrated in Fig. 3 (both left and right), when the first character (*a*) is consumed.
- If none of the candidates fit right away, the procedure explores model-only moves in a depth-first search (DFS) to see whether moving forward in the model first can create a point at which consuming one of the candidates becomes consistent with the reference. This depth-first exploration is bounded to a *maxK* depth (lines 16–47). Fig. 3 (left) illustrates an example in which it is possible to advance with a model move within *maxK* = 2 depth with the processing of the batch $[c/h]$ for the second ambiguous event in input. Notably, while the algorithm is proceeding with a parallel exploration—one branch per character in the batch—, matching character *c* in the left branch while *h* does not have a match effectively disambiguates the event.

Algorithm 3 Step

```

1: function STEP(chars)
2:   frontier  $\leftarrow \emptyset$ 
3:   match  $\leftarrow \emptyset$ 
4:   for [R, l, r, cnum, Q]  $\in$  POP(open) do
5:     for c  $\in \Sigma$  do
6:       l', r'  $\leftarrow$  FORWARDSEARCH(FMIndex, l, r, c)
7:       if l' < r' then
8:         is_event  $\leftarrow$  c  $\in$  chars
9:         Q'  $\leftarrow$  Q + c
10:        nR  $\leftarrow$  ADDROW(R, Q', c, cnum)
11:        if is_event then match  $\leftarrow$  match  $\cup$  {c} end if
12:        APPEND(frontier, [&nR, l', r', cnum + 1, Q'])
13:      end if
14:    end for
15:  end for
16:  if match =  $\emptyset$  then
17:    frontier2  $\leftarrow \emptyset$ 
18:    for extension  $\in$  frontier do
19:      Open2  $\leftarrow \emptyset$ 
20:      PUSH(Open2, [extension, 0])
21:      while Open2  $\neq \emptyset$  do
22:        [[R, l, r, cnum, Q], counter]  $\in$  POP(Open2)
23:        for c  $\in \Sigma$  do
24:          l', r'  $\leftarrow$  FORWARDSEARCH(FMIndex, l, r, c)
25:          if l' < r' then
26:            is_event  $\leftarrow$  c  $\in$  chars
27:            Q'  $\leftarrow$  Q + c
28:            nR  $\leftarrow$  ADDROW(R, Q', c, cnum)
29:            if is_event then
30:              OFFER(frontier2, [&nR, l', r', cnum, Q'])
31:            else
32:              if counter  $\geq$  maxK then continue end if
33:              PUSH(Open2, [[&nR, l', r', cnum, Q'], counter + 1])
34:            end if
35:          end if
36:        end for
37:      end while
38:    end for
39:    if |frontier2| > 0 then
40:      best  $\leftarrow \{e \in \text{frontier2} \mid \forall e' \in \text{frontier2} : e.R[-1] \leq e'.R[-1]\}$ 
41:      events  $\leftarrow \{e.Q[-1] \mid e \in \text{best}\}$ 
42:      PUSH(Open, best)
43:      if |events| == 1 then
44:        return (TRUE, ONE(events)) ▷ Event has been disambiguated
45:      else
46:        return (FALSE, events) ▷ Event could not be disambiguated
47:      end if
48:    else
49:      for [R, l, r, cnum, Q]  $\in$  frontier do
50:        R.v[cnum]  $\leftarrow$  R.v[cnum - 1] + 1
51:        OFFER(frontier2, [&R, l, r, cnum + 1, ">>"])
52:      end for
53:      PUSH(Open, frontier2)
54:      return (TRUE, ">>") ▷ Log move, event considered unambiguous
55:    end if
56:  end if
57:  best  $\leftarrow \{e \in \text{frontier} \mid \forall e' \in \text{frontier} : e.R[-1] \leq e'.R[-1]\}$ 
58:  events  $\leftarrow \{e.Q[-1] \mid e \in \text{best}\}$ 
59:  PUSH(Open, best)
60:  if |events| == 1 then
61:    return (TRUE, ONE(events)) ▷ Event has been disambiguated
62:  else
63:    return (FALSE, events) ▷ Event could not be disambiguated
64:  end if
65: end function

```

Algorithm 4 Online Event Disambiguation

```

1: procedure OED
2:   cases  $\leftarrow \{\}$ 
3:   while true do
4:     trailrec  $\leftarrow$  NEXT
5:     if trailrec = EOF then return end if
6:     for [case, events]  $\in$  trailrec do
7:       if case  $\notin$  cases then
8:         cc  $\leftarrow$  new CCHANDLER(3)
9:         cases[case]  $\leftarrow$  cc
10:      else
11:        cc  $\leftarrow$  cases[case]
12:      end if
13:      (unambiguous, events)  $\leftarrow$  cc  $\rightarrow$  STEP(events)
14:      yield (case, events, unambiguous)
15:    end for
16:  end while
17: end procedure

```

model moves attempt to resynchronize before the algorithm falls back to adding the cost of a log move, ensuring robustness to missing or spurious observations.

As in online settings complete traces are not available until all events are received, the algorithm optimizes alignments incrementally, one event at a time. It follows a greedy strategy that guarantees *local* optimal alignment for each new event, but not the *global* optimum. Thus, it potentially selects an incorrect interpretation for an early ambiguity, if it has the lowest cost. This can induce higher costs for aligning later events, hence a higher global cost, compared to selecting the correct interpretation with a higher cost at first. This tradeoff reflects the inherent limitation that decisions must be made without future knowledge: timely decisions are made, but potentially yielding a suboptimal global alignment.

5 Implementation and Evaluation

We implemented and published the proposed algorithm as a prototype in C++¹. As the prototype is intended for use in an online setting, we implemented the conformance checker as a REST Web service using the C++ Crow framework. We sequentially process the events in a log to simulate an event stream. Using a Python script, we read the log and submit each event (ambiguous or not) to the REST API. On the backend, a new conformance checker instance is created whenever a new case arrives, and a reference to this instance is stored in a hash table. Whenever a new event arrives, the backend retrieves the corresponding conformance checker instance and dispatches the event for processing by calling the Step function. The function returns the result of the (greedy) alignment of the given event: a synchronous move if possible, a log move (i.e., a “>>” character, along with the event that could not be aligned), the list of events skipped in the model (model moves) to reach a synchronous move, etc. In our implementation, the DFS exploration is bounded to a depth of $maxK = 3$ —a

¹ <https://github.com/lgbanuelos/disambiguation>

value chosen to balance responsiveness and alignment optimality in our setting. The result of the alignment for the event is returned in JSON format.

We evaluated the algorithm’s execution time and disambiguation accuracy on a Windows 11 machine with WSL2, Intel i7-13650HX CPU, and 16GB RAM.

5.1 Datasets with Ambiguity

Due to the absence of suitable datasets with ambiguous traces, we evaluated the algorithm using two synthetically generated datasets: one (*BD*) derived from the phlebotomy process (cf. Sect. 2) and one (*SH*) from a smart home process based on the BP-Meets-IoT 2020 challenge dataset [1]. For each process, we generated 15,000 heterogeneous ambiguous traces. To ensure high variability across these already ambiguous traces and at the same time better reflect traces generated from IoT-based process monitoring, we stochastically perturbed the generated traces. Perturbation was implemented using Java’s `java.util.Random` without a fixed seed to probabilistically skip activities, thereby simulating undetected activities by the IoT setup. Moreover, when generating each ambiguous trace, we also generated a corresponding *gold-standard* unambiguous trace to assess the algorithm’s ability to correctly disambiguate events. As ambiguity creates *uncertainty* about an event’s true interpretation, we adopted the XES format extended with uncertain event data (cf. [21]) to encode ambiguous traces, modeling competing interpretations as uncertain activity labels. The datasets, generation details, and complete evaluation results are available in an online appendix ².

5.2 Results and Discussion

Execution time. For each dataset, the algorithm was executed on both ambiguous and gold-standard traces three times, resulting in three paired observations per condition (ambiguous, gold standard). Execution times were recorded cumulatively across batches of 100 traces. To avoid statistical dependence inherent in cumulative measurements, non-cumulative batch runtimes (available online) were obtained by differencing successive measurements, yielding 150 batch runtimes for each of the three runs and condition. For each run, the mean batch runtime and corresponding mean trace runtime were computed to obtain single independent performance estimates (cf. Tables 1–2). Overall, for *BD*, the ambiguous set yielded a mean trace runtime of 15.14 ms ($\sigma = 0.18$), and the gold standard 15.15 ms ($\sigma = 0.18$)—a negligible difference of -0.1% (illustrative paired t-test: $t(2) = -0.08$, $p = 0.94$; $d = -0.05$). For *SH*, the ambiguous set exhibited a mean trace runtime of 8.90 ms , ($\sigma = 1.61$), and the gold standard 7.91 ms ($\sigma = 0.06$)—an approximate 12.5% increase ($t(2) = 1.09$, $p = 0.39$; $d = 0.63$). Overall, the mean observed per-event runtimes (available online) are 1 ms , with no statistically significant runtime degradation detected for ambiguous traces. This suggests that, within our experimental setup, the algorithm is computationally lightweight and suitable for online IoT-based monitoring of the considered processes, though a more detailed latency analysis remains future work.

² <https://doi.org/10.5281/zenodo.18861647>

Accuracy and fitness drop. Table 3 summarizes the results on disambiguation accuracy and fitness drop, comparing our algorithm against a null model baseline for event disambiguation (described online). This baseline, in the absence of model-based guidance, treats competing interpretations as equally plausible and randomly selects one of them, akin to the baseline in [19]. Disambiguation accuracy was defined as the ratio of correctly matched events between each pair disambiguated–gold-standard trace to the total number of events in the gold-standard trace. The algorithm’s mean disambiguation accuracy for *BD* was 0.99, while for *SH* it was 0.81 (baseline: 0.53 for *BD* and 0.54 for *SH*). Fitness drop was computed as the difference between the fitness (cf. [9]) of each gold-standard trace and the respective disambiguated trace. The algorithm’s mean fitness drop was 0 for both datasets (baseline: 0.22 for *BD* and 0.33 for *SH*). The high accuracy observed for *BD* can be explained by the largely sequential structure of the phlebotomy process, which limits the number of competing interpretations. By contrast, the lower accuracy for *SH* suggests that structural characteristics of the process allow for a higher number of competing interpretations. The absence of a fitness drop indicates that, despite occasional mismatches with the gold standard (particularly in *SH*), the disambiguated traces remain conformant with the process model: when multiple interpretations exist, the algorithm selects a model-consistent one, even if it does not correspond to the gold standard. This behavior reflects the greedy, bounded exploration strategy, which prioritizes locally optimal alignments to ensure responsiveness in online settings. Consequently, the algorithm may sacrifice perfect disambiguation accuracy in favor of timeliness and model consistency. These results therefore illustrate the algorithm’s inherent tradeoff between global optimality and responsiveness, while outperforming the baseline in terms of both accuracy and fitness drop.

Complexity. The algorithm’s complexity is the same as Needleman-Wunsch’s [20], which finds optimal pairwise alignments with quadratic time and space complexities. In our setting, the algorithm explores a trace alignment against all possible model runs. The greedy strategy explores the state space up to $\max K$ depth (3 in our experiments). While this induces an overhead with ambiguous events, our experiments show that it is negligible. Hence we expect the algorithm to generalize to more complex processes maintaining these theoretical guarantees.

Threats to validity. A threat to the external validity stems from assuming the availability of a normative process model to derive reference traces for alignment. This may limit applicability to settings where such a model is unavailable or only partially specified. In such cases, a reference model may be obtained via process discovery from historical logs or specified using domain knowledge; however, historical logs may be ambiguous, and real-world executions often deviate from the prescribed process. As a result, the derived reference model may diverge from the actual process behavior. In such cases, model–reality mismatches, missing observations in the log, and ambiguity in event abstraction are possible, potentially overlapping sources of sequence moves during alignment. This can hinder disambiguation accuracy by leading to false positives or false negatives when selecting event interpretations. These limitations in non-normative settings mo-

Table 1: Mean trace runtimes (*BD*)

Condition	Run 1	Run 2	Run 3
Ambiguous	14.94 <i>ms</i>	15.20 <i>ms</i>	15.28 <i>ms</i>
Gold standard	15.30 <i>ms</i>	15.21 <i>ms</i>	14.96 <i>ms</i>

Table 2: Mean trace runtimes (*SH*)

Condition	Run 1	Run 2	Run 3
Ambiguous	10.76 <i>ms</i>	8.00 <i>ms</i>	7.95 <i>ms</i>
Gold standard	7.94 <i>ms</i>	7.84 <i>ms</i>	7.96 <i>ms</i>

Table 3: Mean accuracy, fitness drop: proposed algorithm (A) vs baseline (B)

Dataset	Accuracy (A)	Accuracy (B)	Fitness drop (A)	Fitness drop (B)
<i>BD</i>	0.99	0.53	0.00	0.22
<i>SH</i>	0.81	0.54	0.00	0.33

tivate future work on uncertainty-aware representations or declarative alignment techniques, which may be better suited for such settings.

An additional threat to external validity stems from the limited diversity of process structures considered in the evaluation. Although the two datasets differ in their behavioral characteristics, they do not cover a broad spectrum of process complexity. Therefore, the observed accuracy and performance may not generalize to more complex processes or domains with substantially different control-flow characteristics. Assessing the approach on a wider range of processes remains future work.

A further threat to ecological validity stems from the use of synthetically generated ambiguous traces in the evaluation. Although the perturbation strategy was designed to mimic IoT-based monitoring imperfections by including missing observations, the resulting traces may not fully reflect those encountered in real-world deployments. Consequently, the reported accuracy and runtime results may be optimistic with respect to real IoT-based monitoring environments.

Finally, a threat to conclusion validity stems from fixing the maximum exploration depth to $\max K = 3$. While this choice limits the search space and supports responsiveness, it may prevent the algorithm from capturing longer dependencies in the reference behavior. As a result, interpretations requiring deeper exploration may be missed, which may in turn lower disambiguation accuracy. More generally, the reported results may be biased toward settings in which short-horizon alignment decisions are sufficient. Investigating the selection of $\max K$, the sensitivity of the approach to different values, and the resulting responsiveness–global optimality tradeoff constitute an avenue for future work.

6 Related Work

To the best of our knowledge, our algorithm is the first to address online event disambiguation via conformance checking, although related approaches exist.

Several techniques check process conformance using alignments, e.g., [3,10], but they typically assume certainty in the event log. This assumption is relaxed in the presence of ambiguity and uncertainty. For example, when event timestamps are too coarse or incorrect, total event ordering cannot be guaranteed and

causal dependencies become uncertain. The authors in [16] present an approach specifically aimed at checking conformance under such circumstances. Additional uncertainty arises with unclear event-to-activity mappings, i.e., when events in a log cannot be mapped with certainty to activities of a model. This is addressed in [2] by introducing a probabilistic conformance checking technique that leverages the notion of *behavioral space*. It considers all potential event-to-activity mappings to check process conformance without committing to a single, possibly incorrect mapping. Such uncertain mappings closely relate to event ambiguity, as ambiguity may arise from unclear mappings. Unlike [2], which provides probabilistic conformance results, our algorithm aims to disambiguate events and operates in online settings rather than post-mortem on complete traces.

Our algorithm builds on knowledge from a normative process model that encodes the dependencies between activities to disambiguate events. A similar use of process knowledge is found in [26], where NFC data collected at process run time is matched with process model activities to automatically monitor processes. Unlike [26], which focuses on prohibiting deviations, our algorithm allows deviations from the normative behavior within a user-defined bound.

Our work also relates to online conformance checking, addressed by work such as [6,22,27]. [28] presents an incremental computation of prefix-alignments, including *approximate* ones. While our algorithm computes only optimal alignments, it also disambiguates events—something prior approaches do not achieve.

7 Conclusion

In IoT-based process monitoring, event abstraction often introduces ambiguity, reducing downstream analysis reliability. We proposed an alignment-based algorithm for online event disambiguation. A prototype implementation shows the algorithm’s computational efficiency and suitability for the experimental setup.

The evaluation highlights both the potential and the limitations of our work. In generating traces with stochastic perturbation, we did not conduct a sensitivity analysis across multiple independently generated datasets: this remains future work. Nevertheless, the datasets are publicly available. The algorithm achieved near-perfect disambiguation accuracy on the *BD* dataset, but lower on the *SH*, suggesting that process structure affects accuracy. Moreover, the greedy strategy with fixed exploration depth prioritizes responsiveness over global optimality, which may reduce disambiguation accuracy. Additional evaluations on more complex processes and further investigation of the responsiveness-optimality tradeoff, including varying exploration depth, are necessary to fully assess performance.

Overall, this work highlights how conformance checking can move beyond post-hoc analysis to support decision-making in the presence of ambiguity.

Acknowledgments. This work has received funding from the Swiss National Science Foundation under Grant No. IZSTZ0_208497 (*ProAmbiIon* project).

Disclosure of Interests. The authors have no competing interests to declare.

References

1. BP Meets IoT 2020 Challenge Dataset. https://bp-meet-iot.webs.upv.es/2020/challenge/BP_Meets_Iot2020_Challenge_Dataset.pdf (2020), accessed: February 19, 2026
2. Van der Aa, H., Leopold, H., Reijers, H.A.: Efficient process conformance checking on the basis of uncertain event-to-activity mappings. *IEEE Transactions on Knowledge and Data Engineering* **32**(5), 927–940 (2019). <https://doi.org/10.1109/TKDE.2019.2897557>
3. Van der Aalst, W., Adriansyah, A., Van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **2**(2), 182–192 (2012). <https://doi.org/10.1002/widm.1045>
4. Adriansyah, A., van Dongen, B., van der Aalst, W.: Conformance checking using cost-based fitness analysis. In: *International Enterprise Distributed Object Computing Conference*. pp. 55–64 (2011). <https://doi.org/10.1109/EDOC.2011.12>
5. Burattin, A., Re, B., Rossi, L., Tiezzi, F.: A purpose-guided log generation framework. In: *International Conference on Business Process Management*. pp. 181–198. Springer (2022). https://doi.org/10.1007/978-3-031-16103-2_14
6. Burattin, A., van Zelst, S.J., Armas-Cervantes, A., van Dongen, B.F., Carmona, J.: Online conformance checking using behavioural patterns. In: *Business Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018, Proceedings 16*. pp. 250–267. Springer (2018). https://doi.org/10.1007/978-3-319-98648-7_15
7. Burrows, M.: A block-sorting lossless data compression algorithm. *SRS Research Report* **124** (1994)
8. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: *Conformance Checking: Relating Processes and Models*. Springer (2018)
9. Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: *Process mining handbook*, pp. 155–190. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_5
10. De Leoni, M., Maggi, F.M., van der Aalst, W.M.: Aligning event logs and declarative process models for conformance checking. In: *International Conference on Business Process Management*. pp. 82–97. Springer (2012). https://doi.org/10.1007/978-3-642-32885-5_6
11. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: *Proceedings 41st annual symposium on foundations of computer science*. pp. 390–398. IEEE (2000). <https://doi.org/10.1109/SFCS.2000.892127>
12. Franceschetti, M., Buchegger, D.M., Seiger, R., Weber, B.: Toward iot-based process analytics: Extending event knowledge graphs with ambiguity. In: *International Conference on Business Process Modeling, Development and Support*. pp. 176–192. Springer (2025). https://doi.org/10.1007/978-3-031-95397-2_11
13. Franceschetti, M., Seiger, R., González, M.J.G., García-Ceja, E., Flores, L.A.R., García-Bañuelos, L., Weber, B.: Proambition: Online process conformance checking with ambiguities driven by the internet of things. In: *CAiSE Research Projects Exhibition*. pp. 52–59 (2023)
14. Franceschetti, M., Seiger, R., López, H.A., Burattin, A., García-Bañuelos, L., Weber, B.: A characterisation of ambiguity in bpm. In: *International Conference on Conceptual Modeling*. pp. 277–295. Springer (2023). https://doi.org/10.1007/978-3-031-47262-6_15

15. Janiesch, C., et al.: The internet of things meets business process management: a manifesto. *IEEE Systems, Man, and Cybernetics Magazine* **6**(4), 34–44 (2020). <https://doi.org/10.1109/MSMC.2020.3003135>
16. Lu, X., Fahland, D., van der Aalst, W.M.: Conformance checking based on partially ordered event data. In: *Business Process Management Workshops: BPM 2014 International Workshops, Revised Papers 12*. pp. 75–88. Springer (2015). https://doi.org/10.1007/978-3-319-15895-2_7
17. Maier, J.B., Gram, J., Weisbarth, M., Hennebold, C., Huber, M.F.: Unsupervised event abstraction for automatic process modeling of plc-controlled automation systems. *Procedia CIRP* **120** (2023). <https://doi.org/10.1016/j.procir.2023.09.050>
18. Mangler, J., Seiger, R., Benzin, J.V., Grüger, J., Kirikkayis, Y., Gallik, F., Malburg, L., Ehrendorfer, M., Bertrand, Y., Franceschetti, M., et al.: From internet of things data to business processes: Challenges and a framework. In: *Internet of Things Meets Business Process Management: A Synergistic Integration*, pp. 25–61. Springer (2026). https://doi.org/10.1007/978-3-031-90746-3_2
19. Metzner, C., Schilling, A., Traxdorf, M., Tziridis, K., Maier, A., Schulze, H., Krauss, P.: Classification at the accuracy limit: facing the problem of data ambiguity. *Scientific reports* **12**(1), 22121 (2022). <https://doi.org/10.1038/s41598-022-26498-z>
20. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology* **48**(3), 443–453 (1970). [https://doi.org/10.1016/0022-2836\(70\)90057-4](https://doi.org/10.1016/0022-2836(70)90057-4)
21. Pegoraro, M., Uysal, M.S., van der Aalst, W.M.: An xes extension for uncertain event data (2022). <https://doi.org/10.48550/arXiv.2204.04135>
22. Raun, K., Tommasini, R., Awad, A.: I will survive: An event-driven conformance checking approach over process streams. In: *Proceedings of the 17th ACM International Conference on Distributed and Event-Based Systems*. pp. 49–60 (2023). <https://doi.org/10.1145/3583678.3596887>
23. Rivera-Partida, A., Armas-Cervantes, A., García-Bañuelos, L., Rodríguez-Flores, L.: All optimal k-bounded alignments using the fm-index. In: *International Conference on Cooperative Information Systems*. pp. 75–92. Springer (2024). https://doi.org/10.1007/978-3-031-81375-7_5
24. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M., Weske, M.: Repairing event logs using timed process models. In: *OTM 2013 Workshops*. pp. 705–708. Springer (2013). https://doi.org/10.1007/978-3-642-41033-8_89
25. Seiger, R., Franceschetti, M., Weber, B.: Data-driven generation of services for iot-based online activity detection. In: *International Conference on Service-Oriented Computing*. pp. 186–194. Springer (2023). https://doi.org/10.1007/978-3-031-48424-7_14
26. Stertz, F., Mangler, J., Rinderle-Ma, S.: Nfc-based task enactment for automatic documentation of treatment processes. In: *Enterprise, Business-Process and Information Systems Modeling. BPMDS EMMSAD 2017*. pp. 34–48. Springer (2017). https://doi.org/10.1007/978-3-319-59466-8_3
27. Stertz, F., Mangler, J., Rinderle-Ma, S.: Data-driven improvement of online conformance checking. In: *2020 IEEE 24th international enterprise distributed object computing conference (EDOC)*. pp. 187–196. IEEE (2020). <https://doi.org/10.1109/EDOC49727.2020.00031>
28. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.: On-line conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics* **8**, 269–284 (2019). <https://doi.org/10.1007/s41060-017-0078-6>