Activity Instance Identification Using Bipartite Graph Matching

Chiao-Yun Li[1,2(✉)], Anton Antonov[1], and Wil M.P. van der Aalst[1,2]

[1] Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
{chiaoyun.li,wvdaalst}@pads.rwth-aachen.de,
anton.antonov@rwth-aachen.de
[2] Fraunhofer FIT, Birlinghoven Castle, Sankt Augustin, Germany

Abstract. Process mining provides enterprises with insights into their business processes based on *event data* extracted from information systems. Techniques such as process discovery, conformance checking, and performance analysis enable the modeling, evaluation, and optimization of processes. Meanwhile, an activity execution is recorded as events representing transitions within the lifecycle of an activity, and the collection of these events is referred to as an *activity instance*. To accurately represent activity instances, *event logs*, a commonly used format of event data in process mining, require the correlation of events. However, real-world event logs often lack such information, leading to unreliable or biased results. This paper presents a novel approach to identify activity instances. By correlating events using bipartite-graph matching and alignment, we identify activity instances conforming to the lifecycle of the activity. The experiments demonstrate the effectiveness of the method and its robustness to noise and missing events using event logs across various domains.

Keywords: Process Mining · Activity Instance · Graph Matching

1 Introduction

Process mining allows organizations to derive *factual* insights into their business processes based on execution data extracted from *information systems* supporting process operations [1]. Owing to the strict linearity in information systems logging, execution data are typically logged as *events*, which capture *transitions* within an activity execution, referred to as an *activity instance*.

To ensure reliable process mining, it is essential to capture the entire lifecycle of an activity instance. Beyond the typical attributes of event data—(i) the event class (i.e., the associated *activity*), (ii) the transition, (iii) the timestamp, and (iv) the process instance (i.e., *case*) identifier—the correlation between events associated to the same activity instance is critical. Yet, in real-world event logs, these relationships are frequently absent [8,9]. As shown in Fig. 1, scenarios with identical event footprints for two activity instances within a case, extending the

S. Kallel et al. (Eds.): ICSOC 2024 Workshops, LNCS 15833, pp. 82–95, 2026.
https://doi.org/10.1007/978-981-96-7238-7_7

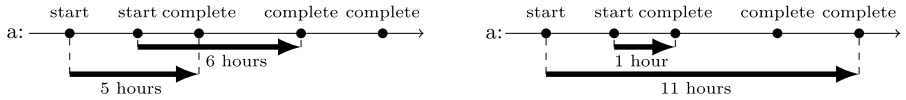


Fig. 1. Two scenarios of activity instance identification with the same footprint

examples from [1], highlight how this lack of correlation leads to ambiguity, resulting in arbitrary identification of activity instances. Consequently, process mining techniques that rely on timestamps to measure efficiency or order events may yield inconsistent and unreliable results [2,3]. Retrofitting information systems to capture these correlations is often prohibitively expensive. Therefore, a method to correlate events belonging to the same activity instance is necessary.

The presence of data quality issues further complicates the identification of activity instances [10]. For example, the uneven number of events for each transition of every activity instance implies the presence of noise or missing events. Figure 2 illustrates the footprint of an activity as recorded in a real-world log. The footprint may be interpreted in various ways: one might pair a start event with a subsequent complete, treating remaining events as extraneous noise, or, alternatively, one could assume the activity was executed twice, with one start event missing before one of the complete events.

In this paper, we propose an approach for activity instance identification. The process begins with preprocessing an *event log*, a structured representation of event data, to filter out noise and insert missing events. Next, we identify the *execution boundary events*, which define the actual start and completion of an activity's execution using Bipartite Graph Matching (BGM). Acknowledging the limitations of intractable issues like missing events, we introduce a variation that incorporates *certainty* levels based on the how close a transition is to the start or complete of an activity, as described in the *Transactional Lifecycle Model* (TLM), which outlines the lifecycle of an activity. This design choice aims to minimize the impact of missing events by accounting for their proximity and extract the maximal number of instances rather than discard the unmatched events [12]. Finally, for each pair of boundary events, we extract all related events such that the entire collection conforms to the TLM, thereby defining a complete activity instance. We evaluate our approach using a synthetic event log and three real-world logs, demonstrating its effectiveness and robustness against noise and missing events.

The paper is structured as follows. Section 2 reviews related work. Section 3 provides the foundation for the method in Sect. 4, evaluated in Sect. 5. Finally, Sect. 6 summarizes our contributions and outlines future directions.

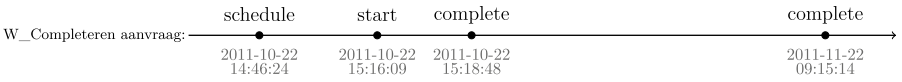


Fig. 2. Footprint of *W_ Completeren aanvraag* in case *179080* from [8]

2 Related Work

This paper focuses on accurately correlating events as activity instances within a case, an issue that has received limited attention. We investigate techniques for event correlation and discuss their applicability to the identified challenges. In the following, we categorize them based on the requirement of predefined models that outline expected sequences and interactions of concepts of interest.

Model-independent approaches identify event correlations without predefined relationships. Van der Aalst suggests using specific attributes, such as names or addresses, to correlate events when available [1]. Another method groups events of the same activity by applying a timeout window, treating events within that timeframe as an activity instance [1]. Clustering-based techniques create event representations and group them by similarity [13,15]. Supervised learning approaches predict event labels based on statistical patterns [4,23], which can be adapted to predict transitions. However, segmenting events for the same activity instance remains a challenge, often relying on solutions like those from [1]. Without explicit constraints, these methods may generate transition sequences that contradict with an activity's lifecycle. Some approaches assume a generally accepted temporal order, where a start event must precede a complete event of an activity instance. In [1], the author proposes to heuristically identify activity instances using a First-In-First-Out (FIFO) principle, as shown in the first scenario of Fig. 1. For instance, events from a wafer scanner are mapped to job steps using FIFO [21], based on the sequential flow of test jobs. However, some events remained unmapped due to incomplete event-to-activity associations, highlighting the limitations of relying solely on domain knowledge. Moreover, the FIFO principle may not be applicable in all scenarios. These methods focus on boundary events based on specific assumptions and overlook other transitions that provide a more detailed view of an execution.

In contrast, model-dependent approaches leverage the established relationships between events. For instance, [7] applies a probabilistic optimization technique to correlate events within the same case by minimizing misalignment between the correlated log and a process model. Another study maps event labels to activities in a model, comparing log behavior with the model to ensure conformity [6]. This approach relies on domain knowledge to resolve ambiguous mappings and assumes an acyclic process structure. Additionally, [5] leverages domain knowledge annotated on a process model to define event-activity relationships through textual analysis, treating events as time-interval transitions of higher-level activity instances. Alignment-based approaches match events to a process model, with the aligned events forming an instance at a higher level [18,19]; however, these methods can be non-deterministic.

3 Preliminaries

Let X be a set. $\mathcal{P}(X)=\{X'|X'\subseteq X\}$ denotes the powerset of X and $|X|$ is the number of elements in X. $\mathcal{A}$ is the universe of *activities*, $\mathcal{C}$ denotes the universe of

case identifiers, and $\mathcal{TS}$ is the universe of timestamps. A *sequence* is a function $\sigma\colon \{1,2,\ldots,n\}\to X$, denoted as $\langle x_1, x_2, \ldots, x_n\rangle$, where $\forall 1\leq i\leq n\colon \sigma(i)=x_i$, and $x_i\in\sigma$ denotes an element in σ. $|\sigma|$ denotes the *length* of σ, and X^* represents the set of all possible sequences over X. Let $\sigma=\langle x_1, x_2, \ldots, x_n\rangle \in X^*$ and $\sigma'\in X^*$. We write $\sigma'\subseteq\sigma$ if $\forall 1\leq i<j\leq|\sigma'|\colon \exists 1\leq m<n\leq|\sigma|\colon \sigma'(i)=\sigma(m)\wedge\sigma'(j)=\sigma(n)$. We insert an arbitrary element y after $0\leq i\leq|\sigma|$ in σ with a function INSERT. For example, given $\sigma=\langle x_1, x_2, x_3\rangle$, $\text{INSERT}(\sigma, y, 2)=\langle x_1, x_2, y, x_3\rangle$. The position of 0 indicates the insertion before the first element of σ, e.g. $\text{INSERT}(\sigma, y, 0)=\langle y, x_1, x_2, x_3\rangle$.

A TLM describes the lifecycle of a *successfully* executed activity.

Definition 1 (Transactional Lifecycle Model). *A transactional lifecycle model of any $a\in\mathcal{A}$ is a tuple $TLM_a=(S^a, T^a, s^a_{init}, s^a_{succ}, S^a_{active}, t^a_s, t^a_c)$, where S^a is a set of* states *and $T^a\subseteq S^a\times S^a$ are transition labels between the states s.t.*

- *$s^a_{init}\in S^a$ is the initial state, where $\nexists(s_1s_2)\in T^a\colon s_2=s^a_{init}$,*
- *$s^a_{succ}\in S^a$ is the successful completion state, where $\nexists(s_1s_2)\in T^a\colon s_1=s^a_{succ}$,*
- *$S^a_{active}\subseteq S^a$ is a set of* active *states,*
- *$t^a_s\in T^a$ is the* start *transition, where $\exists s\in S^a\setminus S^a_{active}\colon \exists s'\in S^a_{active}\colon (s,s')=t^a_s$,*
- *$t^a_c\in T^a$ is the* complete *transition, where $\exists s\in S^a\setminus S^a_{active}\colon \exists s'\in S^a_{active}\colon (s',s)=t^a_c$.*

We define a function $\text{LC}(TLM_a)=\{\sigma\in S^{a*}|\sigma(1)=s^a_{init}, \sigma(|\sigma|)=s^a_{succ}, \forall 1\leq i<|\sigma|\colon (\sigma(i),\sigma(i+1))\in T^a, \exists!1\leq i<j<|\sigma|\colon (\sigma(i),\sigma(i+1))=t^a_s \wedge (\sigma(j),\sigma(j+1))=t^a_c\}$, *extracting all state sequences that describe complete lifecycles of an activity. For every $s\in S^a\colon \exists\sigma\in\text{LC}(TLM_a)\colon s\in\sigma$. The* active transitions *are $T^a_{active}=\{t^a_s, t^a_c\}\cup\{(s_1s_2)\in T^a \mid s_1s_2\in S^a_{active}\}$. $\mathcal{T}^a$ is the set of all transitions for any $a\in\mathcal{A}$.*

Given $t\in\mathcal{T}^a$, if $t\notin T$, t is not a *valid* transition that leads to the successful execution of a, e.g., cancel or abort. For $t_1, t_2\in T^a$, t_1 precedes t_2, denoted as $t_1\prec t_2$, iff there is a *path* $\sigma\in T^*\colon \sigma(1)=t_1\wedge\sigma(|\sigma|)=t_2\wedge\forall 1\leq i<|\sigma|\colon \sigma(i)=(s_1s_2)\wedge\sigma(i+1)=(s_3s_4)\Rightarrow s_2=s_3$, and $t_2\not\prec t_1$. $\text{P}(t_1,t_2)\subseteq T^*$ denotes all paths from t_1 to t_2. Figure 3 visualizes a TLM, derived from [11].

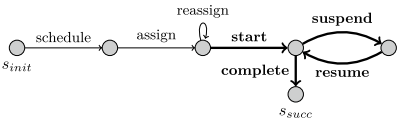


Fig. 3. Visualization of a standard TLM. Nodes represent states, with transitions annotated on arrows and active transitions highlighted.

Events associated with a case are organized into *traces*, forming an *event log*.

Definition 2 (Event, Trace, Event Log). *An event is a tuple $e=(c,a,ts,t)\in\mathcal{C}\times\mathcal{A}\times\mathcal{TS}\times\mathcal{T}^a$ referring to case c, activity a, timestamp t and transition t. We project the information with a function $\#$ s.t. $\#_{case}(e)=c$, $\#_{act}(e)=a$, $\#_{ts}(e)=ts$, and $\#_{tr}(e)=t$. Let $\mathcal{E}=\mathcal{C}\times\mathcal{A}\times\mathcal{TS}\times\mathcal{T}$. A trace is a sequence $\sigma\in\mathcal{E}^*$, where $\forall e_1,e_2\in\sigma\colon \#_{case}(e_1)=\#_{case}(e_2)$ and $\forall 1\leq i<|\sigma|\colon \#_{ts}(\sigma(i))\leq\#_{ts}(\sigma(i+1))$. An event log $L\subseteq\mathcal{E}^*$ is a set of traces where $\forall\sigma_1,\sigma_2\in L\colon \forall e_1\in\sigma_1\forall e_2\in\sigma_2\colon \#_{case}(e_1)=\#_{case}(e_2)\Leftrightarrow\sigma_1=\sigma_2$.*

In the rest of the paper, we refer to the TLM of an event through its associated activity. We denote the activities in L by $\mathsf{A}(L)=\{\#_{act}(e)|\sigma\in L, e\in\sigma\}\subseteq\mathcal{A}$. Given $\sigma\in L$ and $a\in\mathsf{A}(L)$, we project σ onto a s.t. $\sigma_{\uparrow a} \subseteq \sigma$, where $\forall 1\leq i\leq|\sigma|: \#_{act}(\sigma(i)) = a \Rightarrow \sigma(i)\in\sigma_{\uparrow a}$. The projection of L on a is denoted by $L_{\uparrow a}=\{\sigma_{\uparrow a}\subseteq\sigma|\sigma\in L\}$.

An activity instance is a sequence of events in a trace that align with the TLM of the corresponding activity.

Definition 3 (Activity Instance). *Let L be an event log. Given $a\in\mathsf{A}(L)$, $\sigma\in L$, and $\sigma'\subseteq\sigma_{\uparrow a}$, σ' is an activity instance of a if $\exists\sigma_s\in\mathrm{LC}(TLM_a): \forall 1\leq i<|\sigma_s|: (\sigma_s(i),\sigma_s(i+1))=\#_{tr}(\sigma'(i))$. The execution time of σ' is defined as the duration between the start and complete events, i.e., $\exists!1\leq i<j \leq |\sigma'|: \#_{tr}(\sigma'(i))=t_s\wedge\#_{tr}(\sigma'(j))=t_c \Rightarrow \#_{ts}(\sigma'(j))-\#_{ts}(\sigma'(i))$ is the execution time of σ'.*

We formulate the activity instance identification as a BGM problem by constructing a weighted labeled bipartite graph, which is a tuple $G=(V_1,V_2,E,\lambda,w)$, where V_1 and V_2 are disjoint sets of nodes, $E\subseteq V_1\times V_2$ are a set of edges, $\lambda: V_1\cup V_2\to\mathbb{N}$ is a labeling function, and $w: E\to\mathbb{R}_{\geq 0}$ is a weight function, denoted as $w_e=w(e)$ for any $e\in E$. A BGM is a selection $E'\subseteq E$ s.t. $\forall(v_1,v_2)\in E': \exists(v_1,v')\in E'\Rightarrow v'=v_2\wedge\exists(v',v_2)\in E'\Rightarrow v'=v_1$. Let E_M denote all possible matchings. A maximum weighted BGM yields a matching $E'\in E_M$ s.t. $\nexists E''\in E_M: \Sigma_{e\in E''}w_e>\Sigma_{e\in E'}w_e$.

4 Method

In this section, we outline the method in Sect. 4.1. Section 4.2 describes the preprocessing, followed by activity instance identification in Sect. 4.3.

4.1 Overview

Figure 4 illustrates the method. First, we preprocess an event log to ensure that every executed activity has corresponding boundary events within a case, specifically a start event followed by a complete event. We identify three challenges and propose preprocessing steps for each:

1. **Inactive activity filtering:** We remove noise from events related to unexecuted activities, such as scheduling events that do not result in an execution.
2. **Atomic activity completion:** *Atomic* activities are executed instantaneously and typically recorded with a single transition (e.g., a form submission captured solely as a complete event). We address this by inserting complementary events with the same timestamp.
3. **Missing events remediation:** Non-atomic activity instances can span a time interval yet may only capture a single active transition. We insert complementary missing events to guarantee the existence of boundary events.

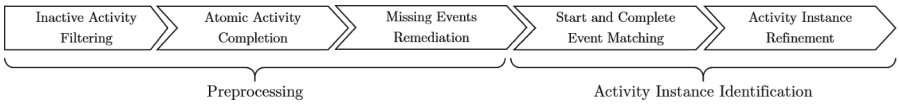


Fig. 4. Method Overview

After preprocessing, we *match* start and complete events to define the boundaries of activity execution. Establishing on the basis, we refine the collection for each instance, ensuring alignment with the associated TLM. Figure 5 visualizes a log with four traces, which we use to illustrate the method. Each chevron represents an event. For instance, σ_{101}=$\langle(101, A, 12{:}53{:}45, resume), (101, B, 12{:}56{:}48, complete), (101, D, 13{:}15{:}45, start), (101, B, 13{:}18{:}04, autoskip), (101, D, 13{:}23{:}12, complete), (101, C, 15{:}12{:}02, complete)\rangle$ is a trace of case 101. This example highlights the challenges that we will discuss in detail shortly.

4.2 Preprocessing

We apply specific assumptions to detect the data quality challenges and preprocess accordingly. The TLM in Fig. 3 is applied to all activities in Fig. 5.

Inactive Activity Filtering. We assume that *an activity is executed if its events with active transitions are observed within a trace or if the transition* may *lead to an activity instance*. For instance, all events related to activity D are excluded as noise in σ_{102}, since no active transitions for D are present within σ_{102}. Also, all events of the invalid transition *autoskip* in Fig. 5 are filtered out.

Atomic Activity Completion. Building upon the *consistency* principle, i.e., uniform representation of data across all events [10,24], we assert that *if only start or complete transition for an activity is observed in a log, the activity is atomic*. Let L be a log. Let T^a_{active} be the active transitions, and t^a_s and t^a_c the start and complete transitions in TLM_a for $a{\in}\mathsf{A}(L)$. The function $\textsc{atom}\colon \mathcal{P}(\mathcal{E}^*){\to}\mathcal{P}(\mathcal{A})$ detects atomic activities as $\textsc{atom}(L){=}\{a{\in}\mathsf{A}(L) | T{=}\{\#_{tr}(e) | \forall\sigma{\in}L \forall e{\in}\sigma_{\uparrow a}\colon \#_{tr}(e){\in}T^a_{active}\}$ and $|T|{=}1 \wedge T{\subseteq}\{t^a_s, t^a_c\}\}$. Definition 1 defines the minimal requirement of an activity instance as

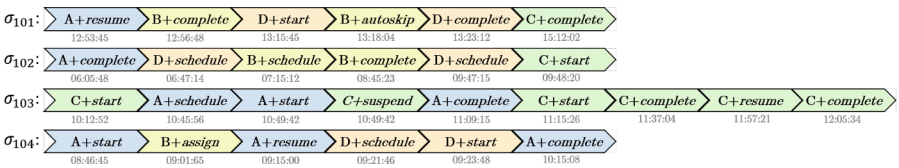


Fig. 5. A motivating example of activity instance identification

a start event followed by a complete event. Hence, $\forall \sigma \in L$ and $1 \leq i \leq |\sigma|$, if $\#_{act}(\sigma(i)) \in \text{ATOM}(L)$:

$$\sigma = \begin{cases} \text{INSERT}(\sigma, (\#_{case}(\sigma(i)), \#_{act}(\sigma(i)), \#_{ts}(\sigma(i)), t_c^a), i), & \text{if } \#_{tr}(\sigma(i)) = t_s^a, \\ \text{INSERT}(\sigma, (\#_{case}(\sigma(i)), \#_{act}(\sigma(i)), \#_{ts}(\sigma(i)), t_s^a), i-1), & \text{otherwise.} \end{cases}$$

Missing Events Remediation. If only *one* active transition is observed within a trace for a non-atomic activity, we assume that *events with other active transitions are absent*. Let L be a log. For each $\sigma \in L$ and $1 \leq i \leq |\sigma|$, if $\#_{act}(\sigma(i)) \notin \text{ATOM}(L)$, $\#_{tr}(\sigma(i)) \in T^a_{active}$, and $\nexists 1 \leq j \leq |\sigma| : i \neq j \wedge \#_{act}(\sigma(i)) = \#_{act}(\sigma(j))$:

- If $\#_{tr}(\sigma(i)) = t_s^a$, we define a function $\text{INSERTCOMPLETE}(\sigma, i)$ to insert the corresponding complete event:

$$\sigma = \begin{cases} \text{INSERT}(\sigma, (\#_{case}(\sigma(i)), \#_{act}(\sigma(i)), \#_{ts}(\sigma(i+1)), t_c^a), i), & \text{if } i < |\sigma|, \\ \text{INSERT}(\sigma, (\#_{case}(\sigma(i)), \#_{act}(\sigma(i)), \#_{ts}(\sigma(i)), t_c^a), i), & \text{otherwise.} \end{cases}$$

- If $\#_{tr}(\sigma(i)) = t_c^a$, we define a function $\text{INSERTSTART}(\sigma, i)$ to insert the corresponding start event:

$$\sigma = \begin{cases} \text{INSERT}(\sigma, (\#_{case}(\sigma(i)), \#_{act}(\sigma(i)), \#_{ts}(\sigma(i-1)), t_s^a), i-1), & \text{if } i > 1, \\ \text{INSERT}(\sigma, (\#_{case}(\sigma(i)), \#_{act}(\sigma(i)), \#_{ts}(\sigma(i)), t_s^a), i-1), & \text{otherwise.} \end{cases}$$

- Otherwise, $\sigma = \text{INSERTCOMPLETE}(\text{INSERTSTART}(\sigma, i), i+1)$.

Figure 6 illustrates the preprocessing. Activity B is considered *atomic*, since the only observed transition throughout the log is *complete*. Hence, we insert a start event with the same timestamp in σ_{101} and σ_{102}. The event of B is excluded in σ_{104} owing to inactive activity filtering. For non-atomic activities, complementary events are inserted as exemplified in σ_{101}, e.g., a start event of C is inserted with the timestamp derived from the preceding event. In σ_{101}, both start and complete events are introduced for A, since *resume* signifies that A was executed, despite the missing start and complete events.

4.3 Activity Instance Identification

We identify the boundary events by formulating it as an optimization problem, adapted from [17]. Next, we present a variation that incorporates the certainty of the information. Finally, we refine the collection as activity instances.

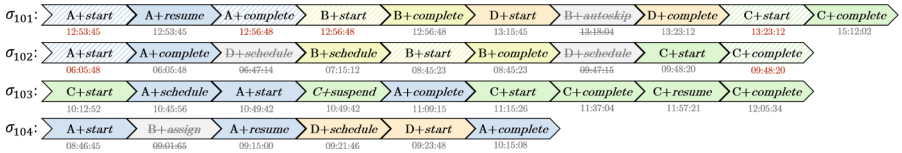


Fig. 6. Event log preprocessing based on Fig. 5. Excluded events are shown in gray, while inserted events are highlighted with their associated timestamps.

Naïve Matching. Let L be a log and t_s^a and t_c^a be the start and complete transitions of TLM_a for $a \in \mathsf{A}(L)$. The function $\textsc{Match}\colon \mathcal{E}^* \to \mathcal{P}(\mathbb{N}\times\mathbb{N})$ returns a set of index tuples $\textsc{Match}(\sigma)$ for any $\sigma \in L$, where $\forall (i,j) \in \textsc{Match}(\sigma)\colon$

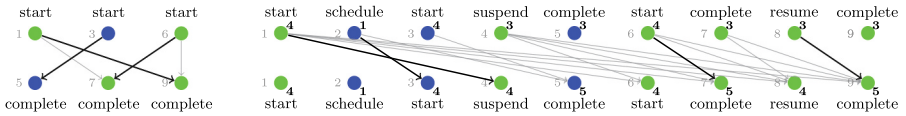
- $1 \leq i < j \leq |\sigma| \wedge \#_{act}(\sigma(i)) = \#_{act}(\sigma(j))$.
- $\#_{tr}(\sigma(i)) = t_s^{\#_{act}(\sigma(i))} \wedge \#_{tr}(\sigma(j)) = t_c^{\#_{act}(\sigma(j))}$.
- $\forall (m,n) \in \textsc{match}(\sigma)\colon \{m,n\} \cap \{i,j\} = \emptyset \vee (m,n) = (i,j)$.

Let $G=(V_1, V_2, E, \lambda, w)$, where $V_1=\{\sigma(i) | 1\leq i\leq|\sigma|\colon \#_{tr}(\sigma(i))=t_s\}$ and $V_2 = \{\sigma(i) | 1\leq i\leq|\sigma|\colon \#_{tr}(\sigma(i))=t_c\}$. The function λ labels a node with its position in σ s.t. $\forall v \in V_1 \cup V_2\colon v=\sigma(\lambda(v))$. Edges are defined as $E=\{(v_1,v_2) \in V_1 \times V_2 \mid \lambda(v_1)<\lambda(v_2) \wedge \#_{act}(\sigma(\lambda(v_1)))=\#_{act}(\sigma(\lambda(v_2)))\}$, and $\forall (v_1,v_2) \in E\colon w_{(v_1,v_2)} = 1/(\lambda(v_2)-\lambda(v_1))$. We match V_1 and V_2 using Integer Linear Programming (ILP) by assigning variable $x_{(v_1,v_2)} \in \{0,1\}$ to each $(v_1,v_2) \in E$, where $\forall v_2 \in V_2\colon \sum_{v_1 \in V_1} x_{(v_1,v_2)} \leq 1$ and $\forall v_2 \in V_2\colon \sum_{v_1 \in V_1} x_{(v_1,v_2)} \leq 1$, s.t. $\sum_{(v_1,v_2)\in E} w_{(v_1,v_2)} x_{(v_1,v_2)}$ is *maximized*. In this formulation, $w_{(v_1,v_2)}$ encourages matching events that are close in a trace, and the edges ensure only events of the same activity are paired. Figure 7a shows the graph with the matching for σ_{103} in Fig. 6.

Matching with Uncertainty. We estimate timestamps for missing events based on the nearest event, introducing a degree of *uncertainty*. For example, for a missing start event (see Fig. 2), we select the "schedule" as a substitute. We prioritize best-effort matching to maximize identified instances. Thereby, we refine the matching by establishing a *certainty ranking*, which assesses the confidence in an event based on its proximity to start or complete transitions.

Definition 4 (Certainty Ranking). *Let $TLM_a=(ss^a, T^a s^a_{init} s^a_{succ}, S^a_{active}, t^a_s, t^a_c)$ for any $a \in \mathcal{A}$. Let $t_1, t_2 \in T^a$. We define a function, denoted as $rnk_a\colon \mathcal{T}^a \times \mathcal{T}^a \not\to \mathbb{N}$, where, if $t_1 \prec t_2 \vee t_2 \prec t_1$, $rnk_a(t_1,t_2) = \min(\{(|\sigma|-1) \mid t_1 \prec t_2\colon \sigma \in \mathrm{P}(t_1,t_2) \wedge t_2 \prec t_1\colon \sigma \in \mathrm{P}(t_2,t_1)\})$; otherwise, $rnk_a(t_1,t_2) = \perp$.*

Let L be a log. For any $\sigma \in L$, let $G=(V_1, V_2, E, \lambda, w)$, where $V_1=\{\sigma(i) | 1\leq i\leq|\sigma|\}$ and V_2 is constructed the same. The labeling function λ assigns positions in σ, and the edges E and weight function w are defined similarly. We incorporate the certainty ranking for the matching, defined as $rnk_s\colon V_1 \not\to \mathbb{N}$ for V_1, where $\forall v \in V_1$, given $e=\sigma(\lambda(v))$, $rnk_s(v) = \max(\{rnk_{\#_{act}(e)}(t_s^{\#_{act}(e)}, t) | t \in T^{\#_{act}(e)}\}) + 1 - rnk_{\#_{act}(e)}(t_s^{\#_{act}(e)}, \#_{tr}(e))$; likewise, $rnk_c\colon V_2 \not\to \mathbb{N}$, where $\forall v \in V_2$, given $e=\sigma(\lambda(v))$, $rnk_c(v) = \max(\{rnk_{\#_{act}(e)}(t_c^{\#_{act}(e)}, t) | t \in T^{\#_{act}(e)}\}) + 1 - rnk_{\#_{act}(e)}(\#_{tr}(e), t_c^{\#_{act}(e)})$. We match the events for the maximal certainty by assigning $x_{(v_1,v_2)} \in \{0,1\}$ to each $(v_1,v_2) \in E$ s.t. $\sum_{(v_1,v_2)\in E} w_{(v_1,v_2)} \cdot (rnk_s(v_1) \cdot rnk_c(v_2)) \cdot x_{(v_1,v_2)}$ is maximal, subjected to $\forall 1 \leq i \leq |\sigma|\colon \sum_{(v'_1,v'_2)\in\{(v_1,v_2)\in E | \lambda(v_1)=i \vee \lambda(v_2)=i\}} x_{(v'_1,v'_2)} \leq 1$. The same example, incorporating certainty, is illustrated in Fig. 7b, with a different matching compared to those obtained through naïve matching.



(a) Naïve Matching (b) Matching with uncertainty, with the ranking annotated

Fig. 7. Comparison of matching based on σ_{103} from Fig. 6: Upper nodes represent V_1 and lower nodes V_2 in a bipartite graph. Nodes are color-coded by activity (blue for A, green for C) and annotated with the transition and trace position.

Activity Instance Refinement. We identify activity instances by aligning every trace with the TLM through alignment [3].[1] For each matching, we *anchor* the boundary events by adjusting the cost of log moves and identify other unmatched events such that the entire collection forms a complete activity instance. Figure 8 shows the correlation among events that constitute activity instances, based on the matching presented in Fig. 7. The naïve matching results in all events assigned to activity instances, whereas the matching with uncertainty identifies the complete event of activity A as noise.

5 Evaluation

We compare the proposed methods with FIFO and alignment. Figure 10 depicts the process for each method. Since FIFO only considers start and complete events, we apply the refinement described in Sect. 4.3. For alignment, we extract event sequences within a trace that conform to the TLM without any prior matching using [3]. Events of an activity within a trace are repeatedly *mapped* to the TLM until no start event is followed by a complete event of the same activity within the trace; subsequently, we discard aligned event sequences that do not contain a start followed by a complete, as per the definition of an activity instance. All methods use the same preprocessed logs.[2]

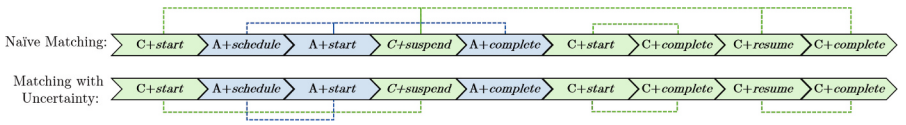


Fig. 8. Activity instance identification for σ_{103}. Events of the same activity instance are connected with dashed lines to show their correlation.

[1] Since the implementation requires a Petri net, we convert the standard TLM into a Petri net based on transitions with only active transitions mandatory, as presented in Fig. 9.

[2] Due to space constraints, a subset of results is presented. Implementation and complete results are available at https://git.rwth-aachen.de/chiaoyun.li/activity-instance-identification.

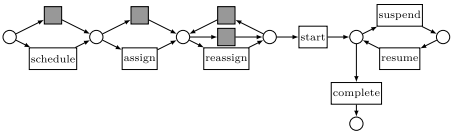


Fig. 9. Petri net of Fig. 3

Fig. 10. Methods to identify activity instances for comparison

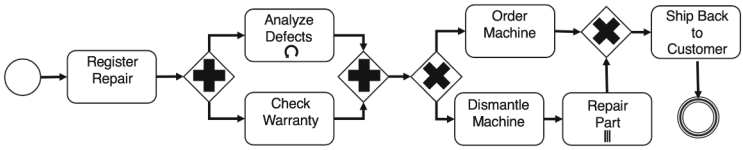


Fig. 11. A process for equipment repair and maintenance service, where *Register repair* and *Order machine* are atomic, while the others are non-atomic and follow the standard TLM in Fig. 3.

We evaluate our approach using a synthetic event log simulated from the process model in Fig. 11. To assess robustness, we introduce noise by inserting 10%, 20%, and 30% noise into the log. Noise events are generated by randomly selecting a case, an activity, a timestamp, and a transition in the TLM of the corresponding activity. We randomly remove events in 5% increments up to 30% to assess their robustness to missing events. Following the assumption from [7], we expect instances of the same activity exhibit similar execution durations. Thus, we report the average normalized Earth Mover's Distance (EMD) of execution times to compare the method's effectiveness from a temporal perspective [20]. The results are reported in Table 1, showing a general decline in performance as noise increases. FIFO achieves higher accuracy compared to other methods, with naïve BGM ranking next, while naïve BGM demonstrates the best recall. The F1-Score reveals that as noise increases, performance shifts from favoring FIFO to naïve BGM, highlighting the robustness of the proposed method. The EMD results further validate that naïve BGM is more resilient to noise. Overall, FIFO and naïve BGM excel in different aspects, followed by alignment, while BGM with uncertainty performs the worst. Figure 12 illustrates the methods' robustness to missing events. Interestingly, while the BGM with uncertainty remains the lowest performance, its robustness improves as the ratio of missing events increases.

We assess the practical applicability of the methods on two publicly available logs from the loan application process [8,9] using the standard TLM and a case study (CS) with sensor data collected from devices distributing packages. In the case study, a domain-specific TLM was developed in collaboration with experts, reflecting the rankings established in [16]. Since no ground-truth for event correlation is available, accuracy and recall cannot be computed. Hence, we evaluate the effectiveness based on the number of activity instances detected and the EMD of the execution times between FIFO and other methods, selected as the baseline due to its common use [14,22] and the accuracy it achieved with the synthetic log previously.

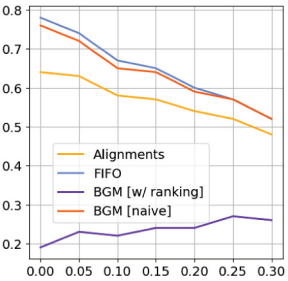
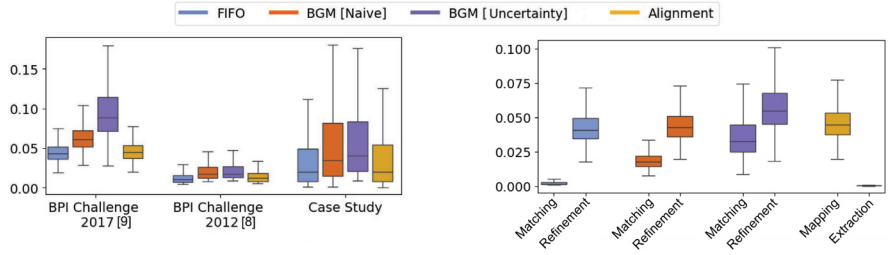


Fig. 12. F1-Score by percentage of missing events

Table 1. Metrics based on synthetic log with increasing noise thresholds

	Accuracy				Recall				F1-Score				EMD			
	0	0.1	0.2	0.3	0	0.1	0.2	0.3	0	0.1	0.2	0.3	0	0.1	0.2	0.3
FIFO	**0.83**	**0.81**	**0.79**	**0.77**	0.74	0.72	0.7	0.68	**0.78**	**0.76**	**0.74**	0.72	**0.01**	0.03	**0.03**	0.05
Alignment	0.58	0.57	0.56	0.55	0.71	0.71	0.7	0.69	0.64	0.63	0.62	0.61	0.05	0.06	0.06	0.08
BGM [Naïve]	0.78	0.76	0.75	0.74	**0.75**	**0.73**	**0.72**	**0.71**	0.76	0.75	0.73	**0.73**	**0.01**	**0.02**	**0.03**	**0.04**
BGM [Uncertainty]	0.64	0.55	0.5	0.45	0.11	0.1	0.1	0.09	0.19	0.17	0.16	0.16	0.24	0.14	0.13	0.13

Similar to the experiments conducted on the synthetic log, FIFO and naïve BGM yield comparable results in the number of instances extracted, while BGM with uncertainty identifies a greater number of instances. Further analysis reveals that the execution of activities such as *W_call after offers* and *W_Call incomplete files* are often captured with only *start* or *suspend* without corresponding *complete* events in [9], indicating the execution of activities with missing events. Regarding the EMD, alignment achieves the lowest values for both [9] and the



(a) Total runtime (b) Runtime per step on [9]

Fig. 13. Runtime per trace in seconds

Table 2. Number of activity instances

	[8]	[9]	CS
FIFO	12,744	12,744	111,080
Alignment	12,524	12,524	110,420
BGM [Naïve]	12,744	12,744	111,080
BGM [Uncertainty]	**12,828**	**12,828**	**131,624**

Table 3. Average EMD to FIFO

	[8]	[9]	CS
Alignment	0.0235	**0.0012**	**0**
BGM [Naïve]	**0**	0.0047	**0**
BGM [Uncertainty]	0.2159	0.0397	0.0062

case study. In contrast, BGM with uncertainty results in larger EMD, suggesting ineffective matching, consistent with findings from previous experiments. Hence, although BGM with uncertainty can extract more instances despite the missing events, these results highlight the need for improvements in the formulation (Tables 2 and 3).

Lastly, we assess the efficiency by measuring the runtime per trace, as illustrated in Fig. 13a. FIFO proves to be the most efficient due to its straightforward approach. In contrast, BGM with uncertainty, while extracting the most activity instances, is the slowest one. Figure 13b breaks down the runtime into the second and third steps of each method. It shows that the refinement is the most time-consuming step, primarily due to the application of alignment.

Limitations and Threats to Validity. Most public event logs lack attributes to correlate events within a trace, limiting the reliability of identified activity instances due to the absence of ground truth. In addition, alignment introduces a degree of nondeterminism, affecting result consistency. The current method, which formulates matching as an ILP problem, systematically pairs start and complete events, but the specific formulation significantly influences the outcome. For instance, matches between events with one position apart are often prioritized, highlighting the impact of the distance-weighting mechanism. Alternative formulations could adjust this emphasis. The formulation is crucial and may be tailored to applications, suggesting aspects for refinement and validation.

6 Conclusion

Process mining emerged as a technology to uncover issues in process operations through event logs. However, while activities occur over time, information systems often log their executions as discrete events, yielding a discrepancy between actual execution and the logged data. Many real-world event logs lack information to correlate events within an activity instance, leading to biased process mining results. Existing methods often rely on arbitrary assumptions and fail to account for all relevant events. Therefore, we propose a method for identifying activity instances. Anchoring on the start and complete, we propose two ways of matching: one where all events are considered complete and another where a certain level of uncertainty is introduced. The matching establishes the boundary and the number of activity instances and, subsequently, alignment based on

the matches and the associated TLM identifies the activity instances. We assess the effectiveness of our method by comparing it with existing approaches that we extend. We evaluate its robustness against noise, performance metrics, and runtime efficiency. For the real-world logs lacking ground truth, we measure effectiveness based on the number of activity instances extracted and conduct further analysis of the results. The findings indicate that the proposed naïve solution is both robust and effective. In contrast, while the method incorporating uncertainty exhibits increasing robustness against missing events, it requires additional refinement. Future work will focus on the automatic discovery of TLMs and enhance scalability. Additionally, investigating various matching formulations with event data of different applications may further validate and refine the method, improving its robustness and applicability.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer (2016)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining Knowl. Discov. **2**, 182–192 (2012)
3. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-based fitness in conformance checking. In: ACSD, pp. 57–66 (2011)
4. Alharbi, A., Bulpitt, A., Johnson, O.A.: Towards unsupervised detection of process models in healthcare. In: MIE. vol. 247, pp. 381–385 (2018)
5. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Inf. Syst. **46**, 123–139 (2014)
6. Baier, T., Rogge-Solti, A., Mendling, J., Weske, M.: Matching of events and activities: an approach based on behavioral constraint satisfaction. In: SAC, pp. 1225–1230 (2015)
7. Bayomie, D., Ciccio, C.D., Mendling, J.: Event-case correlation for process mining using probabilistic optimization. Inf. Syst. **114**, 102167 (2023)
8. van Dongen, B.: BPI challenge 2012 (2012)
9. van Dongen, B.: BPI challenge 2017 (2017)
10. Fischer, D.A., Goel, K., Andrews, R., van Dun, C., Wynn, M.T., Röglinger, M.: Enhancing event log quality: detecting and quantifying timestamp imperfections. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 309–326. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_18
11. Gunther, C.W., Verbeek, H.: Xes - standard definition (2014)
12. ter Hofstede, A.H.M., et al.: Process-data quality: the true frontier of process mining. ACM J. Data Inf. Qual. **15**, 29:1–29:21 (2023)
13. van Hulzen, G., Li, C.-Y., Martin, N., van Zelst, S., Depaire, B.: Mining context-aware resource profiles in the presence of multitasking. Artif. Intell. Medicine **134**, 102434 (2022)
14. Klijn, E.L., Fahland, D.: Performance mining for batch processing using the performance spectrum. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 172–185. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_15
15. de Leoni, M., Dündar, S.: Event-log abstraction using batch session identification and clustering. In: SAC '20: ACM/SIGAPP, pp. 36–44 (2020)

16. Li, C.-Y., et al.: Rectify sensor data in IoT: A case study on enabling process mining for logistic process in an air cargo terminal. In: CoopIS, vol. 14353, pp. 293–310 (2023)
17. Li, C.-Y., van Zelst, S.J., van der Aalst, W.M.P.: A generic approach for process performance analysis using bipartite graph matching. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 199–211. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_17
18. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: From low-level events to activities - A pattern-based approach. EMISA **37**, 47–48 (2017)
19. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: Guided process discovery - a pattern-based approach. Inf. Syst. **76**, 1–18 (2018)
20. Pele, O., Werman, M.: Fast and robust earth mover's distances. In: ICCV, pp. 460–467 (2009)
21. Rozinat, A., de Jong, I., Günther, C.W., van der Aalst, W.M.P.: Process mining applied to the test process of wafer scanners in ASML. IEEE Trans. Syst. Man Cybern. Part C **39**, 474–479 (2009)
22. Suriadi, S., Wynn, M.T., Xu, J., van der Aalst, W.M.P., ter Hofstede, A.: Discovering work prioritisation patterns from event logs. Decis. Support Syst. **100**, 77–92 (2017)
23. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Event abstraction for process mining using supervised learning techniques. In: IntelliSys. vol. 15, pp. 251–269 (2016)
24. Wand, Y., Wang, R.Y.: Anchoring data quality dimensions in ontological foundations. Commun. ACM **39**, 86–95 (1996)