May I Take Your Order?
On the Interplay Between Time and Order in Process Mining

Wil M. P. van der Aalst[1,2(✉)] and Luis Santos[1]

[1] Process and Data Science (Informatik 9), RWTH Aachen University, Aachen, Germany
`wvdaalst@pads.rwth-aachen.de`

[2] Fraunhofer-Institut für Angewandte Informationstechnik (FIT), Sankt Augustin, Germany

Abstract. Process mining starts from event data. The ordering of events is vital for the discovery of process models. However, the timestamps of events may be unreliable or imprecise. To further complicate matters, also causally unrelated events may be ordered in time. The fact that one event is followed by another does not imply that the former causes the latter. This paper explores the relationship between time and order. Moreover, it describes an approach to preprocess event data having timestamp-related problems. This approach avoids using accidental or unreliable orders and timestamps, creates partial orders to capture uncertainty, and allows for exploiting domain knowledge to (re)order events. Optionally, the approach also generates interleavings to be able to use existing process mining techniques that cannot handle partially ordered event data. The approach has been implemented using ProM and can be applied to any event log.

Keywords: Process mining · Event data · Partial orders · Uncertainty

1 Introduction

Most process mining techniques require the events within a case to the totally ordered [2]. For example, nearly all discovery techniques convert the event log into a multiset of traces where each trace is a sequence of activities. To order the events within a case, typically timestamps are used. However, the timestamps may be unreliable or too coarse-grained. Consider, for example, a nurse taking a blood sample from a patient at 16.55 but recording this into the hospital's information system at 17.55 when her shift ends (event e_1). At 17.15, the patient's insurance company approved the operation and this was automatically recorded (event e_2). The same patient also had an X-ray in the evening, but only the date is recorded (event e_3). In this example, the real ordering of events was $\langle e_1, e_2, e_3 \rangle$, but in the event log they may appear as $\langle e_3, e_2, e_1 \rangle$. Event e_1 happened before e_2 but was recorded one hour later. Event e_3 was the last event, but because only the date was recorded, it appeared to have happened at time 00.00. Moreover, events e_1 and e_2 were fully unrelated, so why consider the temporal order? The approval was triggered by a request submitted two days before. Healthcare data are notorious for having data quality problems [10]. However, such issues can be found in any domain [3,11,15].
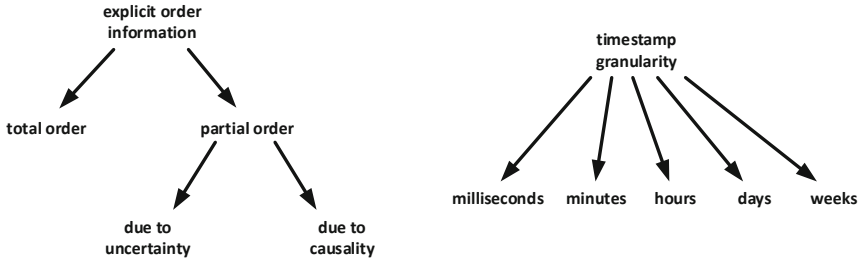
Fig. 1. We assume that events may have explicit order information (left) and have a timestamp (right). However, the ordering is partial and the timestamps can be coarse-grained.

In this paper, we assume that events are *partially ordered* and have a *timestamp* (see Fig. 1). This allows us to reason about the problems just mentioned. Given a set of events E, we assume a strict partial order $\prec_o \subseteq E \times E$. $e_1 \prec_o e_2$ means that event e_1 is before event e_2. $\pi_{time}(e_1)$ and $\pi_{time}(e_2)$ are the timestamps of both events. We assume that events are recorded at a certain granularity, e.g., milliseconds, seconds, hours, days, weeks, months, or years. Events may have more fine-grained timestamps, but we map these onto the chosen level of granularity. For example, "19-05-2021:17.15.00" and "19-05-2021:17.55" are both mapped onto "19-05-2021" when using days as a granularity.

As mentioned, next to timestamps at a selected granularity level, we also assume a partial order on events. Such a partial order can also be more coarse-grained or more fine-grained. One extreme is that the events are totally ordered, i.e., for any two events e_1 and e_2: $e_1 \prec_o e_2$ or $e_1 \succ_o e_2$. Another extreme is that that no two events are ordered ($e_1 \not\prec_o e_2$ or $e_1 \not\succ_o e_2$). The latter case (events are unordered) is similar to assuming that all events have the same coarse-grained timestamp (e.g., same year).

To better explain the problem, we consider the BPMN (Business Process Model and Notation) model shown in Fig. 2 for handling requests for compensation within an airline. Customers may request compensation for various reasons, e.g., a delayed or canceled flight. The process starts by registering the request (*reg*). After this, three checks need to be performed. The customer's ticket and history are checked for all cases, i.e., activities (*ct*) and (*ch*) need to be performed for all requests. There is a choice
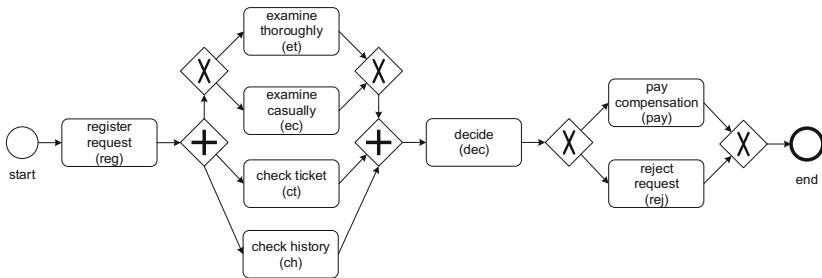

Fig. 2. A BPMN model having 4 partially-ordered runs and $2 \times 2 \times 3! = 24$ sequential runs.

Fig. 3. The four partially-ordered runs of the BPMN model in Fig. 2.

between a thorough examination (et) and a casual examination (ec). After this, a decision is made (dec) and the request is rejected (rej) or some compensation is paid (pay). Examples of sequential runs are $\langle reg, et, ct, ch, dec, pay \rangle$, $\langle reg, ct, ec, ch, dec, rej \rangle$, and $\langle reg, ch, ct, et, dec, pay \rangle$. In total there are $2 \times 2 \times 3! = 24$ sequential runs. Note that in each run there are three concurrent activities (ct, ch, and either et or ec). These can be interleaved in $3! = 6$ ways.

There are only $2 \times 2 = 4$ partially-ordered runs. These are depicted in Fig. 3. Note that the four partially-ordered runs do not need to specify the ordering of concurrent activities. Consider the scenario with 10 concurrent activities; there is just one partially-ordered run, but there are $10! = 3628800$ sequential runs. Figure 3 helps to understand why partial orders are considered in process mining and many other analysis approaches.

Table 1. A fragment of an event log where the timestamps of events are problematic.

Event id	Properties					
	Case id	Activity	Timestamp	Resource	Cost	...
36533	9901	Register request	19-05-2021:11.02.55	Sarah	50	...
36534	9901	Check ticket	19-05-2021:13.02	John	25	...
36535	9902	Register request	19-05-2021:13.02	Sarah	50	...
36536	9902	Check history	20-05-2021:00.00.00	Pete	45	...
36537	9901	Check history	20-05-2021:00.00.00	Pete	45	...
36538	9901	Examine casually	20-05-2021:08.55.34	Mary	55	...
36539	9902	Check ticket	20-05-2021:09.11.21	John	25	...
36540	9902	Examine thoroughly	20-05-2021:10.55	Harry	55	...
36541	9901	Decide	21-05-2021	Angela	55	...
36542	9902	Decide	21-05-2021	Angela	75	...
36543	9902	Reject request	22-05-2021:14.12.45	Sarah	20	...
36544	9901	Pay compensation	22-05-2021:16.52.37	Sarah	150	...
...	...	...	...	...	...	...

Table 1 shows a fragment of an event log corresponding to the BPMN model in Fig. 2. Process discovery techniques aim to learn a process model based on such data. If we assume the events to be sorted based on the identifier in the first column, then case 9901 corresponds to sequential run $\langle reg, ct, ch, ec, dec, pay \rangle$ and case 9902 corresponds to sequential run $\langle reg, ch, ct, et, dec, rej \rangle$. However, a closer inspection of the timestamp column suggests that there several problems. Some events have a precision in seconds, others in minutes, or even days. There are also timestamps of the form "20-05-2021:00.00.00" which suggests that times are sometimes rounded to days. Moreover, we may know that some timestamps show the time of recording and not the actual event. At the same time, we may know that the registration activity (reg) always happens before the check activities.

When timestamps are unreliable or imprecise, like in Table 1, we cannot use them as-is. One approach is to make the timestamps more coarse-grained (e.g., just consider the day). This automatically leads to partially ordered traces. Moreover, there may be explicit information that reveals explicit causal relations. For example, when the concurrent activities do not share any information. We may know that ct, ch, ec, and et use only data collected in ref, but that dec uses the outcomes of the three checks. Such causal dependencies can be derived based on data-flow analysis or explicit domain knowledge, e.g., a payment is always preceded by a decision. As Fig. 1 shows, partial orders can be used to express either *uncertainty* or *explicit causality* (i.e., partial orders have a dual interpretation).

Next to discussing the relationship between time and order, we present a concrete preprocessing approach implemented in ProM (contained in the *PartialOrderVisualizer* package that can be downloaded from `promtools.org`). The approach uses a *time aggregator* and *tiebreaker* to create a *partially-ordered* event log. Moreover, it is possible to create a *k-sequentialization* of the partially-ordered event log to be able to apply conventional approaches.

The remainder of this paper is organized as follows. Section 2 presents related work and Sect. 3 provides a theoretical foundation to reason about the relationship between time and order. Section 4 presents our preprocessing approach, followed by implementation details and an example (Sect. 5). Section 6 concludes the paper.

2   Related Work

For an overview of process mining techniques, we refer to [2]. See [5] for conformance checking and [14] for large-scale applications in organizations such as BMW, Uber, Siemens, EDP, ABB, Bosch, and Telekom.

Recently, many papers on data quality in process mining were published [3,11,15]. Earlier [2,10] already provided a framework for data quality issues and guidelines for logging. Timestamp-related data quality problems are seen as one of the main roadblocks in process mining.

Explicit uncertainty is considered in the work of Pegoraro et al. [12,13]. Event logs are annotated with explicit uncertainty and this information is used when discovering process models or checking conformance. For example, timestamps of events have upper and lower bounds and conformance checking yields optimistic and pessimistic bounds for the actual fitness.

Partial orders are a well-studied topic in modeling checking and concurrency theory. Partially-ordered causal runs are part of the standard true-concurrency semantics of Petri net [7]. See [4] for an example of a synthesis technique using partial orders as input. There are a few process mining techniques that start from partial orders, e.g., the conformance checking technique in [9] and the process discovery technique in [8].

In [1] techniques for partial order resolution are presented. These aim to convert a strict weak ordering into a probability distribution over all corresponding total orders. In [6] also the "same-timestamp problem" is addressed, again aiming at creating total orders. It is impossible to list all partial-order-based approaches here. Moreover, the goal of this paper is not to present new conformance checking or process discovery techniques. Instead, we provide a framework to reason about the relation between order and time, and the corresponding challenges.

In this paper, we focus on the preprocessing of event data while using standard process mining techniques. The main contribution is a discussion on the interplay between time and ordering and a concrete preprocessing tool implemented in ProM. Obviously, our framework can be combined with existing partial-order-based techniques such as [1,4,6,8,9].

3   On the Interplay Between Time and Order in Process Mining

In this section, we define event logs that may have both explicit ordering information and timestamps (possibly rounded to hours, days, or weeks). We relate such event logs to the simplified event logs typically used as input for process discovery.

3.1   Event Logs with Time and Order

We first define universes for events, attribute names, values, activities, timestamps, and attribute name-value mappings. Attribute name-value mappings will be used to assign at least a case, activity, and timestamp to each event.

Definition 1 (Universes). *$\mathcal{E}$ is the universe of event identifiers. $\mathcal{N}$ is the universe of attribute names with $\{case, act, time\} \subseteq \mathcal{N}$, $\mathcal{V}$ is the universe of attribute values, $\mathcal{C} \subseteq \mathcal{V}$ is the universe of case identifiers, $\mathcal{A} \subseteq \mathcal{V}$ is the universe of activity names, $\mathcal{T} \subseteq \mathcal{V}$ is the universe of totally-ordered timestamps, and $\mathcal{M} \subseteq \mathcal{N} \nrightarrow \mathcal{V}$ is the universe of attribute name-value mappings such that for any $m \in \mathcal{M}$: $\{case, act, time\} \subseteq dom(m)$, $m(case) \in \mathcal{C}$, $m(act) \in \mathcal{A}$, and $m(time) \in \mathcal{T}$. For any $n \in \mathcal{N}$ we write $m(n) = \bot$ if $n \notin dom(m)$.*

The properties of an event are described by an attribute name-value mapping that provides at least a case identifier, activity name, and timestamp. Moreover, events may have an explicit order next to timestamp information.

Definition 2 (Event Log). *An event log* $L = (E, \pi, \prec_o)$ *consists of a set of events* $E \subseteq \mathcal{E}$, *a mapping* $\pi \in E \rightarrow \mathcal{M},$[1] *and* $\prec_o \subseteq E \times E$ *such that* $(E, \prec_o)$ *is a strict partial order (i.e., irreflexive, transitive, and asymmetric).*[2]

Table 1 shows a fragment of a larger event log. Consider the first event in the table: $e = 36533$, $\pi_{case}(e) = 9901$, $\pi_{act}(e) = register\ request$, $\pi_{time}(e) = 19\text{-}05\text{-}2021{:}11.02.55$, $\pi_{resource}(e) = Sarah$, and $\pi_{cost}(e) = 50$. Table 1 does not define an explicit order. Possible interpretations are that $\prec_o = \emptyset$ (no order) or a total order based on the order in the table, i.e., $e_1 \prec_o e_2$ if the row corresponding to e_1 appears before the row corresponding to e_2. However, $\prec_o$ may also be based on domain knowledge or data-flow analysis (events can only use a data value produced by an earlier event).

Definition 3 (Notations). *Let* $L = (E, \pi, \prec_o)$ *be an event log.*

- $A(L) = \{\pi_{act}(e) \mid e \in E\}$ *are the activities in* L, $C(L) = \{\pi_{case}(e) \mid e \in E\}$ *are the cases in* L, *and* $E{\upharpoonright}_c = \{e \in E \mid \pi_{case}(e) = c\}$ *are the events of case* $c \in C(L)$.
- $\prec_t = \{(e_1, e_2) \in E \times E \mid \pi_{time}(e_1) < \pi_{time}(e_2)\}$ *is the strict partial order based on the timestamps,*
- $\prec_{ot} = \prec_o \cup \prec_t$ *is the union of the strict partial orders* $\prec_o$ *and* $\prec_t$.
- *If two events* $e_1, e_2 \in E$ *are unordered with respect to* $\prec_o$ *(i.e.,* $e_1 \not\prec_o e_2$ *and* $e_1 \not\succ_o e_2$), *we write* $e_1 \sim_o e_2$. *Similarly,* $e_1 \sim_t e_2 \Leftrightarrow e_1 \not\prec_t e_2 \wedge e_1 \not\succ_t e_2$, *and* $e_1 \sim_{ot} e_2 \Leftrightarrow e_1 \not\prec_{ot} e_2 \wedge e_1 \not\succ_{ot} e_2$.

It is easy to verify that also $(E, \prec_t)$ is a strict partial order (i.e., irreflexive, transitive, and asymmetric). $\sim_o$, $\sim_t$, and $\sim_{ot}$ are reflexive and symmetric by construction. Note that $e_1 \sim_t e_2$ if an only if $\pi_{time}(e_1) = \pi_{time}(e_2)$.

3.2 Consistency

The relation $\prec_{ot}$, which combines $\prec_o$ and $\prec_t$, does not need to be a strict partial order. For example, e_1 happens before e_2 according to $\prec_o$, but e_2 happens before e_1 according to $\prec_t$. Because both ordering relations disagree, $\prec_{ot}$ is not asymmetric. Therefore, we introduce the notion of *consistency*.

Definition 4 (Consistent). *An event log* $L = (E, \pi, \prec_o)$ *is consistent if for any* $e_1, e_2 \in E$: $e_1 \prec_o e_2$ *implies* $\pi_{time}(e_1) \leq \pi_{time}(e_2)$.

This can also be formulated as follows (using transposition): $e_1 \not\prec_o e_2$ or $e_1 \not\succ_t e_2$, for any $e_1, e_2 \in E$. Hence, it is impossible that $e_1 \prec_o e_2$ and $e_1 \succ_t e_2$ hold at the same time. Since both orderings are not conflicting and $\prec_t$ is also a strict weak ordering, the combination yields a strict partial order.

---

[1] We use the shorthand $\pi_n(e) = \pi(e)(n)$. Note that $\pi_{case}(e)$, $\pi_{act}(e)$, and $\pi_{time}(e)$ denote the case, activity, and timestamp of an event $e \in E$.

[2] For any $e, e_1, e_2, e_3 \in E$: $e \not\prec_o e$ (irreflexivity), if $e_1 \prec_o e_2$ and $e_2 \prec_o e_3$, then $e_1 \prec_o e_3$ (transitivity), and if $e_1 \prec_o e_2$, then $e_2 \not\prec_o e_1$ (asymmetry).
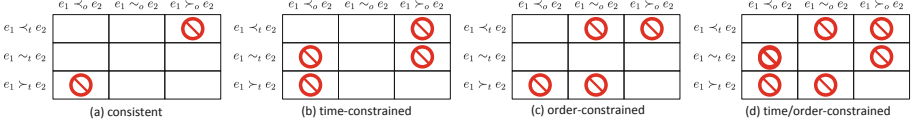
(a) consistent

	$e_1 \prec_o e_2$	$e_1 \sim_o e_2$	$e_1 \succ_o e_2$
$e_1 \prec_t e_2$			🚫
$e_1 \sim_t e_2$			
$e_1 \succ_t e_2$	🚫		

(b) time-constrained

	$e_1 \prec_o e_2$	$e_1 \sim_o e_2$	$e_1 \succ_o e_2$
$e_1 \prec_t e_2$			🚫
$e_1 \sim_t e_2$	🚫		🚫
$e_1 \succ_t e_2$	🚫		

(c) order-constrained

	$e_1 \prec_o e_2$	$e_1 \sim_o e_2$	$e_1 \succ_o e_2$
$e_1 \prec_t e_2$		🚫	🚫
$e_1 \sim_t e_2$			
$e_1 \succ_t e_2$	🚫	🚫	

(d) time/order-constrained

	$e_1 \prec_o e_2$	$e_1 \sim_o e_2$	$e_1 \succ_o e_2$
$e_1 \prec_t e_2$		🚫	🚫
$e_1 \sim_t e_2$	🚫		🚫
$e_1 \succ_t e_2$	🚫	🚫	

Fig. 4. Possible combinations of order and time relations between two events e_1 and e_2 assuming that the event log is (a) consistent, (b) time-constrained ($\prec_o \subseteq \prec_t$), (c) order-constrained ($\prec_t \subseteq \prec_o$), and (d) time-constrained and order-constrained ($\prec_t = \prec_o$).

Proposition 1 (Consistency Implies Strict Partial Ordering). *Let $L = (E, \pi, \prec_o)$ be an event log. $(E, \prec_t)$ is a strict weak ordering (i.e., a strict partial order with negative transitivity[3]), and $(E, \prec_{ot})$ is a strict partial order if L is consistent.*

Proof. $(E, \prec_t)$ is irreflexive, transitive, and asymmetric by construction. Remains to show that negative transitivity holds. Assume that $e_1 \not\prec_t e_2$ and $e_2 \not\prec_t e_3$, i.e., $\pi_{time}(e_1) \geq \pi_{time}(e_2)$ and $\pi_{time}(e_2) \geq \pi_{time}(e_3)$. Hence, $\pi_{time}(e_1) \geq \pi_{time}(e_3)$, i.e., $e_1 \not\prec_t e_3$. Therefore, $(E, \prec_t)$ is a strict weak ordering.

Next, assume that L is consistent. We show that $(E, \prec_{ot})$ is a strict partial order, i.e., for any $e, e_1, e_2, e_3 \in E$: $e \not\prec_{ot} e$ (irreflexivity), if $e_1 \prec_{ot} e_2$ and $e_2 \prec_{ot} e_3$, then $e_1 \prec_{ot} e_3$ (transitivity), and if $e_1 \prec_{ot} e_2$, then $e_2 \not\prec_{ot} e_1$ (asymmetry). Because $\prec_{ot} = \prec_o \cup \prec_t$, irreflexivity follows from $e \not\prec_o e$ and $e \not\prec_t e$. Asymmetry follows directly from consistency: It is impossible that both $e_1 \prec_o e_2$ and $e_1 \succ_t e_2$ hold, so no cycles are introduced. Transitivity relies on the fact that that negative transitivity holds for $\prec_t$. One can use case distinction using the following four cases. (1) $e_1 \prec_t e_2 \ \wedge \ e_2 \prec_t e_3 \Rightarrow e_1 \prec_t e_3 \Rightarrow e_1 \prec_{ot} e_3$. (2) $e_1 \prec_o e_2 \ \wedge \ e_2 \prec_o e_3 \Rightarrow e_1 \prec_o e_3 \Rightarrow e_1 \prec_{ot} e_3$. (3) Assume $e_1 \prec_t e_2 \ \wedge \ e_2 \prec_o e_3 \ \wedge \ e_2 \not\prec_t e_3$. Using consistency, we know that $e_2 \not\succ_t e_3$, hence $e_2 \sim_t e_3$. Since $e_1 \prec_t e_2$ and $e_2 \sim_t e_3$, also $e_1 \prec_t e_3$. (If $e_1 \not\prec_t e_3$, then negative transitivity implies $e_1 \not\prec_t e_3 \ \wedge \ e_3 \not\prec_t e_2 \Rightarrow e_1 \not\prec_t e_2$ leading to a contradiction.) Since $e_1 \prec_t e_3$, also $e_1 \prec_{ot} e_3$. (4) Assume $e_1 \prec_o e_2 \ \wedge \ e_2 \prec_t e_3 \ \wedge \ e_1 \not\prec_t e_2$. Consistency implies $e_1 \not\succ_t e_2$, hence $e_1 \sim_t e_2$. Since $e_1 \sim_t e_2$ and $e_2 \prec_t e_3$, also $e_1 \prec_t e_3$ and $e_1 \prec_{ot} e_3$. Hence, in all four cases transitivity holds, thus completing the proof.

Both $\prec_o$ and $\prec_t$ order events. L is called time-constrained if $\prec_t$ is at least as strict as $\prec_o$, i.e., $e_1 \prec_o e_2$ implies $e_1 \prec_t e_2$. L is order-constrained if $e_1 \prec_t e_2$ implies $e_1 \prec_o e_2$. Figure 4 illustrates these notions.

3.3 Simplified Event Logs

The basic process discovery techniques assume linear traces and only consider the activity names. Therefore, we connect the more involved event log notion $L = (E, \pi, \prec_o)$ (Definition 2) to simplified event logs and standard discovery techniques.

Definition 5 (Simplified Event Log, Process Model, and Discovery Technique). *A trace $\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in \mathcal{A}^*$ is a sequence of activities. $S \in \mathcal{B}(\mathcal{A}^*)$ is a simplified*

---

[3] Recall that negative transitivity means that if $e_1 \not\prec_t e_2$ and $e_2 \not\prec_t e_3$, then $e_1 \not\prec_t e_3$. In a strict weak ordering, incomparability is transitive, i.e., $e_1 \sim_t e_2 \ \wedge \ e_2 \sim_t e_3 \Rightarrow e_1 \sim_t e_3$.

event log, i.e., a multiset of traces. A process model $M \subseteq \mathcal{A}^*$ is a set of traces. A discovery function $disc \in \mathcal{B}(\mathcal{A}^*) \rightarrow \mathcal{P}(\mathcal{A}^*)$ maps an event log onto a process model.

We abstract from the process model notations (e.g., BPMN or Petri nets) and focus on the modeled behavior. This allows us to define a model as a set of possible traces $M \subseteq \mathcal{A}^*$. $M = disc(S)$ is the process model discovered from simplified event log S. A simplified event log is a multiset of traces, e.g., $S = [\langle reg, ct, ch, ec, dec, pay \rangle^3, \langle reg, ch, ct, et, dec, rej \rangle^2]$ contains five traces.

Definition 6 (Sequential Runs). *Let $L = (E, \pi, \prec_o)$ be a consistent event log. For any case $c \in C(L)$, $\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in \mathcal{A}^*$ is a sequential run of c if there is a bijection $f \in \{1, 2, \ldots n\} \rightarrow E{\restriction}_c$ such that $a_i = \pi_{act}(f(i))$ for any $1 \leq i \leq n$ and $e_i \not\succ_{ot} e_j$ for any $1 \leq i < j \leq n$. $seqr_L(c) \subseteq \mathcal{A}^*$ are all sequential runs of case c.*

$\sigma \in seqr_L(c)$ is a trace where each activity refers to an event of case c in such a way that there is a one-to-one correspondence between the elements of σ and $E{\restriction}_c$, and the order does not contradict the combined ordering $\prec_{ot}$. Given a partial order, there may be many linearizations, i.e., total orders that are compatible. In a k-sequentialization of L, we pick k linearizations for each case.

Definition 7 (k-Sequentialization of an Event Log). *Let $L = (E, \pi, \prec_o)$ be a consistent event log. $S = [\sigma_1, \sigma_2, \ldots \sigma_n] \in \mathcal{B}(\mathcal{A}^*)$ is a k-sequentialization of L if (1) there is function $f \in \{1, 2, \ldots n\} \rightarrow L(C)$ such that $\sigma_i \in seqr_L(f(i))$ for any $1 \leq i \leq n$, and (2) $|\{i \in \{1, 2, \ldots, n\} \mid f(i) = c\}| = k$ for any $c \in C(L)$. $seql_k(L) \subseteq \mathcal{B}(\mathcal{A}^*)$ are all possible k-sequentializations of L.*

Definition 7 shows how event log L can be converted into a simplified event log $S \in seql_k(L)$. Each case in L corresponds to k linearizations in S. We leave it open how the linearizations are selected. This can be probabilistic or deterministic. (In our implementation, all linearizations are sampled from $seqr_L(c)$ using equal probabilities).

4   What if Timestamps Are Imprecise?

As described in the introduction, timestamps may be imprecise or partially incorrect. Therefore, we provide transformations of the event log, making time more coarse granular, e.g., all events on the same day have the same timestamp.

Definition 8 (Time Aggregator). *$ta \in \mathcal{T} \rightarrow \mathcal{T}$ is a time aggregator if for any $t_1, t_2 \in \mathcal{T}$ such that $t_1 < t_2$: $ta(t_1) \leq ta(t_2)$.*

For example, $ta(19\text{-}05\text{-}2021{:}13.02) = ta(19\text{-}05\text{-}2021{:}17.55) = 19\text{-}05\text{-}2021{:}00.00$ if a time granularity of days is used, i.e., all timestamps on the same day are mapped onto the same value. By making time more coarse-grained, more events become unordered. These may still be ordered by $\prec_o$, e.g., based on data-flow analysis. Next to $\prec_o$, we may use domain knowledge in the form of a so-called *tiebreaker* to optionally order events having identical coarse-grained timestamps.
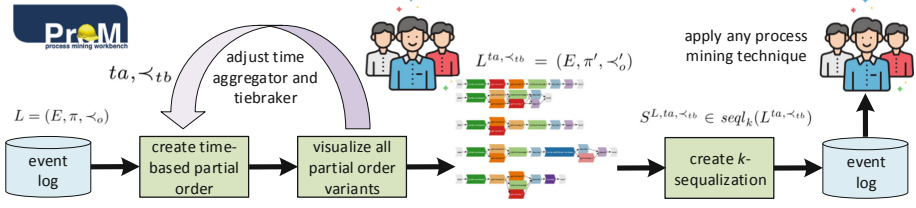
Fig. 5. Overview of the functionality of the *PartialOrderVisualizer* package. The user can change the time granularity and modify the tiebreaker using domain knowledge. Cases with the same partial order are grouped into partial-order variants. These can be sorted and inspected. At any point in time, it is possible to create a regular event log (using k-sequentialization).

Definition 9 (Tiebreaker). *A tiebreaker $\prec_{tb} \subseteq \mathcal{A} \times \mathcal{A}$ is a strict partial order used to order activities having the same aggregated timestamp.*

A tiebreaker adds causal dependencies between events that have the same coarse-granular timestamps and belong to the same case. Using time aggregator ta and tiebreaker $\prec_{tb}$, we can create a new event log $L^{ta,\prec_{tb}}$.

Definition 10 (Preprocessing). *Let $L = (E, \pi, \prec_o)$ be a consistent event log, $ta \in \mathcal{T} \to \mathcal{T}$ a time aggregator, and $\prec_{tb} \subseteq \mathcal{A} \times \mathcal{A}$ a tiebreaker. $L^{ta,\prec_{tb}} = (E, \pi', \prec'_o)$ is the event log after applying the time aggregator ta and tiebreaker $\prec_{tb}$ such that $\pi'_n(e) = \pi_n(e)$ for any $e \in E$ and $n \in \mathcal{N} \setminus \{time\}$, $\pi'_{time}(e) = ta(\pi_{time}(e))$ for any $e \in E$, and $\prec'_o = \prec_o \cup \{(e_1, e_2) \in E \times E \mid \pi_{case}(e_1) = \pi_{case}(e_2) \land \pi'_{time}(e_1) = \pi'_{time}(e_2) \land \pi_{act}(e_1) \prec_{tb} \pi_{act}(e_2)\}$.*

As long as the tiebreaker $\prec_{tb}$ does not contradict $\prec_o$, $\prec'_o$ is a partial order and $L^{ta,\prec_{tb}} = (E, \pi', \prec'_o)$ is a consistent event log.

Hence, we can compute a k-sequentialization of $L^{ta,\prec_{tb}}$ and produce a simplified event log $S^{L,ta,\prec_{tb}} \in seql_k(L^{ta,\prec_{tb}})$. As mentioned before, we leave it open how sequential runs are selected. Using the simplified preprocessed event log, we can apply any process discovery technique to obtain process model $M^{L,ta,\prec_{tb}} = disc(S^{L,ta,\prec_{tb}})$.

5   Implementation

The new ProM package *PartialOrderVisualizer* implements the approach described in Sect. 4 and can be downloaded as part of ProM's nightly builds (http://www.promtools.org/doku.php?id=nightly). It has been implemented as a so-called "visualizer" and can be selected by choosing *Explore Partial Orders (Variants)* in the pull-down menu. The visualizer can be applied to any event log.

Figure 5 shows the main components of the *PartialOrderVisualizer*. Based on time aggregator ta and tiebreaker $\prec_{tb}$ the partial orders are computed and visualized. Initially, ta is set to hours and $\prec_{tb} = \emptyset$. The user can experiment with the time granularity and add ordering constraints. One can inspect partial order variants and details of the corresponding cases. Moreover, the user can create a k-sequentialization of $L^{ta,\prec_{tb}}$. The resulting event log can be analyzed using classical process mining techniques.
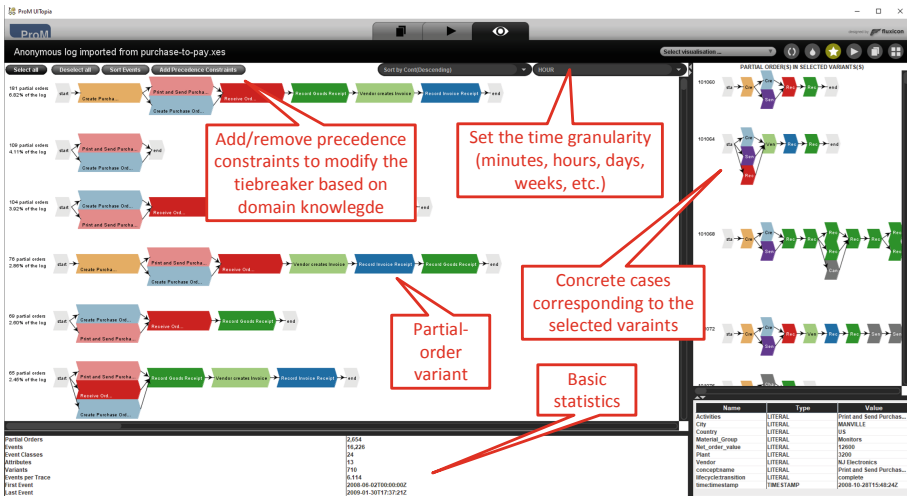
Fig. 6. Data from a Purchase-to-Pay (P2P) process visualized using *PartialOrderVisualizer*. What can be seen is that activities *Create Purchase Order Item* and *Print and Send Purchase Order* often happen in the same hour.

Figure 6 shows the *PartialOrderVisualizer* for the event data of a Purchase-to-Pay (P2P) process with 2,654 cases and 16,226 events. There are 685 trace variants in the original event log. The view shown in Fig. 6 uses a time granularity of one hour leading to a similar number of partial-order variants (i.e., 710). However, one can clearly see that some activities often happen within the same hour. It is possible to add ordering information to make these sequential (if desired). Note that seeing which activities happen in the same period is valuable and provides new insights.

Figure 7 shows the *PartialOrderVisualizer* for the same event data, but now using a time granularity of a week. One can see that more events become unordered, because they happen in the same week. Using this view, we generated a new event log replicating each partially-ordered case 10 times. As expected, this new event log has 26,540 cases and 162,260 events. Interestingly, there are now 8,864 trace variants.

The *PartialOrderVisualizer* has been applied to a range of event logs, e.g., we have used it to analyze the treatment of Covid-19 patients at Uniklinik RWTH Aachen. In this Covid-19 dataset, only the dates are reliable. Naïvely using the ordering in the event log or the recorded timestamps leads to incorrect conclusions.

For the Covid-19 dataset it takes just a few seconds. For a larger data sets like the well-known road fines event log[4], which has over 560.000 events and 150.000 cases, it takes around 10 s (using for example the day, hour, minute, and second abstractions).

What is interesting is that in many applications the number of partially-ordered variants temporarily goes up when coarsening the time granularity. However, by definition,

---

[4] Road Traffic Fine Management Process, 4TU.ResearchData, https://doi.org/10.4121/uuid: 270fd440-1057-4fb9-89a9-b699b47990f5.
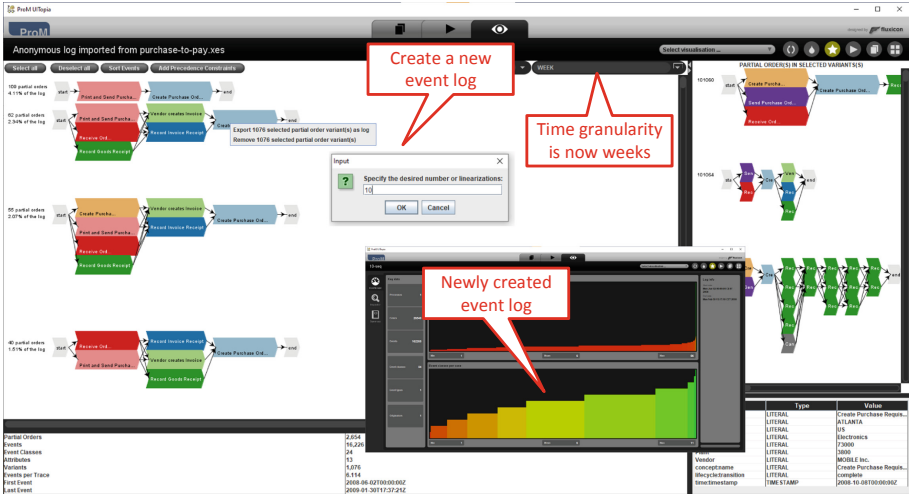
Fig. 7. Another view on the same P2P dataset now using a time granularity of a week. A new event log was created using a k-sequentialization (with $k = 10$).

the number of partially-ordered variants is the smallest when all events are mapped onto the same time period.

6 Conclusion

The contribution of this paper is twofold. On the one hand, we discussed the possible interplay between time and order using an event log definition where events have timestamps and may be ordered by some partial order. Since (rounded) timestamps provide a strict weak ordering, the combination is a partial order (provided that the event log is consistent). On the order hand, we described a new way to preprocess event logs using a time aggregator ta and a tiebreaker $\prec_{tb}$. The time aggregator makes the timestamps more coarse-grained to avoid accidental event ordering due to imprecise or partially incorrect timestamps. The tiebreaker can be used to order events that have the same coarse-grained timestamp (e.g., date). The preprocessed event log can be created and explored using the new *PartialOrderVisualizer* implemented in ProM. Viewing the event log at different time scales provides novel insights and helps to counteract data quality problems. Moreover, the *PartialOrderVisualizer* can also generate a k-sequentialization of the partially-ordered event log. This allows for the application of regular process mining techniques (e.g., discovery and conformance checking).

Although some process mining techniques have been developed for partially-ordered event logs, we feel that more research is needed. Partial orders may be the result of uncertainty or explicit causal information. These need to be treated differently. We also plan to integrate frequent item-set mining into our approach. The fact that certain combinations of activities often happen in the same period can be used to create event logs where high-level events refer to sets of low-level events.

References

1. van der Aa, H., Leopold, H., Weidlich, M.: Partial order resolution of event logs for process conformance checking. Decis. Support Syst. **136**, 113347 (2020)
2. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
3. Andrews, R., van Dun, C.G.J., Wynn, M.T., Kratsch, W., Röglinger, M.K.E., ter Hofstede, A.H.M.: Quality-informed semi-automated event log generation for process mining. Decis. Support Syst. **132**, 113265 (2020)
4. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Synthesis of petri nets from term based representations of infinite partial languages. Fund. Inform. **95**(1), 187–217 (2009)
5. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99414-7
6. Conforti, R., La Rosa, M., ter Hofstede, A.H.M., Augusto, A.: Automatic repair of same-timestamp errors in business process event logs. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 327–345. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_19
7. Desel, J.: Validation of process models by construction of process nets. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 110–128. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_8
8. van Dongen, B.F., Desel, J., van der Aalst, W.M.P.: Aggregating causal runs into workflow nets. In: Jensen, K., van der Aalst, W.M., Ajmone Marsan, M., Franceschinis, G., Kleijn, J., Kristensen, L.M. (eds.) Transactions on Petri Nets and Other Models of Concurrency VI. LNCS, vol. 7400, pp. 334–363. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35179-2_14
9. Lu, X., Fahland, D., van der Aalst, W.M.P.: Conformance checking based on partially ordered event data. In: Fournier, F., Mendling, J. (eds.) BPM 2014. LNBIP, vol. 202, pp. 75–88. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15895-2_7
10. Mans, R., van der Aalst, W.M.P., Vanwersch, R.: Process Mining in Healthcare: Evaluating and Exploiting Operational Healthcare Processes. Springer Briefs in Business Process Management, Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16071-9
11. Martin, N.: Data quality in process mining. In: Fernandez-Llatas, C. (ed.) Interactive Process Mining in Healthcare. Health Informatics, pp. 53–79. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-53993-1_5
12. Pegoraro, M., van der Aalst, W.M.P.: Mining uncertain event data in process mining. In: Carmona, J., Jans, M., La Rosa, M. (eds.) International Conference on Process Mining (ICPM 2019), Aachen, Germany, pp. 89–96. IEEE Computer Society (2019)
13. Pegoraro, M., Uysal, M.S., van der Aalst, W.M.P.: Efficient construction of behavior graphs for uncertain event data. In: Abramowicz, W., Klein, G. (eds.) BIS 2020. LNBIP, vol. 389, pp. 76–88. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53337-3_6
14. Reinkemeyer, L.: Process Mining in Action: Principles, Use Cases and Outlook. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40172-6
15. Suriadi, S., Andrews, R., ter Hofstede, A.H.M., Wynn, M.T.: Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. Inf. Syst. **64**, 132–150 (2017)