Visualizing Trace Variants From Partially Ordered Event Data

Daniel Schuster[1,2][0000−0002−6512−9580], Lukas Schade[2], Sebastiaan J. van Zelst[1,2][0000−0003−0415−1036], and Wil M. P. van der Aalst[1,2][0000−0002−0955−6940]

[1] Fraunhofer Institute for Applied Information Technology FIT, Germany
{daniel.schuster,sebastiaan.van.zelst}@fit.fraunhofer.de
[2] RWTH Aachen University, Aachen, Germany
wvdaalst@pads.rwth-aachen.de

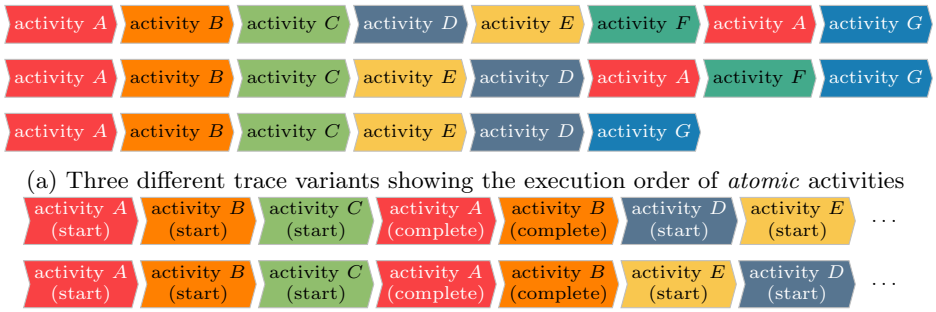
Abstract. Executing operational processes generates event data, which contain information on the executed process activities. Process mining techniques allow to systematically analyze event data to gain insights that are then used to optimize processes. Visual analytics for event data are essential for the application of process mining. Visualizing unique process executions—also called trace variants, i.e., unique sequences of executed process activities—is a common technique implemented in many scientific and industrial process mining applications. Most existing visualizations assume a total order on the executed process activities, i.e., these techniques assume that process activities are atomic and were executed at a specific point in time. In reality, however, the executions of activities are *not* atomic. Multiple timestamps are recorded for an executed process activity, e.g., a start-timestamp and a complete-timestamp. Therefore, the execution of process activities may overlap and, thus, cannot be represented as a total order if more than one timestamp is to be considered. In this paper, we present a visualization approach for trace variants that incorporates start- and complete-timestamps of activities.

Keywords: Process Mining · Visual analytics · Interval order.

1 Introduction

The execution of operational processes, e.g., business and production processes, is often supported by information systems that record process executions in detail. We refer to such recorded information as *event data*. The analysis of event data is of great importance for organizations to improve their processes. *Process mining* [1] offers various techniques for systematically analyzing event data, e.g., to learn a process model, to check compliance, and to obtain performance measures. These insights into the processes can then be used to optimize them.

As in other data analysis applications, *visual analytics* for event data are important in the application of process mining. A state-of-the-art process mining methodology [6] lists *process analytics* including visual analytics as a key component next to the classic fields of process mining: process discovery, conformance checking, and process enhancement.



(a) Three different trace variants showing the execution order of *atomic* activities

(b) Two different trace variants showing the execution order of *non-atomic* activities, i.e, each activity is split into start and complete

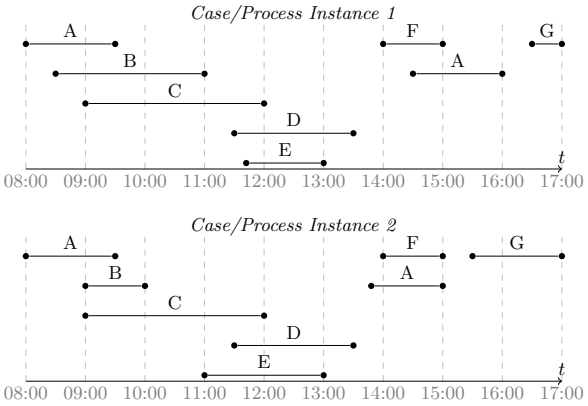
Fig. 1: Classic trace variant visualizations for *(non)-atomic* process activities

A visualization approach that is used across various process mining tools, ranging from industry to scientific tools, is called the *variant explorer*. Consider Figure 1a for an example. In classic trace variant visualizations, a *variant* describes a unique sequence of executed process activities. Thus, a *strict total order* on the contained activities is required to visualize such sequence. Recorded timestamps of the executed activities are usually used for ordering them.

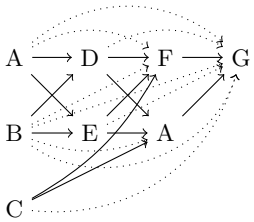
This classic trace variant visualization has two main limitations. (1) Assume atomic process activities, i.e., a single timestamp is recorded for each process activity. A *strict* total order cannot be derived if multiple activities have the same timestamp. In such cases, the sequential visualization, indicating temporal execution, of process activities is problematic because a second-order criteria is needed to obtain a strict total order. (2) In many real-life scenarios, process activities are performed over time, i.e., they are *non*-atomic. Thus, the execution of activities may intersect with each other. Consider Figure 2a for an example. Considering both start- and complete-timestamps, a strict total order cannot be obtained if the executions of activities overlap. The classic trace variant explorer usually splits the activities in start and complete as shown in Figure 1b to obtain atomic activities. However, the parallel behavior of activities is not easily discernible from the visualization. In addition, the first limitation remains.

In this paper, we propose a novel visualization of trace variants to overcome the two aforementioned limitations. We define a variant as an *interval order*, which can be represented as a graph. For instance, Figure 2b shows the interval order of the two process executions shown in Figure 2a. The graph representation of an interval order (cf. Figure 2b) is, however, not easy to read compared to the classic trace variant explorer (cf. Figure 1). Therefore, we propose an approach to derive a visualization from interval orders representing trace variants.

The remainder of this paper is structured as follows. Section 2 presents related work. Section 3 introduces concepts and definitions used throughout this paper. Section 4 introduces the proposed visualization approach. Section 5 presents an experimental evaluation, and Section 6 concludes this paper.



(a) Time plots visualizing activity instances, i.e., each activity has a start-timestamp and a complete-timestamp, executed within two different cases/process instance



(b) Visualization of the corresponding interval order. Vertices represent activity instances. Arcs indicate an ordering between activity instances

Fig. 2: Visualizing partially ordered event data. Each interval shown in Figure 2a, i.e., an activity instance, describes the execution of an activity. $A, \ldots, G$ represent activity labels. Both visualized cases/process instances (Figure 2a) correspond to the same interval order (Figure 2b). Note that we consider two activity instances to be unrelated if they overlap in time

2 Related Work

For a general overview of process mining, we refer to [1]. Note that the majority of process mining techniques assume totally ordered event data. For example, in process discovery few algorithms exist that utilize life cycle information, i.e., more than one timestamp, of the recorded process activities. For instance, the Inductive Miner algorithm has been extended in [9] to utilize start- and complete-timestamps of process activities. Also in conformance checking there exist algorithms that utilize life cycle information, e.g., [10]. A complete overview of techniques utilizing life cycle information is outside the scope of this paper.

In [6], the authors present a methodology for conducting process mining projects and highlight the importance of visual analytics. In [8], open challenges regarding visual analytics in process mining are presented. The visualization of time-oriented event data—the topic of this paper—is identified as a challenge.

The classic variant explorer as shown in Figure 1 can be found in many different process mining tools, e.g., in ProM[3], which is an open-source process mining software tool. In [3], the authors present a software tool to visualize event data. Various visualizations of event data are offered; however, a variant explorer, as considered in this paper, is not available. In [2], the authors present a plugin for ProM to visualize partially ordered event data. The approach considers events

[3] https://www.promtools.org

Table 1: Example of event data

Event-ID	Case-ID	Activity Label	Start-timestamp	Complete-timestamp	Resource	...
1	1	activity A	07/13/2021 08:00	07/13/2021 09:30	staff	...
2	1	activity B	07/13/2021 08:30	07/13/2021 11:00	staff	...
3	1	activity C	07/13/2021 09:00	07/13/2021 12:00	staff	...
4	1	activity D	07/13/2021 11:30	07/13/2021 13:30	staff	...
5	1	activity E	07/13/2021 11:40	07/13/2021 13:00	supervisor	...
6	1	activity F	07/13/2021 14:00	07/13/2021 15:00	manager	...
7	1	activity A	07/13/2021 14:30	07/13/2021 16:00	staff	...
8	1	activity G	07/13/2021 16:30	07/13/2021 17:00	staff	...
9	2	activity A	07/13/2021 08:00	07/13/2021 09:30	staff	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

to be atomic, i.e., an event representing the start and an event representing the completion of an activity are considered to be separate events. Based on a user-selected time granularity, events within the same time segment are aggregated, i.e., they are considered and visualized to be executed in parallel. This offers the advantage that the user can change the visualization depending on how accurately the timestamps are to be interpreted. Compared to our approach, we consider non-atomic activity instances, i.e., we map start and complete events of a process activity to an activity instance. Next, we relate these activity instances to each other instead of atomic events as proposed in [2]. Therefore, both approaches, the one presented in [2] and the one presented in this paper, can coexist and each have their advantages and disadvantages.

3 Preliminaries

In this section, we present concepts and definitions used within this paper.

Event data describes the historical execution of processes. Table 1 shows an example of said event data. Each row corresponds to an event, i.e., in the given example an *activity instance*.[4] For example, the first event, identified by event-id 1, recorded that activity A has been executed from 08:00 until 09:30 at 07/13/2021 within the process instance identified by case-id 1.

In general, activity instances describe the execution of a process activity within a specific case. A *case* describes a single execution of a process, i.e., a process instance, and it is formally a set of activity instances that have been executed for the same case. Activity instances consist of at least the following attributes: an identifier, a case-id, an activity label, a start-timestamp, and a complete-timestamp. Since we are only interested in the order of activity instances within a case and not in possible additional attributes of an activity instance, we define activity instances as a 5-tuple.

[4] Note that in some event logs, the start and the completion of an activity are separate events (i.e., separate rows). Observe that such records are easily transformed to our notion of event data.

Definition 1 (Universes). *$\mathcal{T}$ is the universe of totally ordered timestamps. $\mathcal{L}$ is the universe of activity labels. $\mathcal{C}$ is the universe of case identifiers. $\mathcal{I}$ is the universe of activity instance identifiers.*

Definition 2 (Activity Instance). *An activity instance $(i, c, l, t_s, t_c) \in \mathcal{I} \times \mathcal{C} \times \mathcal{L} \times \mathcal{T} \times \mathcal{T}$ describes the execution of an activity labeled l within the case c. The start-timestamp of the activity's execution is t_s, and the complete-timestamp is t_c, where $t_s \leq t_c$. Each activity instance is uniquely identifiable by i. We denote the universe of activity instances by $\mathcal{A}$.*

Note that any event log with only one timestamp per executed activity can also be easily expressed in terms of activity instances, i.e., $t_s = t_c$. For a given activity instance $a = (i, c, l, t_s, t_c) \in \mathcal{A}$, we define projection functions: $\pi^i(a) = i$, $\pi^c(a) = c$, $\pi^l(a) = l$, $\pi^{t_s}(a) = t_s$, and $\pi^{t_c}(a) = t_c$.

Definition 3 (Event Log). *An event log E is a set of activity instances, i.e., $E \subseteq \mathcal{A}$ such that for $a_1, a_2 \in E \wedge \pi^i(a_1) = \pi^i(a_2) \Rightarrow a_1 = a_2$. We denote the universe of event logs by $\mathcal{E}$.*

For a given event log $E \in \mathcal{E}$, we refer to the set of activity instances executed within a given case $c \in \mathcal{C}$ as a *trace*, i.e., $T_c = \{a \in E \mid \wedge \pi^c(a) = c\}$. As shown in Figure 2a, we can visualize a trace and its activity instances in a time plot.

Note that each activity instance $a = (i, c, l, t_s, t_c) \in \mathcal{A}$ defines an interval on the timeline, i.e., $[t_s, t_c]$. A collection of intervals—in this paper we focus on traces—defines an *interval order*. In general, given two activity instances $a_1, a_2 \in \mathcal{A}$, we say $a_1 < a_2$ iff $\pi^{t_c}(a_1) < \pi^{t_s}(a_2)$. Note that interval orders are a proper subclass of strict partial orders [7]; hence, interval orders satisfy: irreflexivity, transitivity, and asymmetry. Interval orders additionally satisfy the *interval order condition*, i.e., for any $x, y, w, z : x < w \wedge y < z \Rightarrow x < z \vee y < w$ [7].

In this paper, we represent an interval order as a directed, labeled graph that consists of vertices V, representing activity instances, and directed edges $V \times V$, representing ordering relations between activity instances. Figure 2b shows the interval order of the traces shown in Figure 2a. We observe that the first two activity instances labeled with A and B are incomparable to each other because there is no arc from either A to B or vice versa. Thus, the first execution of A and B are executed in parallel, i.e., their intervals overlap. For example, activity C is related to F, G and the second execution of A. Thus, C is executed before F, G and the second execution of A. Next, we formally define the construction of the directed graph representing the interval order of a trace.

Definition 4 (Interval Order of a Trace). *Given a trace $T_c \subseteq \mathcal{A}$, we define the corresponding interval order as a labeled, directed graph (V, E, λ) consisting of vertices V, directed edges $E = (V \times V)$, and a labeling function $\lambda : V \to \mathcal{L}$. The set of vertices is defined by $V = T_c$ with $\lambda(a) = \pi^l(a)$. Given two activity instances $a^1, a^2 \in T$, there is a directed edge $\big(\pi^i(a_1), \pi^i(a_2)\big) \in E$ iff $\pi^{t_c}(a_1) < \pi^{t_s}(a_2)$. We denote the universe of interval orders by $\mathcal{P}$.*

Next, we define the induced interval order.

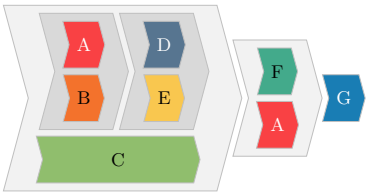


Fig. 3: Proposed visualization for the interval order shown in Figure 2b

Definition 5 (Induced Interval Order). *Given* $(V, E, \lambda) \in \mathcal{P}$. *For* $V' \subseteq V$, *we define the induced interval order, i.e., the induced subgraph,* $(V', E', \lambda') \in \mathcal{P}$ *with* $E' = E \cap (V' \times V')$ *and* $\lambda'(v) = \lambda(v)$ *for all* $v \in V'$.

4 Visualizing Trace Variants

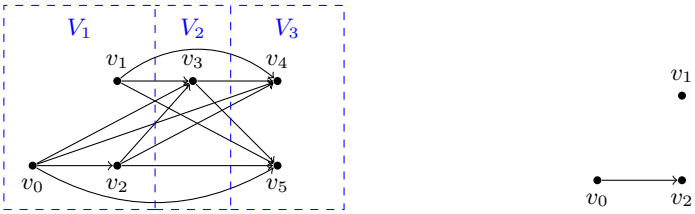
This section introduces the proposed approach to visualize trace variants from partially ordered event data. Section 4.1 introduces the approach, and Section 4.2 proves that the approach is deterministic. Section 4.3 discusses the potential limitations of the approach. Finally, Section 4.4 covers the implementation.

4.1 Approach

The proposed visualization approach of trace variants is based on chevrons, a graphical element known from classical trace variant visualizations (cf. Figure 1). Figure 3 shows an example of the proposed visualization for the interval order given in Figure 2b. The interpretation of a chevron as indicating sequential order is maintained in our approach. Additionally, chevrons can be nested and stacked on top of each other. Stacked chevrons indicate parallel/overlapping execution of activities. Nested chevrons relate groups of activities to each other. In the given example, the first chevron indicates that C is executed in parallel to A, B, D, and E. The two upper chevrons indicate that A and B are executed in parallel, but are executed before D and E, both of which are also executed in parallel.

The proposed approach assumes an interval order, representing a trace variant, as input and *recursively* partitions the interval order by applying cuts to compute the layout of the visualization (cf. Figure 3). In general, a cut is a partition of the nodes of a given interval order. Based on the partition, induced interval orders are derived. Each application of such a cut corresponds to chevrons and their positioning in the final visualization, e.g., stacked or side-by-side chevrons. Nested chevrons result from the recursive manner. Next, we define the computation of the proposed layout, i.e., we define two types of cuts.

An *ordering cut* partitions the activity instances into sets such that these sets can be totally ordered, i.e., all activity instances within a set can be related to all other activity instances from other sets. In terms of the graph representation of an interval order, this implies that all nodes from one partition have a directed edge to all nodes from the other partition(s). We depict an example of an ordering



(a) Interval order and a maximal ordering cut (b) Induced interval order based on V_1

Fig. 4: Example of an ordering cut, i.e., a partition of the nodes into V_1=$\{v_0, v_1, v_2\}$, V_2=$\{v_3\}$, V_3=$\{v_4, v_5\}$, and one corresponding induced interval order for V_1

cut in Figure 4. Note that all nodes in V_1 are related to all nodes in V_2 and V_3. Next, we formally define an ordering cut for an interval order.

Definition 6 (Ordering Cut). *Assume an interval order* $(V, E, \lambda) \in \mathcal{P}$. *An ordering cut describes a partition of the nodes* V *into* $n>1$ *non-empty subsets* $V_1, \ldots, V_n$ *such that:* $\forall 1 \leq i < j \leq n \Big(\forall v \in V_i, v' \in V_j \big((v, v') \in E \big) \Big)$.

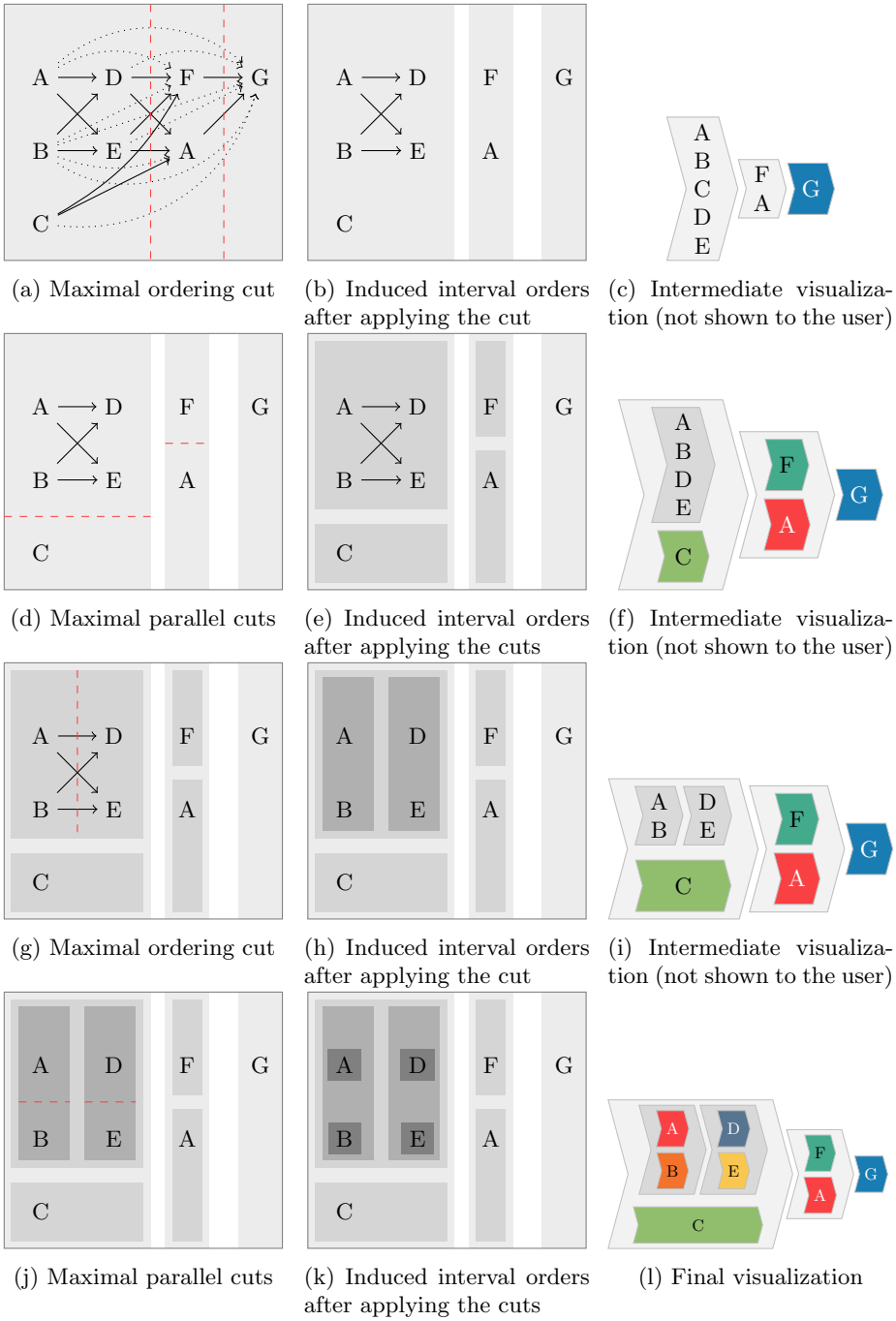
A *parallel cut* indicates that activity instances from one partition overlap in time with activity instances in the other partition(s), i.e., activity instances from different partitions are unrelated to each other. Thus, we are looking for *components* in the graph representation of an interval order.

Definition 7 (Parallel Cut). *Assume an interval order* $(V, E, \lambda) \in \mathcal{P}$. *A parallel cut describes a partition of the nodes* V *into* $n \geq 1$ *non-empty subsets* $V_1, \ldots, V_n$ *such that* $V_1, \ldots, V_n$ *represent connected components of* (V, E, λ), *i.e.,* $\forall 1 \leq i < j \leq n$ $\Big(\forall v \in V_i \forall v' \in V_j \big((v, v') \notin E \wedge (v', v) \notin E \big) \Big)$.

We call a cut *maximal* if n, i.e., the number of subsets, is maximal.

Figure 5 shows an example of the proposed visualization approach. We use the interval order from Figure 2b as input. The visualization approach recursively looks for a maximal ordering or parallel cut. In the example, we initially find an ordering cut of size three (cf. Figure 5a). Given the cut, we create three induced interval orders (cf. Figure 5b). As stated before, each induced interval order created by a cut represents a chevron. In general, an ordering cut indicates the horizontal alignment of chevrons while a parallel cut indicates the vertical alignment of chevrons. Since we found an ordering cut of size three, the intermediate visualization consists of three horizontally-aligned chevrons (cf. Figure 5c). If an induced interval order only consists of one element (e.g., the third induced interval order in Figure 5b), we fill the corresponding chevron with a color that is unique for the given activity label (cf. Figure 5c). As in the classic trace variant explorer, colors are used to better distinguish different activity labels.

We now recursively apply cuts to the induced interval orders. In the first two interval orders, we apply a parallel cut (cf. Figure 5d). The third interval order consists only of one node labeled with G; thus, no further cuts can be applied.



(a) Maximal ordering cut

(b) Induced interval orders after applying the cut

(c) Intermediate visualization (not shown to the user)

(d) Maximal parallel cuts

(e) Induced interval orders after applying the cuts

(f) Intermediate visualization (not shown to the user)

(g) Maximal ordering cut

(h) Induced interval orders after applying the cut

(i) Intermediate visualization (not shown to the user)

(j) Maximal parallel cuts

(k) Induced interval orders after applying the cuts

(l) Final visualization

Fig. 5: Example of recursively applying ordering and parallel cuts to an interval order and the corresponding visualization

Figure 5e shows the induced interval orders after applying the two parallel cuts. As stated before, time-overlapping activity instances are indicated by stacked chevrons. Since both applied parallel cuts have size two, we create two stacked chevrons each within the first and the second chevron (cf. Figure 5f). After another ordering cut (cf. Figure 5g-5i) and two more parallel cuts (cf. Figure 5j), the visualization approach stops because all induced interval orders consist of only one activity instance. Figure 5l shows the final visualization.

4.2 Formal Guarantees

Next, we show that the proposed approach is deterministic, i.e., the same visualization is always returned for the same interval order. We therefore show that different cuts cannot coexist, i.e., either a parallel cut, an ordering cut, or no cut exists in an interval order. Further, we show that maximal cuts are unique.

Lemma 1 (Cuts Cannot Coexist). *In an interval order $(V, E, \lambda) \in \mathcal{P}$ a parallel and an ordering cut cannot coexist.*

Proof. Let $(V, E, \lambda) \in \mathcal{P}$ be an interval order with an ordering cut $V_1, \ldots, V_n$ for some $n \geq 2$. Assume there exists a parallel cut, too, i.e., $V'_1, \ldots, V'_m$ for some $m \geq 2$. For $1 \leq j \leq m$, assume that for an arbitrary $v \in V$ it holds that $v \in V'_j$ such that $v \in V_i$ for some $i \in \{1, \ldots, n\}$. Since an ordering cut exists, we know that $\forall w \in V_{i+1} \cup \ldots \cup V_n \big((v, w) \in E\big)$ and $\forall w' \in V_1 \cup \ldots \cup V_{i-1} \big((w', v) \in E\big)$. Since $V'_1, \ldots, V'_m$ is a parallel cut, i.e., each $V'_k \in \{V'_1, \ldots, V'_m\}$ represents a connected component (Definition 7), also all w and w' must be in V'_j. Hence, $V'_j = \{v\} \cup V_1 \cup \ldots \cup V_{i-1} \cup V_{i+1} \cup \ldots \cup V_n$. Further, since $\forall w' \in V_1 \cup \ldots \cup V_{i-1} \forall w \in V_{i+1} \cup \ldots \cup V_n \big((w', w) \in E \wedge (w', v) \in E \wedge (v, w) \in E\big)$ it follows by Definition 7 that $V'_j = V_1 \cup \ldots \cup V_n = V$. Hence, $\forall V'_k \in \{V'_1, \ldots, V'_m\} \setminus \{V'_j\} (V'_k = \emptyset)$ since $V'_1, \ldots, V'_m$ is a partition of V. This contradicts our assumption that there exists a parallel cut, too. The other direction is symmetrical. □

Since cuts cannot coexist (cf. Lemma 1), one cut is applicable for a given interval order at most. Next, we show that maximal cuts are unique.

Lemma 2 (Maximal Ordering Cuts Are Unique). *If an ordering cut exists in a given interval order $(V, E, \lambda) \in \mathcal{P}$, the maximal ordering cut is unique.*

Proof. Proof by contradiction. Assume an interval order $(V, E, \lambda) \in \mathcal{P}$ having two *different* maximal ordering cuts, i.e., $V_1, \ldots, V_n$ and $V'_1, \ldots, V'_n$.
$\Rightarrow \exists i \in \{1, \ldots, n\} \forall j \in \{1, \ldots, n\} \big(V_i \neq V'_j\big) \Rightarrow V_i \neq V'_i$
$\Rightarrow \exists v \in V_i \cup V'_i \big((v \in V_i \wedge v \notin V'_i) \vee (v \notin V_i \wedge v \in V'_i)\big)$
Assume $v \in V_i \wedge v \notin V'_i$ (the other case is symmetric)
$\Rightarrow v \in V'_1 \cup \ldots \cup V'_{i-1} \cup V'_{i+1} \cup \ldots \cup V'_n = V \setminus V'_i$
1) Assume $v \in V'_1 \cup \ldots \cup V'_{i-1}$
$\xRightarrow{Definition\ 6} \forall v' \in V_i \big((v, v') \in E\big)$
$\xRightarrow{v \in V_i} (v, v) \in E$ contradicts the assumption (V, E, λ) represents an interval order because irreflexivity is not satisfied.

2) Assume $v \in V'_{i+1} \cup \ldots \cup V'_n$
$\xRightarrow{Definition\ 6} \forall v' \in V_i \big((v', v) \in E\big)$
$\xRightarrow{v \in V_i} (v, v) \in E$ contradicts the assumption (V, E, λ) represents an interval order because irreflexivity is not satisfied. □

Lemma 3 (Maximal Parallel Cuts Are Unique). *If a parallel cut exists in a given interval order $(V, E, \lambda) \in \mathcal{P}$, the maximal parallel cut is unique.*

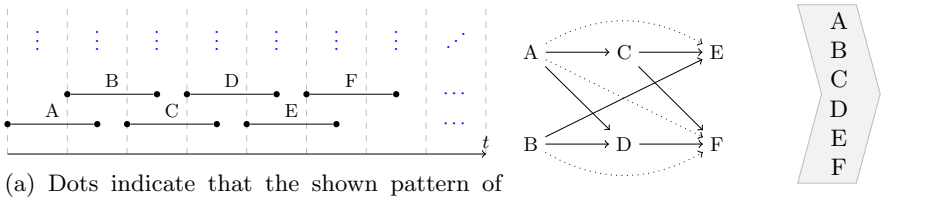
Proof (Lemma 3). By definition, components of a graph are unique. □

Lemma 2 and Lemma 3 show that maximal cuts, both ordering and parallel, are unique. Together with Lemma 1, we derive that the proposed visualization approach is *deterministic*, i.e., the approach always returns the same visualization for the same input, because for a given interval order only one cut type is applicable at most and if a cut exists, the maximal cut is unique.

4.3 Limitations

In this section, we discuss the limitations of the proposed visualization approach.

Reconsider the example in Figure 5. Cuts are recursively applied until one node, i.e., an activity instance, remains in each induced interval order (cf. Figure 5k). However, there are certain cases in which the proposed approach cannot apply cuts although more than one node exists in an (induced) interval order.



(a) Dots indicate that the shown pattern of chained activity instances can be extended arbitrarily, horizontally as well as vertically

(b) Corresponding interval order

(c) Corresponding visualization

Fig. 6: Example trace and interval order in which no cuts are applicable

Consider Figure 6a, showing an example of a trace where no cuts can be applied. Since each activity instance is overlapping with some other activity instance, we cannot apply an ordering cut. Also, since there is no activity instance that overlaps with all other activity instances, we cannot apply a parallel cut. Note that the visualized pattern of chained activity instances can be arbitrarily extended by adding more activity instances vertically and horizontally, indicated by the dots in Figure 6a. Figure 6b shows the corresponding interval order.

For the example trace, the proposed approach visualizes the activities $A, \ldots, F$ within a single chevron, indicating that the activities are executed in an unspecified order (cf. Figure 6c). Thus, the visualization highly simplifies the observed process behavior in such cases. Alternatively, it would be conceivable to show the interval order within a chevron if an (induced) interval order cannot be cut anymore. However, we decided to keep the visualization simple and show all activities within a single chevron. Note that this design decision entails that the expressiveness of the proposed visualization is lower than the graphical notation of interval orders, i.e., different interval orders can have the same visualization.

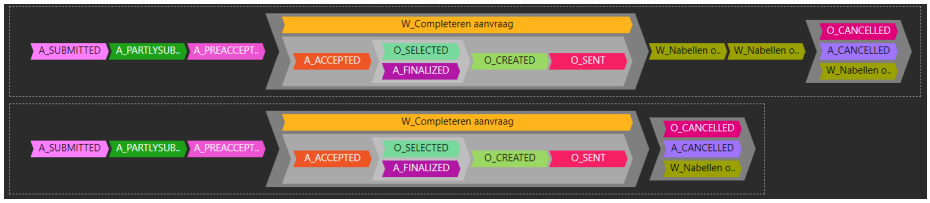


Fig. 7: Screenshot of Cortado's variant explorer showing real-life event data [4]

Table 2: Evaluation results based on real-life event logs

	Log statistics		Calculation time (s) of interval ordered variants				Variants statistics		
Event log	#cases (avg. #events per case)	multiple timestamps per activity available	Total calculation	Pre-processing event data	Creating interval orders	Cutting interval orders	#classic variants (only start-time-stamp considered)	#interval ordered variants	#interval ordered variants with limitations
BPI Ch. 2017 [5]	31,509 (≈38)	yes	≈39	≈21.6	≈5.7	≈11.3	15,930	5,854	335 (≈6%)
BPI Ch. 2012 [4]	13,087 (≈20)	yes	≈22.6	≈5.9	≈5.4	≈11.2	4,366	3,830	0 (≈0%)
Sepsis [11]	1,050 (≈14)	no	≈1.9	≈0.3	≈0.3	≈1.2	846	690	0 (≈0%)

4.4 Implementation

The proposed visualization approach for partially ordered event data has been implemented in *Cortado* [12][5], which is a standalone tool for interactive process discovery. Figure 7 shows a screenshot of Cortado visualizing an event log with partially ordered events. The implemented trace variant explorer works for both, partially and totally ordered event data. The tool assumes an event log in the `.xes` format as input. If the provided event log contains start- and complete-timestamps, the visualization approach presented in this paper is applied.

5 Evaluation

In this section, we evaluate the proposed visualization approach. We focus thereby on the performance aspects of the proposed visualization. Further, we focus on the limitations, i.e., no cuts can be applied anymore, although the (induced) interval order has more than one element, as discussed in Section 4.3.

We use publicly available, real-life event logs [4,5,11]. Table 2 shows the results. The first three columns show information about the logs. Two logs [5,4] contain start- and complete-timestamps per activity instance while one log [11] contains only a single timestamp per activity instance. Regarding the total calculation time, we note that the duration of the visualization calculation is reasonable from a practical point of view. We observe that the recursive application of cuts takes up most of the computation time in all logs, as expected. Regarding

[5] Available from version 1.2.0, downloadable from: https://cortado.fit.fraunhofer.de/

the variants, we observe that the number of classic variants is higher compared to the number of variants derived from the interval order for all event logs. We observe this even for the third event log [11] because some activities within the cases share the same timestamp. Regarding the limitations of the approach, as discussed in Section 4.3, we observe that only in the first log [5] approximately in 6% of all trace variants patterns occur where it was not possible to apply cuts anymore. Note that the limitation cannot occur in event logs where only a single timestamp per activity is available, e.g., [11].

6 Conclusion

This paper introduced a novel visualization approach for partially ordered event data. Based on chevrons, known from the classic trace variant explorer, our approach visualizes the ordering relations between process instances in a hierarchical manner. Our visualization allows to easily identify common patterns in trace variants from partially ordered event data. The approach has been implemented in the tool *Cortado* and has been evaluated on real-life event logs.

References

1. van der Aalst, W.M.P.: Data Science in Action. Springer Berlin Heidelberg (2016)
2. van der Aalst, W.M.P., Santos, L.: May i take your order? on the interplay between time and order in process mining. arXiv preprint arXiv:2107.03937 (2021)
3. Bodesinsky, P., Alsallakh, B., Gschwandtner, T., Miksch, S.: Exploration and Assessment of Event Data. In: EuroVis Workshop on Visual Analytics (EuroVA). The Eurographics Association (2015)
4. van Dongen, B.: BPI Challenge 2012 (2012), https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204
5. van Dongen, B.: BPI Challenge 2017 (2017), https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884
6. van Eck, M.L., Lu, X., Leemans, S.J.J., van der Aalst, W.M.P.: PM^2: A process mining project methodology. In: Advanced Information Systems Engineering. Springer (2015)
7. Fishburn, P.C.: Intransitive indifference with unequal indifference intervals. Journal of Mathematical Psychology **7**(1) (1970)
8. Gschwandtner, T.: Visual analytics meets process mining: Challenges and opportunities. In: Data-Driven Process Discovery and Analysis. Springer (2017)
9. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: Business Process Management Workshops. Springer (2016)
10. Lu, X., Fahland, D., van der Aalst, W.M.P.: Conformance checking based on partially ordered event data. In: Business Process Management Workshops. Springer (2015)
11. Mannhardt, F.: Sepsis cases - event log (2016), https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639
12. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Cortado—an interactive tool for data-driven process discovery and modeling. In: Application and Theory of Petri Nets and Concurrency. Springer (2021)