Repairing Event Logs with Missing Events to Support Performance Analysis of Systems with Shared Resources

Vadim Denisov[1,3], Dirk Fahland[1], and Wil M.P. van der Aalst[1,2]

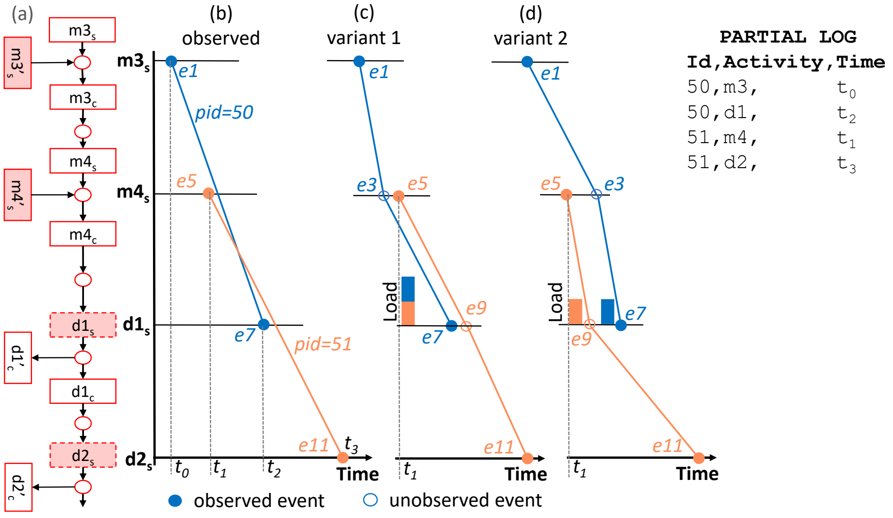
[1] Eindhoven University of Technology, Eindhoven, The Netherlands,
[2] Process and Data Science (Informatik 9), RWTH Aachen University, Aachen, Germany,
[3] Vanderlande Industries, Veghel, The Netherlands,
v.denisov@tue.nl, d.fahland@tue.nl, wvdaalst@pads.rwth-aachen.de

Abstract. To identify the causes of performance problems or to predict process behavior, it is essential to have correct and complete event data. This is particularly important for distributed systems with shared resources, e.g., one case can block another case competing for the same machine, leading to inter-case dependencies in performance. However, due to a variety of reasons, real-life systems often record only a subset of all events taking place. For example, to reduce costs, the number of sensors is minimized or parts of the system are not connected. To understand and analyze the behavior of processes with shared resources, we aim to reconstruct bounds for timestamps of events that must have happened but were not recorded. We present a novel approach that decomposes system runs into token trajectories of cases and resources that may need to synchronize in the presence of many-to-many relationships. Such relationships occur, for example, in warehouses where packages for N incoming orders are not handled in a single delivery but in M different deliveries. We use linear programming over token trajectories to derive the timestamps of unobserved events in an efficient manner. This helps to complete the event logs and facilitates analysis. We focus on material handling systems like baggage handling systems in airports to illustrate our approach. However, the approach can be applied to other settings where recording is incomplete. The ideas have been implemented in ProM and were evaluated using both synthetic and real-life event logs.

Keywords: Log repair · Process mining · Performance analysis · Modeling · Material handling systems

1 Introduction

Precise knowledge about actual process behavior and performance is required for identifying causes of performance issues [16], as well as for predictive process monitoring of important process performance indicators [14]. For Material Handling Systems (MHS), such as Baggage Handling Systems (BHS) of airports, performance incidents are usually investigated offline, using recorded event data for finding root causes of problems [10], while online event streams are used as input for predictive performance models [4]. Both analysis and monitoring heavily rely on the completeness and accuracy of input data. For example, events may not be recorded and, as a result, we do not know when they happened even though we can derive that they must have happened. Yet, when different



```
PARTIAL LOG
Id,Activity,Time
50,m3,      t0
50,d1,      t2
51,m4,      t1
51,d2,      t3
```

Fig. 1. An MHS model example (a), observed imprecise behavior for two cases 50 and 51 (b), possible actual behaviors (c,d).

cases are competing for shared resources, it is important to reconstruct the ordering of events and provide bounds for non-observed timestamps.

However, in most real-life systems, items are not continuously tracked and not all events are stored for cost-efficiency, leading to incomplete performance information which impedes precise analysis. For example, an MHS tracks the location of an item, e.g., a bag or box, via hardware sensors placed throughout the system, generating tracking events for system control, monitoring, analysis, and prediction. Historically, to reduce costs, a tracking sensor is only installed when it is strictly necessary for the correct execution of a particular operation, e.g., only for the precise positioning immediately before shifting a bag from one conveyor onto another. Moreover, even when a sensor is installed, an event still can be discarded to save storage space. As a result, the recorded event data of an MHS are typically incomplete, hampering analysis based on such incomplete data. Therefore, it is essential to repair the event data before analysis. Fig. 1 shows a simple MHS where events are not always recorded. The process model is given and for two cases the recorded incomplete sets of events are depicted using the so-called *Performance Spectrum* [10].

Fig. 1(b) shows item pid=50 entering the system via $m3$ at time t_0 (event e_1) and leaving the system via $d1$ at time t_2 (e_7), and item pid=51 entering the system via $m4$ at time t_1 (e_5) and leaving the system via $d2$ at time t_3 (e_{11}). As only these four events are recorded, the event data do not provide information in which order both cases traversed the *segment* $m4 \rightarrow d1$. Naively interpolating the movement of both items, as shown in Fig. 1(b), suggests that item pid=51 overtakes item pid=50. This contradicts that all items are moved from $m4$ to $d1$ via a conveyor belt, i.e., a FIFO queue: item 51 cannot have overtaken item 50. In contrast, Fig. 1(c) and Fig. 1(d) show two possible behaviors that are consistent with our knowledge of the system. We know that a conveyor belt

(FIFO queue) is a shared resource between $m4$ and $d1$. Both variants differ in the order in which items 50 and 51 enter and leave the shared resource, the speed with which the resource operated, and the load and free capacity the resource had during this time. In general, the longer the duration of naively interpolated segment occurrences, the larger the potential error. Errors in load, for example, make performance outlier analysis [10] or short-term performance prediction [9] rather difficult. Errors in order impede root-cause analysis of performance outliers, e.g., finding the cases that caused or were affected by outlier behavior.

Problem. In this paper, we address a novel type of problem as illustrated in Fig. 1 and explained above. The behavior and performance of the system cannot be determined by the properties of each case in isolation, but depends on the *behavior of other cases* and the *behavior of the shared resources* involved in the cases. Crucially, each case is handled by multiple resources and each resource handles multiple cases, resulting in *many-to-many* relations between them. The concrete problem we address is to reconstruct *unobserved* behavior and performance information of *each case* and each *shared resource* in the system that is *consistent* with both observed and reconstructed unobserved behavior and performance of all other cases and shared resources. More specifically, we consider the following information as given: (1) an event log L_1 containing the case identifier, activity and time for recorded events where intermediate steps are not recorded (i.e., the event log may be incomplete), (2) a model of the process (i.e., possible paths for handling each individual case), and (3) a description (model) of the resources involved in each step (e.g., queues, single server resources and their performance parameters such as processing and waiting time). Based on the above input, we want to provide a complete event log L_2 that describes (1) for each case the exact sequence of process steps, (2) and for each unobserved event a time-window of earliest and latest occurrence of the event so that (3) either all earliest or all latest timestamps altogether describe a consistent execution of the entire process over all shared resources.

Contribution. We propose a solution to this problem for a limited class of systems. We focus on processes where each step is served by one single-server resource and resources are connected by strict FIFO queues only. These assumptions are reasonable for a large class of MHSs. Our current solution formulation assumes the process to be *acyclic* which suffices for many real-life problem instances. Sect. 2 presents related work while elaborating on the problem. Specifically, prior work either only considers the case or the resource perspective explicitly, making implicit assumptions about their complex interplay. To overcome this limitation, we use *synchronous proclets* [11] in Sect. 3 to conceptually decompose a run of a system into individual *token trajectories* of cases, resources, and queues. Token trajectories synchronize when a resource or queue is involved in a case, allowing to explicitly describe their many-to-many relations in the run. Sect. 4 then formally captures token trajectories in terms of partial orders of events and defines the general problem. We solve the problem in Sect. 5 by formulating a Linear Programming (LP) problem [19] in terms of timestamps along the different token trajectories. To evaluate the approach, we compare the restored event logs with the ground truth for synthetic logs and estimate errors for real-life event logs for which the ground truth is unavailable (Sect. 6). We discuss our findings and future work in Sect. 7.

2 Related Work

In all operational processes (logistics, manufacturing, healthcare, education and so on) complete and precise event data, including information about workload and resource utilization, is highly valuable since it allows for process mining techniques uncovering compliance and performance problems. Event data can be used to replay processes on top of process models [2], to predict process behavior [5,9], or to visualize detailed process behavior using performance spectra [10]. All of these techniques rely on complete and correct event data. Since this is often not the case, we aim to transform *incomplete* event data into *complete* event data.

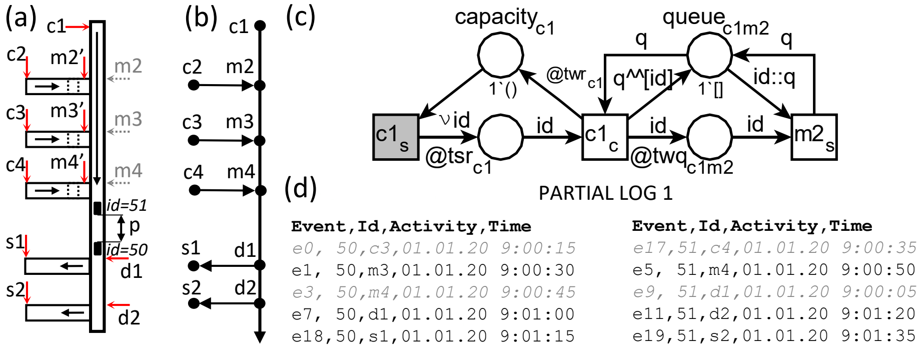
Various approaches exist for dealing with incomplete data of processes with non-isolated cases that compete for scarce resources. In call-center processes, thoroughly studied in [12], queueing theory models can be used for load predictions under assumptions about distributions of unobserved parameters, such as customer patience duration [6], while assuming high load snapshot principle predictors show better accuracy [21]. For time predictions in congested systems, the required features are extracted using congestion graphs [20] mined using queuing theory.

Techniques to repair, clean, and restore event data before analysis have been suggested in other works. An extensive taxonomy of quality issue patterns in event logs is presented in [22]. The taxonomy also lists approaches to repair inadvertent time intervals [22] in [8]. In [15] resource availability calendars are retrieved from event logs without the use of a process model, but assuming *start* and *complete* life-cycle transitions as well as a case arrival time present in a log. Using a process model, classical trace alignment algorithms [7] restore missing events but do not restore their timestamps. The authors conclude (see [7], p. 262) that incorporating other dimensions, e.g., resources, for multi-perspective trace alignment and conformance checking is an important challenge for the near future. Recently, also techniques for process discovery and conformance checking over uncertain event data were presented [17,18]. The output of our approach can provide the input needed for these techniques.

Our work contributes to the problem of reconstructing behavior of cases and limited shared resources for which the cases compete. We use the notion of *proclets* first introduced in [1] and adapted for process mining in [11] to approach the problem from control-flow and resource perspectives at once. We assume a system model given as a composition of a control-flow proclet (process) and resource/queue proclets. We restore missing events through classical trace alignments over control-flow proclets. The dynamic synchronization of proclets [11] allows us to infer how and when resource tokens must have traversed over the control-flow steps, which we express as a linear programming problem to compute timestamp intervals for the restored events. Event logs repaired in this way enable the use of analysis assuming complete event logs.

3 Modeling Inter-Case Behavior via Shared Resources

Prior work (cf. Sect. 2) approaches the problem of analyzing the performance of systems with shared resources primarily either from the control-flow perspective [15,17,18,5,9] or the resource/queuing perspective [12,6,21,20], leading to information loss about the



```
Event,Id,Activity,Time              Event,Id,Activity,Time
e0, 50,c3,01.01.20 9:00:15          e17,51,c4,01.01.20 9:00:35
e1, 50,m3,01.01.20 9:00:30          e5, 51,m4,01.01.20 9:00:50
e3, 50,m4,01.01.20 9:00:45          e9, 51,d1,01.01.20 9:00:05
e7, 50,d1,01.01.20 9:01:00          e11,51,d2,01.01.20 9:01:20
e18,50,s1,01.01.20 9:01:15          e19,51,s2,01.01.20 9:01:35
```

Fig. 2. A baggage handling system fragment (a) and its material flow diagram (b). Conveyor belts of check-in counters $c1 - c4$ merge at points $m2 - m4$, further downstream bags can divert at $d1$ and $d2$ to X-Ray security scanners $s1$ and $s2$. Red arrows show sensor (logging) locations. Conveyor $c1 : m2$ is modeled as a coloured Petri net model (c). An example of an incomplete event log of the system in (a) is shown in (d), where missing events are shown in the grey color.

other perspective. In the following, we show how to conceptualize the problem from both perspectives at once using *synchronous proclets* [11]. This way we are able to capture both control-flow and resource dynamics and their interaction as synchronizing token trajectories. We introduce the model in Sect 3.1 and use it to illustrate how incomplete logging incurs information loss for performance analysis in Sect. 3.2.

3.1 Processes-Aware Systems with Shared Resources

We explain the dynamics of process-aware systems over shared resources using a BHS handling luggage. The process control-flow takes a bag from a source (e.g., check-in or transfer from another flight), to a destination (e.g., the airplane, transfer) along intermediate process steps (e.g., baggage scanning, storage). BHS resources are primarily single-server machines (e.g., baggage scanners) connected via conveyor belts, i.e., FIFO queues. Fig. 2(a) shows a typical system design pattern involving the control-flow and resource perspective: four parallel check-in desks (c1-c4) merge into one *linear conveyor* through *merge points* (m2-m4). *Divert points* (d1 and d2) can route bags from the linear conveyor to *scanners* (s1 and s2). Each merge point and scanner is preceded by a FIFO queue for buffering incoming cases (bags) in case the corresponding resource is busy. Fig. 2(b) shows the plain control-flow of this BHS (also called Material Flow Diagram (MFD)). A real-life BHS may contain hundreds of process steps and resources, and conveyors may also form loops.

Modeling with Coloured Petri Nets. Fig. 2(c) shows a Coloured Petri Net (CPN) model for the segment $c1 \rightarrow m2$. In the model, transitions $c1_s$ and $c1_c$ describe *start* and *completion* of the check-in step $c1$. At the occurrence of $c1_s$ a new bag ($\nu\ id$) represented by a token with an *id* is inserted. Step $c1$ is served by a single resource (place $capacity_{c1}$) which has service-time tsr_{c1} to complete the step and waiting time trw_{c1} until the next bag can go through $c1$. All Resources in a BHS may require a waiting time to ensure sufficient “operating space” between two subsequent cases. After completion of $c1$, the bag enters a FIFO queue (modeling a conveyor belt) to the *start* of the merge step $m2_s$. Time annotation twq_{c1m2} models the minimum time it takes for a bag to travel from $c1$ to $m2$. Only then a bag may leave the queue at $m2_s$. The CPN model in Fig. 2(c) describes the impact of limited resource capacity and queues on the progress of a case, but does not model the resource itself as its own entity. The absence of the resource in the described behavior makes it impossible to reason about its behavior explicitly.

Modeling with Synchronous Proclets. The synchronous proclet system in Fig. 3 describes the entire BHS of Fig. 2(a) by using three types of proclets.

1. The *process proclet* (red border) is a Petri net describing the control-flow perspective of how bags may move through the system. It directly corresponds to the MFD of Fig. 2(b). It is transition-bordered and each occurrence of one of its initial transitions creates a new case identifier, see [11] for details.
2. Each *resource proclet* (green border) models a resource as its own entity with a cyclic behavior. For example, the *PassengerToSystemHandover* proclet (top left) identifies a concrete resource by token id $c1$; its life-cycle models that starting a task ($c1_s$) makes the resource *busy* and takes service time tsr_{c1}, after completing the task ($c1_c$) the resource has waiting time twr_{c1} before being *idle* again in the same way as Fig. 2(c). All other resource proclets follow the same pattern, though some resources such as *MergingUnit-m2* and *DivertingUnit-d1* may have two transitions to become *busy* or *idle*, respectively.
3. Each queue proclet (blue border) describes a FIFO queue as in Fig. 2(c). However, the queue state (the list) is accompanied by a queue identifier in place q. Items entering the queue are remembered by their number (generated from the *count* place).

The proclet system synchronizes process, resources, and queues via *synchronous channels* between transitions. Transitions linked via synchronous channels may only occur when all linked transitions are enabled; when they occur, they occur in a single synchronized event. For example, transition $c1_s$ is always enabled in *Process*, generating a new bag id, e.g., id=49, but it may only occur together with $c1_s$ in *PassengerToSystemHandover*, i.e., when resource $c1$ is *idle*, thereby synchronizing the process case for bag $id = 49$ with the resource with identifier $c1$. By annotation *init c1,1:1* $c1$ is now correlated to $id = 49$. The subsequent correlation annotation *=c1, 1:1* on the channel of the *complete* transition $c1_c$ ensures that resource $c1$ only synchronizes with the process case on which it started the step, i.e., $id = 49$; the next occurrence of $c1_s$ will create a new correlation to another process case, see [11] for details. In the example, each resource is statically linked to one process step, but the model also allows for one resource to participate in multiple different process steps, and multiple resources to be required for one process step. In the following, we call a proclet system that defines proclets for processes, queues, and resource that are linked via synchronous channels as described above, a *PQR system*.

Proclets describe synchronizing token trajectories. We now highlight how the partial-order semantics of synchronous proclets [11] preserves the identities of process, resources, and queues as "token trajectories". Figure 4(b) shows a partially-ordered run of the PQR system of Fig. 3 for two bags $id = 50$ and $id = 51$. The run in Fig. 4(b) can

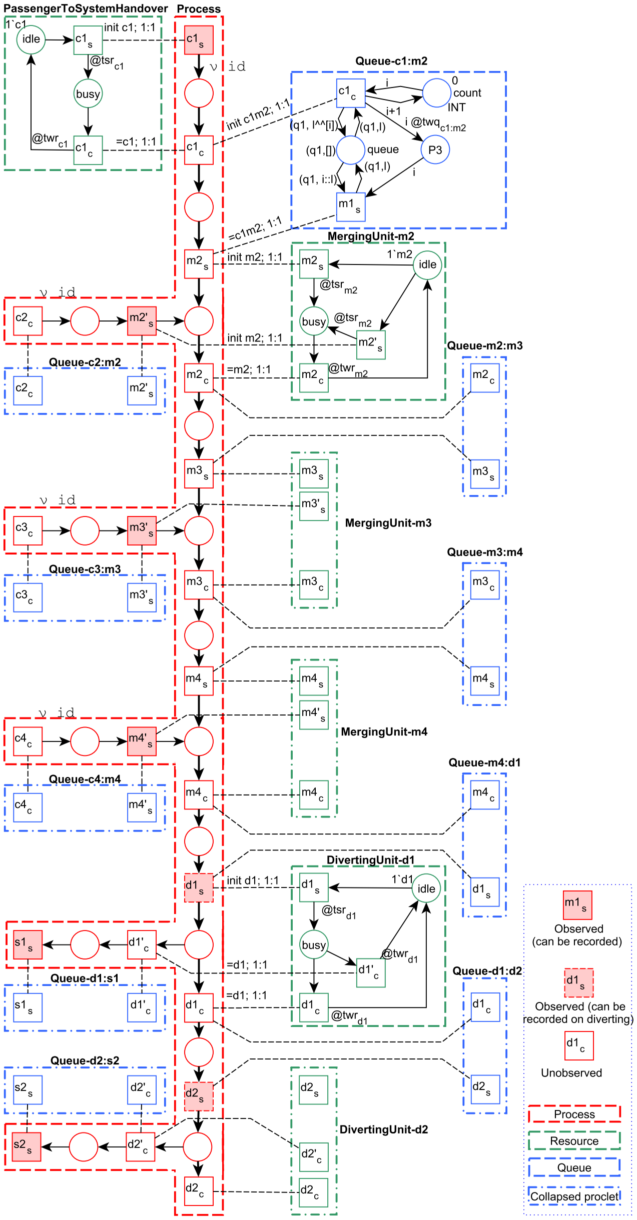


Fig. 3. The synchronous proclet model of the system shown in Fig. 2(a) consists of three types of proclets: *Process* for modeling a system layout and process control flow (red), *Resource* for modeling connector and sensor resources (services), and *Queue* for modeling conveyors transporting bags in the FIFO order. Only filled transitions can be observed in an event log.

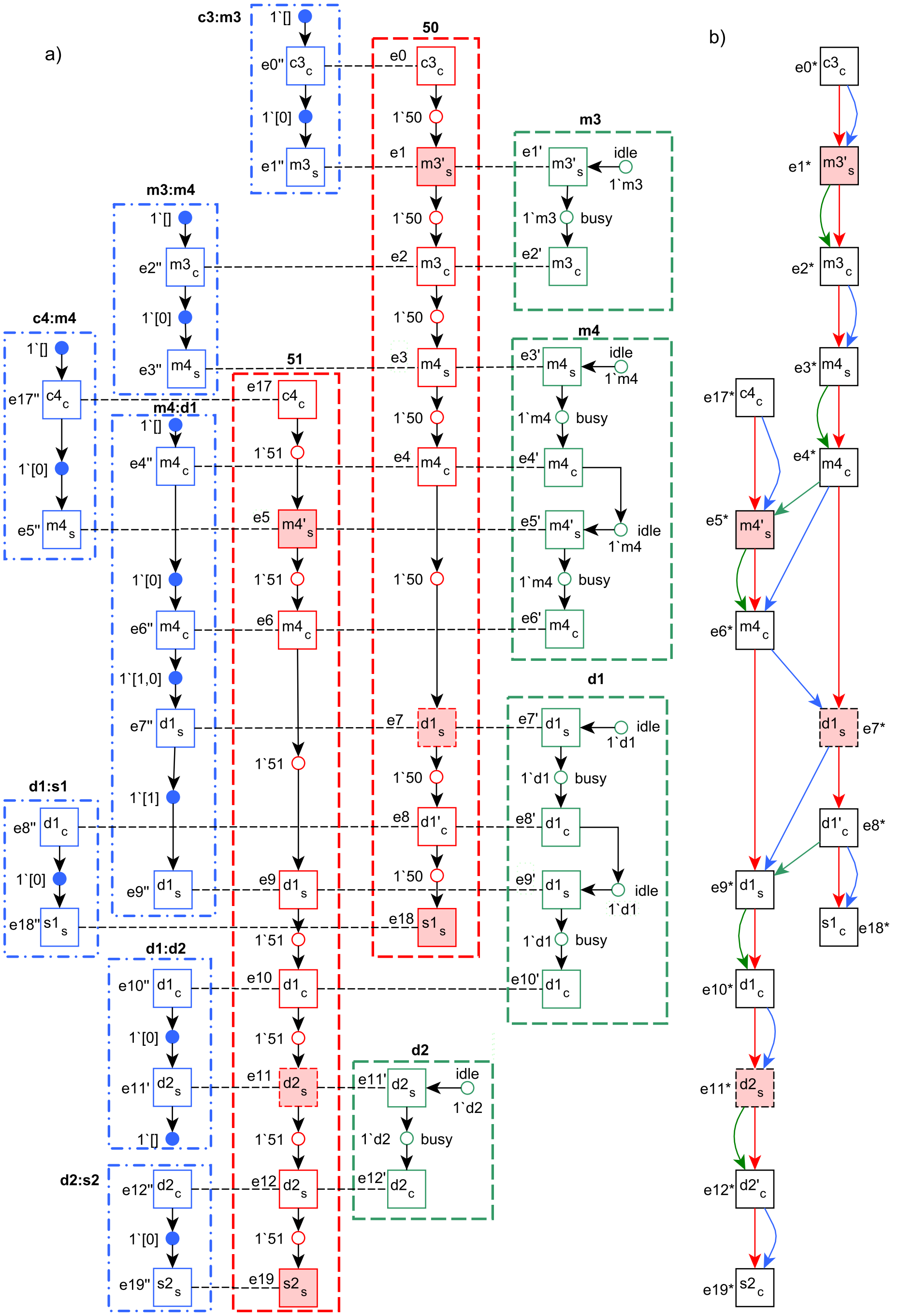


Fig. 4. Synchronization of multiple sub-runs of the synchronous proclet system in Fig. 3 over shared resources and queues (a), and a global partial order obtained by the union of partial orders of each sub-run (b) for synchronized events, shown by red, green and blue arrows for partial orders $<_{pid}$, $<_{rid}$ and $<_{qid}$ respectively.

be understood as a synchronization of multiple runs of the process, resource, and queue proclets, one for each case, resource, or queue involved as shown in Fig. 4(a).

Bag 50 gets inserted via input transition $c3_c$ (event e^*_0 in Fig. 4(b)). This event is a *synchronization* of events $e0$ ($c3_c$ occurs for bag 50 in the *Process* proclet) and $e0'$ ($c3_c$ occurs for the *c3:m3* queue) in Fig. 4(a). The minimal waiting time twq_{c3m3} must pass before bag 50 reaches the end of the queue and process step $m3$ can start. The process step $m3$ merges bag 50 from the check-in conveyor $c3$ onto the main linear conveyor and may only start via transition $m3_s$ when *MergingUnit-m3* is *idle*. As this is the case, bag 50 leaves the queue ($e1''$ in *c3:m3*), $m3$ starts merging ($e1'$ in *m3*), the bag starts the merging step (event $e1$ in *Process*), resulting in the synchronized event $e1^*$ in Fig. 4(b).

By e'_1, resource *m3* switches from *idle* to *busy* and takes time tsr_{m3} before it can complete the merge step with $m3_c$ (event $e2'$) on bag 50 (event $e2$); this merge step also inserts bag 50 into queue *m3:m4* ($e2''$) resulting in synchronized event $e2^*$. Subsequently, bag *50* leaves queue *m3:m4* ($e3^*$) is pushed by merge unit *m4* into queue *m4:d1* ($e4^*$).

Concurrently, bag 51 is inserted via input transition $c4_c$ (event $e17^*$), moves via queue *c4:m4* also to merge unit *m4* to enter queue *m4:d1*, i.e., both bags 50 and 51 now compete for merge unit *m4* and the order of entering *m4:d1*. In the run in Fig. 4, *m4* executes $m4_s$ and $m4_c$ for bag 51 ($e5^*$ and $e6^*$) *after completing* this step for bag 50 ($e3^*$ and $e4^*$). Thus, 51 enters the queue ($e6^*$) after 50 entered the queue ($e5^*$) but before 50 leaves the queue $e7^*$. Consequently, divert unit *d1* first serves 50 ($e7^*$ and $e8^*$) to reach scanner *s1* ($e18^*$) before serving 51 ($e9^*$ and $e10^*$) to reach scanner *s2* ($e19^*$).

Fig. 4(b) shows how the process tokens of bag 50 and 51 synchronized with the resources and queue tokens along the run, forming sequences or *trajectories* of events where this token was involved. For example, bag 50 followed the trajectory $e0^*, e1^*, \ldots, e8^*, e18^*$ and queue *m4:d1* followed trajectory $e4^*, e6^*, e7^*, e9^*$ thereby synchronizing with both bag 50 and bag 51.

3.2 Information Loss Because of Incomplete Logging

Although event data on objects that are tracked can be used for various kinds of data analysis [4,9], in practice sensors are placed only where it is absolutely necessary for correct operation of the system, e.g., for merge and divert operations, without considering data analysis needs. Applied to our example, only the transitions that are shaded in Fig. 3 would be logged, i.e., $c1_s, m2'_s, m3'_s, m4'_s, d1_s, d2_s, s1_s, s2_2$ would be logged from the *control-flow* perspective only. The run of Fig. 4 would result in a “typical” but highly incomplete event log as shown in Fig. 2(d).

According to this incomplete log, bag 50 silently passes *m4* and is tracked again only at *d1* ($e7$) and finally at *s1* ($e18$) whereas 51 silently passes *d1* (as it moves further on the main conveyor) and is tracked again only at *d2* ($e11$). Based on this incomplete information the bags 50 and 51 may have traversed *m4:d1* in different orders and at different speeds resulting also in different loads as illustrated in Fig. 1. As a result, in case of congestion, we cannot determine the ordering of cases [10], cannot compute the exact load on each conveyor part for (predictive) process monitoring [9,5]. The longer an unobserved path (e.g., $c1 \rightarrow d2$), the higher the uncertainty about the actual behavior and the less accurate performance analysis outcome.

Although minimal (or even average) service and waiting times on conveyor belts and resource are known, we need to determine the *exact timestamps* of all missing events and their order to reconstruct for how long resources were occupied by particular cases and in which order cases were handled, e.g., did 50 precede 51 on *m4:d1* or vice versa?

The objective of this paper is to reconstruct from a subset of events logged from the control-flow perspective only the remaining events (including time information), so that the time order is consistent with a partially ordered run of the entire system, including resource and queue proclets. For example, from the recorded events of the event log in Fig. 2(d) we reconstruct the remaining events (Fig. 4(a)) with time information so that the resulting order (by time) is consistent with the partially ordered run in Fig. 4(b).

4 System Runs and Partial Event Logs

In Sect. 3, we showed how the behavior of resource and queue-aware processes can be modeled as a PQR system, a particular type of a synchronous proclet system. The partially-ordered run of a PQR system decomposes into token trajectories for process cases, resources, and queues. In this section, we first formalize this relation between a partially ordered run of a system and its token trajectories through projection on partially ordered sets. We then formalize *partial* and *complete* event logs of a system run within this model and state the formal problem we address.

We use the following notion. Let A be a set of *event classifiers*; A is usually the set of activity names or the set of locations in case of an MHS. Let T be the set of time durations and timestamps, e.g., the rational or real numbers. Let $\mathcal{E}$ be the *universe of unique events* with *attributes*, let AN be a set of attribute names. For any $e \in \mathcal{E}, n \in AN$, $\#_n(e)$ is the value of attribute n for event e ($\#_n(e) = \perp$ if attribute n is undefined for e). Each event has a mandatory attribute *act*, $\#_{act}(e) \in A$, a mandatory attribute *lt* for a life-cycle transition, $\#_{lt}(e) \in \{start, complete\}$ and an optional attribute *time*, $\#_{time}(e) \in T$. Finally, we allow events to be related to multiple case notions. Let $\mathcal{Z}$ be the *universe of case identifiers* and $ID \subset AN$ be a *set* of *case notions*. If $\#_{id}(e) = z$, then event e is related to case z under case notion $id \in ID$.

From partial orders to token trajectories. A run of a proclet system [11] can be observed in terms of a Strict Partially Ordered Set (SPOSET) $\pi = (E, <)$ of events $E \subseteq \mathcal{E}$. As usual, we write $e_1 < e_2$ if event e_1 *precedes* event e_2 and we write $e_1 \lessdot e_2$ iff e_1 *directly* precedes e_2, i.e., $e_1 < e_2$ and there is no other event e_3 with $e_1 < e_3 < e_2$. In a PQR system we can distinguish three case notions $ID = \{pid, rid, qid\}$ to distinguish cases of the process, resources, and queues. Each event $e \in \mathcal{E}$ in a run of a PQR system has one or more case notions from ID. For example, in Fig. 4, $\#_{pid}(e5^*) = 51$, $\#_{rid}(e5^*) = m4$, $\#_{qid}(e5^*) = c4{:}m4$.

Restricting $<$ of a system run to events of the same case notion $id \in ID$ results in a *case notion-specific* partial order $e_1 <_{id} e_2$ iff $e_1 < e_2$ and $\#_{id}(e_1) = \#_{id}(e_2) \neq \perp$. Only events which share the same case notion and case identifier are ordered by $<_{id}$ – events of different cases are unordered. For example, in Fig. 4(b) $e4^* <_{rid} e5^*$ but $e4^* \not<_{pid} e5^*$. Consequently, $<_{pid}$ orders all events wrt. the process perspective whereas $<_{rid}$ and $<_{qid}$ order all events wrt. the resource and queue perspective, respectively. For a given case notion $id \in ID$ and case identifier $z \in \mathcal{Z}$, the events $E^z_{id} = \{e \in E | \#_{id}(e) = z\}$ of case z and

Strict Partial Order (SPO) $<^z_{id} |_{E^z_{id} \times E^z_{id}}$, restricted to the events of the same case, define a *sub-run* $\pi^z_{id} = (E^z_{id}, <^z_{id})$. Each such sub-run formalizes one *token trajectory* in the system run π. For example, Fig. 4(a) shows the sub-runs, viz. token trajectories, of all cases of the run of Fig. 4(b), i.e., π^{50}_{pid} and π^{51}_{pid} from the perspective of the process, $\pi^{m3}_{rid}, \pi^{m4}_{rid}, \pi^{d1}_{rid}, \pi^{d2}_{rid}$ from the perspective of the resources, and $\pi^{c3:m3}_{qid}, \pi^{m3:m4}_{qid}, \pi^{c4:m4}_{qid}, \pi^{m4:d1}_{qid}, \pi^{d1:d2}_{qid}, \pi^{d1:s1}_{qid}, \pi^{d2:s2}_{qid}$ from the perspective of the queues. In this way, our model shows that events of different process cases ($pid = 50$ and $pid = 51$) are independent under the classical control-flow perspective $<_{pid}$, e.g., $e4^* \not<_{pid} e5^* \not<_{pid} e7^*$, but mutually depend on each other under $<_{rid}$ and $<_{qid}$, e.g., $e4^* <_{rid} e5^* <_{rid} e6^*$ and $e6^* <_{qid} e7^*$. Each sub-run π^z_{id} is a "proper" run of case z in the corresponding proclet (Lemma 2 in [11]).

Event logs and token trajectories. Starting point for our analysis is the notion of a classical control-flow event log, which we express in our model of SPOSETs using *pid* as case notion. An *event log* $L = (E, <)$ is a finite set of events E where each event e has an activity $\#_{act}(e)$, a process case id $\#_{pid}(e)$. Note that e may have additional case identifiers $\#_{rid}(e)$ and $\#_{qid}(e)$ as attributes.

Adopting [13] to our setting, the optional timestamps $\#_{time}(e)$ induce the log's partial order, i.e., two events e_1 and e_2 are ordered if e_1 time-wise precedes e_2 and both are related in some case (for any $id \in ID$), i.e., $e_1 < e_2$ iff $\perp \neq \#_{time}(e_1) < \#_{time}(e_2) \neq \perp$ and there exists $id \in ID$ with $\#_{id}(e_1) = \#_{id}(e_2)$. If all events are only related to id cases, then $<$ and $<_{id}$ are identical. Further, each sub-run L^z_{id} (projection onto events with $\#_{id}(e) = z$) is a *trace* for case z under case notion id.

Event logs and token trajectories. Given a model M of a PQR system, i.e., a system defining proclets for pid, rid, qid, we call log L *complete* wrt. events and ordering iff there is a run π of M such that L and M are isomorphic wrt. attributes act, pid, qid, rid. Note that the ordering in L is induced by event timestamps only, thus a complete log defines the "right" timestamps. Further note that in a complete log L, each trace L^z_{id} is also complete and describes a token trajectory, i.e., a sub-run π^z_{id} of π that fits the corresponding proclet. Further, all traces of process cases (pid) are ordered relative to each other via the shared resources and queues as described in M.

In reality often only a subset of activities $B \subseteq A$ and the control-flow case notion pid have been recorded in a log, making it *partial*. In this paper, we call a log L' *partial* if there exists a complete log $L = (E, <)$ of M (viz. system run π) such that $L' = (E', <')$ is the projection of L onto activities in B, $L|_B = (E_B, < |_{E_B \times E_B})$, $E_B = \{e \in E \mid \#_{act}(e) \in B\}$ such that additionally

1. each $e \in E'$ has only case notion pid, i.e., $\#_{pid}(e) \neq \perp, \#_{rid}(e) = \#_{qid}(e) = \perp$,
2. $\#_{time}(e)$ is defined, and
3. for each process case z occurring in L, L' contains at least the first and last event of the complete trace.

Thus, L' contains for each case z at least one *partial trace* L'^z_{pid} recording the entry and exit of the case and preserving the order of observed events, i.e., it can be completed to fit the model. An MHS typically records a partial log as defined above. Fig. 2(d) shows a partial event log of the run on Fig. 4. In a partial event log, events of different process cases are less ordered, e.g., observed events $e1^*$ and $e5^*$ in Fig. 4 are unordered wrt. any a)

```
Event,Id,Activity,Time
  f1, 53, c1, 8:00:00
  f3, 53, m2, 8:00:15
  f5, 53, m3, 8:00:30
  f6, 53, m4, 8:00:45
  f9, 53, d1, 8:01:00
  f0, 54, c3, 8:01:20
  f12,54, m3, 8:00:35
  f14,54, m4, 8:00:50
  f16,54, d1, 8:01:05
```

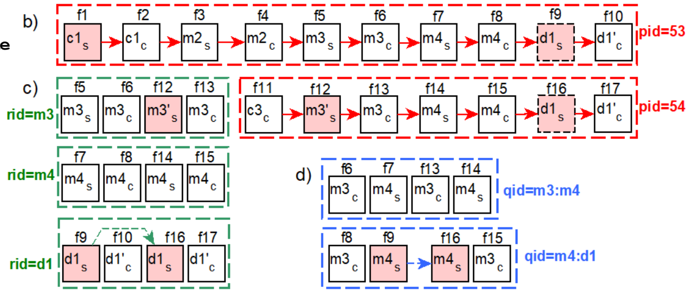


Fig. 5. Another partial event log of the system in Fig. 3 for bags 53 and 54 (a), partially complete traces of the Process (b), Resource (c) and Queue (d) proclets, restored by oracles O_1, O_2. Only observed events are ordered, e.g., $f9 <^{d1}_{rid} f16$, while the other events are isolated.

resource or queue whereas they are ordered in the complete run. In the following, we investigate how to restore this lost ordering.

Problem Formulation. Reconstructing a complete log from a partial log as defined above requires to reconstruct all missing events, all missing case notion attributes, and their timestamp. Restoring the *exact* timestamp is generally infeasible and for most use cases also not required. We, therefore, formulate the problem as restoring *time-windows* providing minimal and maximal timestamps for each unobserved event.

Let M be a model of a PQR system defining life-cycles of process, resource, and queue proclets, which resources and queues synchronize on which process step, and for each resource the minimum service time tsr and waiting time twr and for each queue the minimum waiting time twq. Given M and a partial log $L_1 = (E_1, <_1)$ of M, we want to (1) reconstruct unobserved events E_u for all process cases in L_1 and their relations to queues and resources, and (2) for each unobserved event $e \in E_u$ a time-window of earliest and latest occurrence of the event $\#_{tmin}(e), \#_{tmax}(e) \in T$ so that (3) $L_2 = (E_1 \cup E_u, <_2)$ is a complete log of M when $<_2$ is inferred from $\#_{tmin}(e)$ or from $\#_{tmax}(e)$.

5 Inferring Timestamps Along Token Trajectories

In Sect. 4, we presented the problem of restoring missing events and time-windows for their timestamps from a partial event log L_1 such that the resulting log is consistent with resource and queueing behavior. In this section, we solve the problem for PQR systems with *acyclic* process proclets by casting it into a constraint satisfaction problem, that can be solved using Linear Programming (LP) [19]. In Sect. 5.1, we show how to infer unobserved events (from M) and how to infer resource and queue identifiers from M to construct an intermediate SPO $(E_2, <_2)$. All unobserved events $E_2 \setminus E_1$ have no timestamp, i.e., they are unordered in $<_2$. In Sect. 5.2 we then show how to determine minimal and maximal timestamps for each unobserved event (through a linear program) that preserves the already known ordering $<_2$. Inferring $<_3$ from the minimal (or maximal) timestamps refines $<_2$ and results in an SPO $L_3 = (E_2, <_3)$ which is a complete log of M and has L_1 as a partial log. We explain our approach using another (more compact running) example shown in Fig. 5(a) for two bags 53 and 54 processed in the system of Fig 3. The events in grey italic (i.e., f3, f5, f6, f14) are unobserved.

5.1 Infer Potential Complete Runs From a Partial Run

We first derive for the partial log $L_1 = (E_1, <_1)$ an intermediate log $L_2 = (E_2, <_2)$ so that each trace $L^z_{2,pid}$ of a process case z is complete (i.e., fits the process proclet in M). In a second step, we relate each unobserved event $e \in E_u = E_2 \setminus E_1$ to a corresponding resource and/or queue case identifier which orders observed events wrt. $<_{rid}$ and $<_{qid}$, resulting in an SPO $\pi = (E_\pi, <_\pi)$ (with $E_\pi = E_2$). All unobserved events $e \in E_u$ lack a timestamp and hence are left unordered wrt. $<_{rid}$ and $<_{qid}$ in π; we later refine $<_\pi$ in Sect 5.2.

We specify how to solve each of the steps in terms of two *oracles* O_1 and O_2 and describe concrete implementations for either. Oracle O_1 has to return $L_2 = (E_2, <_2\ ) = O_1(E_1, <_1, M)$ by completing each partial trace $L^z_{1,pid}$ of some process case z into a complete trace $L^z_{2,pid}$ that fits the process proclet M. The restored *unobserved* events $E_u = E_1 \setminus E_1$ only have attributes *act*, *pid* and *lt* (life-cycle transition) and events are *totally ordered* along pid, i.e., $<_2 = <_{pid}$ is a total order. O_1 can be implemented using well-known trace alignment [3]. For example, applying O_1 on the partial log of Fig. 5(a) results in the complete process traces of Fig. 5(b). Note that, slightly deviating from our model, O_1 constructs $<_2$ explicitly (not based on timestamps).

Oracle O_2 has to enrich events in L_2 with information about queues and resources so that for each $e \in E_2$ if resource r is involved in the step $\#_{act}(e)$, then $\#_{rid}(e) = r$ and if queue q was involved, then $\#_{qid}(e) = q$. Moreover, in order to formulate the linear program to derive timestamps in a uniform way, each event e has to be annotated with the performance information of the involved resource and/or queue. That is, if e is a start event and $\#_{rid}(e) = r \neq \perp$, then $\#_{tsr}(e)$ and $\#_{twr}(e)$ hold the minimum service and waiting time of r, and if $\#_{qid}(e) = q \neq \perp$, then $\#_{twq}(e)$ hold the minimum waiting time of q. For the concrete PQR systems considered in this paper, we set $\#_{rid}(e) = r$ based on the model M if r is the case id of the resource proclet that synchronizes with transition $t = \#_{act}(e)$ via a channel (there is at most one). Attributes $\#_{tsr}(e)$, $\#_{twr}(e)$, can be set from the model as they are parameters of the resource proclet. To ease the LP formulation, if e is unrelated to a resource, we set $\#_{rid}(e) = r^*$ to fresh identifier and $\#_{tsr}(e) = \#_{twr}(e) = 0$; $\#_{qid}(e)$ and $\#_{twq}(e)$ are set correspondingly. By annotating the events in E_2, we obtain SPO $\pi = (E_\pi, <_\pi), E_\pi = E_2$ that also contains sub-runs for each queue and resource containing all events to be complete wrt. M but only observed events are ordered (due to their timestamps). For example, Fig. 5(d) shows the sub-run $\pi^{m4:d1}_{qid}$ containing events f_8, f_9, f_{16}, f_{15} with only $f_9 <_{qid} f_{16}$. Next, we define constraints based on the information in this intermediate log π to infer timestamps for all unobserved events.

5.2 Restoring Timestamps of Unobserved Events by Linear Programming

The SPO $\pi = (E, <)$ obtained in Sect. 5.1 from partial log $L_1 = (E_1, <_1)$ includes all unobserved events $E_u = E \setminus E_1$ of the complete log, but lacks timestamps for each $e \in E_u, \#_{time}(e) = \perp$. Each observed $e \in E_1$ has a timestamp $\#_{time}(e)$ and we also added minimum service time $\#_{tsr}$, waiting time $\#_{twr}(e)$ of the resource $\#_{rid}(e)$ involved in e and minimum waiting time $\#_{twq}(e)$ of the queue involved in e. We now define a constraint satisfaction problem that specifies the earliest $\#_{tmin}(e)$ and latest $\#_{tmax}(e)$ timestamps for each $e \in E_u$ so that all earliest (latest) timestamps yield a consistent ordering of all events in E wrt. $<_{pid}$ (events follow the process), $<_{rid}$ (events follow resource lifecycle), and $<_{qid}$ (events satisfy queueing behavior). The problem formulation propagates the known $\#_{time}(e)$ values along with the different case notions $<_{pid}$, $<_{rid}$, $<_{qid}$, using tsr, twr, twq. For that, we introduce variables x_e^{tmin}, $x_e^{\text{tmax}} \geq 0$ for representing event attributes $tmin, tmax$ of each $e \in E_u$. For all observed events $e \in E_1$, we set $x_e^{\text{tmin}} = x_e^{\text{tmax}} = \#_{time}(e)$ as here the correct timestamp is known. We now define two groups of constraints to constrain the x_e^{tmin} and x_e^{tmax} values for the unobserved events further. In the following, we assume for the sake of simpler constraints presented in this paper, that all observed events are start events (which is in line with logging in an MHS). The constraints can easily be reformulated to assume only complete events were observed (as in most business process event logs) or a mix (requiring further case distinctions).

The first group propagates constraints for $\#_{time}(e)$ along $<_{pid}$, i.e., for each token trajectory (viz. trace) π^z_{pid} of pid in π. By the steps in Sect. 5.1, events in π^z_{pid} are totally ordered and we write $\pi^z_{pid} = \langle e_1 ... e_m \rangle$ as a sequence of events. Each process step has a start and a complete event in π^z_{pid}, i.e., $m = 2 \cdot y, y \in \mathbb{N}$, odd events are start events and even events are complete events. For each process step $1 \leq i \leq y$, the time between start event e_{2i-1} and complete event e_{2i} is at least the service time of the resource involved (which we stored as $\#_{tsr}(e_{2i-1})$ in Sect. 5.1). Thus the following constraints must hold for the earliest and latest time of e_{2i-1} and e_{2i}.

$$x_{e_{2i}}^{\text{tmin}} = x_{e_{2i-1}}^{\text{tmin}} + \#_{tsr}(e_{2i-1}), \tag{1}$$

$$x_{e_{2i}}^{\text{tmax}} = x_{e_{2i-1}}^{\text{tmax}} + \#_{tsr}(e_{2i-1}). \tag{2}$$

For the remainder, it suffices to formulate constraints only for *start* events. We make sure that *tmin* and *tmax* define a proper interval for each start event:

$$x_{e_{2i-1}}^{\text{tmin}} \leq x_{e_{2i-1}}^{\text{tmax}}. \tag{3}$$

We write $e^s_i = e_{2i-1}$ for the start event of the i-th process step in π^z_{pid} and $\theta^z_{pid} = \langle e^s_1, ..., e^s_m \rangle$ for the sub-trace of start events of π^z_{pid}. Any event $e^s_i \in \theta^z_{pid}$ that was observed in L_1, i.e., $e^s_i \in E_1$, has $\#_{time}(e^s_i) \neq \perp$ defined. By the assumption in Sect. 4, π^z_{pid} as well as θ^z_{pid} always start and end with observed events, i.e., $e^s_1, e^s_y \in E_1$ and $\#_{time}(e^s_1), \#_{time}(e^s_y) \neq \perp$. An unobserved event e^s_i has no timestamp $\#_{time}(e^s_i) = \perp$ yet, but $\#_{time}(e^s_i)$ is bounded by $\#_{time}(e^s_1)$ (minimally) and $\#_{time}(e^s_y)$ (maximally). Furthermore, any two succeeding start events in $\theta^z_{pid} = \langle ..., e^s_{i-1}, e^s_i, ... \rangle$ are separated by the service time $\#_{tsr}(e^s_{i-1})$ of step e^s_{i-1} and the waiting time $\#_{twq}(e_i)$ of the queue from e_{i-1} to e_i. Similar to Eq. 1 and 2, we formulate this constraint for both x_e^{tmin} and x_e^{tmax} variables:

$$x_{e^s_k}^{\text{tmin}} \geq x_{e^s_{k-1}}^{\text{tmin}} + (\#_{tsr}(e^s_{k-1}) + \#_{twq}(e^s_k)), \tag{4}$$

$$x_{e^s_k}^{\text{tmax}} \leq x_{e^s_{k+1}}^{\text{tmax}} - (\#_{tsr}(e^s_k) + \#_{twq}(e^s_{k+1})). \tag{5}$$

Fig. 6 uses the *Performance Spectrum* [10] to illustrate the effect of applying our approach step by step to the partially complete traces of Fig. 5 obtained in the steps of Sect. 5.1. The straight lines in Fig. 6(a) from f_1 to f_9 (for pid=53) and from f_{12} to f_{16} (for pid=54)

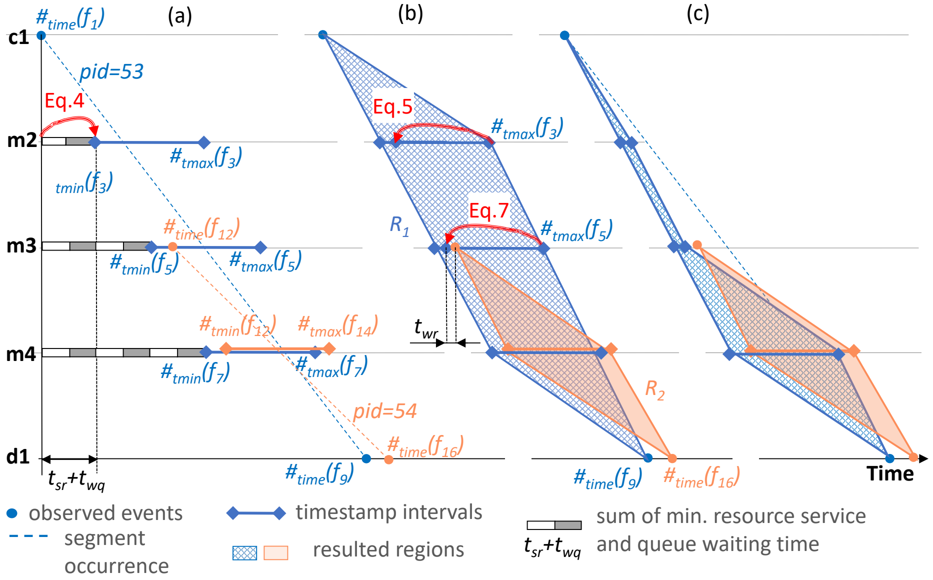


Fig. 6. Equations 1-5 define time intervals for unobserved events (a), defining regions for the possible traces (b). Equations 6-7 propagate orders of cases observed on one resource to other resources (b), resulting in tighter regions (c).

illustrate that L_2 (after applying O_1) contains all intermediate steps that both process cases passed through but not their timestamps. Further (after applying O_2), we know for each process step the resources (i.e., c1, m2, m3, m4, d1) and the queues (c1:m2, m2:m3 etc.), and their minimum service and waiting times tsr, twr, twq. The sum $tsr + twq$ is visualized as bars on the time axis in Fig. 6(a), the duration of twr is shown in Fig. 6(b). We now explain the effect of applying Eq. 4 on pid=53 for f_3, f_5 and f_7. We have $\theta^{53}_{pid} = \langle f_1, f_3, f_5, f_7, f_9 \rangle$ with f_1 and f_9 observed, thus $x^{\text{tmin}}_{f_i} = x^{\text{tmax}}_{f_i} = \#_{time}(f_i)$ for $i \in \{1, 9\}$. By Eq. 4, we obtain the lower-bound for the time for f_3 by $x^{\text{tmin}}_{f_3} \geq x^{\text{tmin}}_{f_1} + \#_{tsr}(f_1) + \#_{twq}(f_3)$ with $\#_{tsr}(f_1)$ and $\#_{twq}(f_3)$ the service time of resource c1 and waiting time of queue c1:m2. Similarly, Eq. 4 gives the lower bound for f_5 from the lower bound from f_3 etc. Conversely, the upper bounds $x^{\text{tmax}}_{f_i}$ are derived from f_9 "downwards" by Eq. 5. This way, we obtain for each $f_i \in \theta^{53}_{pid}$ an initial interval for the time of f_i between the bounds $x^{\text{tmin}}_{f_i} \leq x^{\text{tmax}}_{f_i}$ as shown by the intervals in Fig. 6(a). As $x^{\text{tmin}}_{f_1} = x^{\text{tmax}}_{f_1} = \#_{time}(f_1)$ and $x^{\text{tmin}}_{f_1} = x^{\text{tmax}}_{f_1} = \#_{time}(f_9)$, the lower and upper bounds for the unobserved events in θ^{53}_{pid} form a polygon as shown in Fig. 6(b). Case 53 must have passed over the process steps and resources as a path inside this polygon, i.e., the polygon contains all admissible solutions for the timestamps of the unobserved events of θ^{53}_{pid}; we call this polygon the *region* of case 53. The region for case 54 overlays with the region for case 53.

We now introduce a second group of constraints by which we infer more tight bounds for $x^{\text{tmin}}_{e_i}$ and $x^{\text{tmax}}_{e_i}$ based on the overlap with other regions. While the first group of constraints traversed token trajectories along *pid* (i.e., process traces), the second group of constraints traverses token trajectories for resources along *rid*. Each resource trace $\pi^r_{rid} = (E^r_{rid}, <^r_{rid})$ in π, contains all events E^r_{rid} resource r was involved in - *across* multiple different process traces. The SPO $<^{r}_{rid}$ orders *observed* events of this resource trace due to their known timestamps; e.g. in Fig. 6(b) $f_9 <^{m1}_{rid} f_{16}$ with f_9 from pid=53 and f_{16} from pid=54. The order of the two events $e^s_{p1} <^{r}_{rid} e^s_{p2}$ for the *same* step $\#_{act}(e^s_{p1}) = \#_{act}(e^s_{p2}) = t_1$ in different cases $\#_{pid}(e^s_{p1}) = p1 \neq \#_{pid}(e^s_{p2}) = p2$ propagates "upwards" and "downwards" the process traces π^{p1}_{pid} and π^{p2}_{pid} as follows. Let events $f^s_{p1} \in E^{p1}_{pid}$ and $f_{p2} \in E^{p2}_{pid}$ be events in process traces π^{p1}_{pid} and π^{p2}_{pid} of the same step $\#_{act}(f^s_{p1}) = \#_{act}(f^s_{p2}) = t_n$. We say t_1 and t_n are *in FIFO relation* iff there is a unique path $\langle t_1 ... t_n \rangle$ between t_1 and t_n in the process proclet (i.e., no loops, splits, parallelism) so that between any two consecutive transitions t_k, t_{k+1} only synchronize with single-server resources or FIFO queues. If t_1 and t_n are in FIFO relation, then also $f^s_{p1} <^{r2}_{rid} f^s_{p2}$ on the resource r2 involved in t_n (as the case cannot overtake the case along this path). Thus $x^{\text{tmin}}_{f^s_{p1}} \leq x^{\text{tmin}}_{f^s_{p2}}$ must hold. More specifically, $x^{\text{tmin}}_{f^s_{p1}} + \#_{tsr}(f^s_{p1}) + \#_{twr}(f^s_{p1}) \leq x^{\text{tmin}}_{f^s_{p2}}$ must hold as the service time and waiting time of the resource involved in f^s_{p1} must elapse.

For any pair $e^s_{p1}, e^s_{p2} \in E^{r}_{rid}$ with $e^s_{p1} <^{r}_{rid} e^s_{p2}$ and any other trace θ^{r2}_{rid} for resource $r2$ and any pair $f^s_{p1}, f^s_{p2} \in E^{r2}_{rid}$ such that $\#_{pid}(e^s_{p1}) = \#_{pid}(f^s_{p1}), \#_{pid}(e^s_{p2}) = \#_{pid}(f^s_{p2})$ and transition $\#_{act}e^s_{p1}$ is in FIFO relation with $\#_{act}(f^s_{p1})$, we generate the following constraint for *tmin*:

$$x^{\text{tmin}}_{f^s_{p1}} \leq x^{\text{tmin}}_{f^s_{p2}} - (\#_{tsr}(f^s_{p1}) + \#_{twr}(f^s_{p1})), \tag{6}$$

and the following constraint for *tmax*:

$$x^{\text{tmax}}_{f^s_{p1}} \leq x^{\text{tmax}}_{f^s_{p2}} - (\#_{tsr}(f^s_{p1}) + \#_{twr}(f^s_{p1})), \tag{7}$$

In the example of Fig. 6(b), we observe $f_9 <^{d1}_{rid} f_{16}$ (both of transition $d1_s$) along resource $d1$ at the bottom of Fig. 6(b). By Fig. 3, $d1_s$ and $m3_s$ are in FIFO-relation. Applying Eq. 7 yields $x^{\text{tmax}}_{f_5} \leq \#_{time}(f_{12}) - (\#_{tsr}(f_5) + \#_{twr}(f_5))$, i.e., f_5 occurs at latest before f_{12} minus the service and waiting time of $m3$. This operation significantly reduces the initial region R_1. By Eq. 5, the tighter upper bound for f_5 also propagates along the trace pid=53 to f_3, i.e., $x^{\text{tmax}}_{f_3} \leq x^{\text{tmax}}_{f_5} - (\#_{tsr}(f_3) + \#_{twq}(f_5))$, resulting in a tighter region as shown in Fig. 6(c). If another trace $\langle m3_s, d1_s \rangle$ were present *before* trace 53, then this would cause reducing the *tmin* attributes of the events of trace 53 by Eq. 4,6 in a similar way. In general, the more cases interact through shared resources, the more accurate timestamp intervals can be restored by Eq. 1-7 as we will show in Sect. 6.

To construct the linear program, we generate equations 1 to 5 by iteration of each process trace in L_2. Further, iterate over each resource trace and for each pair of events $e_{p1} <^{r}_{rid} e_{p2}$ we generate equations 6,7 for each other pair of events $f_{p1} <^{r2}_{rid} f_{p2}$ that is in FIFO relation. The objective function to maximize is the sum of all intervals $\sum_{e \in E_2}(x^{\text{tmax}}_{e} - x^{\text{tmin}}_{e})$ to maximize the coverage of possible timestamp values by those intervals.

6 Evaluation

To evaluate our approach, we formulated the following questions. (Q1) Can timestamps be estimated in real-life settings and used to estimate performance reliably? (Q2) How

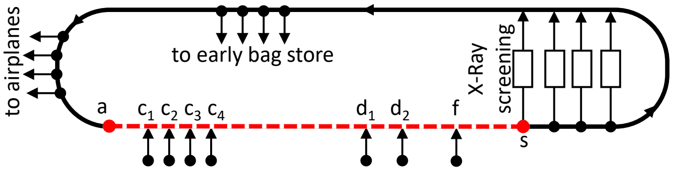


Fig. 7. In the BHS bags come from check-in counters c_{1-4} and another terminals d_{1-2}, f, go through mandatory screening and continue to other locations.

accurately can the load (items per minute) be estimated for different system parts, using restored timestamps? (Q3) What is the impact of sudden deviations from the minimum service/waiting times, e.g., the unavailability of resource or stop/restart of an MHS conveyor, on the accuracy of restored timestamps and the computed load? For that, we extended the interactive ProM plug-in "Performance Spectrum Miner" with an implementation of our approach that solves the constraints using heuristics[4]. As input we considered the process of a part of real-life BHS shown in Fig. 7 and used Synthetic Logs (SL) (simulated from a model to obtain ground-truth timestamps) and Real-life Logs (RL) from a major European airport. Regarding Q3, we generated SL with regular performance and with *blockages* of belts (i.e., a temporary stand-still); the RL contained both performance characteristics. All logs were partial as described in Sect. 4. We selected the acyclic fragment highlighted in Fig. 7 for restoring timestamps of steps c_{1-4}, d_{1-2}, f, s.

We evaluated our technique against the ground truth known for SL as follows. For each event we measured the error of the estimated timestamp intervals $[t_{min}, t_{max}]$ against the actual time t as $\max\{|t_{max} - t|, |t_{min} - t|\}$ normalized over the sum of minimal service and waiting times of all involved steps (to make errors comparable). We report the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) of these errors. Applying our technique to SL with regular behavior, we observed very narrow time intervals for the estimated timestamps, shown in Fig. 8(a), and a MAE of $< 5\%$. The MAE of the estimated load (computed on estimated timestamps), shown in Fig. 8(e), was $< 2\%$. For SL with blockage behavior, the intervals grew proportionally with the duration of blockages (Fig. 8(b)), leading to a proportional growth of the MAE for the timestamps. However, the MAE of the estimated load (Fig. 8(f)) was at most 4%. The load MAE for different processing steps for both scenarios are shown in Table 1. Notably, both observed and reconstructed load showed load peaks each time the conveyor belt starts moving again.

When evaluating on the real-life event log, we measured errors of timestamps estimation as the length of the estimated intervals (normalized over the sum of minimal service and waiting times of all involved steps). Performance spectra built using the restored RL logs are shown in Fig. 8(c,d), and the load computed using these logs is shown in Fig. 8(g,h). The observed MAE was $< 5\%$ in regular behavior and increased proportionally as observed on SL. The load error could not be measured, but similarly to synthetic data, it showed peaks after assumed conveyor stops.

[4] The simulation model, simulation logs, ProM plugin, and high-resolution figures are available on https://github.com/processmining-in-logistics/psm/tree/rel.

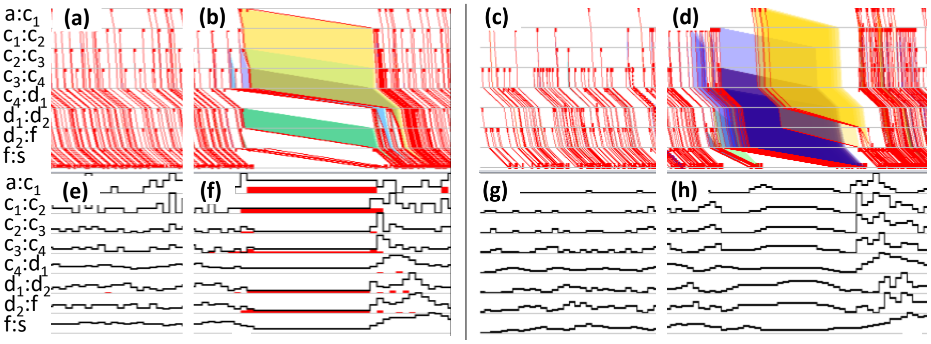


Fig. 8. Restored Performance Spectrum for synthetic (a,b) and real-life (c,d) logs. The estimated load (computed on estimated timestamps) for synthetic (e,f) and real-life (g,h) logs. For the synthetic logs, the load error is measured and shown in red (e,f).

The obtained results on SL show that the timestamps can be always estimated, and the actual timestamps are always within the timestamp intervals (Q1). When the system resources and queues operate close to the known performance parameters tsr, twr, twq, our approach restores accurate timestamps resulting in reliable load estimates in SL (Q2). During deviations in resource performance, the errors increase proportionally with performance deviation while the estimated load remains reliable (error < 4% in SL) and shows known characteristics from real-life systems on SL and RL (Q3).

Scenario	MAE, $c_4 : d_1$	RMSE, $c_4 : d_1$	MAE, $d_1 : d_2$	RMSE, $d_1 : d_2$	MAE,$f : s$	RMSE, $f : s$
no blockages	0.16	1.01	0.22	1.66	0.17	0.89
blockages	1.67	4.8	3.19	7.17	0.15	0.75

Table 1. The estimated load (computed on estimated timestamps) Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are shown in % of max. load.

7 Conclusion

In this paper, we studied the problem of repairing a partial event log with missing events for the performance analysis of systems where case interact and compete for shared limited resources. We addressed the problem of repairing partial event logs that contain only a subset of events which impede the performance analysis of systems with shared limited resources and queues. To study and solve the problem, we used synchronous proclets [11] to model processes served by resources and queues (a PQR system). The model allows to decompose the interactions of resources and queues over multiple process cases into token trajectories for process cases, resources and queues that synchronize on shared events. We exploit the decomposition when restoring missing events along the process token trajectories using trace alignment [7]. We exploit the synchronization when formulating linear programming constraints over timestamps of restored events along, both, the process and the resource token trajectories. As a result, we obtain timestamps which are consistent for all events along the process, resource, and queue dimensions. The evaluation of our implementation in synthetic and real-life data shows errors of the estimated timestamps and of derived performance characteristics (i.e., load) of $< 5\%$ under regular performance, while correctly restoring real-life dynamics (i.e. load peaks) after irregular performance behavior.

Limitations. The work made several limiting assumptions. (1) Although the proclet formalism allows for arbitrary, dynamic synchronizations between process steps, resources, and queues, we limited ourselves in this work to a static known resource/queue id per process step. The limitation is not severe for some use cases such as analyzing MHS, but generalizing oracle O_2 to a dynamic setting is an open problem. (2) The LP constraints to restore timestamps assume an acyclic process proclet without concurrency. Further, the LP constraints assume 1:1 interactions (at most one resource and/or queue per process step). Both assumptions do not hold in business processes in general; formulating the constraints for a more general setting is an open problem. (3) Our approach ensures consistency of either all earliest or all latest timestamps with the given model, it does not suggest how to select timestamps between the latest and earliest such that the consistency holds. (4) When the system performance significantly changes, e.g., due to sudden unavailability of resources, the error of restored timestamps is growing proportionally the duration of deviations. Points (3) and (4) require attention to further improve event log quality for performance analysis.

Acknowledgements The research leading to these results has received funding from Vanderlande Industries in the project “Process Mining in Logistics”. We also thank Mitchel Brunings for his comments that greatly improved our approach.

References

1. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. International Journal of Cooperative Information Systems **10**(04), 443–481 (2001). https://doi.org/10.1142/S0218843001000412
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
3. Aalst, W.M.P., Adriansyah, A., Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining and Knowledge Discovery **2**, 182–192 (03 2012). https://doi.org/10.1002/widm.1045
4. Ahmed, T., Pedersen, T.B., Calders, T., Lu, H.: Online risk prediction for indoor moving objects. In: 2016 17th IEEE International Conference on Mobile Data Management (MDM). vol. 1, pp. 102–111 (June 2016). https://doi.org/10.1109/MDM.2016.27
5. A.Senderovich, C.D.Francescomarino, F.M.Maggi: From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring. Information Systems (2019), https://doi.org/10.1016/j.is.2019.01.007
6. Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S., Zhao, L.: Statistical analysis of a telephone call center. Journal of the American Statistical Association **100**(469), 36–50 (2005). https://doi.org/10.1198/016214504000001808, https://doi.org/10.1198/016214504000001808
7. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)
8. Conforti, R., La Rosa, M., ter Hofstede, A.: Timestamp repair for business process event logs. Tech. rep. (2018/04/05 2018), http://hdl.handle.net/11343/209011

9. Denisov, V., Fahland, D., van der Aalst, W.M.P.: Predictive performance monitoring of material handling systems using the performance spectrum. In: 2019 International Conference on Process Mining (ICPM). pp. 137–144 (June 2019). https://doi.org/10.1109/ICPM.2019.00029
10. Denisov, V., Fahland, D., van der Aalst, W.M.P.: Unbiased, fine-grained description of processes performance from event data. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) Business Process Management. pp. 139–157. Springer International Publishing, Cham (2018)
11. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Donatelli, S., Haar, S. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 3–24. Springer International Publishing, Cham (2019)
12. Gans, N., Koole, G., Mandelbaum, A.: Telephone call centers: Tutorial, review, and research prospects. Manufacturing & Service Operations Management **5**, 79–141 (03 2003)
13. Lu, X., Fahland, D., van der Aalst, W.M.P.: Conformance checking based on partially ordered event data. In: BPM'14 Workshops. LNBIP, vol. 202, pp. 75–88. Springer (2014). https://doi.org/10.1007/978-3-319-15895-2_7, https://doi.org/10.1007/978-3-319-15895-2_7
14. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: A survey. IEEE Transactions on Services Computing **11**(6), 962–977 (Nov 2018). https://doi.org/10.1109/TSC.2017.2772256
15. Martin, N., Depaire, B., Caris, A., Schepers, D.: Retrieving the resource availability calendars of a process from an event log. Information Systems **88**, 101463 (2020). https://doi.org/https://doi.org/10.1016/j.is.2019.101463, http://www.sciencedirect.com/science/article/pii/S0306437919305150
16. Maruster, L., van Beest, N.R.T.P.: Redesigning business processes: a methodology based on simulation and process mining techniques. Knowl. Inf. Syst. **21**(3), 267–297 (2009)
17. Pegoraro, M., Aalst, W.: Mining uncertain event data in process mining. pp. 89–96 (06 2019). https://doi.org/10.1109/ICPM.2019.00023
18. Pegoraro, M., Uysal, M.S., van der Aalst, W.M.P.: Discovering process models from uncertain event data. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) Business Process Management Workshops. pp. 238–249. Springer International Publishing, Cham (2019)
19. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Chichester (1986)
20. Senderovich, A., Beck, J., Gal, A., Weidlich, M.: Congestion graphs for automated time predictions. Proceedings of the AAAI Conference on Artificial Intelligence **33**, 4854–4861 (07 2019). https://doi.org/10.1609/aaai.v33i01.33014854
21. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining - predicting delays in service processes. In: CAiSE (2014)
22. Suriadi, S., Andrews, R., ter Hofstede, A., Wynn, M.: Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Information Systems **64**, 132 – 150 (2017). https://doi.org/https://doi.org/10.1016/j.is.2016.07.011, http://www.sciencedirect.com/science/article/pii/S0306437915301344