Estimating Activity Start Timestamps in the presence of Waiting Times via Process Simulation

Claudia Fracca[1,2], Massimiliano de Leoni[1], Fabio Asnicar[2], and Alessandro Turco[2]

[1] University of Padua, Italy
deleoni@math.unipd.it
[2] ESTECO SpA, Trieste, Italy
{asnicar, fracca, turco}@esteco.com

Abstract. Process Mining aims to analyze and improve processes to enable organizations to provide better services or products. The starting point of Process Mining is an event log that is extracted from the organization's information systems that support the process' executions. Several techniques require event logs to record the timestamp when process' activities have started and been completed. Unfortunately, information systems do not always record the timestamps when process activities start, preventing the application of these techniques. This paper reports on a technique based on process simulation that aims to estimate the start event timestamps when missing. In a nutshell, the idea is to build an accurate process model from the initial event log without start timestamps, to simulate it with alternative activity-duration profiles, and to select the model with the profile that generates the runs that are the closest to the initial log. This activity-duration profile is used to add the missing, start timestamps to the initial log. Experiments were conducted with two event logs with start timestamps, and aimed at their rediscovery: the results show our estimation of the start event timestamps is more accurate than the state of the art.

Keywords: Start timestamps · Time perspective · Waiting time · Log repair · Process simulation.

1 Introduction

Process mining is a research discipline that sits between machine learning and data mining on the one hand and process modeling and analysis on the other hand. The idea behind it is to discover, monitor, and improve real processes to provide better services or products by extracting knowledge from event logs readily available in today's organization's information systems [1]. As briefly elaborated in Section 2, several process mining techniques need event logs that store both start and complete life-cycle transitions of activities. These techniques, e.g., [8, 13], specifically rely on knowing the timestamps when process' activities were started. Unfortunately, most information systems that support the process executions only record the completion of the different process' phases and activities. As a consequence, event logs often only store the events when activities are completed, thus missing events when activities started.

Existing techniques naïvely estimate the timestamps related to the start of activities under the assumptions that activity instances start as soon as possible, namely when the

activities that come before are completed and a suitable resource is available (e.g.,[14]). This corresponds to assuming no waiting times, and it is often unrealistic in practice [4, 8]: *(i)* resources work on multiple processes and continuously switch from one to the other while event logs refer to one process, *(ii)* take breaks during the working days (e.g., when tired), *(iii)* carry on additional duties that lead no trail in the event logs (e.g., when answering the phone).

This paper proposes a new technique to estimate the timestamps of the start events, namely of the events related to the start of activities. The starting point is an event log $\mathcal{L}$ of a given process $\mathcal{P}$, and a simulation model $\mathcal{M}$ of $\mathcal{P}$. This simulation model consists of a model of a process $\mathcal{P}$ (e.g., a BPMN model)[3] extended with additional information for the simulation aspects (case inter-arrival time, activity durations, routing probabilities, resource allocation, and utilization, etc.). The simulation model can be constructed by combining different process mining techniques (see the case studies discussed in Section 5). In a nutshell, the idea is that event log $\mathcal{L}$ is augmented with the missing start events using n different activity duration profiles, thus obtaining a set of event logs $\mathcal{L}_1, \ldots, \mathcal{L}_n$. Using each log $\mathcal{L}_i$, it is possible to compute the activity duration probabilities to be included in $\mathcal{M}$, lead to n simulation model $\mathcal{M}_1, \ldots, \mathcal{M}_n$. Each simulation model $\mathcal{M}_i$ is used to generate a simulated event log $\mathcal{L}_i^{sim}$. The technique returns the event log $\mathcal{L}_i$ with start events such that the corresponding simulation model $\mathcal{M}_i$ generated an event log $\mathcal{L}_i^{sim}$ that is the closest to $\mathcal{L}$ with the respect that certain properties, such as the probability distribution of trace duration.

The technique has been evaluated in two case studies using two event logs about the process for credential recognition of students in a university and a purchase to pay process (cf. Section 5). These event logs contain both start and completion events. For the evaluation, we removed the start events and aimed to rediscover through our technique. The evaluation outcome shows that the our technique was able to rediscover start-event timestamps that were more accurate than what could be achieved by existing techniques that naïvely assume no waiting time.

2 Related Work

Several techniques rely on the availability of the start activity timestamps, namely the timestamps of the start events. In business process simulation complete and precise event logs, including information about completion and start of activities, is highly valuable and necessary as the starting point for many simulation parameters discovery algorithms, such as techniques for multi-perspective information extraction. For example, Martin et al. [13] present a discover technique for the resource availability calendars and in [8] a technique to detect the presence of multitasking. These techniques start from event log assuming both start and complete life-cycle transitions of activities. A complete event log is necessary as input also for certain techniques for automated discovery of business process models, such as in [10], to better distinguish between true concurrency and interleaving. The construction of a business simulation model that accurately model resources also requires the presence of the timestamps of start and completion

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

events [5, 18]. In particular, Carmargo et al. [5] confirm the difficulty to find real-life event logs with both start and completion timestamps, thus limiting the applicability of their simulation model construction. The same applies to techniques to discover the resource availability calendars [13] or the detection of multitasking [8].

The problem of estimating the start activity timestamps is somehow related to queue mining, namely assessing the queue lengths and waiting times of process activities [2, 19]. However, queue mining uses stochastic approaches to provide probability distributions, confidence intervals, etc., without focusing on computing punctual values for each start event. Rogge-Solti and Weske use stochastic Petri nets to determine the probability distribution of the duration of process instances, but their goal is not to estimate the start-event timestamps [17]. Techniques to repair, clean, and restore event data before analysis have been suggested in other works: classical trace alignment algorithms are used to restore missing events but without restoring their timestamps [6]. In [7], Denisov et al. propose an alternative technique to restoring missing timestamps, under the strong assumption, which does not generally hold, that the model is acyclic and contains no parallelism constructions (e.g., AND splits). Pegoraro et al. have proposed a repertoire of process mining techniques over missing event data, but they have aimed at process discovery and conformance checking [15, 16], without focusing on repairing event logs, and adding the missing start events and their respective timestamps.

3 Preliminaries

An event log and a simulation model are the starting points of our framework to estimate the starting timestamps for computing the activity durations. Since we do not need to make any assumption of the modeling language and the simulation parameters, we remain very abstract in this respect. A **simulation model** is a tuple $\mathcal{M} = (\mathcal{N}, \mathcal{S})$ composed by a business process model $\mathcal{N}$ (e.g., a BPMN model), and a set $\mathcal{S}$ of the parameters to define the simulation specifications for the different process perspectives (time, resources, decisions, etc.). Examples of parameters are the case inter-arrival time, activity durations, routing probabilities, resource allocation, and utilization. Events and logs are defined as follows.

Definition 1 (Events). *Let $\mathcal{A}$ be a set of activity labels. Let $\mathcal{T}$ be the universe of timestamps. Let $\mathcal{R}$ be a set of resources. Let $\mathcal{I} = \{\texttt{start}, \texttt{complete}\}$ be the life-cycle information. An event $e \in \mathcal{A} \times \mathcal{T} \times \mathcal{R} \times \mathcal{I}$ is a tuple consisting of an activity label, a timestamp of occurrence, a resource performing the activity, and the life-cycle information.*

In the remainder, given an event $e = (a, t, r, i)$, $act(e) = a$ returns the activity label, $time(e) = t$ returns the timestamp, $res(e) = r$ returns the resource, and $life(e) = i$ is the information whether e refers to the starting or completion of an activity.
In practice, several event logs are composed of events where the life-cycle information is not present. In this case, we assume that those events refer to the completion. The log might include events not related to starting or completing activities; we ignore those events. Additional details attached to the events are also ignored. Sometimes events carry a payload consisting of attributes taking on values; we also ignore these attributes.

Table 1. A fragment of an event log of a train ticket compensation request procedure.

CASE ID	ACTIVITY	TIMESTAMP	RESOURCE	LIFE-CYCLE
123	Check Ticket	16-07-21 00:21	Paul	complete
124	Register Request	16-07-21 00:27	Ann	start
124	Register Request	16-07-21 00:32	Ann	complete
124	Check Ticket	16-07-21 00:40	Paul	start
124	Check Ticket	16-07-21 00:49	Paul	complete
123	Decide	16-07-21 00:50	Ann	start
123	Decide	16-07-21 01:10	Ann	complete
124	Decide	16-07-21 01:20	Ann	start

Definition 2 (Traces and Event Logs). *Let $\mathcal{E}_A$ the universe of the events defined over a set $\mathcal{A}$ of (labels of) activities. A trace $\sigma = \langle e_1, \ldots, e_m \rangle \in \mathcal{E}_A^*$ is a sequence of events, with the constraint that, for all $0 < i < j \leq m$, $time(e_i) \leq time(e_j)$. An event log $\mathcal{L}$ is a set of traces, namely $\mathcal{L} \subset \mathcal{E}_A^*$.*

In the remainder, we use the shortcut $e \in \mathcal{L}$ to indicate that there is a trace $\sigma \in \mathcal{L}$ such that $e \in \sigma$. Also, the subscript A is omitted when it is clear from the context.

Example 1. Consider the fragment of an event log presented in Table 1. This event log contains information about the handling of a request for compensation of train tickets. The arrival of a ticket compensation request initiates a process instance. After the request is received, the ticket is checked, and a decision is made. The compensation request is either rejected or paid. Each case, identified by CASE ID, is composed of a list of events that has an activity label, a timestamp of occurrence, a resource performing the activity, and life-cycle information, i.e., starting or completion of activities. As mentioned before, we assume that the timestamps of the events are ordered within the case.

Definition 3 (Trace Duration). *Let $\sigma = \langle e_1, \ldots, e_m \rangle$ a trace in an event log $\mathcal{L}$. Let denote with $TD(\sigma)$ the trace duration related to the trace σ. A trace duration $TD(\sigma) = time(e_m) - time(e_1)$ is the difference between the timestamps of the last and the first event in the trace σ.*

Given an event log $\mathcal{L}$, we aim to compute the **probability distribution of trace durations of** $\mathcal{L}$ as a function $\mathcal{D} : \mathbb{R}_0^+ \rightarrow [0, 1]$ that best fits the multiset of trace durations $\uplus_{\sigma \in \mathcal{L}} TD(\sigma)$.[4] Note that the domain of function $\mathcal{D}$ coincides with the possible trace durations, namely any non-negative real value.

The discussion of our technique requires the introduction of the concept of the duration of an activity instance, namely the difference between the timestamp of the completion event and that of the start event:

Definition 4 (Activity Instance Duration). *Let $\sigma = \langle e_1, \ldots, e_i, \ldots, e_m \rangle$ be a trace in $\mathcal{L}$. Let $e_i \in \sigma$ be an event such that $life(e_i) =$ `complete`. Let e_j be the latest, previous event in σ referring to the starting of the same activity as e_i , namely $j < i$, $act(e_j) = act(e_i)$, $life(e_j) =$ `start`, and there is no start event for the same activity between the j-th and the i-th event, namely $\forall k \in]j, i[$. $act(e_k) \neq act(e_j) \vee life(e_k) \neq$ `start`.*

[4] Symbol $\uplus$ indicates the union of elements to form a multiset.

The duration of the activity instance related to the completion event e_i is $AD(e_i) = time(e_i) - time(e_j)$.

Similarly, the concept of waiting time of an activity instance is necessary hereafter, intended as the difference between the timestamp of start event e and that of the latest event that precedes e in the trace:

Definition 5 (Activity Instance Waiting Time). *Let $\sigma = \langle e_1, \ldots, e_i, \ldots, e_m \rangle$ be a trace in $\mathcal{L}$. Let $e_i \in \sigma$ be an event such that $life(e_i) =$ `start`. Let e_j be the latest, previous event in σ referring to the completion of an activity, namely $j < i$, $life(e_j) =$ `complete`, and there is no completion event between j and i, namely $\forall k \in]j, i[$. $life(e_k) \neq$ `complete`. The waiting time of the activity instance related to the start event e_i is $WD(e_i) = time(e_i) - time(e_j)$.*

Example 2. Considering the example before. Let take case 124 and the event related to the activity *Check Ticket*. In this case, the start timestamp is 16-07-21 00:40, and the completion timestamp is 16-07-21 00:49, and the previously completed timestamp is 16-07-21 00:32. Therefore, in this case, the activity instance duration is equal to 9 minutes, and the activity instance waiting time is equal to 8 minutes.

Let $\mathcal{L}$ be an event log defined over a set $\mathcal{A}$ of activities. For each activity $a \in \mathcal{A}$, it is possible to compute the **activity-duration probability distribution** as the probability distribution functions $d_{p,a} : \mathbb{R}_0^+ \longrightarrow [0, 1]$ that best fits the multisets $\uplus_{e \in \mathcal{L} | act(e)=a} AD(e)$ of duration of instances of activity a in $\mathcal{L}$. Similarly, the **waiting time probability distribution** of a is the distribution $d_{w,a} : \mathbb{R}_0^+ \longrightarrow [0, 1]$ that best fits the multisets $\uplus_{e \in \mathcal{L} | act(e)=a} WD(e)$ of waiting times for instances of activity a ins $\mathcal{L}$.

In the remainder, we denote with $dist\_p : \mathcal{A} \rightarrow S$, the function that for each activity a returns $d_{p,a}$ the activity-duration probability distribution in the universe S of probability distribution functions $f : \mathbb{R}_0^+ \longrightarrow [0, 1]$. And with $dist\_w : \mathcal{A} \rightarrow S$ the function that returns for an activity a the waiting time probability distribution $d_{w,a}$.

The following definition presents the concept of a previous event in a trace by control-flow, which, given an event $e \in \sigma$ related to completion, represents the previous event in σ that also refers to a completion.

Definition 6 (Previous Event in a Trace by Control-Flow). *Let $\sigma = \langle e_1, \ldots, e_m \rangle$ a trace in an event log $\mathcal{L}$. For each $e_i \in \sigma$ s.t. $life(e_i) =$ `complete` and $\exists j < i$: $life(e_j) =$ `complete`, the previous event $prev\_t_{\mathcal{L}}(e_i)$ in a trace by control-flow related to the event e_i, is defined as follows:*

$$prev\_t_{\mathcal{L}}(e_i) = e_k \text{ s.t. } life(e_k) = \texttt{complete} \text{ and } \forall\, k < l < i \;\; life(e_l) \neq \texttt{complete}.$$

If the event log contains resource information, we can thus define the previous event $prev\_r_{\mathcal{L}}(e)$ as the completion event with the closest timestamp smaller than $time(e)$, among those referring to activities performed by the same resource $res(e)$:

Definition 7 (Previous Event Performed by a Resource). *Let e an event in the event log $\mathcal{L}$. Let now assume that $life(e) =$ `complete` and $\exists\, e_j \in \mathcal{L} : life(e_j) =$ `complete`,*

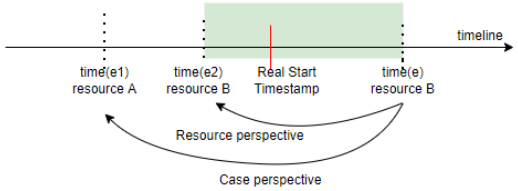

Fig. 1. Schema representing the concepts of previous event in a trace by control-flow and previous event performed by a resource. For the event e, $time(e)$ is the timestamp related to the event e, and $res(e) = resourceB$ the resource performing the activity $act(e)$. Looking at the trace perspective, e_1 is the previous event in a trace by control-flow $prev\_t_{\mathcal{L}}(e)$, and e_2 the previous event performed by a resource $prev\_r_{\mathcal{L}}(e)$ with $res(e_2) = res(e) = resourceB$. The green box represents the timeline range in which the event e could be started.

$res(e_j) = res(e)$ *and* $time(e_j) < time(e)$. *The previous event performed by the resource* $res(e)$*, is defined as follows:*

$$prev\_r_{\mathcal{L}}(e) = e_k \;\; \textit{s.t.} \;\; life(e_k) = \texttt{complete},\ res(e_k) = res(e) \;\; \textit{and}$$
$$\nexists\ \hat{e} \in \mathcal{L} \mid res(\hat{e}) = res(e),$$
$$life(\hat{e}) = \texttt{complete},\ \textit{and}$$
$$time(prev\_r_{\mathcal{L}}(e)) \leq time(\hat{e}) \leq time(e).$$

In Figure 1 a schema representing these two concepts. Note that, given an event $e \in \mathcal{L}$, $prev\_t_{\mathcal{L}}(e)$ and $prev\_r_{\mathcal{L}}(e)$ may not always be defined, e.g. when the current activity instance is the first one executed by a resource, or when the activity instance is the first one in a trace, or for the first activity instance recorded in the event log.

Example 3. Considering the example before. Let consider the case 123 and the event related to the activity *Decide* with life-cycle information equal to `complete`. In this case the event e is the tuple (*Decide*, 16-07-21 01:10, Ann, `complete`). For this event the related $prev\_t_{\mathcal{L}}(e)$ is the tuple (*Check Ticket*, 16-07-21 00:21, Paul, `complete`) and it is the previous event completed in the same trace 123. Therefore $time(prev\_t_{\mathcal{L}}(e))$ is equal to 16-07-21 00:21. And the related $prev\_r_{\mathcal{L}}(e)$ is (*Register Request*, 16-07-21 00:32, Ann, `complete`) and it is the previous completion event performed by the same resource Ann. Therefore $time(prev\_r_{\mathcal{L}}(e))$ is equal to 16-07-21 00:32.

The estimation of the timestamp related to the starting of activity requires defining the minimum timestamp when the activity could have started, considering the different process constraints, e.g., on resource and control flow. Given an event log $\mathcal{L}$, denoted the set of completion events in $\mathcal{L}$ with $E_{\mathcal{L}} = \{e \in \mathcal{L} : life(e) = \texttt{complete}\}$, this computation is abstracted as a *minimum-timestamp oracle* $mintime_{\mathcal{L}} : E_{\mathcal{L}} \rightarrow \mathcal{T}$ where $\mathcal{T}$ is the universe of timestamps. For each event in $e \in E_{\mathcal{L}}$, $mintime_{\mathcal{L}}(e)$ is the earliest timestamp when the activity instance related to e could have started.

The abstraction as a minimum-timestamp oracle is motivated by the fact this can be defined in multiple ways, also depending on the information available in the $\mathcal{L}$. As an

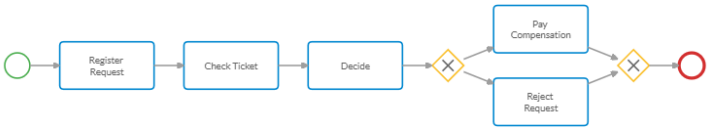

Fig. 2. BPMN model related to the event log in Table 1.

example, if we have both defined for an event e the events $prev\_t_{\mathcal{L}}(e)$ and $prev\_r_{\mathcal{L}}(e)$, the related $mintime\_res\_time_{\mathcal{L}}(e) = max(time(prev\_t_{\mathcal{L}}(e)), time(prev\_r_{\mathcal{L}}(e)))$. Another example, if it is possible to compute $prev\_t_{\mathcal{L}}(e)$ but we do not have information about the resource perspective, then $mintime\_time_{\mathcal{L}}(e) = time(prev\_t_{\mathcal{L}}(e))$. In the remainder, when evident from the context, we omit the subscript $\mathcal{L}$.

Example 4. Considering the example before. Let consider the case 123 and the event related to the activity *Decide*, therefore $e =$ (*Decide*, 16-07-21 01:10, Ann, `complete`). In this case the $mintime\_res\_time_{\mathcal{L}}(e)$ is equal to 16-07-21 00:32, i.e. the maximum between the $time(prev\_t_{\mathcal{L}}(e))$ and $time(prev\_r_{\mathcal{L}}(e))$.

4 Technique

In the remainder, event logs are assumed to contain no events related to the starting of activities. This is to keep the explanation simple: however, the extension is simple to tackle the hybrid case where a fraction of the start events are present. Given an event log $\mathcal{L}$ without start events defined over a set $\mathcal{A}$ of activities, the technique aims to build a new event log $\mathcal{L}'$ that include the start events. In particular, for each trace $\sigma \in \mathcal{L}$, $\mathcal{L}'$ includes a trace σ' that contains every event of σ. For each event $e \in \sigma$, σ' additionally includes a matching start event e': $act(e') = act(e)$, $res(e') = res(e)$ and $life(e') = \texttt{start}$. The timestamp $time(e')$ needs to be estimated. To do so, we formulate the problem as finding a value for a parameter $\alpha(e) \in [0, 1]$ related to the event e such that:

$$time(e') = \alpha(e) \cdot mintime(e) + (1 - \alpha(e)) \cdot time(e) \tag{1}$$

where $mintime(e)$ is some instance of the minimum-timestamp oracle (cf. Section 3). Note that, if $\alpha(e) = 0, time(e') = time(e)$, namely the duration of the activity instance to which e refers is zero. Conversely, if $\alpha(e) = 1$, $time(e') = mintime(e)$, namely the activity instance to which e refers starts at the earliest possible moment.

In practice, to keep the problem tractable, we assume to find the same $\alpha(e)$ for all events e related to the same activity a, namely:

$$time(e') = \alpha(a) \cdot mintime(e) + (1 - \alpha(a)) \cdot time(e) \tag{2}$$

where $act(e') = a$. The assumption is that waiting times for different instances of the same activity a are similar, i.e., these instances are executed by the same type of resources, which exhibit similar behavior. The remainder of this section details how our technique computes function $\alpha : \mathcal{A} \rightarrow [0, 1]$, namely each value $\alpha(a)$ for every activity a.

Along with event log $\mathcal{L}$, we need an initial simulation model $\mathcal{M}_0 = (\mathcal{N}, \mathcal{S}_0)$ without a specification about activity durations.

Example 5. The event log in Table 1 could be potentially associated to a simulation model $\mathcal{M}_0 = (\mathcal{N}, \mathcal{S}_0)$ where $\mathcal{N}$ is the process model in Figure 2, and the simulation parameters $\mathcal{S}_0$ contain: (i) the case inter-arrival time parameterized as an exponential distribution with mean 20 minutes, (ii) the routing probabilities for the XOR gateway: 65% for the Pay Compensation branch and 35% for the other, and (iii) a resources allocation where Ann performs the activities $Register Request$ and $Decide$, while Paul does the others.

Given n functions $\alpha_1 : \mathcal{A} \rightarrow [0, 1], \ldots, \alpha_n : \mathcal{A} \rightarrow [0, 1]$, we extend the original event log $\mathcal{L}$ with the starting events computed using Equation 2. Using $\alpha_j : \mathcal{A} \rightarrow [0, 1]$, we obtain an event log $\mathcal{L}_j^{sc}$, which can be used to compute the probability distribution function $\mathcal{D}_j^{sc}$ of trace duration, along with the waiting time probability distribution $dist\_w_j^{sc}(a)$ for each activity $a \in \mathcal{A}$ (see Section 3).

The probability distributions $dist\_p_j^{sc}(a)$ for all $a \in \mathcal{A}$ can be added to the initial simulation model $\mathcal{M}_0 = (\mathcal{N}, \mathcal{S}_0)$, yielding a simulation model $(\mathcal{N}, \mathcal{S}_j)$. The simulation model $(\mathcal{N}, \mathcal{S}_j)$ can be run so as to obtain an event log $\mathcal{L}_j^{sim}$. Log $\mathcal{L}_j^{sim}$ can be used to compute the probability distribution function $\mathcal{D}_j^{sim}$ of trace duration, and the waiting time probability distribution function $dist\_w_j^{sim}(a)$ for each activity $a \in \mathcal{A}$.

Event logs $\mathcal{L}_j^{sc}$ and $\mathcal{L}_j^{sim}$ can now be compared considering the distance of the respective trace-duration distributions:

Definition 8 (Trace Duration Distance). *Let $\mathcal{L}_1$, $\mathcal{L}_2$ be two event logs. Let $\mathcal{D}_1 : \mathbb{R}_0^+ \rightarrow [0, 1]$ and $\mathcal{D}_2 : \mathbb{R}_0^+ \rightarrow [0, 1]$ be the probability distribution functions of the trace durations of $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. The trace duration distance is the integral difference between the probability distribution functions: $\varepsilon_{(\mathcal{L}_1, \mathcal{L}_2)} = \int_0^{+\infty} |\mathcal{D}_1(x) - \mathcal{D}_2(x)| dx$.*

Logs $\mathcal{L}_j^{sc}$ and $\mathcal{L}_j^{sim}$ can also be compared with respect to the distance of the waiting-time distributions:

Definition 9 (Waiting Time Distance). *Let $\mathcal{L}_1$, $\mathcal{L}_2$ be two event logs defined over the same set $\mathcal{A}$ of activities. For each activity $a \in A$, let $d_{w,a}^1 : \mathbb{R}_0^+ \rightarrow [0, 1]$ and $d_{w,a}^2 : \mathbb{R}_0^+ \rightarrow [0, 1]$ be the probability distribution functions of the waiting times for activity instance of a in $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. The waiting time distance for a is the integral difference between the probability distribution functions of waiting times: $\phi_{(\mathcal{L}_1, \mathcal{L}_2)}(a) = \int_0^{+\infty} |d_{w,a}^1(x) - d_{w,a}^2(x)| dx$.*

As mentioned, for each alpha function $\alpha_k \in \{\alpha_1, \ldots, \alpha_n\}$, we obtain a real event log $\mathcal{L}_k^{sc}$ augmented with start events, and a simulated event log $\mathcal{L}_k^{sim}$. We opt for the α_k that minimizes the distance $\Delta(\mathcal{L}_k^{sc}, \mathcal{L}_k^{sim})$ between $\mathcal{L}_k^{sc}$ and $\mathcal{L}_k^{sim}$, which consider the distances between the respective trace-duration and waiting-time distributions:

input : Event log: $\mathcal{L}$.
input : Simulation model: $\mathcal{M}_0 = (\mathcal{N}, \mathcal{S}_0)$.
input : Mintime function: $mintime : \mathcal{E}_{\mathcal{L}} \rightarrow \mathcal{T}$.
input : Granularity parameter: δ.

$\alpha_{best} \leftarrow set\_start\_alpha(\mathcal{L}, \mathcal{M}_0, \delta)$;
$\epsilon_{best} \leftarrow compute\_distance(\alpha_{best}, \mathcal{L}, \mathcal{M}_0, mintime)$;
for $a \in activities(\mathcal{N})$ **do**
  $\mathcal{Q}_{tried} \leftarrow [\alpha_{best}(a)]$;
  $\mathcal{Q}_{next} \leftarrow [prev\_succ(\alpha_{best}, \delta), next\_succ(\alpha_{best}, \delta)]$;
  while $\mathcal{Q}_{next} \neq []$ **do**
    $\alpha \leftarrow \alpha_{best}$;
    $\alpha(a) \leftarrow pick\_and\_remove(\mathcal{Q}_{next})$;
    if $\alpha(a) \notin \mathcal{Q}_{tried}$ **then**
      $\mathcal{Q}_{tried} \leftarrow \mathcal{Q}_{tried} \cup [\alpha(a)]$;
      $\epsilon \leftarrow compute\_error(\alpha, \mathcal{L}, \mathcal{M}_0, mintime)$;
      if $\epsilon \leq \epsilon_{best}$ **then**
        $\alpha_{best} \leftarrow \alpha,$;
        $\epsilon_{best} \leftarrow \epsilon$;
        $\mathcal{Q}_{next} \leftarrow [prev\_succ(\alpha_{best}, \delta), next\_succ(\alpha_{best}, \delta)]$;
      end
    end
  end
end
return α_{best}

Function $compute\_distance(\alpha, \mathcal{L}, \mathcal{M}_0, mintime) : \mathbb{R}_0^+$
  $\mathcal{L}^{sc} \leftarrow add\_start\_event(\mathcal{L}, mintime, \alpha)$;
  $dist\_p \leftarrow find\_processing\_time(\mathcal{L}^{sc})$;
  $\mathcal{M} \leftarrow set\_duration\_distribution(\mathcal{M}_0, dist\_p)$;
  $\mathcal{L}^{sim} \leftarrow simulate(\mathcal{M})$;
  $logs\_distance \leftarrow \Delta(\mathcal{L}^{sc}, \mathcal{L}^{sim})$;
  return $logs\_distance$
end

Algorithm 1: Local search-based algorithm to estimate activity start timestamps via simulation.

Definition 10 (Logs Distance). *Let $\mathcal{L}^{sc}$ be the original event log augmented with start events. Let $\mathcal{L}^{sim}$ be the event log obtained via simulation. Let $\varepsilon_{(\mathcal{L}^{sc}, \mathcal{L}^{sim})}$ be the trace duration distance for the two logs $\mathcal{L}^{sc}$ and $\mathcal{L}^{sim}$. Let $\phi_{(\mathcal{L}^{sc}, \mathcal{L}^{sim})}(a)$ be the waiting time distance for any activity $a \in \mathcal{A}$. The distance between $\mathcal{L}^{sc}$ and $\mathcal{L}^{sim}$ is computed as follows:*

$$\Delta(\mathcal{L}^{sc}, \mathcal{L}^{sim}) = \varepsilon_{(\mathcal{L}^{sc}, \mathcal{L}^{sim})} + \sum_{a \in \mathcal{A}} \phi_{(\mathcal{L}^{sc}, \mathcal{L}^{sim})}(a) \tag{3}$$

So far, the set of configurations were given. However, these configurations need to be computed on the fly to find a (sub)optimal minimum error. To this aim, we will use a local search based algorithm for this minimization problem. The pseudo-code in Algorithm 1. The proposed method takes as input an event log $\mathcal{L}$, an initial simulation model $\mathcal{M}_0 = (\mathcal{N}, \mathcal{S}_0)$, a $mintime_{\mathcal{L}}$ function, and a granularity parameter $\delta \in (0, 1)$. Using the δ parameter we can define the succession $alpha\_succ(\delta) = \{x_t | x_t = x_{t-1} + \delta, x_0 = 0, x_t \leq 1\}$, in such way we can obtain a different configuration obtained via function $\alpha(a) = x_t$ for each activity $a \in \mathcal{A}$. For example, using the parameter $\delta = 0.1$, the succession is $alpha\_succ(\delta) = \{0, 0.1, 0.2, \ldots, 1\}$.

The first step of the Algorithm 1 is to initialize the function α with random values. Starting from the initial function α, we select one activity $a \in \mathcal{A}$ and try to optimize

$\alpha(a)$ using local search. In particular, for each activity and the corresponding value $\alpha(a) = x_t \in alpha\_succ(\delta)$, for the next values of $\alpha(a)$ in $\mathcal{Q}_{next}$, we add the previous value $x_{t-1} \in alpha\_succ(\delta)$ and the consecutive value $x_{t+1} \in alpha\_succ(\delta)$. We store the value with the smallest logs distance and we keep going in the next updates in the direction with decreasing logs distance until no improvements are permitted, see Algorithm 1. To compute the logs distance given a configuration of the function α, we can create the new event log $\mathcal{L}^{sc}$ as discussed above, here abstracted as a function $add\_start\_event()$. The next step is, given the completed event log $\mathcal{L}^{sc}$, compute the function $dist\_p(a)$ that for each $a \in \mathcal{A}$. Then we incorporate the activity-duration probability distribution function $dist\_p()$ in the structure of the simulation model $\mathcal{M}_0$, obtaining the updated simulation model $\mathcal{M} = (\mathcal{N}, \mathcal{S})$. Given as input the simulation model $\mathcal{M} = (\mathcal{N}, \mathcal{S})$ the next step is to call the $simulate(\mathcal{M})$ function that calls a simulator and returns the simulated event log $\mathcal{L}^{sim}$ related to the simulation model $\mathcal{M}$. The last step is to calculate the logs distance $\Delta(\mathcal{L}^{sc}, \mathcal{L}^{sim})$, according to Equation 3. If the logs distance decreases then we update the function α_{best} and the logs distance ϵ_{best}, and we continue until no further improvement are observed.

5 Implementation and Experiments

This section assess the quality of the estimation of the timestamp of start activities for two case studies. In our experiments, we used the inductive miner algorithm [10] for the discovery of the process model because it guarantees the soundness of the discovered models. The resulting Petri-net models are later translated into BPMN models [11]. Then, using process mining and statistical techniques, we complement the BPMN model with the other simulation parameters, namely:

The resource perspective. We extract the pool of resources from the log events, and we group them in roles, finally linking BPMN-model tasks to roles. In particular, we leverage on the role-discovery algorithm by Burattin et al. [3].

Working calendar. We discover the working calendar hours for each role by analyzing the day of the week and the hour of the day in which each role most frequently completed tasks.

The inter-arrival time. We calculate the time difference between subsequent traces, also consider the working calendar. Then, we compute the inter-arrival time distribution that best fits, namely which minimizes the error.

Branching probabilities. For each XOR split in the BPMN model, we compute the probability to continue via each available branch.

These parameters focus on different process perspectives (resource, time and control-flow, respectively) and allow configuring a sufficiently-precise simulation model, with evident benefits on the accuracy of the estimation of the start events and their respective timestamps. The BPMN-model branching probabilities are derived via from the corresponding probabilities on the Petri net discovered through the Inductive leveraging, by using the Multi-perspective Process Explorer [12]. The other simulation parameters are obtained through a combination of the results of the process-mining library PM4Py[5] with other Python libraries for machine learning, data science and statistics.

[5] https://pm4py.fit.fraunhofer.de/documentation

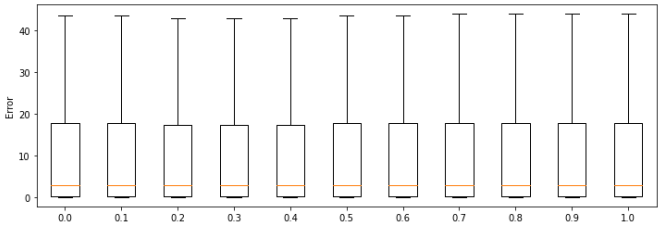

Fig. 3. The estimation error when varying the noise parameter of the inductive miner algorithm for the process for student credential recognition. The estimation error is measured in days, and is the absolute number of the between the estimated and the real timestamps.

To put together all the simulation parameters with the BPMN model into a single data structure the implementation is through a Python library BPSimpy [9]. It can be downloaded from the Github repository.[6] We used the Lanner simulator (L-sim)[7] using the generated simulation model as input, to perform the simulation. All the experiments were run on an Intel Core i7-8550U CPU @ 1.80GHz with 16GB RAM with duration of ca. 3.5 hours per analysis.

5.1 Case Study of the Process for Student Credential Recognition

This section reports a case study of a real business process for credential recognition of students in a Colombian University that contains both starting and completion events.[8] It contains 954 traces, 18 activities, 6870 events and involves 561 resources. In the remainder, we translate the activity labels into English for clarity. The start events were removed, aiming to assess the accuracy of their rediscovery via our simulation-based techniques. Once we discovered the BPMN process model and the simulation parameters, we have the initial simulation model $\mathcal{M}_0$. Using the $mintime_{\mathcal{L}}$ oracle based on control-flow and resources (cf. Section 3) and the granularity parameter $\delta = 0.1$, we apply Algorithm 1 to find the best α function.

Sensitivity with respect to the model. The first question of the evaluation is how much the quality of the model influences our technique, in terms of precision and fitness [1]. As mentioned we employed inductive miner to mine model, which allows the discovery of models with different sensitivity levels as function of a noise parameter that takes values between 0 and 1: value 0 indicates that no part of behaviour is considered as noise and the model allows for the whole behavior observed in the event log, whereas larger and larger values indicate that less and less event-log behavior is incorporated into the model. Since lower values produce models with a larger set of admissible behavior, in general lower values produce models with higher fitness but less precise. Figure 3 shows the estimation error when applying our technique and using the different models that

[6] https://github.com/claudiafracca/BPSimpyLibrary

[7] https://www.lanner.com/en-us/technology/l-sim-bpmn-simulation-engine.html

[8] https://github.com/AutomatedProcessImprovement/Simod/blob/master/inputs/ConsultaDataMining201618.xes

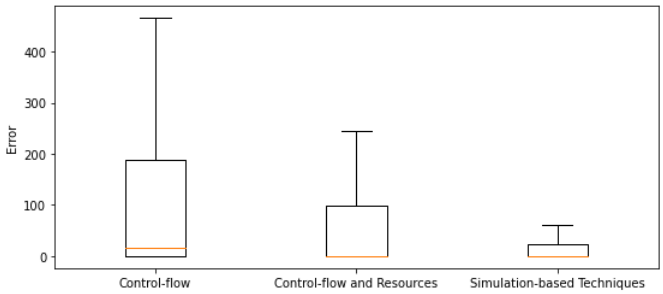

Fig. 4. Comparison of the estimation error of the start timestamp for the student credential-recognition process when using our simulation-based technique and the naïve techniques The first and second boxplots respectively report on the naïve techniques that use the control-flow information only, and both the control-flow and resource information. The third refers to our technique. Errors are reported in days.

are obtained varying the noise parameter of the inductive-miner algorithm and extended with the simulation parameters mentioned above. The estimation error is measured in days and is the absolute number of the between the estimated and the real timestamps. We can notice that the error does not change with the noise parameter and, hence, with the model precision and fitness. This can be explained: the possible lack of precision is balanced by the branching probabilities for the XOR splits. If the model is not precise, the probabilities for certain infrequent branches might become so low that it is in fact equivalent to not having them.

Comparison with naïve techniques. The second evaluation question refers to a comparison between our and some naïve techniques. The naïve techniques assume no waiting time and that the new activities start as soon as possible. In a first case, the new activity starts when the previous completes; a second case also considers the availability of resources and assumes that the new activity starts when the previous completes and when the resource that performs the new activity is available. Figure 4 uses boxplots to show the distribution of the estimation error between the two naïve techniques and our technique. The error for our technique is built on the model that scores the best when varying the noise parameter (cf. Figure 3). Our technique clearly outperforms the naïve technique that is based on only control-flow information: The mean and media values of the estimation error are certainly smaller, and the error's standard deviation reduced to ca. 10% of the case of the naïve technique based on control-flow information. Compared with the naïve technique that also employs resource information, our technique is characterized by a similar error's mean value, but the error's standard deviation is reduced to ca. 20% of the naïve-technique case that also uses the resource information.

Accuracy with respect to the α configuration. The next question of the evaluation is how the estimation error varies for a specific activity a with respect to $\alpha(a)$. Figure

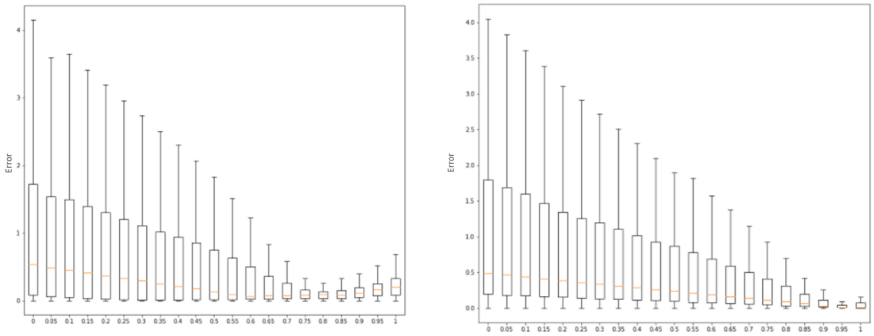

Fig. 5. Comparison by estimation error in days for the activities *Course revision (Revisar curso)* and *Validate the application (Validar solicitud)* between different parameter values α in $alpha\_succ(\delta) = \{0, 0.1, 0.2, \ldots, 1\}$ for the process for student credential recognition.

5 reports for the activities *Course revision (Revisar curso)* and *Validate the application (Validar solicitud)* a comparisons between error boxplots one for each values of function α in $alpha\_succ(\delta) = \{0, 0.1, 0.2, \ldots, 1\}$. This figure shows a typical distribution of the error function when varying α: the error has a convex trend, with the minimum for some value between zero and one. In this case, the minimum corresponds to the $\alpha(a) = 0.8$ and $\alpha(a) = 0.95$, illustrating that activities tend to be started soon after it is possible, but not immediately (cf. Equation 2).

Effects on Process-Instance Duration in Simulation. In sum, our technique allows a better estimation of the duration of the activity instances observed in the event log. We previously mentioned that this provides large benefits to the application of several Process-Mining techniques, including for more accurate simulation models. Assuming no waiting times, naïve techniques would lead to estimating distributions of activity-instance durations that are larger than the reality. If activity-instance durations are simulated to be larger than reality, simulations would highlight a unreal overestimation of the utilization of the resources that perform such activities, as well as they would report on overly long duration of process instances. To empirically verify this, we first used

Table 2. Average and standard deviation of process-instance duration of simulated process executions when activity duration is estimated via the naïve approach (first two table rows) and ours (third row). The last row refers to actual average and standard deviation of process-instance durations, observed in the event log.

	Average	Standard Deviation
Estimating using control-flow only	138d 09h 11m 28s	180 d 19h 25m 53s
Estimating using control-flow and resources	58d 08h 10m 32s	107d 09h 34m 31s
Estimating using our simulation-based technique	10d 19h 01m 56s	18d 20h 35m 21s
Actual process-instance durations in the event log	14d 10h 18m 11s	27d 03h 40m 51s

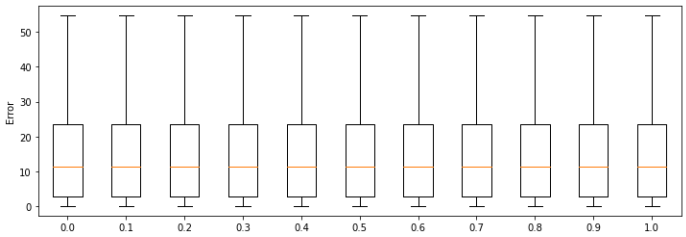

Fig. 6. The estimation error when varying the noise threshold of the inductive miner algorithm for the purchase process case study.

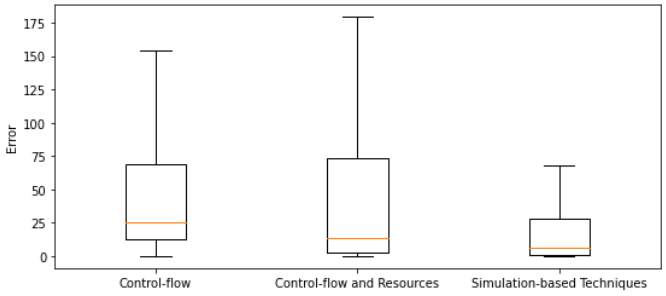

Fig. 7. Comparison of the estimation error of the start timestamp for the purchase process case study, when using our simulation-based technique and some naïve techniques that assume no waiting time.

the durations estimated via our techniques and via the naïve techniques to learn the probability distributions of the durations of the instances of different process' activities, which were later added to the simulation model. So-constructed simulation models were run for so many instances as those recorded in the event log. We measured the average and standard deviations of process-instance durations. Table 2 summarizes the findings: the comparison highlights that our technique allows analysts to build simulation models that simulate realistic process-instance durations. Indeed, the average of 10 days and ca. 19 hours and the standard deviation of 18 days and ca. 21 hours is close to the actual durations (see last table row). Conversely, the two naïve techniques estimated activity-instance durations that produced distributions for the simulation model that generate process-instance runs whose durations are of an order of magnitude larger than reality.

5.2 Purchase Process Case Study

This section reports the experiment results based on an event log that record executions of a purchase-to-pay process of 21 activities and 27 resources.[9] This event log is composed by 9119 events, both start and complete, divided in 608 traces. Analogously to the

[9] https://fluxicon.com/academic/material/

previous case study, we removed the start events, aiming to assess the accuracy of their rediscovery via our simulation-based techniques. As the case study before, the simulation model was created via Inductive Miner, extended with the simulation parameters that are mentioned at the beginning of this section and discovered as there indicated. The same sensitivity analysis of the model wrt. different noise-parameter values was carried out (see Figure 6): analogously to the previous case study, the accuracy of the estimations was not influenced by the quality of the model to well balance fitness and precision. So did we compare our techniques with the same naïve techniques that use control-flow and, possibly, resource information. The results are shown in Figure 7, which confirm the findings of the first case study: our technique reduces both the average and the standard deviation of the estimation error.

6 Conclusion

Process mining techniques allows process analysts to discover, monitor, and extract information from an event log related to different perspectives of a business process. However, most of the techniques required event logs that stored both start and complete life-cycle transitions to extract the different multi perspectives aspects. Unfortunately, event logs often record the completion only. This paper focuses on the approximating of the start events. Naïve techniques assume activity instances to start as soon as possible, thus assuming no waiting times. In reality, resources can work on different processes, take breaks, and perform other duties that are not recorded in the logs. In these cases, the actual durations are over-estimated, yielding simulations that report an untrue over-utilization of resources and/or process-instance durations that are far longer than the reality. This paper reports on an technique to estimate the start event, assuming waiting times to be possible. The idea is to estimate the start event by simulating different activity-duration configurations and comparing the simulation results to find the configuration for which the process-instance durations follow a probability distribution that is close to that of the original event log. The idea hint is that the duration of simulated traces are closer to those of the original event log if the activity durations are better estimated. The validation has been conducted on processes whose event logs record both starting and completion timestamps for each activity instance. The starting timestamps were removed and estimated: the results show that our technique computes better duration estimation with respect to techniques that assume no waiting time.

As future work, we aim to optimize our technique. Each configuration (i.e. vector) of α parameters needs to be simulated, and currently we simulate as many traces as those of the original event log in an attempt to fade any simulation warm-up effects. In fact, it might actually be statistically sufficient to simulate fewer traces, so as to speed up the simulation steps, while still fading the warm-up effects.

We acknowledge that there might be some threats to the validity of our technique, on which we aim to work in the future. First and foremost, the accuracy of the estimations of the start timestamps might depend on the accuracy of the simulation model. Secondly, our technique still assumes resources to work on at most one activity instance at the same time, which might not always be true: we aim to extend support for resources that carry on multiple activity instances at the same time [8].

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer (2016)
2. Berkenstadt, G., Gal, A., Senderovich, A., Shraga, R., Weidlich, M.: Queueing inference for process performance analysis with missing life-cycle data. In: Proceedings of 2nd International Conference on Process Mining (ICPM 2020). pp. 57–64. IEEE (2020)
3. Burattin, A., Sperduti, A., Veluscek, M.: Business models enhancement through discovery of roles. In: Proceedings of the 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM). pp. 103–110 (2013)
4. Camargo, M., Dumas, M., González Rojas, O.: Learning accurate business process simulation models from event logs via automated process discovery and deep learning. arXiv **abs/2103.11944** (2021)
5. Camargo, M., Dumas, M., González-Rojas, O.: Automated discovery of business process simulation models from event logs. Decision Support Systems (2020)
6. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer Publishing Company Incorporated, 1st edn. (2018)
7. Denisov, V., Fahland, D., van der Aalst, W.M.P.: Repairing event logs with missing events to support performance analysis of systems with shared resources. In: Proceedings of the 41st Intl. Conference on the Application and Theory of Petri Nets and Concurrency (PETRINET 2020). LNCS, vol. 12152, pp. 239–259. Springer (2020)
8. Estrada-Torres, B., Camargo, M., Dumas, M., García-Bañuelos, L., Mahdy, I., Yerokhin, M.: Discovering business process simulation models in the presence of multitasking and availability constraints. Data & Knowledge Engineering **134**, 101897 (2021)
9. Fracca, C., Bianconi, A., Meneghello, F., de Leoni, M., Asnicar, F., Turco, A.: BPSimpy: A python library for WfMC-standard process-simulation specifications. Proceedings of the Demo Session at the 19th International Conference on Business Process Management (2021)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: Proc. of the 35th Intl. Conference on the Application and Theory of Petri Nets and Concurrency. LNCS, vol. 8489, pp. 91–110. Springer (2014)
11. de Leoni, M., van der Aalst, W.M.P.: The FeaturePrediction Package in ProM: Correlating Business Process Characteristics. In: Proceedings of the Demo Session at the 14th International Conference on Business Process Management (BPM 2014). CEUR, vol. 1295 (2014)
12. Mannhardt, F., de Leoni, M., Reijers, H.A.: The multi-perspective process explorer. In: Proceedings of the Demo Session at the 13th International Conference on Business Process Management. vol. 1418, pp. 130–134. CEUR-WS.org (2015)
13. 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)
14. Nakatumba, J.: Resource-aware business process management: Analysis and Support. Ph.D. thesis, Technische Universiteit Eindhoven (2013)
15. Pegoraro, M., van der Aalst, W.M.P.: Mining uncertain event data in process mining. In: Proc. of the 2nd Intl. Conference on Process Mining (ICPM 2019). pp. 89–96. IEEE (2019)
16. Pegoraro, M., Uysal, M.S., van der Aalst, W.M.P.: Discovering process models from uncertain event data. In: Proceedings of the Business Process Management Workshops. vol. 362, pp. 238–249. Springer (2019)
17. Rogge-Solti, A., Weske, M.: Prediction of business process durations using non-markovian stochastic petri nets. Information Systems **54**, 1–14 (2015)
18. Rozinat, A., Mans, R., Song, M., van der Aalst, W.: Discovering simulation models. Information Systems **34**(3), 305–327 (2009)
19. Senderovich, A., Leemans, S.J.J., Harel, S., Gal, A., Mandelbaum, A., van der Aalst, W.M.P.: Discovering queues from event logs with varying levels of information. In: Proceedings of the Business Process Management Workshops. LNBIP, vol. 256. Springer (2016)