Data & Knowledge Engineering 134 (2021) 101897

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Discovering business process simulation models in the presence of multitasking and availability constraints

Bedilia Estrada-Torres [a,b,*], Manuel Camargo [b], Marlon Dumas [b], Luciano García-Bañuelos [c], Ibrahim Mahdy [b], Maksym Yerokhin [b]

[a] *Universidad de Sevilla, Sevilla, Spain*
[b] *University of Tartu, Tartu, Estonia*
[c] *Tecnológico de Monterrey, Monterrey, Mexico*

ARTICLE INFO

Keywords:
Multitasking
Process mining
Process simulation
Resource availability
Timetables

ABSTRACT

Business process simulation is a versatile technique for quantitative analysis of business processes. A well-known limitation of process simulation is that the accuracy of the simulation results is limited by the faithfulness of the process model and simulation parameters given as input to the simulator. To tackle this limitation, various authors have proposed to discover simulation models from process execution logs, so that the resulting simulation models more closely match reality. However, existing techniques in this field make certain assumptions about resource behavior that do not typically hold in practice, including: (i) that each resource performs one task at a time; and (ii) that resources are continuously available (24/7). In reality, resources may engage in multitasking behavior and they work only during certain periods of the day or the week. This article proposes an approach to discover process simulation models from execution logs in the presence of multitasking and availability constraints. To account for multitasking, we adjust the processing times of tasks in such a way that executing the multitasked tasks sequentially with the adjusted times is equivalent to executing them concurrently with the original times. Meanwhile, to account for availability constraints, we use an algorithm for discovering calendar expressions from collections of time-points to infer resource timetables from an execution log. We then adjust the parameters of this algorithm to maximize the similarity between the simulated log and the original one. We evaluate the approach using real-life and synthetic datasets. The results show that the approach improves the accuracy of simulation models discovered from execution logs both in the presence of multitasking and availability constraints.

1. Introduction

Business process simulation (BPS) is a widely used technique for analyzing quantitative properties of business processes. The basic idea of BPS is to execute a large number of instances of a process, based on a process model enhanced with simulation parameters, with the goal of collecting performance measures such as waiting times of tasks, processing times, execution cost, and cycle time [1,2]. BPS tools (simulators) allow analysts to identify performance bottlenecks [3] and to estimate how a given change to a process may affect its performance [4].

The accuracy of a business process simulation, and hence the usefulness of the conclusions drawn from it, is to a large extent dependent on how faithfully the process model and simulation parameters capture the observed reality. Traditionally, process models

* Corresponding author at: Universidad de Sevilla, Sevilla, Spain.
E-mail address: iestrada@us.es (B. Estrada-Torres).

https://doi.org/10.1016/j.datak.2021.101897
Received 6 December 2020; Received in revised form 30 April 2021; Accepted 4 May 2021
Available online 21 May 2021

are manually designed by analysts for the purpose of communication and documentation. As such, these models do not capture all the intricacies of how the process is actually performed. In particular, manually designed process models tend to focus on frequent pathways, leaving aside exceptions. Yet, in many cases, exceptions occur in a non-negligible percentage of instances of a process. Moreover, simulation parameters for BPS are traditionally estimated based on expert intuition, sampling, and manual curve fitting, which do not always lead to an accurate reflection of reality [3].

To tackle these limitations, several authors have advocated the idea of automatically discovering simulation models from business process execution logs (also known as *event logs*) [5–7]. Simulation models discovered in this way are generally more faithful since they capture not only common pathways, but also exceptional behavior. Moreover, automated approaches to simulation model discovery typically explore a larger space of options when tuning the simulation parameters compared to what an analyst is able to explore manually.

Additionally, the automated discovery of BPS models from event logs opens up the possibility of capturing resource behavior at a finer granularity than manual BPS modeling approaches. However, existing techniques in this field make highly restrictive assumptions about resource behavior, which do not typically hold in practice [8], notably:

1. They assume that each resource performs one task at a time. In other words, they exclude the possibility of *multitasking*. In this setting, multitasking refers to the situation where a resource executes multiple task instances simultaneously, meaning that the resource divides its attention across multiple active task instances [9].
2. They assume that resources are continuously available (24 hours a day, 7 days a week). In other words, they do not take into account resource availability constraints stemming from work schedules and natural circadian cycles.

This article proposes an approach to discover BPS models from execution logs that takes into account multitasking and resource availability constraints. To account for multitasking, we adjust the processing times of tasks in such a way that executing the multitasked tasks sequentially with the adjusted times is equivalent to executing them concurrently with the original times. Once the event log is adjusted to account for multitasking, we discover a BPS model using an existing BPS model discovery technique, namely SIMOD [10].

Meanwhile, to account for availability constraints, we apply an algorithm for discovering calendar expressions from collections of time-points to infer resource timetables from an execution log. We then use the discovered calendar expressions to construct a resource availability timetable and we inject these timetables into BPS models discovered by SIMOD. Finally, we adjust the parameters used to discover calendar expressions so as to maximize the similarity between the log generated by the simulation (the simulated log) and the original log.

This article is an extension of a previous conference paper [11]. The conference paper focused on the problem of accounting for multitasking behavior. In this paper, we extend this approach in order to discover resource availability timetables (i.e. to address the second limitation outlined above).

The rest of this article is structured as follows. Section 2 motivates our research. Section 3 introduces basic concepts and related work. Section 4 describes the proposed approach to identify the multitasking behavior, while Section 5 explains the approach for handling resource availability constraints. Section 6 reports the proposal's evaluation from the two points of view: multitasking and resource availability effects. Section 7 describes the threats to validity identified. Finally, Section 8 draws conclusions and outlines directions for future work.

2. Motivation

This section discusses the notions of multitasking and availability constraints, as well as the implications of not taking into account these notions during the discovery of a BPS model from an event log.

2.1. Multitasking behavior

The execution of a business process is often tracked by events stored in so-called *event logs*. Each event in the event log represents a change of status in the life-cycle of an instance of a task (herein called a *work item*). Each event conveys information about the resource that executed a work item, as well as data used and/or produced by the work item.

Assuming that an event log captures the start time and the end time of each work item, it is possible to estimate the duration of each work item. We can assert that the resource is active (available to work) when the work item is started and when the work item is completed. However, we cannot assert with full certainty that the work item is available during other time points in between the start and the end timestamp. For example, a resource might start a work item at 17:00 on a Friday and finish it at 10:00 on Monday. We can ascertain that the resource was at work on Friday at 17:00 and on Monday at 10:00, but we cannot ascertain that the resource is available for example on Saturday or Sunday. Accordingly, in the general case we assume that it is possible to discover the duration of a work item for complete "granules" of time adjusted to the timestamps associated to the start and end of each work item, but not any time points in-between them.

Sometimes, the work items associated with a given resource in an event log may show that the resource started a task instance before completing a previous one. Hence, during some periods, the resource simultaneously performs multiple task instances, a situation known as *multitasking*. Multitasking arises, for example, when a resource postpones the completion of a task due to missing information. While this information becomes available, the resource may start another task instance to avoid idle times.

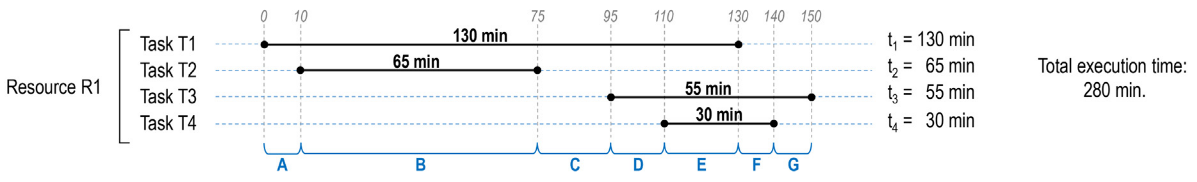


Fig. 1. Example of multitasking for the resource $R1$.

Fig. 1 represents a subset of four tasks carried out by resource $R1$, where each continuous line represents the duration of each task. These four tasks result in seven execution intervals. In intervals A(0–10), C(75–95) and G(140–150) only one task is executed, T1, T1 and T3, respectively. Other segments reflect multitask execution: in B(10–75), D(95–110) and F(130–140) two tasks are executed (T1, T2), (T1, T3) and (T3, T4), respectively; and in interval E(110–130) multitasking is performed between three tasks (T1, T3, T4). These tasks may belong to one or more traces. A *trace* contains the ordered sequence of work items observed for a given process instance [2]. An event log is composed of one or more traces.

Given this event log segment's data, a traditional simulator would calculate a total execution time of 280 min because it would take each duration individually, one task after the other. However, in Fig. 1, it is possible to see that all tasks are executed between the 0 and the 150 minute. That means that the resource $R1$ divided its time and attention into more than one task during certain intervals. Therefore, it would not be correct to consider as total task execution time the time between the start and end record of the task, but the time should be distributed among all the tasks that overlap in a given period.

Usually, there is no detailed record of each resource's specific time on the execution of each task in multitasking scenarios. Therefore, we consider it necessary to propose a mechanism to adjust processing times to reflect the time spent by each resource more accurately.

2.2. Resource availability constraints

Existing approaches to discover BPS models from event logs assume that the resources involved in the business process do not have any availability constraints. In other words, they assume that resources are available to perform tasks continuously, 24 h a day, 7 days a week. In practice, resources are only available during certain time periods, usually determined by weekly or daily cycles, and possibly also seasonal (annual) cycles. For example, in a typical organization, employees only carry out activities Monday through Friday, from 8:00 to 17:00. Outside this timetable, one would not expect a worker to be available to perform tasks.

Similarly, in some organizational settings, new instances of a process are only be created during certain time-slots. For example, consider a wholesaler that distributes products to retailers. Typically, the purchasing managers of the retailers only approve new purchase orders during working hours. Hence, new cases of the wholesaler's order-to-cash process are only created during business hours and not during weekends or at night-time. In other words, the creation of new cases in this process is constrained to a certain timetable.[1]

A BPS model that does not take availability constraints into account is likely to produce simulations that are not temporally aligned with reality. For example, let us consider a loan application handling process in which loan applications are accepted online 24/7, except Sundays between midnight and 3 am due to weekly server maintenance. Let us further assume that the loan application handlers and credit officers responsible of performing the tasks in the process are only available Monday to Friday 9:00–17:00. In such a process, loan applications that arrive during the weekend need to wait until Monday to be processed. This means that those applications will have a longer total cycle time (case duration). Also, this means that the loan handlers and credit officers are likely to have a higher workload on Mondays, which will cause that applications received on Monday will also have slower cycle times compared to those who arrive later in the week, once the weekend backlog has been handled. The BPS model will not be able to take these factors into account, leading to an inaccurate distribution of cycle times.

Existing business process simulators are capable of taking into account availability constraints via so-called *timetables*. For example, an example of a timetable is: *Mondays–Thursdays 8:00–12:00 and 13:00–17:00, and Fridays 8:00–12:00 and 13:00–16:30*. Such timetables can be attached to resource pools. For example, one can specify that loan handlers work according to the above timetable. A timetable can also be used to temporally constrain the creation of new cases of the process. For example, one can specify that new cases are only created on Mondays–Fridays, 7:00–18:00. In this article, we proposed to enhance automatically discovered BPS models with timetables (both resource timetables and case timetables) discovered from an event log, and we study the impact of adding such timetables on the accuracy of the resulting BPS models.

3. Related work

Existing business process simulation techniques make various simplifying assumptions that restrict their ability to accurately capture reality. In particular, existing business process simulators have several deficiencies in the way they capture the behavior

[1] Note that this is not a hard constraint. Purchase orders may be submitted by a retailer outside business hours. However, this happens rarely. For practical purposes, we can consider that cases are only created during business hours.

of resources as highlighted in [12]. This latter work notes that existing techniques adopt a resource allocation model based on a First-In-First-Out allocation approach. This approach works as follows. When a task T is ready to be executed within an instance of the process, a *work item* is created. Each task T is associated with a *resource pool*, which consists of a set of resources. If a resource is available within the resource pool associated with T, the resource is assigned to one such resource. Otherwise the work item is added to a first-in-first-out queue until a resource becomes available. Once the work item is assigned to a resource, the work item immediately starts. The simulator determines the work item's duration and the work item is blocked until its duration has elapsed. When the duration has elapsed, the work item is marked as completed, and the resource that had been assigned to it becomes available to be assigned to another work item [2].

The above approach does not take into account that resources may be assigned to tasks in a range of different ways, and not just according to this first-in-first-out allocation approach. A wide of different approaches for assignment of resources to tasks is provided by Russel et al. [13] have identified a range of other ways in which resources are assigned to tasks in practical scenarios. Existing simulators do not take into account this range of possible workflow resource patterns [1,8].

Afifi et al. [14,15] propose an extension of BPSim, a Business Process Simulation Standard [16], to take into account the workflow resource patterns of Russel et al. [13]. The authors do not provide a concrete implementation of their proposed BPSim extension (they leave it as future work). With respect to this previous work, the present study differs in that it focuses specifically on handling multitasking and availability constraints, which are resource behaviors not captured by the resource patterns defined in [13].

Ling et al. [17] describe a simulation tool that considers differences between resources based on their experience and on personnel movements such as recruitment, transfer, and resignation. However, this proposal does not consider neither multitasking nor availability constraints.

Ouyang et al. [9] point out that real business processes are resource-intensive, where multitasking situations are typical. Their study focuses on proposing an approach to model and to schedule the use of shared material resources, such as surgical material shared by multiple doctors during a surgical operation that involves multitasking. Unlike our proposal, the authors do not analyze real execution data nor do they deal with the question of how to capture multitasking in a business process simulation.

Rusinaite et al. [3] propose to classify resources (human or not) into *shareable resources* (a resource that can be used by multiple activities simultaneously) and *non-shareable resources* in the context of process simulation. A resource is defined by means of attributes of *capacity* (reusable or consumable) and *shareability* (shareable and non-shareable). Multitasking can be seen as a situation where the same resource (a worker) is "shared" by multiple work items. As such, there is a similarity between our work and that of Rusinaite et al. [3]. However, this latter work does not deal with the question of how to discover that human resources are shared by multiple work items (and to what extent) by analyzing an event log.

In [18] a method is described for discovering so-called *resource availability calendars* from event logs. The method consists of two phases. In the first phase, the method derives, for each of the resources referenced in the log, a summary of the periods of time when a given resource is actively working on a *work item* and the periods of time when said resource is idle. To this end, the method uses various heuristics. For instance, it combines the processing intervals of two consecutive activities, even when there is a (short) pause between these activities. At the end, for each resource and each calendar day (e.g. for the day 10 November 2020), the method derives a so-called *working day specification*, i.e., a summary of the periods during which the resource is either working or idle during that calendar day. In a second phase, the method generates a *resource availability calendar* for a given period (e.g., one or two weeks) by randomly sampling from the working day specifications generated in the first phase.[2]

Martin et al. [18] suggest that the resource availability calendars generated by their approach can be used in the context of business process simulation. However, they do not evaluate if and to what extent their approach would enhance the accuracy of a BPS model. One of the limitations of their method is that it discovers calendars for each individual resource. However, many existing business process simulators require that the calendars are defined at the level of resource pools. In contract, in this article we propose an alternative method that discovers calendars at any level (a calendar for an individual resource or for an entire resource pool). A second limitation of the method in [18] is that it does not rely on a measure of support or confidence. In other words, the method does not allow one to tune the accuracy of the resulting calendars (relative to the observed data). In contrast, in this paper we use a calendar discovery technique that relies on confidence and support measures, making it possible to tune the accuracy of the discovered calendars. Finally, [18] does not consider the discovery of calendars associated with the creation of cases, whereas the approach we propose in this paper can discover both resource availability calendars as well as case creation calendars.

4. Handling multitasking

As explained in Section 2, a resource can start a *work item* before finishing one or more *work item* he/she started before, but simulators are not capable of taking this behavior into account. To cope with this lack, we propose to pre-process event logs to adjust the processing times, which is to proportionally divide the interval of execution time where different tasks intersect by the number of tasks involved. In this way, multitasking can be approached without modifying the structure and operations of the simulators. Fig. 2 shows how the duration of multitasked intervals (Fig. 1) are distributed proportionally among the number of tasks in each interval. For example, in interval *B*, the total time (65 min) is divided proportionally between tasks T1 and T2 (32.5 min for each); or in segment *C*, three tasks are executed, so the 20 min of its duration are divided between tasks T1, T3, and T4. In this way, the new task execution times are more similar to the real dedication of the resource.

[2] [18] also proposes an alternative approach in which the resources are first partitioned into clusters, and the centroids of these clusters are used as the samples (as an alternative to random sampling).

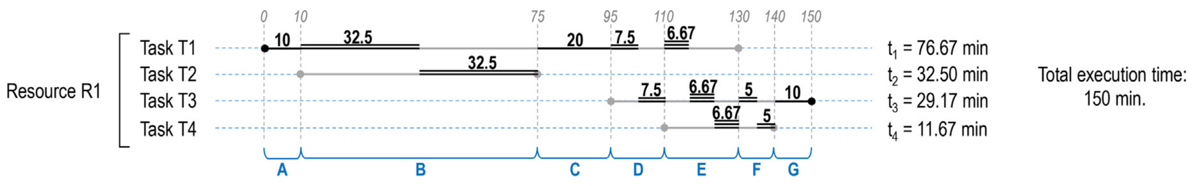


Fig. 2. Example of time adjustments derived from multitasking.

The objective of pre-processing event log is, on the one hand, to identify the resources that perform multitasking, determine in which periods the multitasking execution is performed, and to make an adjustment of the *work item* duration times according to the multitasking periods. On the other hand, to determine how the multitasking execution intervals influence the general performance of the business process. We assume that resources are involved in only one business process at a time.

The following definitions describe step-by-step how the event log is pre-processed. To do this, we begin by formally defining the concepts of *event*, *trace*, *event log* and *work item*.

Definition 1 (*Events, Attribute*)**.** Let $\mathcal{E}$ be the set of all possible events that occur during a process execution. Let us assume an event e can be described by means of a set of attributes att, where $att = \{id, type, r, st, et\}$, id is the identifier of the event; $type$ represents the event type, the activity name; r represents the resource that performs the event; st indicates the event start timestamp; et indicates the event end timestamp. In such a way that, for example, $att_r(e) = e_r = r_1$, where r_1 is a particular resource performing e.

Definition 2 (*Trace*)**.** Let $\mathcal{T}$ be the set of all possible traces defined as a sequence of events, such that, $\sigma \in \mathcal{T}, \sigma = \langle e_1, e_2, \ldots, e_n \rangle$

Definition 3 (*Event Log*)**.** An event log can be defined as a set of traces, $\mathcal{L} \subseteq \mathcal{T}$, where $\mathcal{L} = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$

Definition 4 (*Work Item*)**.** Let wi be a *work item* representing the execution of a task in a business process simulation. A trace can be represented as a sequence of *work items*, such that $\sigma \in \mathcal{T}, \sigma = \langle wi_1, wi_2, \ldots, wi_n \rangle$.

As with events, a *work item* has the set of attributes att. For example, $att_r(wi) = wi_r = r_1$, where r_1 is the resource that has the wi assigned to it.

Multitasking can be generated by *work items* generated in a single trace or by *work items* belonging to different traces. In this proposal, the broadest case is considered, so every trace in which each resource participates is considered. In order to identify the task (and *work items*) in which a resource perform multitasking, the log $\mathcal{L}$ is divided into as many *Segment per Resource* as there are resources in the log. Each segment consists of all the *work items* of each resource in $\mathcal{L}$, which will be ordered according to the start timestamp of each *work item* (wi_{st}). Fig. 1 represents one *Segment per Resource* (sr_1) with four *work items* for resource $R1$.

Definition 5 (*Segment per Resource*)**.** Given an event log $\mathcal{L}$, $\mathcal{R}$ represents the set of all possible resources that execute at least one work item in any trace in a log $\mathcal{L}$. Such that, $\forall r \in \mathcal{R}, \exists wi \subseteq \mathcal{T} \subseteq \mathcal{L} \mid wi_r = r$

Then, let $\mathcal{S}$ be the set of all possible ordered subsets of work items conforming the traces of a log, in such a way that $\mathcal{L} = \{sr_1, sr_2, \ldots, sr_n\}$, where $\forall sr_i \in \mathcal{S}, sr_i = \langle wi_j, \ldots, wi_m \rangle \mid (wi_{j_r} = wi_{j+1_r} = \cdots = wi_{m_r}) \wedge (wi_{j_{st}} \leq wi_{j+1_{st}} \leq \cdots \leq wi_{m_{st}})$, where $1 \leq j \leq n$.

Having divided $\mathcal{L}$ into different (sr_i), the *Sweep Line algorithm* [19] is applied to each sr_i to identify intersection points between *work items* determined by their start and end timestamps. For each pair of intersection points between the different *work items*, *auxiliary work items* ($wiaux$) are created.

To identify the set of $wiaux$, first, for each *segment per resource* sr_i an ordered list of time ($ordtimes$) is created, where $ordtimes = \{\ point_1, point_2, \ldots, point_n\ \}$. Each element of the list, called $points$, is a tuple $point_i = (tstamp_i, wiid_i, symbol_i)$, where $tstam_i$ could be a start timestamp or an end timestamp of any of the work items in sr_i; $wi_{i_{id}}$ is the identifier of the work item with start or end timestamp equals to $tstamp_i$; $symbol_i$ could be '+' if $tstamp_i$ corresponds to a start timestamp, or '-' if it is an end timestamp; and wi_i is the complete *work item* used to obtain the other values of the tuple.

Definition 6 (*Ordered List of Times*)**.** $\forall sr_i \subseteq \mathcal{L}, \exists times_i, ordtimes_i \mid \{(wi_{j_{st}}, wi_{j_{id}}, \text{'+'}), (wi_{j_{et}}, wi_{j_{id}}, \text{'−'}), \ldots, (wi_{n_{st}}, wi_{n_{id}}, \text{'+'}), (wi_{n_{et}}, wi_{n_{id}}, \text{'−'})\} \wedge ordtimes_i = \{(tstamp_k, wiid_k, symbol_k), (tstamp_{k+1}, wiid_{k+1}, symbol_{k+1}), \ldots, (tstamp_l, wiid_l, symbol_l)\} \wedge tstamp_k \leq tstamp_{k+1} \leq \cdots \leq tstamp_l \wedge |times_i| = |ordtimes_i|$, where $((tstamp_x = wi_{x_{st}} \vee tstamp_x = wi_{x_{et}}); wiid_x = wi_{x_{id}}; symbol_x \subset \{\text{'} + \text{'}, \text{'} - \text{'}\}$.'

Once the $ordtimes_i$ has been created, concrete intervals of time $intervals_i$ are specified, identifying also the work items wi_n that are being executed for each interval, $intervals_i = \{(start\_int_1, end\_int_1, list\_wiids_1), \ldots, (start\_int_k, end\_int_k, list_w iids_k)\}$, where $start\_int$ and end_int represent the start and end timestamp of the intersected work items collected in $list\_wiids$. For each element in $list\_wiids$ an auxiliary work item $wiaux$ is created, in such a way that $wiaux = (start\_int, end\_int, id, duration)$, where $wi = \{wiaux_1, \ldots, wiaux_n\}$ and $wi_{et} - wi_{st} = \sum_{n=1}^{n} wiaux_{n_d}$. The $duration$ of each $wiaux$ is determined by the number of $wiaux$ generated from a given

Algorithm 1: Creating $wiaux$ elements in a $lwiaux$

```
Input: Ordered list of times $ordtimes_i$
Output: List of auxiliary work items $lwiaux$
temp_ids = []; intervals = []; lwiaux = []; id = 1
for i in range(0,len(ordtimes)-1) do
    if (exists(ordtimes[i+1])) then
        if ordtimes[i]['symbol'] == '+') then
            temp_ids.append(ordtimes[i]['wiid'])
        else
            temp_ids.remove(ordtimes[i]['wiid'])
        intervals.append(ordtimes[i]['tstamp'], ordtimes[i+1]['tstamp'], temp_ids)
for interval in intervals do
    for wiid in interval['list_wiid'] do
        lwiaux.append(id, interval['start_int'], interval['end_int'], interval[list_wiid]['wiid'])
        id += 1
```

Table 1
Intermediate values obtained from Definition 6 and Algorithm 1.

ordtimes	=	{(0, A, '+'), (10, B, '+'), (75, B, '-'), (95, C, '+'), (110, D, '+'), (130, A, '-'), (140, D, '-'), (150, C, '-')}
intervals	=	{(0, 10, 'A'), (10, 75, 'A,B'), (75, 95, 'A'), (95, 110, 'A,C'), (110, 130, 'A,C,D'), (130, 140, 'C,D'), (140, 150, 'C')}
lwuiaux	=	{(0, 10, 'A', 10), (10, 75, 'A', 32.5), (10, 75, 'B', 32.5), (75, 95, 'A', 20), (95, 110, 'A', 7.5), (95, 110, 'C', 7.5), (110, 130, 'A', 6.67), (110, 130, 'C', 6.67), (110, 130, 'D', 6.67), (130, 140, 'C', 5), (130, 140, 'D', 5), (140, 150, 'C', 10)}

$interval, duration = (end\_int - start\_int)/\ len(list_w iids)$. For example, from $interval = (10, 75, \text{‘}wi_1, wi_2\text{’})$ two $wiaux$ are generated $wiaux_1 = (10, 75, \text{‘}wi_1\text{’}, 32.5)$, $wiaux_2 = (10, 75, \}wi_2\text{’}, 32.5)$, $wiaux_2 = (10, 75, \text{‘}wi_2\text{’}, 32.5)$. The list $lwiaux$ contains all $wiaux$ generated.

Based on the above definitions, Algorithm 1 describes how the adjustment of task execution times is performed, taking into account the number of tasks that are simultaneously executed by a resource, by means of the creation of the $lwiaux$ list. Applying Definition 6 and the Algorithm 1 to the scenario depicted in Figs. 1 and 2, the set of values presented in Table 1 are obtained.

Given an event log $\mathcal{L}, len(\mathcal{L})$ indicates the number of work items in $\mathcal{L}$. And according to the above definitions, it is possible to state that $lwiaux = \mathcal{L}'$, where $\mathcal{L}'$ is defined as:

Definition 7 (*Auxiliary Event Log ($\mathcal{L}'$)*)**.** Given a $\mathcal{L}$, $\forall \mathcal{L} = \langle wi_1, wi_2, \dots, wi_n \rangle, \exists \mathcal{L}' \mid \mathcal{L} \equiv \mathcal{L}' \wedge \mathcal{L}' = \langle wiaux_1, wiaux_2, \dots, wiaux_m \rangle$, where $wi_i =< wiaux_j, \dots,\ wiaux_k >, 1 \le i \le n, 1 \le j \le k, m \ge len(\mathcal{L})$.

From $\mathcal{L}'$ it is possible to generate a "coalesced log" $\mathcal{L}''$ that contains a set of coalesced work items $wicoal$. Each $wicoal$ is the result of the sum of the pre-processed times ($wiaux$) of each original wi in $\mathcal{L}$.

Definition 8 (*Coalesced Log ($\mathcal{L}''$)*)**.** $\forall \mathcal{L} = \langle wi_1, \dots, wi_n \rangle, \mathcal{L}' =< wiaux_1,\ \dots, wiaux_m > \exists \mathcal{L}'' = \langle wicoal_1, \dots, wicoal_n \rangle \mid wicoal_{i_{id}} = wi_{i_{id}} \wedge wicoal_{i_{type}} = wi_{i_{type}} \wedge wicoal_{i_r} = wi_{i_r} \wedge wicoal_{i_{st}} = wi_{i_{st}} \wedge wicoal_{i_{et}} = (wicoal_{i_{st}} + sum_{t=1}^{m} wiaux_{t_d}) \wedge len(\mathcal{L}) = len(\mathcal{L}'') \wedge [sum_{t=1}^{n}(wi_{t_{et}} - wi_{t_{st}}) = sum_{t=1}^{n}(wicoal_{t_{et}} - wicoal_{t_{st}})]$.

From the above definitions we can deduce that: $\forall \mathcal{L} \exists \mathcal{L}', \mathcal{L}'' \mid \mathcal{L} \equiv \mathcal{L}' \equiv \mathcal{L}'' \wedge len(\mathcal{L}) \le len(\mathcal{L}') \wedge len(\mathcal{L}') \ge len(\mathcal{L}'') \wedge len(\mathcal{L}) = len(\mathcal{L}'')$.

In addition, if $len(\mathcal{L}) == len(\mathcal{L}')$ there is no multitasking, because the execution times do not intersect for any work item of any resource in the event log $\mathcal{L}$ and $\forall wi_i \in \mathcal{L} \mid wi_i = \{wiaux_i\}$.

In other words, the coalesced event log has the same total workload per resource (hence same resource utilization) as the original log, while not containing any multitasking. Accordingly, the coalesced log can be used to discover a simulation model that can be executed using existing business process simulators (without multitasking support), while ensuring that the resulting resource utilization is the same as that of the original event log.

5. Handling resource availability constraints

In order to discover resource availability timetables, we use as a starting point the approach for discovering calendar expressions from collections of time points proposed by Yingjiu et al. [20]. They explain that event repeat over time according to a temporal pattern, such regularity could be capture by a time granularity or multiple granularities. For example, an organization performs maintenance operations for its services every 15th day of the month; in this case, *month* and *day* are time granularities.

Definition 9 (*Time Granularity*)**.** A granularity expression or time granularity is a countable set of time granules that do not overlap, where each granule is a subset of a time domain. Granularity expression takes the form of $(gn, gn-1, gn-2, \dots, g2, g1)$ where each field is a granule subset of a time domain.

Table 2
Set of calendar expressions — example.

1	$\langle year: 2012, week: 1, day\_of\_week: 3, hour: 3\rangle$
2	$\langle year: 2012, week: 1, day\_of\_week: 4, hour: 3\rangle$
3	$\langle year: 2012, week: 1, day\_of\_week: 4, hour: 7\rangle$
4	$\langle year: 2012, week: 1, day\_of\_week: 6, hour: 6\rangle$
5	$\langle year: 2012, week: 2, day\_of\_week: 1, hour: 6\rangle$
6	$\langle year: 2012, week: 2, day\_of\_week: 1, hour: 14\rangle$
7	$\langle year: 2012, week: 2, day\_of\_week: 2, hour: 3\rangle$
8	$\langle year: 2012, week: 3, day\_of\_week: 3, hour: 7\rangle$
9	$\langle year: 2012, week: 3, day\_of\_week: 4, hour: 7\rangle$
10	$\langle year: 2012, week: 3, day\_of\_week: 5, hour: 13\rangle$
11	$\langle year: 2012, week: 4, day\_of\_week: 2, hour: 2\rangle$
12	$\langle year: 2012, week: 4, day\_of\_week: 3, hour: 3\rangle$
13	$\langle year: 2012, week: 5, day\_of\_week: 5, hour: 13\rangle$
14	$\langle year: 2012, week: 6, day\_of\_week: 1, hour: 13\rangle$
15	$\langle year: 2012, week: 6, day\_of\_week: 6, hour: 2\rangle$
16	$\langle year: 2012, week: 6, day\_of\_week: 6, hour: 3\rangle$

Table 3
Calendar expressions supporting SP: $\langle year: 2012, week: 3, day\_of\_week: 4, hour: 7\rangle$.

1	$\langle year: 2012, week: 3, day\_of\_week: 4, hour: 7\rangle$
2	$\langle year: 2012, week: 3, day\_of\_week: 4, hour: 7\rangle$

Each granule of a valid *granularity expression* in gn has to be a union of some granules in $gn-1$. That is, a granule $g2$ has to cover $g1$. For example, the expression $(day, hour)$ is accepted since a day consists of a set of identified hours. Other valid examples are: $(year, month, day\_of\_month)$, $(year, week, day\_of\_week)$, $(day, hour, quarter\_of\_hour)$, etc. However, $(week, hours)$ is not acceptable since a week does not contain a set of unique hours.

A *calendar schema* is a schema which has fields f_i and corresponding domains D_i, where $(1 \leq i \leq n)$, $R = (f_n : D_n, f_{n-1} : Dn-1, \dots, f_i : D_1)$. Each field represents a *time granularity* and each domain can be an integer or a wild card symbol (*). The domain D_i may be omitted in a calendar schema. A *time point* (also called a *time granule* or a *granule* for short) is an instance of a calendar schema, (e.g. week 24, day 3, hour 5 is a time point). A *calendar expression* is a set of time granules, represented in the same way as a single time granule, but using the wildcard symbol '*' to indicate that for a given granularity, the calendar expression matches all possible values of that granularity (relative to the parent granularity). For example, $\langle day: *, hour: 10\rangle$ is a calendar expression, which can be interpreted as "everyday at 10 hours". This notion of calendar expression is captured by the following definition.

Definition 10 (*Calendar Expression*)**.** Let D_i be the set of possible granules of a granularity i $(1 \leq i \leq n)$. Moreover, let $\hat{D}_i = D_i \cup \{*\}$ be the domain consisting of all time granules of granularity i, augmented with the wildcard symbol. A *calendar expression* is a tuple $\langle d_1, \dots, d_n\rangle$, s.t. $d_i \in \hat{D}_i$.

Definition 11 (*Simple Calendar-based Pattern*)**.** A *simple calendar-based pattern (SP) is a pair* $\langle e, \Gamma\rangle$, where e is a calendar expression and $\Gamma=(\gamma_n, \gamma_{n-1}, \dots, \gamma_1)$ is a constraint, and each γ_i is a set of granules.

For example, $\langle 2021, *, 1\rangle$ is as SP that follows the granularity $\langle year, month, day\rangle$. Intuitively, it means the first day of 'every' month of year 2021. Based on a set of calendar expressions provided as input, the approach proposed in [20] aims to find all simple calendar-based patterns that follow a given granularity expression and satisfy a required predefined constraints. The constraints referred to are threshold values recognized as *support* and *confidence* rates as stated in [20].

Yingjiu et al. [20] employ techniques similar to association rule mining (a priori algorithm) in order to generate calendar expressions from a set of granules. Like other association rule mining methods, the interestingness of a calendar expression is measured with respect to a user-specified minimum support and confidence levels. Given an observed collection of time granules *TS*, the *support* of a calendar expression is the percentage of granules in *TS* that fall inside the set of granules defined by a calendar expression. Note that the support is 100% if all granules in *TS* are covered by the calendar expression. On the other hand, *confidence* is the percentage of time granules of the calendar expression that can be found in *TS*. Confidence is 100% when every granule matched by the calendar expression is observed in *TS*.

Table 2 has a set of calendar expressions following the granularity expression $\langle year, week, day\_of\_week, hour\rangle$. For example, the SP $\langle year: 2012, week: 3, day\_of\_week: 4, hour: 7\rangle$ is supported by the calendar expressions in Table 3. The support rate for the respected pattern is therefore $2/16 = 0.125$. For an SP with a wild card '*' such as $\langle year: 2012, week: 1, day\_of\_week: *, hour: *\rangle$, the supporting expression are demonstrated in Table 4. Support value for the respected pattern equals $4/16 = 0.25$

Over a time period, the confidence of a simple calendar-based pattern is the percentage of basic time units (a simple calendar-based pattern that does not contain any wild cards), among all the basic time unit points given by the simple calendar-based pattern, that contain the given events.

Table 4
Calendar expressions supporting SP: $\langle year : 2012, week : 1, day\_of\_week :*, hour :*\rangle$.

1	$\langle year : 2012, week : 1, day\_of\_week : 3, hour : 3\rangle$
2	$\langle year : 2012, week : 1, day\_of\_week : 4, hour : 3\rangle$
3	$\langle year : 2012, week : 1, day\_of\_week : 4, hour : 7\rangle$
4	$\langle year : 2012, week : 1, day\_of\_week : 6, hour : 6\rangle$

Table 5
Basic time units unfolded by SP: $\langle year : 2012, week :*, day\_of\_week : 4, hour : 7\rangle$ for the period between week 1 and 6.

1	$\langle year : 2012, week : 1, day\_of\_week : 4, hour : 7\rangle$
2	$\langle year : 2012, week : 2, day\_of\_week : 4, hour : 7\rangle$
3	$\langle year : 2012, week : 3, day\_of\_week : 4, hour : 7\rangle$
4	$\langle year : 2012, week : 4, day\_of\_week : 4, hour : 7\rangle$
5	$\langle year : 2012, week : 5, day\_of\_week : 4, hour : 7\rangle$
6	$\langle year : 2012, week : 6, day\_of\_week : 4, hour : 7\rangle$

Table 6
Calendar expressions contain basic time units by SP: $\langle year : 2012, week :*, day\_of\_week : 4, hour : 7\rangle$.

1	$\langle year : 2012, week : 1, day\_of\_week : 4, hour : 7\rangle$
2	$\langle year : 2012, week : 3, day\_of\_week : 4, hour : 7\rangle$

For example, given the SP: $\langle year : 2012, week :*, day\_of\_week : 4, hour : 7\rangle$ and the dataset of calendar expressions in Table 2, it is observed that the period of time the events fall into is between the first week and the sixth week. Therefore, the basic time units yielded by the SP for the respected period are the ones displayed in Table 5. Additionally, Table 6 indicating the basic time units existing in the given set of calendar expressions. As a result, the confidence of the respected pattern is $6/2 = 3$.

To use the technique from [20], we need to map the event log to a set of granules (*TS*) capturing the points in time where a given resource pool is "active". Here, we have the choice between using only the start timestamp of each work item (herein denoted $att_{st}(e)$), only the end timestamp ($att_{st}(e)$) or both timestamps. In the rest of this article, we use both timestamps because both the start and the end of a work item are time points during which a resource is known to have been actively working.

We choose to discover calendar expressions at the level of quarter-hours (15-minutes duration per granule) in order to capture timetables such as Monday and Tuesday 10:15–12:00 and 14:45–15:45. Accordingly, we preprocess the event log so that all start and end timestamps are rounded to the nearest quarter-hour, for example 2020-10-05T10:09:54 is rounded to 2020-10-05T10:00, whereas 2020-10-05T10:19:54 is rounded to 2020-10-05T10:15.

Note that the use of quarter-hour as the lowest granularity is arbitrary. We could apply the same technique to discover calendar expressions with lower granularity (e.g. at the level of minutes as in "every Monday 14:31–15:58 and 16:03–17:44"), but arguably this would lead to less readable timetables (and the execution times would be higher due to a higher set of possible time granules). Similarly, in the empirical evaluation reported later in this article, we focus on discovering weekly calendars (i.e. the coarsest granularity we consider is the week), but the approach can discover calendar expressions involving coarser granularities (e.g. month).

Given the set of time granules corresponding to the start and end timestamps of work items in a log, we build a variant of the data structure *calendar structure* (CS) defined in [20] and exemplified in Fig. 3. This data structure captures, for each resource pool (role) represented in the event log, all possible calendar expressions over a given set of granularities, in this example the three granularities DayOfWeek, HourOfDay, and QuarterOfHour.

Yingjiu et al. [20] describe four algorithms to derive a calendar structure (CS) capturing the set of calendar expressions that meet a given (user-defined) support and confidence level for a given set of time points. Algorithm 2 describes a variant of the *Enumeration Algorithm with Intra-level Pruning* (EAP) presented in [20], which is the algorithm that performs best in their experiments. We adapted this algorithm to include information about the resource pools (roles) recorded in the event log. The resource pool associated to an event is denoted as $att_r(e)$.

The algorithm is decomposed into two functions, namely $discover$ (the main function) and $addToCS$ (a recursive function that adds information incrementally to build the data structure CS). To illustrate the algorithm, let us assume we use the granules [DayOfWeek, Hour, QuarterHour] and that we want to discover a calendar expression from an event log that contains a trace $\sigma_1 = \langle e_1, e_2, \ldots\rangle$. Moreover, let us assume that the resource associated with e_1 is r_1 (i.e. $att_r(e_1) = r_1$, and that the start and end timestamps associated with e_1 are Monday 2020-10-05T10:00:00 and Monday 2020-10-05T10:22:40, respectively. The algorithm starts by creating an empty CS data structure in line 2, and then it iterates over the set of events in the log. Fig. 4(a) corresponds to the creation of CS root node. In lines 4–5, the algorithm determines the resource associated with the event and tries to access the corresponding child node in CS. Since CS is initially empty, a new CSNode is created and added to the root of the tree-like CS data structure (see Fig. 4(b)). Then, in line 9, the algorithm starts an iteration over the set of start and end timestamps associated with e_1 and it expands the tree by calling $addToCS$ in line 10. Note that $addToCS$ is called with the reference to the node pointed with the resource r_1 as the parent node, with the start timestamp and granule DayOfWeek. From that, one can see that line 23

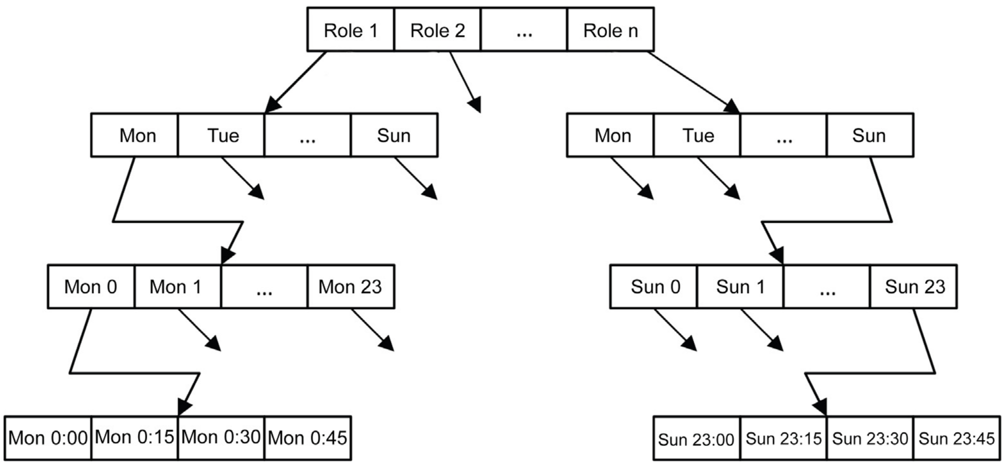


Fig. 3. Data structure *calendar structure*.
Source: (based on [20]).

Algorithm 2: Discovery of calendar expressions (adapted from the EAP algorithm in [20])

```
1  Function discover($\mathcal{L}$: EventLog, sup: double, conf: double, gs: [Granule]) is
       // Construct the CS data structure
2      CS = new CSNode()
3      for event $e$ in event log $\mathcal{L}$ do
4          role = $att_r(e)$
5          child = CS[role]
6          if child != null then
7              child = new CSNode()
8              CS[role] = child
9          child.count += 1
10         for timestamp in $\{att_{st}(e), att_{et}(e)\}$ do
11             addToCS(child, timestamp, gs[0], gs)
       // Initialize the sets of calendar expressions $P_1$ and $P_2$
12     $CE_{disc} = \emptyset$ // Calendar expressions having failed to satisfy support requirement
13     $CE_{kept} = \emptyset$ // Calendar expressions satisfying support and confidence requirements
14     for calendar expression $ce$ generated from role/timestamps represented in CS do
15         if exists $ce'$ in $CE_{disc}$ s.t. $ce'$ covers $ce$ then
16             discard $ce$
17         else
18             if support($ce$) $\geq$ sup and confidence($ce$) $\geq$ conf then
19                 $CE_{kept} = CE_{kept} \cup \{ce\}$
20             else if support($ce$) < sup then
21                 $CE_{disc} = CE_{disc} \cup \{ce\}$
22     return $CE_{kept}$
23 Function addToCS(parent: CSNode, t: Timestamp, g: Granule, gs: [Granule]) is
24     granuleValue = getGranuleValue(t, g)
25     child = parent[granuleValue]
26     if child is null then
27         child = new CSNode()
28         parent[granuleValue] = child
29     child.count += 1
30     next = nextGranule(gs, g)
31     if next != null then
32         addToCS(child, t, next, gs);
```

will set $granuleValue$ to Monday. Since CS is virtually empty, the algorithm will add a new CSNode for Monday in lines 24–27 (see Fig. 4(c)). A counter is kept for every node in CS to enable the computation of the support and confidence values. In line 29, we determine what is the next (finer) granule to analyze which, in this case would be Hour and which leads to a recursive call to $addToCS$. Fig. 4(d) shows CS after inserting the two timestamps associated with e_1. Note that we have kept track on the number of timestamps that can be associated with each calendar expression represented in CS.

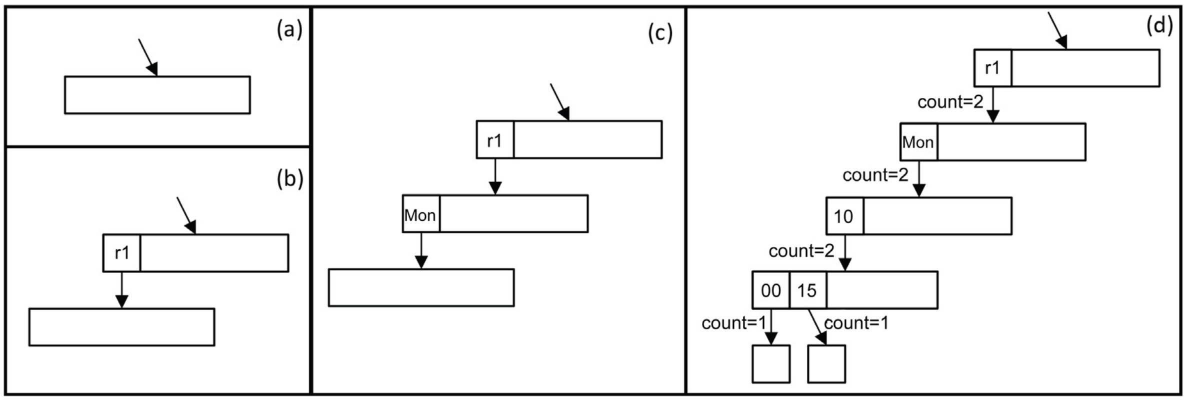


Fig. 4. Example of a CS structure from an empty root node (a) until the insertion of two timestamps (d).

The second part of the algorithm, lines 12 to 22, enumerates the list of candidate calendar expressions and keeps only those that fulfill the support and confidence requirements. Conceptually, for each resource/timestamp pair, e.g. r_1 and Monday 2020-10-05T10:00:00, the algorithm should check the following calendar expressions: $\langle r_1, *, *, * \rangle$, $\langle r_1, \text{Mon}, *, * \rangle$, $\langle r_1, *, 10, * \rangle$, $\langle r_1, *, *, 0 \rangle$, $\langle r_1, \text{Mon}, 10, * \rangle$, $\langle r_1, \text{Mon}, *, 0 \rangle$, and $\langle r_1, \text{Mon}, 10, 0 \rangle$. Hence, the number of patterns to verify might be very large. However, it can be noted that pattern $\langle r_1, \text{Mon}, 10, * \rangle$ "covers" pattern $\langle r_1, \text{Mon}, 10, 0 \rangle$. Moreover, if $\langle r_1, \text{Mon}, 10, * \rangle$ does not meet the support and/or confidence requirements, we can avoid checking $\langle r_1, \text{Mon}, 10, 0 \rangle$. Based on this observation, Algorithm 2 reduces the number of candidate calendar expressions to be checked (cf. lines 15 and 16). The for-loop that enumerates the candidate calendar expressions is implemented via a depth-first-search traversal of CS.

The above algorithm discovers calendar expressions capturing the availability of resources (i.e. a resource availability timetable). We use the same approach to discover a case creation timetable from an event log. Concretely, we first extract the first event associated of each case/trace and we retain the start timestamp of this work item. We then discover a set of calendar expressions from the collection of all such "start timestamps".[3]

6. Evaluation

In this section, we describe the experiments carried out to evaluate the proposal presented for the discovery of business process simulation models, taking into account the effects of multitasking and the availability of resources. The evaluation is based on discovering different business process simulation models taking into account the behavior of the resources, and then calculating a set of measures of goodness that will allow the comparison of the discovered models. The discovery of the business process models was done by means of the SIMOD tool [10]. This tool uses the hyper-parameter optimization technique "to search in the space of possible configurations in order to maximize the similarity between the behavior of the simulation model and the behavior observed in the log". Process models are discovered using the Split Miner algorithm [22], which considers different levels of sensibility and depends on two parameters: the parallelism threshold, epsilon(ϵ) that determines the quantity of concurrent relations between work items to be captured; and the percentile for frequency threshold, eta (η), that acts as a filter over the incoming and outgoing edges of each node and retains only the most frequent percentiles. Both parameters are defined in a range between 0 and 1. The resulting simulation models can be executed using simulators such as Scylla [23] and BIMP [24]. As in [6], we use BIMP because it allows a wider set of distribution probabilities to be used, thus widening the space for configuration options.

6.1. Datasets

To evaluate our proposal, we need event logs with the following Data Requirements (DRs):

DR1 The log should contain records of executions of task instances (work items) with their start and end timestamps. This requirement is necessary in order to calculate the processing time of each work item, which is a pre-requisite for discovering a simulation model.

DR2 The log should have resource identifiers. This requirement is also a pre-requisite for discovering resource pools, which are a required component of a business process simulation model.

[3] Martin et al. [21] observed that the case creation time may occur earlier than the start timestamp of the first work item. They proposed a method to estimate the case creation time under some assumptions. We acknowledge that this method could be used to pre-process the log in order to obtain more accurate estimates of case creation times.

We undertook a search of event logs fulfilling such characteristics. To this end, we scanned the 4TU data centrum collection of real-life event logs.[4] We also considered the event logs previously used to evaluate the Simod method for automated discovery of event logs. Out of this pool of event logs, we found that the following ones fulfill the requirements DR1 and DR2:

- *ACR:* This is a real event log that represents an academic credentials recognition (ACR) process in a university during the first semester of 2016. This log has 954 traces, 18 tasks, 6870 events, and involves 561 resources. After analyzing a set of event logs, we identified that this is the only event log reflecting multitasking.
- *Purchasing Example:* This is a synthetic event logs generated from a purchase-to-pay process model not available to the authors.[5] This event log has 608 traces, 21 tasks, 9119 events, and involves 27 resources. Although this event log does not contain multitasking, it was used as a starting point (PE_0P_log) to create a set of logs in which, given a numerical value $x = \{5, 10, 15, 20, 25\}$, a log with x percentage of multitasking was generated. The detail of how these logs were generated will be described in Section 6.3.
- *MP*: This is a real event log that belongs to a manufacturing production process, exported from an Enterprise Resource Planning system [25]. The tasks in this process refer to steps (or "stations';) in the manufacturing process. This event log has 225 traces, 26 tasks, 4953 events, and involves 48 resources. This event log does not contain multitasking.
- BPI Challenge 2012W: This is a subset of a real event log belonging to a Dutch financial institution's loan application process.[6] The W subset is composed of the events performed by human resources (i.e., only activities that have a duration). This event log has 8616 traces, 6 tasks, 59302 events, and involves 58 resources. This event log does not contain multitasking.
- BPI Challenge 2017 W (filtered): This event log is an updated version of the BPIC2012 log.[7] We carried out the W-subset extraction by following the recommendations reported by the winning teams participating in the BPIC 2017 challenge.[8] Due to the large size of this log and the required time to process it with the SIMOD tool, we decided to reduce its size. The reduced version of the log includes the events that occurred between (06-25-2016 and 10-07-2016) and has 8941 traces, 7 tasks, 63764 events, and involves 113 resources. This event log does not contain multitasking.

In order to evaluate the approach for handling multitasking, we needed event logs that additionally fulfill the following requirement:

DR3 The event log should contain work items with overlapping execution periods, performed by the same resource.

Among the event logs listed above, we found that the Production Manufacturing, the BPI Challenge 2012w and BPI Challenge 2017w do not contain any multitasking (except for a negligible number of instances of multitasking in he BPI Challenge 2017w). The only log fulfilling DR3 was the ACR log, which contains a moderate amount of multitasking. In order to test the performance of our proposal on event logs with different levels of multitasking (from low to high), we took the Purchasing Example log listed above, and we synthetically altered it in order to generate event logs with different levels of multitasking.

6.2. Measures of goodness

To evaluate the goodness of an automatically discovered simulation model, we perform a simulation of this model to produce a simulated event log with the same number of traces as the original event log. We then compare the simulated log to the original log in order to determine their similarity, both in terms of the sequences of activities they contain, and their associated timestamps and case durations (cycle times). Following the approach taken to evaluate the Simod method, we use the following measures of similarity/distance between the simulated log and the original log:

- *Mean Absolute Error (MAE):* This metric assesses the distance in seconds between the cycle times of two logs' traces. We calculated the MAE by first measuring the cycle times absolute error between all the traces on both event logs, then pairing them using the Hungarian algorithm, and finally computing the mean distance of the pairs. It is important to highlight that this measure is relative acting at the trace level; therefore, it must be complemented by employing other measures that provide a clearer idea of the event logs' global and event-level temporal dynamics.
- *Log Mean Absolute Error (LMAE):* This metric evaluates the distance between two event logs' global times with the same number of traces. LMAE provides a global perspective of the times in the event logs that can be affected by factors such as the rate of instances generation and restrictions over the resources' availability. We calculated this measure as the absolute error between the timeframes of the two event logs. Each log timeframe corresponds to the time elapsed between the first event start timestamp and the last event complete timestamp expressed in seconds. We compared the ground-truth log against the simulated logs generated in each experiment repetition and calculated the mean of the distances guaranteeing the convergence of the results.
- *Earth Mover's Distance (EMD):* We used the EMD to evaluate the temporal dynamics of the logs at the event level. This metric is calculated by comparing the normalized histograms of the events in different time windows in the ground-truth log vs. the same histogram computed from the simulated logs. We chose the time windows according to the most frequent circadian

[4] https://data.4tu.nl/

[5] The log is provided as part of the Fluxicon Disco tool – https://fluxicon.com/

[6] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

[7] https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

[8] https://www.win.tue.nl/bpi/doku.php?id=2017:challenge

cycles of resources. In particular, we grouped the timestamps by weekday/hour and by calendar day. The EMD between two histograms is the minimum number of mass units needed to transform one histogram into another. The EMD is zero if the observed times in the two logs are identical, and it tends to one the more they differ.

- *Timed String Distance (TSD):* This metric is a modification of the distance measure called Damerau–Levenshtein (DL) that assesses the distance between two strings and has been extensively used to assess the similarity between two process traces. In addition to the sequences distance, TSD includes a penalty related to the time difference in processing and waiting times, providing a single measure of the traces' accuracy [6].
- *DL-Mean Absolute Error (DL-MAE):* The DL-MAE is a complementary measure to assesses the similarity between two traces, assigning 50% of importance to the distance between the event sequences w.r.t DL and 50% importance to the temporal distance to the cycle time of traces normalized w.r.t MAE.

6.3. Evaluation of multitasking approach

The pre-processing of an event log for the identification of multitasking *work items*, the overlapping time periods, the adjustment of the execution times for these *work items* and the calculation of multitasking indexes is done by means of a Python script called *Sweeper*. This script receives as input a base event log ($\mathcal{L}$) and generates as output a coalesced event log (cf. Definition 8).

As a first step to conduct the multitasking evaluation, we performed a preliminary analysis of all of our available logs to identify those that reflected multitasking as a result of their original event logs. Of those logs, only the *ACR log* reflected overlap between some of the event logs. For this reason, and in order to be able to perform further analysis regarding the characteristics of multitasking, we decided that the evaluation would be two-pronged. The first, based on a real event log in which multitasking is originally present (*ACR log*), and the second, in which work item overlap is introduced into the log. For this second phase we start from the *Purchasing Example log*. The event logs[9] and scripts[10] are available online. The experiments were carried out on a computer using Windows 10 Enterprise (64-bit), a processor Inter Core i5-6200U, CPU 2.3 GHz, and 16.0 GB RAM.

6.3.1. Multitasking evaluation using a real-life event log

This section aims to identify the accuracy gains of the proposal using a real-life event log. The hypothesis in this scenario is that adjusting execution times derived from multitasking leads to more accurate simulation results, reduces the total execution time of tasks and processes; avoids an over-estimation of resource utilization by a simulator that executes work items without multitasking; and maintains the correct alignment of the model generated according to the original model derived from the log.

Experimental setup. Based on the *ACR log*, the first step of the evaluation was to generate a new event log, namely the *ACR adjusted log*, which contains the adjusted times for the ACR log (i.e. the coalesced version of the ACR log as per Definition 8). Then, we discovered a BPS model from the original log and from the coalesced log using SIMOD. In the hyper-parameter phase of SIMOD, 50 BPS models were generated using different parameter configurations. Parameters ϵ and η were varied from 0.0 to 1.0. Each discovered BPS model was simulated 5 times. Finally, we used the Apromore process mining tool[11] to compare the processing times of the logs and BIMP[12] to analyze the resource utilization values.

Results. Executing SIMOD using the *ACR log* gave very similar results were obtained to those presented in [6], using half number of simulations. In that proposal, the similarity measure TSD is equal to *0.9167*, while in our experiment, TSD is equal to *0.906* with ϵ=0.615 and η=0.559. Executing SIMOD on the *ACR adjusted log* led to a similarity measure of *0.929* with ϵ=0.484 and η=0.591. When comparing the average cycle times of the BPS models generated by the original and the coalesced logs, we noticed a gain in accuracy of approximately 14% for the BPS model discovered from the *ACR adjusted log* relative to the BPS model discovered from the *ACR log*.

The above relatively small accuracy gain (particularly with respect to the TSD measure) should be cast in light of the characteristics of the ACR log. Table 7 shows the characteristics of the original and the adjusted (coalesced) ACR log. Out of the 18 tasks in the event log, 17 have at least one overlapping pair of instances (work items). From the 6870 events, 1267 are overlapping with at least one other work item. Out of the 561 resources involved in the log, 76 executed at least one pair of overlapping work items. Finally, after grouping all work items according to the resource that executed them, 1116776 pairs of work items were identified (total number of pairs of work items performed by the same resource). Among them, 1036 overlap in some period of their execution time. This last observation highlights that the log used for this analysis has a low volume of multitasked work items, which in turn explains the relatively low accuracy gain (in term of TSD).

We now look at the results from the perspective of the Multitasking Log Index (*MTLI*). For the *ACR log*, $MTLI = 1.05\%$. If *MTLI* is low, one would expect the rate of improvement in the analysis to be low as well, but as the level of multitasking in the log increases, the measure should improve proportionally. This indicates that when analyzing all possible pairs of work items, only the 1.05% of the time of those work items were overlapped, which represents low level of multitasking. For the same log, the *Multitasking Work Item Index* is $MTWII = 58.54\%$, which indicates that, for those work items where multitasking has been identified, the pairs of work items that overlap with each other do so for 58.54% of their total duration. Both indexes are described in detail in next section.

9 https://github.com/AdaptiveBProcess/Simod/tree/master/inputs/multitasking_logs
10 https://github.com/AdaptiveBProcess/Simod/tree/master/support_modules/multitasking
11 http://apromore.org
12 http://bimp.cs.ut.ee/

Table 7
Differences between original log and adjusted (coalesced) ACR log.

	Tasks	Events	Resources	Pairs
Original log characteristics	18	6870	561	1116776
Coalesced log characteristics	17	1267	76	1039

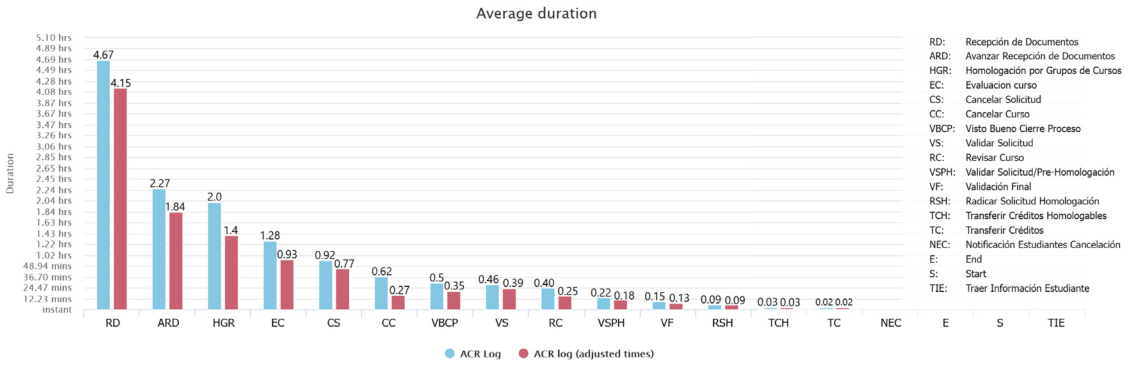


Fig. 5. Comparison of average duration between *ACR log* and *ACR adjusted log*.

Using Apromore, we analyzed the *ACR log* and the *ACR adjusted log* and compared them in terms of their temporal characteristics. Fig. 5 shows the average duration of tasks. Blue bars represent the average time duration of the tasks of the *ACR log* while the red bars represent the average time duration of tasks in the *ACR adjusted log*. Both *RD* and *HGR* reflect a difference of 0.6 h (h), followed by *HGR* with 0.43 h; *EC* and *CC* with 0.35h; *CS*, *VBPC* and *RC* with 0.15 h; *VS* 0.07; *VSPH* 0.04; *VF* with 0.02 h; *RSH* does not show improvement; and the last 6 task do not reflect improvement either, but can be considered activities of instant duration. Finally, we calculated the resource utilization of the three resource pools using BIMP. We noticed only slight differences in the resource utilization.

In general, the above results show that with the pre-processing of log it is possible to maintain and/or improve similarity between traces involved in each log. Besides, we figured that the level of improvement in the results depends on the log's multitasking level: the number of overlapping work item pairs and the percentage of overlap between each overlapping pair of work items. As we have only been able to identify and use one real-life log with multitasking characteristics to show dependence between the amount of multitasking and the result improvements, in the following subsection, a synthetic log was modified to generate a log set with different multitasking levels.

6.3.2. Multitasking evaluation using a set of synthetic event logs

While the experiment with a real-life log reported above shows that the proposed approach can enhance the accuracy of automatically discovered simulation model, it does not allow us to assess the potential benefits of the proposed approach for different types of event logs. Accordingly, we complement the above experiment with a second experiment aimed at assessing the accuracy enhancements produced by our proposal on event logs with varying levels of multitasking. Our hypothesis is that the accuracy enhancement will be higher for event logs with higher levels of multitasking.

Experimental setup. To validate the above hypothesis, we tested our approach on a collection of synthetic event logs with varying levels of multitasking. To this end, we took as starting point the *Purchasing Example* log discussed in Section 6.1. This log, herein referred to as *PE_0P_log*, does not contain any multitasking. To produce logs with varying level of multitasking, we perturbed this event log by altering the end times or start times of *work items* in the log assigned to the same resource in such a way as to create some overlap between them. This perturbation is implemented by a script (namely *percentage*) which, given a percentage of shifting (between 0.0 and 1.0), generates a new event log, including work items that overlap for that percentage of their processing times. The script proceeds as follows.

- The base event log is divided by grouping the work items that are executed by a particular resource (see Definition 5, herein called a *resource segment*).
- The work items of each *resource segment* were ordered according to their start timestamps.
- For each *segment per resource*, the first work item was taken as the pivot, and the next adjacent work item was searched among the remaining work items. Two work items (e_1, e_2) are adjacent if the end timestamp of e_1 has the same value as the start timestamp of e_2. In Fig. 6.a, the first pair of work items shown are adjacent.
- When a pair of adjacent work items were identified, the timestamps were shifted depending on the percentage assigned. In Fig. 6.b, 20% of shifting is applied, while in Fig. 6.c, the shifting is 40%.

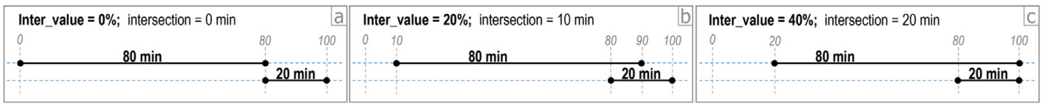

Inter_value = 0%; intersection = 0 min
a
0
80
100
80 min
20 min
Inter_value = 20%; intersection = 10 min
b
0
10
80
90
100
80 min
20 min
Inter_value = 40%; intersection = 20 min
c
0
20
80
100
80 min
20 min

Fig. 6. Overlapping of work items according to a percentage of shifting.

- The two work items of a pair of adjacent work items are excluded from the following search. The next work item in the *segment per resource* is taken as a pivot, and the search is repeated. If no adjacent work item is found, it is not modified, and the search is repeated with a new pivot.
- The search of adjacent work items was repeated for all *segment per resources*.

We can see that by varying the percentage of shifting, we obtain event logs with varying levels of multitasking.

We recall that the hypothesis we intend to test is that the proposed approach leads to higher increases in accuracy for event logs with higher levels of multitasking. To test this hypothesis, we need a way of measuring the level of multitasking in an event log. Intuitively, the level of multitasking in a log is determined by the amount of overlap between the execution times of pairs of work items assigned to the same resource, relative to the total number of pairs of work items that can be formed between the work items of each resource (sr_i). Accordingly, to determine the level of multitasking in a log $\mathcal{L}$, we propose a measure called ***Multitasking Log Index (MTLI)***. This index can take a value between 0 and 1, where 0 indicates that no overlap was identified and 1 would reflect a total overlap between pairs of work items, that is, if all work items were executed in parallel, with the same start and end point, for a resource. From this we derive the intuitive properties of minima and maxima, where the higher the value, the more overlap between pairs of work items was identified in the log.

Concretely, to calculate the *MTLI* of a log $\mathcal{L}$, we based on the idea that a log is divided by grouping all the work items of a given resource (r), generating $sr_i \in \mathcal{S}$ (See Definition 5). The multitasking of a log is derived from the overlap between the execution times of two work items executed by the same resource. Therefore, for each sr_i, let WI_{sr} be the set of all possible work items in sr_i and $SRWI_r$ be the set of all possible pairs of work items in sr_i.

$$SRWI_r = \{(wi_1, wi_2) \in WI_{sr} \times WI_{sr} \mid wi_1 \neq wi_2 \wedge wi_1.r = wi_2.r\}$$

For each pair of work items $(wi_1, wi_2)_i \in SRWI_r$, $1 < i <| SRWI_r |$, we measure the size of their intersection. We note that the intersection of two intervals can be calculated as the difference between the minimum of their end timestamps and the maximum of their start timestamps (or zero if this latter difference is negative, which implies that the two intervals do not intersect). To normalize this intersection, we divide it by the maximum of the durations of the two work items.

$$overlap(wi_1, wi_2)_i = \frac{max((min(wi_1.et, wi_2.et) - max(wi_1.st, wi_2.st)), 0)}{max((wi_1.et - wi_1.st), (wi_2.et - wi_2.st))}$$

Given the above, it becomes possible to calculate a *Multitasking Resource Index ($MTRI_r$)*, which is an index of multitasking per resource sr_i in an event log. For each sr_i, all overlap values are summed; and that sum is multiplied by the value of 1 divided number of pair of work items in $SRWI_r$.

$$MTRI_r = \frac{1}{| SRWI_r |} \sum_{(wi_1,wi_2)_i \in SRWI_r}^{|SRWI_r|} overlap(wi_1, wi_2)_i$$

Finally, *MTLI* is calculated as the average of all $MTRI_r$ in the log.

$$MTLI = \frac{\sum_{j=1}^{|\mathcal{S}|} MTRI_j}{| \mathcal{S} |}, \mathcal{S} = \{sr_1, \dots, sr_n\}$$

The *MTLI* index may be biased in situations where the number of non-overlapping pairs of work items in the log is very large. In this case, the denominator of the *MTLI* index will be very large. To mitigate this potential bias, we additionally define a second multitasking index, namely the ***Multitasking Work Items Index (MTWII)***. This latter index is calculated in a very similar way as *MTLI*, but it only considers overlapping pairs of work items, and it is therefore not affected when there is a large amount of non-overlapping work items. In other words, the *MTLI* index captures the global level of overlap in the log, while *MTWII* captures the extent of overlap among those work items that do overlap.

To define the *MTWII* index, we consider the set of all possible overlapping pairs of work items for a resource is defined as follows.

$$\begin{aligned} RWI_{o_r} = \{(wi_1, wi_2) \in WI_{sr} \times WI_{sr} \mid wi_1 \neq wi_2 \wedge wi_1.r = wi_2.r \\ \wedge (min(wi_1.et, wi_2.et) - max(wi_1.st, wi_2.st)) > 0\} \end{aligned}$$

The function $overlap(wi_1, wi_2)$ is calculated the same way. Now, the value of $MTRI_r$ is calculated only for those pairs of overlapping work items ($MTRI_{o_r}$).

$$MTRI_{o_r} = \frac{1}{| RWI_{o_r} |} \sum_{(wi_1,wi_2)_i \in RWI_{o_r}}^{|RWI_{o_r}|} overlap(wi_1, wi_2)_i$$

Table 8
Comparison between the synthetic logs created using a percentage of shifting.

Shifting (%)	MTWII (%)	MTLI	Overlapping pairs	DL_MAE	MAE (segs)
0	0	0	0	0.8883	1073208
5	5.596	1.468e−05	876	0.8889	1145181
10	10.381	2.754e−05	950	0.8893	1098788
15	14.694	3.953e−05	1006	0.8895	1091332
20	18.860	5.087e−05	1041	0.8841	1049593
25	22.266	6.147e−05	1073	0.8866	1117721

Finally, $MTWII$ is calculated as the average of all $MTRI_{o_r}$, where S_o represents the set of all resources that have at least on pair of work items with multitasking. $S_o = \{sr_{o_1}, \dots, sr_{o_j}\}$, where $1 < i < j$; $sr_{o_i} = \{wi_1, \dots, wi_k\} \mid \exists(wi_n, wi_m) \in MTRI_{o_r}$, where $1 < n, m < j; wi_n \neq wi_m$.

$$MTWII = \frac{\sum_{r=1}^{|S_o|} MTRI_{o_r}}{\mid S_o \mid}$$

Note that the two indexes defined above (*MTLI* and $MTWII$) are used for the sole purpose of characterizing the volume of multitasking in a log. In other words, we do not use these indexes as measures of goodness of the proposed approach (i.e. as dependent variables), but rather we use them as independent variables. The measures of goodness used for evaluation purposes are those discussed in Section 6.2.

Having generated a number of event logs with varying levels of multitasking, we discovered a BPS model from each log using SIMOD. To take advantage of SIMOD's hyperparameter optimization capabilities, 100 BPS models were generated using different parameter configurations. Parameters $epsilon$ and eta varied from 0.0 to 1.0. Each simulation model was executed 5 times, for that, 500 simulations were evaluated for each log in the set as part of the hyperparameter optimization phase of SIMOD. We used the BIMP simulator to simulate the resulting BPS models and to extract statistics from the resulting simulations, such as the resource utilization percentages.

Results. Based on the *PE_0P_log*, 5 more logs were built using *Percentage* script. The percentages of shifting used were 5%, 10%, 15%, 20% and 25%. With regard to resources, in *PE_0P_log* participate 27 resources, and 11 of them reflect multitasking. From the 9119 events in *PE_0P_log*, 2625561 pairs of work items were identified. From these, 789 are adjacent work items to be used for time-shifting of the logs, excluding in this set all instantaneous events.

Table 8 shows the percentage of shifting applied to the adjacent work items in each generated log; the multitasking indexes (*MTWII* and *MTLI*) for each log in the set; the number of pairs of work items in which overlapping was identified (multitasking); and the value of two similarity measures obtained using SIMOD.

The percentage reflected in the column *MTWII* should be the same as *Shifting*, because the number of adjacent work items on which the shifts were made was the same for all the logs. However, certain *MTWII* values are slightly higher since the shifting of some work items may generate overlapping between work items that initially were not adjacent. This is also reflected in the column *Overlapping Pairs*, where the number of pairs of work items overlapped is greater than 789 and increases as the shifting increases. Above a certain amount of shifting, the value of *MTWII* is less than the percentage of shifting. This is because a shift can cause one work item to be embedded within another (Fig. 6.c), and if the shifting increases, the work item is still embedded and does not provide more multitasking to the log. As could be deduced by identifying the number of work items in the log, the *MTLI* is quite low. However, like *MTWII*, it increases when the percentages of shifting increase. Similarly, since the amount of multitasking in the log is low, the difference between *DL_MAE* values varies and improves slightly for those cases where the percentage of shifting is quite similar to *MTWII* (0, 5, 10, 15) and worsen slightly for those cases where the shifting and index vary more. Finally, when calculating the MAE we see that although there is a significant difference between a log with and without multitasking, as the multitasking is increased, and the adjustment in the logs, the MAE is gradually reduced, which means that the discovered BPS models are more accurate. BPS models were simulated using BIMP. 5 resource pools were discovered, two of them with a high percentage of resource utilization (RU). The RU in BPS models derived from multitasking is reduced, especially for those resource pools where the RU is higher.

6.4. Evaluation of resource availability effects

This section aims to determine if the proposed calendar discovery technique improves the global representation of times in business process simulation. The hypothesis behind this experiment is that restricting the availability of resources and the generation of instances according to the schedules observed in the event log helps improve the simulation models' precision. The evaluation was two folded and performed using all the logs described in Section 6.1. The experiments were carried out on a computer using Windows 10 Enterprise (64-bit), a processor Intel Core i5-8250U, CPU 1.6 GHz, and 16.0 GB RAM.

Table 9
Accuracy evaluation results.

Event log	Scenario	TSD	LMAE	MAE	EMD (weekday)	EMD (hour)
ACR	Baseline	0.8142	3258874	338802	0.4001	0.6062
	GI	**0.8204**	6209179	369895	0.3714	0.2871
	LI	0.8163	**2602998**	**296616**	**0.2572**	**0.1956**
PurchasingExample	Baseline	0.7811	9156301	4993402	0.5953	**0.8678**
	GI	**0.8073**	8674894	5250932	**0.4572**	0.8725
	LI	0.7662	**4391892**	**4269479**	0.4714	0.8875
MP	Baseline	0.2249	951211	209036	0.6286	0.7829
	GI	**0.2538**	8004078	**208856**	0.4000	0.7957
	LI	0.1844	**156314**	223114	**0.2334**	**0.6642**
BPI2012W	Baseline	0.4637	2339552	**481631**	0.3667	0.4335
	GI	**0.5873**	**1978978**	494508	0.3143	0.2400
	LI	0.4485	2015657	492679	**0.2857**	**0.2000**
BPI2017W	Baseline	0.5707	1271119	750593	***0.2572***	0.4370
	GI	**0.6155**	1874672	867037	0.5953	0.3854
	LI	0.5615	**870689**	**591604**	0.2857	**0.3566**

Experimental setup. Given the large number of possible hyper-parameter configurations that can affect an automatically discovered simulation model's accuracy, we decided to split the hyper-parameter optimization into two phases. The first phase does not make use of resource availability calendars, while the second phase focuses on finding the best possible calendar to fit the log.

In the first phase, we used the SIMOD optimizer to discover a base scenario that uses a 7/24 calendar. In this phase, SIMOD discovered the optimal values of the discovery parameters required to obtain the simulation model with the best precision using TSD as a loss function. In this phase, we carried out 30 hyper-parameter optimization iterations. We executed five simulation runs for each event log and each hyper-parameter configuration, to ensure that the results were not affected by stochastic variations inherent to simulations.

Once the base simulation scenario was fixed, in the second phase we computed two alternative simulation scenarios: one using a Global Improvement (GI) approach and another using a Local Improvement (LI) approach. The GI approach used the SIMOD optimizer to discover all the discovery parameters, including the support and confidence parameters required for the calendar discovery. This scenario required more iterations by the optimizer (i.e., 50 per log) since the search space is larger than the base scenario. Again, we used the TSD metric as a loss function and performed five simulation runs again for each event log and configuration.

On the other hand, the LI approach uses the same values of the discovery hyper-parameters as the base scenario discovered in the first phase. It then uses SIMOD's hyper-parameter optimizer to optimize the support and confidence parameters of the calendar discovery algorithm. This approach uses as the EMD as a loss-function with a weekday/hour time window instead of the TSD. The rationale for using EMD in this context is that EMD captures the TSD metric relies on relative timestamps (w.r.t. the creation of each case) whereas EMD uses absolute times. Given that the discovery of calendars relies on absolute timestamps, we hypothesize that this approach gives more accurate approach than the GI approach where TSD is used as the loss function. In other words, while the TSD is suitable for optimizing the hyper-parameters that determine the distribution of task durations (processing times), we hypothesize that the EMD metric is more suitable as a loss function when optimizing the discovery of the resource availability and case creation calendars. As for GI, we carried out 30 hyper-parameter optimization iterations for each event log, and five simulation runs for each event log and each configuration.

Once we found the best simulation model for each approach on the training fold, we evaluated their accuracy on the testing fold using the TSD, MAE, LMAE, and EMD measures defined in Section 6.1. To smooth out stochastic variations, we report the mean of each of these measures across five simulation runs.

Results. Fig. 7 and Table 9 present the results of the three scenarios per each event log. As can be seen, overall GI and LI's improvement scenarios outperform the baseline in most cases. Regarding the accuracy in terms of TSD (see Fig. 7 TSD), the GI approach obtained the best results in all the event logs, demonstrating that automatic calendar discovery, in conjunction with other process mining techniques, improves the accuracy of the resulting simulation models at the trace level.

As hypothesized above, the LI approach shows a significant improvement relative the base and the GI approach. The MAE and LMAE distances show that the absolute error between the ground-truth and the simulated logs significantly decreases in the LI approach. Specifically, this improvement is evidenced in three of the five logs, in which the distance in the logs timeframes w.r.t., LMAE, and the cycle times of traces w.r.t. MAE is clear. In the other two event logs, i.e., MP and BPI2012 W, the improvement is not always clear; however, it can be explainable according to the logs nature. In the MP log case, the LI approach dramatically decreases the LMAE distance and remains close to MAE distance. This result indicates that the calendar discovery technique improves the representation of instances generation in the simulation. It also indicates that resource contention (the access conflicts to resources shared by different instances of the process) is not a determining factor since limiting the resources' availability does not affect the instances cycle times. On the other hand, in log BPI2012 W, the local improvement scenario obtained LMAE and MAE results consistent with the other two scenarios, although slightly higher, but there is no clear trend. This behavior may be due to this

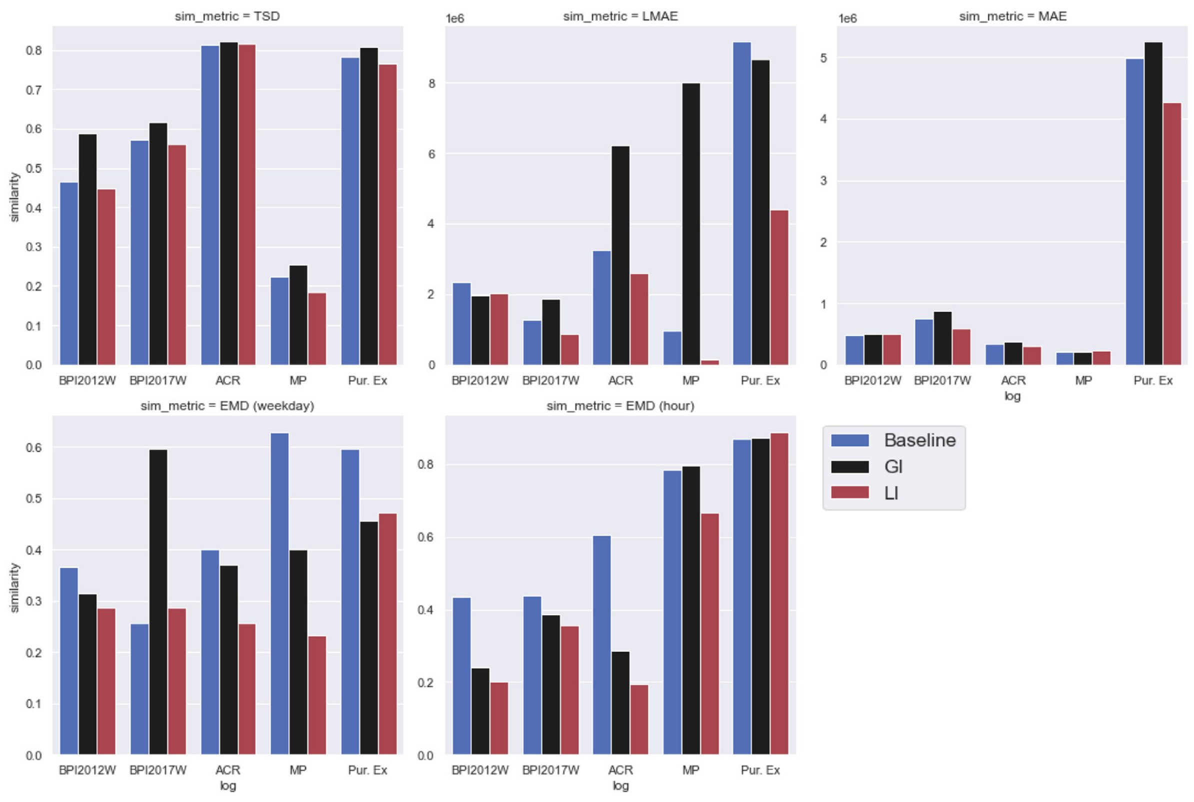


Fig. 7. Accuracy results. MAE and LMAE in seconds.

process does not present substantial restrictions on the availability of resources or the generation of instances, and other unknown factors affect its performance.

Moreover, the metrics for representing times at the event level, i.e., EMD by day of the week and hour, show the same improvement trend in the LI approach. This trend is especially significant for log ACR and log MP. These results are because these logs present circadian behaviors that are adequately captured by the calendar discovery technique.

7. Threats to validity

The reported evaluation has a number of threats to validity. First, a potential threat to internal validity, which may affect the conclusions drawn from the evaluation of execution times, is the fact that we conducted experiments using a single computing environment. The results might differ on other computing environments. To mitigate this threat to validity, we executed each experiment five times and examined the variations in execution times. We did not observe major variations between executions. Furthermore, to ensure the reproducibility of the results, we have relied on publicly available logs and we have publicly released the implementations of the proposed techniques.

With respect to construct validity, we acknowledge that none of the measures of goodness used in the evaluation provides a full picture of the temporal dynamics of the process. To the best of our knowledge, there is no universal measure to assess the goodness of an automatically discovered simulation model. Each measure of goodness has limitations and biases that must be considered in their interpretation. In the case of the MAE measures (MAE and LMAE), one of the limitations is that these measures are not scaled, which prevents the identification of how big or small the reported error is. Another limitation of these measures is that they are not particularly sensitive to outliers, so some differences between the compared entities tend to be minimized. Third, these measures focus on comparing the cycle times of cases and the timeframes of the logs, and as such, they do not consider the durations of individual tasks and their order. Accordingly, we used other complementary measures, specifically the DL_MAE, which takes into account the duration of each work item and the order in which tasks occur in a trace, and the EMD, which measures the differences between the distributions of task durations. These latter measures, in turn, have their own limitations. Regarding the DL_MAE, the distance between sequences of task labels is evaluated using the well-known Damerau–Levenshtein (DL) editing measure. A limitation of this measure, when applied to business processes, is that it does not consider parallel relations between activities. In the case of the EMD, a possible limitation is its interpretability since the comparison made between the entities is independent of the units since it is carried out at the level of normalized histograms, which can make its interpretation challenging. That is why we must analyze the results reported in this paper holistically, considering all the reported metrics.

One of the hypothesis we sought to validate is that the proposed approach for handling multitasking produces higher gains for event logs with higher volumes of multitasking. To validate this hypothesis, we defined two indexes to measure the volume of multitasking in an event log, namely *MTLI* and *MTWII*. The key idea of both of these indexes is to divide the temporal intersection between two multitasked work items by the maximum of their durations, and then to aggregate these intersections across all overlapping work items in the log. Other approaches to measure multitasking are conceivable such as simply counting the number of multitasked pairs of work items. However, any potential bias in these indexes would not invalidate the general conclusion that a higher amount of overlapping between work items leads to a higher gain in accuracy (although the magnitude of overlap could be measured differently).

Related to the above, there is a potential threat to construct validity stemming from the approach used to generate event logs with multitasking from a log without multitasking (cf. Section 6.3.2). Specifically, we generated overlaps between pairs of adjacent work items by shifting their timestamps according to a percentage value given as a parameter. In this respect, we note that a higher percentage does not always lead to more overlap, since beyond a certain value, one of the work items will be shifted so much that it will move away from the reference work item, and thus the overlap between work items will be reduced instead of increased. We verified that this threat to validity did not materialize by measuring the volume of multitasking in each perturbed log using the two multitasking indexes mentioned above. As we increased the percentage value, the multitasking index kept increasing.

Finally, a potential threat to external validity is the use of a limited number of logs: 2 logs for the evaluation of the approach for handling multitasking, and 5 logs for the evaluation of the resource availability discovery approach. This situation stems from the limited availability of publicly available real-life logs that have start and end timestamps as well as resource identifiers. To mitigate this threat to validity we supplemented the real-life event logs with synthetic event logs constructed by perturbing a base log.

8. Conclusion

This article outlined an approach to discover business process simulation models from event logs in a way that takes into account multitasking and resource availability constraints.

To handle multitasking, the article put forward an approach to pre-process an event log in order to discover multitasking behavior and to adjust the processing times of tasks in such a way that the resulting log does not contain multitasking behavior, yet the resource utilization in the resulting log is equivalent to that in the original log. In this way, the BPS model discovered from the pre-processed log takes into account the multitasking behavior and can be simulated using traditional process simulators (e.g., BIMP).

To handle availability constraints, we adapted an algorithm for discovering calendar expressions from collections of time points [20]. For each resource pool, we extract the timestamps of the work items performed by this pool, and we use them to discover an availability calendar for the pool. Similarly, we extract the timestamps of the first work item of each trace, and we use them to discover a case creation calendar. The algorithm in [20] relies on user-defined support and confidence thresholds for discovering calendars. We propose to treat these two thresholds as hyper-parameters and to optimize them using the same optimization technique that is used to optimize other parameters used to discover the business process simulation model.

The empirical evaluation showed that the proposed approach improves the accuracy of the discovered simulation models both in the presence of multitasking and in the presence of availability constraints. Regarding multitasking, we identified that the higher the percentage of overlap between multitasked work items in a log, the more the approach improves the accuracy of the generated simulation models. When it comes to availability constraints, the empirical results showed, as expected, that BPS models that incorporate resource availability calendars have higher accuracy that those that do not. Furthermore, we found that it is best to first optimize the parameters used to discover the base simulation model, and then to optimize separately the confidence and support thresholds used by the calendar discovery algorithm. Furthermore, the latter two parameters are best optimized using a loss function that relies on absolute timestamps.

This article demonstrated the importance of taking into account different types of resource behaviors during the discovery of simulation models from even logs. The article focused on only two such behaviors (multitasking and availability constraints). In future work, we plan to further extend the approach to take into account other common types of resource behaviors, such as batching, task prioritization (giving higher priorities to some tasks than others), fatigue effects (resources being slower after working for several hours in a row), and differentiated resource performance (different resources having different performance levels for the same task). We also plan to investigate the question of how to take into account the fact that a give resource (or a resource pool) may be shared not only by multiple concurrent cases of the same process, but also by concurrent cases across multiple processes. In other words, we plan to lift the assumption that resources are fully dedicated to one single process.

CRediT authorship contribution statement

Bedilia Estrada-Torres: Conceptualization, Software, Validation, Investigation, Writing - original draft, Writing - review & editing. **Manuel Camargo:** Software, Validation, Investigation, Writing - review & editing. **Marlon Dumas:** Conceptualization, Methodology, Investigation, Writing - original draft, Writing - review & editing. **Luciano García-Bañuelos:** Writing - review & editing. **Ibrahim Mahdy:** Software. **Maksym Yerokhin:** Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is funded by the European Research Council (ERC Advanced Grant - Project PIX 834141) and the European Commission (FEDER) and the Spanish R&D&I programmes (grants RTI2018-101204-B-C22 (OPHELIA), P18-FR-2895 (EKIPMENT PLUS)).

References

[1] W.M.P. van der Aalst, Business process simulation revisited, in: Proceedings of EOMAS Workshop - CAiSE 2010, 2010, pp. 1–14.
[2] M. Dumas, M.L. Rosa, J. Mendling, H.A. Reijers, Fundamentals of Business Process Management, second edition, Springer, 2018.
[3] T. Rusinaite, O. Vasilecas, T. Savickas, T. Vysockis, K. Normantas, An approach for allocation of shared resources in the rule-based business process simulation, in: Proceedings of CompSysTech 2016, 2016, pp. 25–32.
[4] S. Peters, R.M. Dijkman, P. Grefen, Quantitative effects of advanced resource constructs in business process simulation, in: Proceedings of EDOC, 2018, pp. 115–122.
[5] N. Martin, B. Depaire, A. Caris, The use of process mining in business process simulation model construction - structuring the field, Bus. Inf. Syst. Eng. 58 (1) (2016) 73–87.
[6] M. Camargo, M. Dumas, O. González-Rojas, Automated discovery of business process simulation models from event logs, Decis. Support Syst. (2020).
[7] M. Pourbafrani, S.J. van Zelst, W.M.P. van der Aalst, Supporting automatic system dynamics model generation for simulation in the context of process mining, in: 23rd International Conference on Business Information Systems (BIS), 389, Springer, 2020, pp. 249–263.
[8] W.M.P. van der Aalst, Business process simulation survival guide, in: Handbook on Business Process Management 1: Introduction, Methods, and Information Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 337–370.
[9] C. Ouyang, M.T. Wynn, C. Fidge, A.H. ter Hofstede, J.-C. Kuhr, Modelling complex resource requirements in business process management systems, in: Proceedings of ACIS 2010, ACIS, 2010.
[10] M. Camargo, M. Dumas, O.G. Rojas, Simod: A tool for automated discovery of business process simulation models, in: Proceedings of Demonstration Track - BPM 2019, 2019, pp. 139–143.
[11] B. Estrada-Torres, M. Camargo, M. Dumas, M. Yerokhin, Discovering business process simulation models in the presence of multitasking, in: F. Dalpiaz, J. Zdravkovic, P. Loucopoulos (Eds.), Research Challenges in Information Science, Springer International Publishing, Cham, 2020, pp. 381–397.
[12] W.M. Van der Aalst, J. Nakatumba, A. Rozinat, N. Russell, Business process simulation: How to get it right?, in: BPM Center Report BPM-08-07, 285, BPMcenter. org, 2008, pp. 286–291.
[13] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, D. Edmond, Workflow resource patterns: Identification, representation and tool support, in: Proceedings of CAiSE, 2005, pp. 216–232.
[14] N. Afifi, A. Awad, H.M. Abdelsalam, RBPSim: A resource-aware extension of BPSim using workflow resource patterns, in: Proc. CEUR-WS, 2018, pp. 32–39.
[15] N. Afifi, A. Awad, H.M. Abdelsalam, Extending BPSim based on workflow resource patterns, in: Proceedings of BIS, 2018, pp. 206–222.
[16] Workflow Management Coalition (WfMC), Business process simulation specification, 2016, http://www.bpsim.org/specifications/2.0/WFMC-BPSWG-2016-01.pdf.
[17] J. Ling, Q. Feng, L. Zhang, A business process simulation method supporting resource evolution, in: Proceedings of ICSSP 2014, 2014, pp. 169–177.
[18] N. Martin, B. Depaire, A. Caris, D. Schepers, Retrieving the resource availability calendars of a process from an event log, Inf. Syst. 88 (2020) 101463.
[19] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, J.S. Vitter, Scalable sweeping-based spatial join, in: Proceedings of VLDB, 1998, pp. 570–581.
[20] Y. Li, X.S. Wang, S. Jajodia, Discovering temporal patterns in multiple granularities, in: Proceedings of International Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining-Revised Papers, in: TSDM '00, 2001, pp. 5–19.
[21] N. Martin, B. Depaire, A. Caris, Using event logs to model interarrival times in business process simulation, in: Proceedings of the Business Process Management Workshops (BPM Workshops), Springer, 2015, pp. 255–267.
[22] A. Augusto, R. Conforti, M. Dumas, M.L. Rosa, Split miner: Discovering accurate and simple business process models from event logs, in: Proceedings of ICDM, 2017, pp. 1–10.
[23] L. Pufahl, T.Y. Wong, M. Weske, Design of an extensible BPMN process simulator, in: Business Process Management Workshops, 2017, pp. 782–795.
[24] A. Madis, Lightning fast business process simulator, University of Tartu, 2011.
[25] D. Levy, Production Analysis with Process Mining Technology, NooL - Integrating People & Solutions, 2014, http://dx.doi.org/10.4121/uuid:68726926-5ac5-4fab-b873-ee76ea412399.

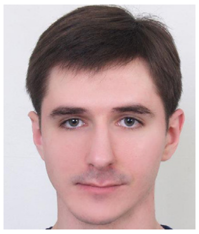
Bedilia Estrada-Torres is a Lecturer and Researcher at the University of Seville (Spain), and a member of the ISA Research Group. She received her Ph.D. with honors in 2018. She has collaborated and carried out research stays in Brazil and Estonia, and has participated in Spanish and European research projects. Her research interests include business process management, the analysis, modeling and management of process performance indicators in different scenarios such as structured processes, variability in process families, knowledge-intensive processes, and their relationship with decision-making processes.

Manuel Camargo is Junior Research Fellow of Information Systems at University of Tartu, Estonia, also is a Ph.D. student in Computer Sciences at University of Tartu, and Ph.D. student in Engineering at the Universidad de los Andes, Colombia. His research interests focus on business process simulation and deep learning techniques.

Marlon Dumas is Professor of Information Systems at University of Tartu, Estonia and Co-Founder at Apromore — a company dedicated to commercializing open-source process mining solutions. His research focuses on process mining, process simulation, and AI methods for automated process improvement. He is co-author of the textbook "Fundamentals of Business Process Management" (Springer, 2013).

Luciano García-Bañuelos is Research Professor at Tecnológico de Monterrey, México and Associate Professor at University of Tartu, Estonia. His research interests span across the fields of software engineering and business process management.

Ibrahim Mahdy received a Bachelor of Science in Computer and Software Engineering from the Helwan University, Egypt. He is currently pursuing a master's degree at the University of Tartu, Estonia, specializing in Software Engineering. Ibrahim was a research assistant in Business Process Management at the University of Tartu.



Maksym Yerokhin was born in Kharkiv, Ukraine. He has obtained his B.S. in Kharkiv National University of Radioelectronics in 2016 and M.S. in the University of Tartu in 2019, both in Software Engineering. In 2019–2020 was conducting research on Business Process Modeling concerning privacy field and multivariate analysis, based on the faculty of Computer Science of the University of Tartu. Contributed to Pleak and PIX projects, associated with the University of Tartu. Currently works as an independent expert in information technology.