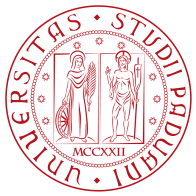

University of Padua

Department of Mathematics "Tullio Levi-Civita"

Master's Degree Course in Computer Science

Discovery of Resource Working Calendars from Process Event Log

Supervisor
Prof. Massimiliano de Leoni
University of Padua

Master Candidate
Alessandro Pegoraro

Student ID
1240466

Academic Year
2021-2022

Is the glass half empty or half full?
If you are filling it is half full,
If you are emptying it is half empty.

Acknowledgements

Thanks:

to Professor Massimiliano De Leoni from UNIPD
to Claudia, Francesca, Fabio and Alessandro from ESTECO
to my family from Scorzè
to my grandma, grandpa & grandma
to my friends.

Abstract

Business Process Simulation (BPS) refers to techniques for the simulation of business processes behaviours that allow analysts to compare alternative scenarios and contribute to the analysis and improvement 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 inputs to the simulator.
One of those inputs is the availability of roles and resources.
In this respect, it should be considered that staff members are not permanently available and that they can be involved in multiple processes within the company. While the default assumption of availability for these resources is either a trivial specification of availability e.g. from predetermined timetables or a continuous availability (24/7), various authors have described how both those assumptions can affect the flow of work and the execution time.

This thesis presents new methods to automatically retrieve the general availability constraints of both resources and roles from event logs containing process execution information and compares its results with the available state-of-the-art methods for calendar discovery.

Contents

List of Figures

List of Tables

1 Introduction

Business process simulation (BPS) is a widely used technique for analyzing the quantitative properties of business processes. The basic idea of BPS is to execute a large number of instances of a process, based on a simulation model enhanced with pinpoint parameters, to collect performance measures such as waiting times of tasks, processing times, execution cost, and cycle time [1][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 simulation model and given parameters can reproduce the observed reality.
When a process model is manually designed by an analyst, it may not be able to capture all the dependencies and patterns of how the process is performed in reality.
To solve these limitations the preferred solution is to automatically discover simulation models from business process execution logs (also known as event logs). Simulation models with parameters discovered in this way are generally more faithful since they do not only capture common patterns, but also uncommon behaviours. Furthermore, in contrast to what an analyst manages to explore and discover manually, these automated approaches typically explore a larger space of options when tuning parameters, also allowing one to find and express operations and processes at a finer granularity.
However, the existing techniques for business process simulation tend to ignore or make a generic assumption about the availability dependencies.
They either assume that resources are continuously available (24 hours a day, 5 to 7 days a week) or simply base their specifications on the "formal" availability provided and defined by the company (work-shifts timetables), disregarding the possibility of breaks, recurrent meetings and other uncommon behaviours that an analysis of the event log may instead provide. This thesis proposes new approaches to solve the problem of roles and resources' availability constraint discovery from event logs. We will then evaluate and compare our approaches with the current solutions available in the literature. These comparisons show that our approaches are a better fit to discover availability calendars in the presence of rare execution of activity occurring outside the usual work time (that hereafter we will call noise).

1.1 Organization

The thesis is organized in the following sections.

Section 2: the section presents the main theoretic concepts that can be useful for the reader;

Section 3: the section presents all the main related works;

Section 4: the section presents the proposed methods and their implementation;

Section 5: the section reports the experiments made to evaluate the proposed methods;

Section 6: the section describes the conclusion of this work and some possible future works.

2 Preliminaries

In this section are presented the main theoretic concepts that can be useful for reading this thesis.

2.1 Process Mining

Process Mining is a research field located between Data-Science and machine learning. The main purpose of process mining is to extract information from data recorded by information systems and turn them into insights and actions. The starting point for process mining is the event log. In modern organizations, almost every activity is performed by employing an information system that records all the operations. All these recordings are then collected into event logs.

2.2 Event Log

They represent a register of past executions of a process. Every operation is recorded as an event, which represents one specific execution of an activity.
Most event logs store additional information about events. Whenever possible, process mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the starting or completing *timestamp* of the event, or *data elements* recorded within the event (e.g. the size of an order).
In this thesis besides the case and activity to which an event is related, the timestamp, resource and transaction type also need to be recorded. Two transaction types that have to be registered are the start and completion of activities.
These event log requirements are formalized using the notation introduced by van der Aalst [1].
(Event Log Requirements): An event log E contains a set of events e. Each event e represents the start or completion of the execution of an activity a by resource r on case c at time τ.
Hence, each event e is represented as a tuple $e = (\text{c, a, r, } \tau, \varphi)$ with:

- $\#_c$(e) representing the case associated with event e
- $\#_a$(e) representing the activity associated with event e
- $\#_r$(e) representing the resource associated with event e
- $\#_\tau$(e) representing the timestamp associated with event e
- $\#_\varphi$(e) representing the transaction type associated with event e

For the remaining of the thesis, the *event log* is supposed to have been first converted into an *activity log*. This is done by mapping start events to their corresponding completion events, creating entries with both "timestamp:start" and "timestamp:complete". When multiple start and complete events are present for a combination of a case, activity and resource, the first occurring start event will iteratively be mapped to the first occurring unmapped complete event.

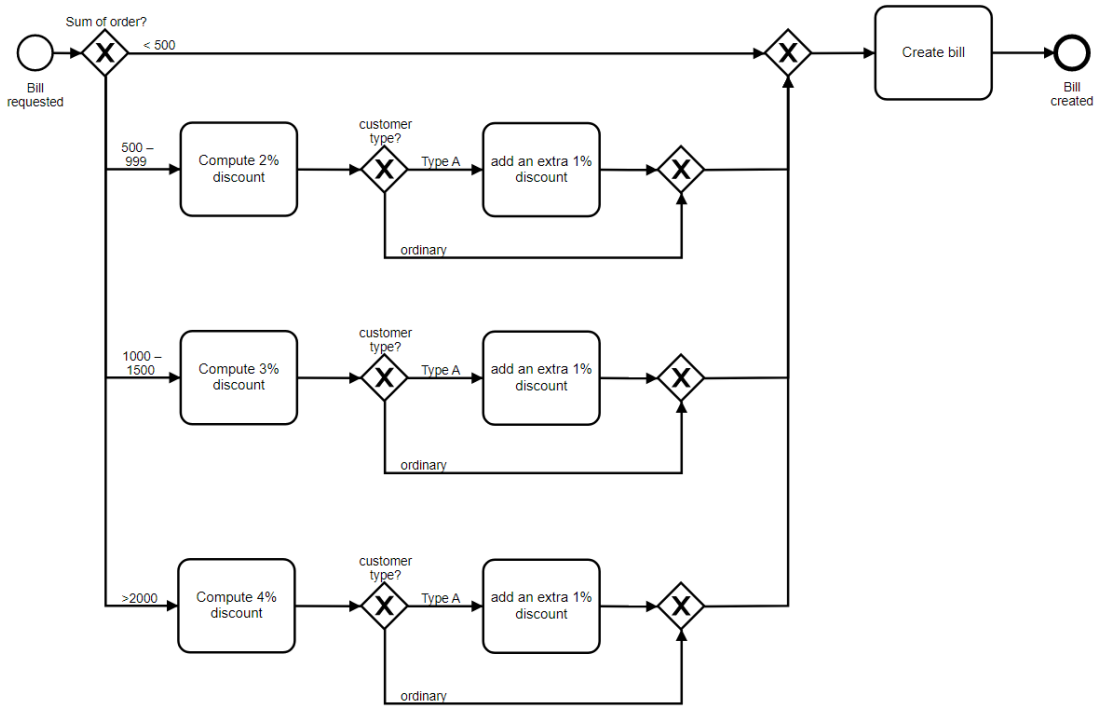

Figure 1: Example of a BPMN diagram for the creation of a bill

2.3 BPMN

Business Process Model and Notation (BPMN) is a graphical representation for specifying business processes in a business process model. It allows organizations to represent their processes with an intuitive notation and improves the communication between organizations.
BPMN consists of four core elements: activity, event, sequence flow and gateway. Each element can be of several types. For example, the gateway element can be exclusive, parallel, inclusive etc. Figure 1 shows an example of a BPMN diagram for the creation of a bill. The latest version of this document is BPMN 2.0.2 [8] published in January 2014

2.4 Simulation

Business Process Simulation (BPS) is a widely used technique for quantitative analysis of business processes [4]. The main idea of BPS is to generate a set of possible execution traces of a process from a BPMN diagram annotated with parameters such as the arrival rate of new process instances, the processing time of each activity and, for the scope of this Thesis, the calendar of availability of the resources simulated.

2.5 BPSim

BPSim (Business Process Simulation Specification) is a standard from WfMC (Workflow Management Coalition) [2] that defines the rules for the configuration of the simulation model and it allows the integration with several simulation software.

The BPSim specification describes in detail how to configure and assign resources to activities/tasks and how to raise events, decision making and other real-world capabilities.
The BPSim's simulation model is organized into scenarios, which contain the initial conditions and also the simulator configurations.
Parameters are provided to configure the scenario by setting the duration of the entire simulation, the number of replication, the start of the simulation and, specific to our case, the roles or resources availability calendars.
In turn, a scenario is composed of a collection of Element Parameters. Each Element Parameter of a scenario represents a specific element of a process such as time, control or resource perspective. For each BPMN element (task, gateway, events, etc.) is possible to create an Element Parameter and also define the related simulation parameters. For the scope of this Thesis, calendars defining the availability constraints of roles and resources are implemented using the *iCalendar* (RFC 5545) [3] format. Notably, we defined the resource's availability as the intervals during each day of the week in which the process participants (human or machine) are available and capable to perform an activity.
While the role's availability is considered as the intervals during each day of the week in which any activity belonging to the set constituting the role can be performed by a resource.
Both constraints are allowed in the simulation model and grant the ability to perform different simulations and study the development of different business processes.

2.6 Calendar

Also called "availability calendar" or "working calendar", it defines the availability constraint either for the roles or for the resources. It is made up of one (e.g. standard 8 a.m. to 6 p.m. working calendar for a full time employed "human" resource) or more availability intervals. Those constraints could change as the execution of the simulated process evolves e.g. a resource could work the morning on even months (February, April, etc) and the afternoon on odd months (January, March, etc), another could work all the day only on Monday, but never on Friday, etc.
So it is necessary to "enrich" its characterization in the simulation model with the work-Shifts of roles or resources, to obtain a better representation of the processes that we want to simulate, understand and improve.

3 Related Works

In this section, we present all the main related works. These studies were used as starting point for our work. Follows then a brief discussion about the weaknesses found compared to our proposal.
While it is worth noticing that there exists a rich body of literature to optimally create resource timetables to, for example, minimize staffing costs while maintaining service levels, there is strikingly little work on empirically deducing the resource availability prevailing in reality [7]. They instead either expect 24-hour availability or express the "formal" availability of a resource for the company as a whole.

3.1 "Discovering business process simulation models in the presence of multitasking and availability constraints" by Estrada-Torre et Al.

The study [5] tries to find a *calendar-based pattern* (SP) made up of a pair of calendar expressions and constraints.
A calendar expression is a tuple of time granules where each subsequent time granule is a subset of the previous, i.e. (year: 2022, week: 1, day_of_week: 3, hour: 7) is a calendar expression and year, week etc. are time granules. It then tries to apply an algorithm for discovering calendar expressions from collections of time points to infer resource timetables from an execution log.
The study supposes that a time point refers to both "timestamp:start" and "timestamp:complete" because both the start and the end of a work item are points in time during which a resource is known to have been actively working. While in the "time periods" between these time points it should be considered unknown if the resource was actively working, this is true, especially in the case of multitasking or breaks that may not be registered (lunch, meetings, events starting and ending on different days, etc.).
The algorithm proposed returns a calendar expression that should cover the majority of the time point extracted from the event log. Concretely, to infer the resource timetable it first extracts the first event associated with each case/trace and it retains the "timestamp:start" of this work item. It then discovers a set of calendar expressions from the collection of all such "timestamp:start". But this approach does not consider the end of availability for the resource, nor does it consider the possibility of work-shifts. Nonetheless, the hypothesis of the uncertainty of work between the start and completion of events allows us to remove the type of noise generated by blindly considering the resource always active during periods defined by events with a high (many hours, different days) delta difference between start and complete timestamp.

3.2 "Retrieving Resource Availability Insights from Event Logs" by Martin et Al.

The paper [6] is the first to recognize the importance of defining the availability constraints for roles and resources in the simulation model. Most human resources tend to be involved in multiple processes and they are not permanently available unlike what is expected from their timetable. Or the availability of the resource is known in terms of the number of hours, but it is unknown how these hours are distributed during the working day.
A naive simulation model, following only the specification obtained from the timetable, could therefore return scenarios less precise and that resemble less the equivalent reality that we want to model.
The algorithm proposed is focused on the daily resource availability for all the dates in the event log. This allows us to define, for each resource, daily active periods by merging events that follow each other. It also allows one to define boundary pending cases, and implicitly unavailable periods, by examining the event log and finding time frames in which new cases are present and the resource could perform an activity, but no resource activity is recorded.

The main drawback of this approach is that all insights are relegated to the days recorded in the event log, so the simulation model has no way to know the possible availability of the possible availability of the resource outside the aforementioned days.
Another problem of this approach is the lack of generality, it is easy to imagine that for large simulation models it is unfeasible to define the availability of each resource by listing all active periods for each day. While a more generic calendar may result in less accurate scenarios our approach tries to cover this with the introduction of work-shifts for each resource.

3.3 "Retrieving the resource availability calendars of a process from an event log" by Martin et Al.

The paper [7] takes from the aforementioned article by the same author to resolve the problem of lack of generality and it proposes two methods to discover availability calendars from the daily availability record:

- **direct sampling**
- **cluster-based sampling**

Those approaches offer a more generalized result, but still suffer from the problem described before, that is the limitation of having to create a calendar for every single day of the event log that we want to reproduce in the simulation model.
Also, these approaches heavily depend on obtaining a good enough sampling, both introducing randomness in the results. Our methods instead can be considered "pure" i.e. they do not depend on anything other than their parameters, so when invoked in a different context or at a different time with the same arguments, they will produce the same result.

3.3.1 Direct Sampling

First, this method groups all records of availability of dates of the specific day of the week (all that are Monday, Tuesday, etc) and then it randomly chooses a record to create the availability calendar. This approach heavily leans on the fact that abnormal patterns and noise should be less frequent and so the probability that records with those noisy intervals are drawn is lower.

3.3.2 Cluster-Based Sampling

This method identifies a limited number of "representative" daily availability records to sample from when composing a resource availability calendar.
All availability records are grouped for each day of the week, then each group is subject to separate cluster analysis. First, through hierarchical clustering, the number of clusters is decided, after which partitioning around medoids (similar to k-means) is used to obtain a cluster solution.
the medoids of the final cluster solution will be considered "representative" for their respective cluster in the final sampling for the availability calendar composition. The size of each cluster will determine the probability of selection.

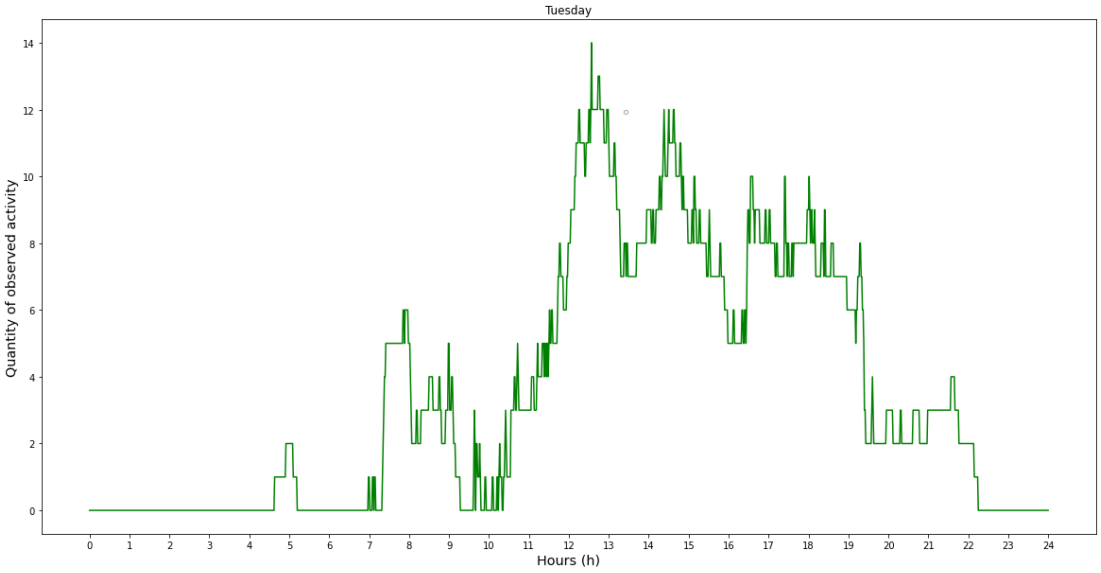

Figure 2: Example of distribution of activities of a role, for all Tuesdays in an event log

4 Algorithm

In this section we will introduce the three main algorithms, the first one handles the discovery of calendar for roles and the second and third ones handle the discovery of work-shifts for roles and individual resources. All algorithms take a filtered event log as input and, using a Data-Science approach, are not bound to any subjective opinion or input from the process stakeholders.
In those sections, we refer to *day of the event log* as a specific date in the event log, i.e. Friday 2022-05-06, and *day of the week* as one of the seven days of the week or all the days recorded in the event log with that specific day of the week, i.e. if we refer to Monday as the day of the week we refer to all Mondays in the event log.

4.1 Working Calendar Discovery for Roles

This algorithm observes the distribution of work of the resources when executing an activity belonging to the role and tries to extract the calendar that defines their availability constraints.

In this section, we refer to the event log as one filtered by the selected role and the selected day of the week. For example Figure 2 shows the distribution of activities executed by the resources of a role, calculated as the sum of all the activities registered during all Tuesdays in the event log. From this data, the algorithm tries to extract a possible calendar using a Data-Science approach, to obtain both a good enough generalization (with the removal of possible noise) and a calendar with which we can simulate a new event log as similar as possible to the original.

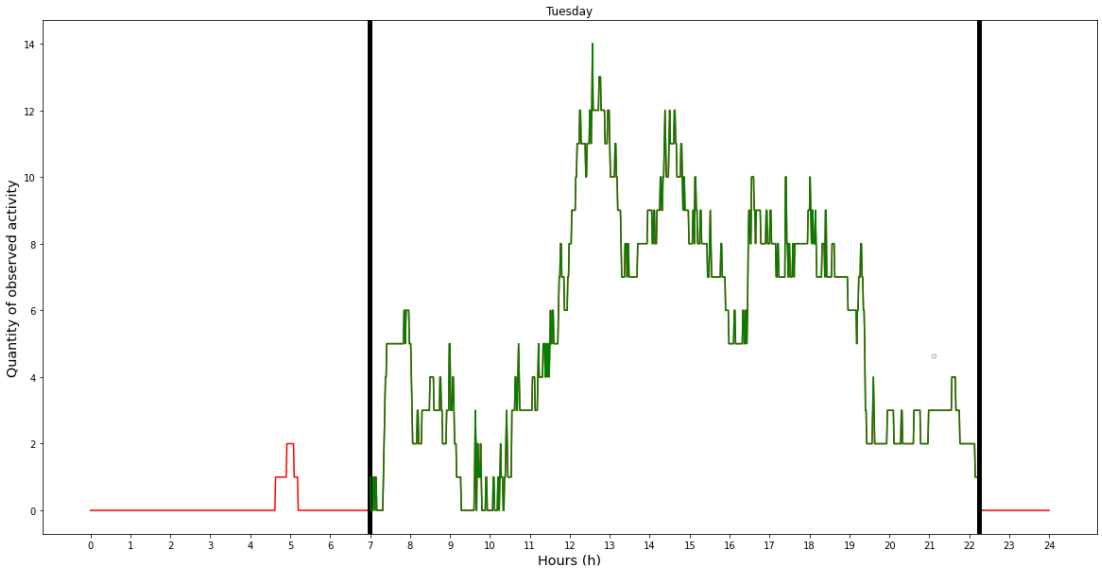

Figure 3: Example of role's availability calendar on Tuesday: 7:00 a.m. to 10:20 p.m., highlighted in red the intervals considered noise

The algorithm for role's calendar discovery depends on two parameters:

- **Threshold**: the minimum duration of activity that would not be considered noise, e.g. with a threshold equal to 10% all the intervals smaller than a tenth of the total length would be eliminated.
- **Tolerance**: the maximum distance between two consecutive intervals that would make the algorithm consider them as one, e.g. two consecutive activity intervals smaller than the threshold, but that are distant from each other by less than the tolerance, can be counted as one single interval of activity, that can be bigger than the threshold and not labelled as noise and removed.

Even slightly different combinations of values for threshold and tolerance can lead the algorithm to totally different working calendars. To choose which pair of values return the calendar that best reflects the working availability of the role we try to maximize the output of an objective function defined as the harmonic mean (F_1 score) of precision and recall, minus the numerosity, plus the calendar size.

$$\gamma = 2 * \frac{precision * recall}{precision + recall} - numerosity + calendar\ size \tag{1}$$

The different elements of γ (1) are computed as follows:

- **Precision**: one minus the sections of intervals in the discovered calendar that are not present in the distribution of activity of the event, divided by the sum of all intervals composing the calendar. For example, in Figure 3, the discovered interval states that there should be an active execution of work from 9:20 to 9:45, but then as we can see there is no activity recorded on that segment. That and the other segments, which are covered by the

intervals but have a value of zero in the distribution, are then divided by the total duration of all intervals in the calendar i.e. from 7:00 a.m. to 10:20 p.m.

$$1 - \frac{activities\ not\ in\ event\ log,\ but\ expected\ in\ calendar}{total\ duration\ covered\ by\ calendar}$$

- **Recall**: the number of activities in the event log that are completely covered by the calendar, divided by the total number of activities in the event log.

$$\frac{activities\ in\ event\ log\ completely\ covered\ by\ calendar}{total\ number\ of\ activities\ in\ event\ log}$$

- **Numerosity**: the number of intervals in the calendar divided by 24 (we suppose that during a day the minimum length of an interval is one hour). This parameter incentivizes calendars with fewer intervals, boosting the score of calendars discovered using higher tolerance and counterbalancing the negative effect of precision: discovering calendars with high tolerance means that we could have large intervals that are the union of two or more consecutive intervals, those by definition should have a segment between the two, and this segment is composed of only zero values (otherwise it would be considered another interval), when calculating the precision this segment is covered by an interval of the calendar, but those activities are not in the event log, decreasing the value of precision.

$$\frac{number\ of\ intervals\ in\ calendar}{24\ intervals}$$

- **Calendar Size**: the total length of all intervals in the calendar divided by the total length of the day (dependent on the specified time unit: 24 hours, 1440 minutes, 864000 seconds etc.). This parameter incentivizes calendars to cover as much area as possible while still being restricted by the threshold to disregard segments labelled as noise.

$$\frac{length\ intervals\ in\ calendar}{time\ unit}$$

To find the pair of threshold and tolerance that maximize (1), we apply the algorithm testing all possible combinations in a customizable range (in our experiments the range was $1\% - 30\%$ for the threshold and $0 - 30$ minutes for the tolerance).
Figure 3 shows that for our example the availability on Tuesday is from 7:00 to 22:20, while the work recorded around 5:00 should be considered noise.
The calendar in the example in Figure 3 was obtained with a threshold of 0.01% and a tolerance of 21 minutes, this pair of values gives the maximum value of 1.5542 for γ (1), computed with the following values:

- Precision: 0.9355
- Recall: 0.9867
- Numerosity: 0.0416
- Calendar Size: 0.6354

4.2 Implementation of Calendar Discovery

Algorithm 1: Discovery of calendar by optimization of threshold and tolerance

Data: Event log filtered by role and day of the week
Result: List of intervals in the specified time unit

```
1  best_γ ← 0                          /*Keeps the maximum value found
2  calendar ← [ ]                      /*Keeps the best calendar found
3
4  for threshold ← 0.01 to 0.3 do
5      for tolerance ← 0 to 30 do
6          intervals ← find_calendar(event_log, threshold, tolerance)
7          precision ← find_precision(event_log, intervals)
8          recall ← find_recall(event_log, intervals)
9          numerosity ← intervals.size() / 24
10         calendar_size ← find_total_length(intervals) / 1440
11
12         current_γ ← F1(precision, recall) − numerosity + calendar_size
13
14         if current_γ > best_γ then
15             best_γ ← current_γ
16             calendar ← intervals
17         end
18     end
19 end
```

Algorithm 1 at line 6 finds the intervals that constitute the calendar that best respect the threshold and tolerance set for the current cycle using the function defined in Algorithm 2.
Precision, recall, numerosity and calendar size are calculated on lines $7-10$ as described in Section 4.1.
On line 12 we calculate function γ (1) and on line 14 we check if we have found a better value, i.e. a better pair of threshold and tolerance if so we memorize the new calendar.

The function $find\_calendar$ described in Algorithm 2 takes as input the chosen threshold and tolerance and an event log filtered by role and day of the week in which all entries contain both the "start" and "complete" timestamp as shown in Table 1.
The function starts by populating an array of length relatives to the time unit chosen (in our case it's 1440 minutes) to obtain the distribution of work shown in Figure 2.
Note that in line 5 both $event["start"]$ and $event["complete"]$ are the representation in time unit (minute in our case), respectively, of "timestamp:start" and "timestamp:complete" of the activity described in the event.

Case Id	Activity	start	complete	Resource	...
...	...	...	...	...	...
105	C	2022-01-01 14:30	2022-01-01 15:30	R1	...
81	A	2022-01-08 9:43	2022-01-08 13:15	R4	...
111	C	2022-02-19 8:17	2022-02-19 9:55	R2	...
36	B	2022-01-08 9:33	2022-01-08 11:22	R1	...
...	...	...	...	...	...

Table 1: Example of event log filtered on Saturday

For example, an event with "timestamp:start" 2022-01-01 14:30 is represented in the range $0 - 1440$ as 870, and "timestamp:complete" 2022-01-01 15:30 is represented as 930, reading this event will increase by one unit all the elements in the array in the range: $870 - 930$.
We then begin searching for the set of intervals that compose the calendar, starting at line 12 when we first find the mark of activity, we begin searching for the end of the interval under research using the variable j. When we no longer found any event of activity we start to check if, in the range defined by the tolerance, we can find the beginning of another interval, to merge the two. We use the variable *patience* that assumes the values in the range $0 - tolerance$.
At line 25 we check two possible cases:

- $tolerance\_check == False$ if we found a second interval to merge in the range allowed by the input *tolerance*
- $patience + j < 1440$ this checks that we do not exceed the range of the array i.e. the current interval ends at 12 p.m.

If both cases are true, then we continue with the search for the ending point of the interval under observation. Otherwise, we have found the end of the current interval (or we have reached the end of the array) and we can check if it satisfies the given threshold. On lines 29 - 31 we calculate the *ratio* between the length of the discovered interval and the sum of the total lengths of all intervals.
This is done by adding all the values within the start and end of the interval and dividing the result by the sum of all values in the array i.e. the sum of all the intervals. If this *ratio* is smaller than the *threshold* given as input then the interval is discarded following the definition in Section 4.1

Algorithm 2: Discovery of calendar for the given threshold and tolerance

```
Data: Event log filtered by role and day of the week
      Threshold
      Tolerance
Result: List of intervals in the specified time unit
Function find_calendar(event_log, threshold, tolerance):
    activities ← [0] * 1440                    /*Starting array with all 0
    for case in event_log do
        for event in case do
            activities[event["start"], event["complete"]] ← +1
        end
    end
    intervals ← [ ]
    start ← 0
    end ← 0
    for i ← 0, i <= 1440 do
        if activities[i] ≠ 0 then
            start ← i                          /*Found start of interval
            search_interval ← True
            for j ← i, j <= 1440 & search_interval do
                if activities[j] == 0 then
                    tolerance_check ← True
                    patience ← 1
                    for tolerance_check & patience <= tolerance &
                                                  patience + j <= 1440 do
                        if activities[j + patience] ≠ 0 then
                            tolerance_check ← False
                        end
                    end
                    if tolerance_check == False & patience + j < 1440
                     then
                        j ← +patience  /*End of interval not found
                    else
                         /*Found end of interval
                        end ← j − 1
                        num ← SUM(activities[start, end])
                        den ← SUM(activities)
                        ratio ← num /den
                        if ratio >= threshold then
                            intervals.append(start, end)
                        end
                        search_interval ← False
                        i ← j + 1
                    end
                end
            end
        end
    end
return intervals
```

Resource	Case Id	Activity	start	complete	...
R1	...	...	...	...	...
R1	105	C	2022-01-01 8:30	2022-01-01 12:03	...
R1	81	A	2022-01-08 13:15	2022-01-08 18:04	...
R1	...	...	...	...	...
R2	...	...	...	...	...
R2	111	C	2022-02-19 13:12	2022-02-19 18:09	...
R2	36	B	2022-01-08 13:20	2022-01-08 18:00	..
R2	...	...	...	...	...
R3	...	...	...	...	...
R3	105	D	2022-01-01 15:33	2022-01-01 17:45	...
R3	36	D	2022-01-08 9:48	2022-01-08 12:01	..
R3	...	...	...	...	...

Table 2: Example of event log filtered on Saturday for work-shift discovery

Resource	date	interval
R1	...	...
R1	2022-01-01	8:30 - 12:03
R1	2022-01-01	13:11 - 17:57
R1	2022-01-08	13:15 - 18:04
R1	...	...
R2	...	...
R2	2022-01-08	13:20 - 18:00
R2	2022-02-19	13:12 - 18:09
R2	...	...

Table 3: Resource's work-shift for each Saturday in the event log of example

4.3 Work-shift Discovery for Roles

In this section, we refer to the event log as one filtered by the selected role and the selected day of the week. From this data, the algorithm tries to extract all possible work-shifts of the resources in the role and then it merges all similar work-shifts to avoid all duplicates and obtain the optimal set of intervals for the simulation.

First, the algorithm tries to eliminate any possible noise, filtering out all events in the event log with *timestamps* outside the bounding defined by the calendar discovered using Algorithm 1.

Then for each resource, we merge consecutive active periods [6] having the same date in the event log.

At the end of these steps, we should have the specific intervals for each day in the event log in which the resources are registered as active, as shown in Table 3.

For the next step, we want to obtain all possible work-shift that each resource can cover on the selected day of the week. To manage this, first, we merge all intervals (of the same resource) that overlap by more than a certain percentage (in our experiment we used a degree of similarity of 70%).

Resource	Work-shift
R1	8:30 - 12:03
R1	13:11 - 18:04
R2	13:12 - 18:09

Table 4: Saturday work-shift for each resource

Role	Work-shift
{A, B, C, D}	8:30 - 12:03
	13:11 - 18:09

Table 5: Saturday work-shift for the role

The overlapping of intervals is calculated as a degree of similarity using the following formula dependent on the chosen time unit (in our experiment 1440 minutes):

$$\begin{aligned} SIM(interval_1, interval_2) &= \frac{\sum_{t=0}^{1440} interval_1(t) * interval_2(t)}{\sum_{t=0}^{1440} SIGN(interval_1(t) + interval_2(t))} \\ &= \frac{interval_1 \cap interval_2}{interval_1 \cup interval_2} \end{aligned} \tag{2}$$

with

$$interval_n(t) = \begin{cases} 1, & \text{if } t \in interval_n \\ 0, & \text{otherwise} \end{cases}$$

and

$$SIGN(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

We can observe the result of this second-last step in Table 4. The last step of the algorithm is to merge the work-shift of each resource into a unique set of work-shifts for the role. To do this we repeatedly apply the similarity function until we reach a set of disjunctive work-shifts as shown in Table 5.

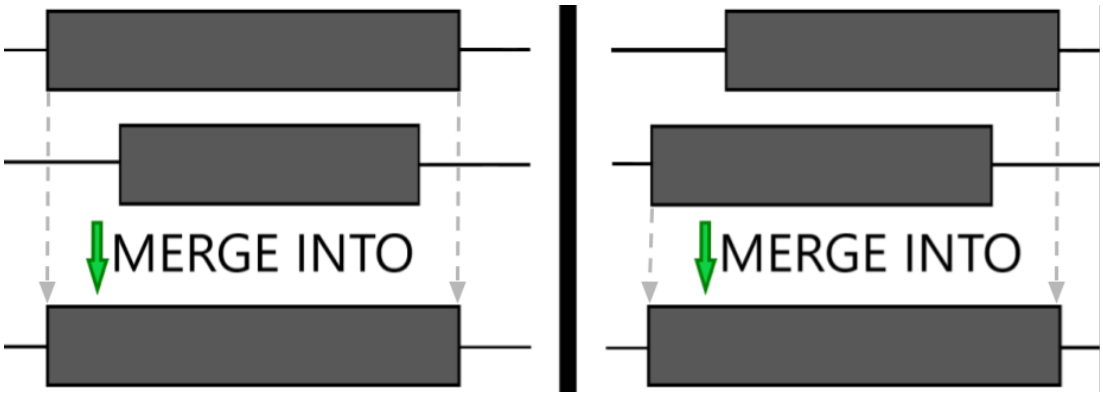

Figure 4: The two possible types of intervals merge

4.4 Implementation of Work-shift Discovery for Role

Algorithm 3 takes as input an event log filtered by any day of the week and a specific role. Starting at line 1 it uses the function defined on Algorithm 4 to both filter the event log by role and to remove all events considered noise. On line 2 the function $extract\_resources()$ return the list of all the resources that appear in the event log, then for each resource it cycles all dates in the event log in chronological order, starting from the first date (line 5) and ending with the last (line 6).
On lines 11 and 12, it checks if the current resource belongs to the event under examination and if the date of the "timestamp:start" is the one that it is currently checking. If this is true it saves the event in a temporary list. When all cases and events for the current date and resource have been checked, it tries to merge all events in the temporary list on line 18.
The function $merge\_events()$ takes as input a list of events and for each pair of events if they are registered as consecutive active periods [6] it merges them into a single event. It then tries to repeat this process until no further pair can be merged. The output is a set of events that we convert into a list of intervals that we add to the global list of intervals: $work\_shifts$.

Lines 19 to 26 and 31 to 38 describe the same subroutine, the first is bounded by the intervals of a single resource, while the second operates on the intervals of all resources. The subroutine takes a pair of intervals and if their degree of similarity, calculated using (2), is higher than the selected percentage it creates a new interval made from their union and it removes the two old intervals. Since the new interval is re-added to the cycled list, we are assured that both subroutines will repeatedly apply the similarity function until they reach a set of disjunctive work-shifts.

Algorithm 3: Discovery of role's work-shifts

```
Data: Event log filtered by day of the week
      Role
Result: List of work-shifts in the specified time unit
event_log ← filter_event_log_role(event_log, role)
resource_list ← extract_resources(event_log)
work_shifts ← [ ]
for RES in resource_list do
    curr_date ← extract_first_date(event_log)
    end_date ← extract_last_date(event_log)
    while curr_date <= end_date do
        events_to_merge ← [ ]
        for case in event_log do
            for event in case do
                if event["org : resource"] == RES then
                    if event["start"].date() == curr_date then
                        events_to_merge.append(event)
                    end
                end
            end
        end
        interval_list ← merge_events(events_to_merge)
        for (INTER_1, INTER_2) in interval_list do
            similarity ← SIM(INTER_1, INTER_2)
            if similarity >= 0.7 then
                INTER_3 ← merge_intervals([INTER_1, INTER_2])
                interval_list.delete([INTER_1, INTER_2])
                interval_list.append(INTER_3)
            end
        end
        work_shifts.append(interval_list)
        curr_date ← +7 days
    end
end
for (INTER_1, INTER_2) in work_shifts do
    similarity ← SIM(INTER_1, INTER_2)
    if similarity >= 0.7 then
        INTER_3 ← merge_intervals([INTER_1, INTER_2])
        work_shifts.delete([INTER_1, INTER_2])
        work_shifts.append(INTER_3)
    end
end
```

Algorithm 4: Filter an event log by role and discard all events that do not match it's calendar

```
   Data: Event log
         Role
   Result: Filtered event log
 1 Function filter_event_log_role(event_log, role):
 2 |  event_log ← filter(event_log, role)
 3 |  calendar ← calendar_discovery(event_log)
 4 |  new_event_log ← [ ]          /*Filtered event log starts empty
 5 |  for case in event_log do
 6 |  |  new_case ← [ ]
 7 |  |  for event in case do
 8 |  |  |  if event["start"] & event["complete"] ∈ calendar then
 9 |  |  |  |  new_case.append(event)
10 |  |  |  end
11 |  |  end
12 |  |  if new_case.size() > 0 then
13 |  |  |  new_event_log.append(new_case)
14 |  |  end
15 |  end
16 return new_event_log
```

The function filter_event_log_role described in Algorithm 4 takes as input an event log and a specific role (set of activities).
First, the function filters the event log by role, removing all events that do not have one of the activities from the role, then at line 3, using the calendar discovery defined in Section 4.1, discover the set of intervals that it will use to filter the event log.
The function cycle each case composing the event log and from those cases extract the corresponding events, then on line 8, it checks that both "timestamp:start" and "timestamp:complete" of the event are covered by an interval of the calendar Lastly, it uses all events that passed the test to create a *new_case* to add to the resulting filtered event log.

Case Id	Activity	start	complete	Resource	...
...	...	...	...	...	...
105	C	2022-01-01 8:30	2022-01-01 12:03	R4	...
81	A	2022-01-08 9:43	2022-01-08 13:15	R4	...
111	H	2022-02-19 13:12	2022-02-19 13:16	R4	...
36	B	2022-02-19 10:06	2022-01-08 13:10	R4	..
93	E	2022-02-26 8:29	2022-02-26 11:47	R4	...
12	A	2022-02-26 11:52	2022-02-26 12:59	R4	..
...	...	...	...	...	...

Table 6: Example of event log filtered on Saturday for resource R4's work-shift discovery

Resource	date	interval
R4	...	...
R4	2022-01-01	8:30 - 12:03
R4	2022-01-08	9:43 - 13:15
R4	2022-02-19	10:06 - 13:16
R4	2022-02-26	8:29 - 12:59
R4	...	...

Table 7: R4's work-shift for each Saturday in the event log

Resource	Work-shift
R4	8:29 - 13:16

Table 8: Saturday work-shift for the resource R4

4.5 Work-shift Discovery for Resources

In this section, we refer to the event log as one filtered by the selected resource and the selected day of the week. From this data, the algorithm tries to extract all the possible work-shifts of the resource and then it merges all similar work-shifts to avoid all duplicates and obtain the optimal set of intervals for the simulation.

First, the algorithm tries to eliminate any possible noise, filtering out all events in the event log that are not covered by the calendar discovered using Algorithm 1.
Given that a resource can belong to multiple roles, but an activity belongs to only one role, the algorithm must filter out the events that it considers noisy according to the appropriate calendar for the role containing the activity in the event.
The algorithm then merges consecutive active periods [6] having the same date in the event log.

At the end of these steps, we should have the specific working intervals for the days in which the resource's activity is registered in the event log as shown in Table 7.

The final step of the algorithm is to apply the similarity function (2) and repeatedly merge each interval into a unique set of work-shifts for the resource, obtaining the work-shift shown in Table 8.

4.6 Implementation of Work-shift Discovery for Resource

Algorithm 5 takes as input an event log filtered by any day of the week and a specific resource. Starting at line 1 it uses the function defined in Algorithm 6 to both filter the event log by resource and to remove all events considered noise.
Then it cycles all dates in the event log starting from the chronological first date (line 3) and ending with the last (line 4).
Lines 9 checks if the date of the "timestamp:start" is the one that it is currently checking. If this is true it saves the event in a temporary list, then when all cases and events for the current date have been checked, it tries, on line 14, to merge all events saved in the temporary list.
The function $merge\_events()$ takes as input a list of events and for each pair of events if they are registered as consecutive active periods [6] it merges them into a single event. It then tries to repeat this process until no further pair can be merged. The output is a set of events that we convert into a list of intervals that we add to the global list of intervals: $work\_shifts$. Lines 18 to 25 describe a subroutine that takes a pair of intervals from $work\_shifts$ and if their degree of similarity, calculated using (2), is higher than the selected percentage it creates a new interval made from their union and it removes the two old intervals. Since the new interval is re-added to the cycled list, we are assured that the subroutine will repeatedly apply the similarity function until it reaches a set of disjunctive work-shifts. The function filter_event_log_resource described in Algorithm 6 takes as input an event log and a specific resource.
First, the function filters the event log by resource, removing all events that are not performed by the specific resource, then at line 3, it extracts all roles belonging to this filtered event log. Then on lines 5 to 8, with each role found it builds a dictionary (map) with their specific calendar obtained from the **original** event log filtered by the role.
The function surveys each case composing the event log filtered by resource and from those cases extracts the corresponding events. On line 13 it cycles all roles searching for the one containing the activity in the current event, then when it founds a match it begins to check, on line 15, that both "timestamp:start" and "timestamp:complete" are covered by an interval of the calendar. Line 19 contains a break because we know that activities can not belong to more than one role, therefore continuing the search into $role\_list$ is useless and time-consuming Lastly, it uses all events that passed the test to create a new_case to add to the resulting filtered event log.

Algorithm 5: Discovery of resource's work-shifts

Data: Event log filtered by day of the week
Resource
Result: List of work-shifts in the specified time unit

```
event_log ← filter_event_log_resource(event_log, resource)
work_shifts ← [ ]
curr_date ← extract_first_date(event_log)
end_date ← extract_last_date(event_log)
while curr_date <= end_date do
    events_to_merge ← [ ]
    for case in event_log do
        for event in case do
            if event["start"].date() == curr_date then
                events_to_merge.append(event)
            end
        end
    end
    interval_list ← merge_events(events_to_merge)
    work_shifts.append(interval_list)
    curr_date ← +7 days
end
for (INTER_1, INTER_2) in work_shifts do
    similarity ← SIM(INTER_1, INTER_2)
    if similarity >= 0.7 then
        INTER_3 ← merge_intervals([INTER_1, INTER_2])
        work_shifts.delete([INTER_1, INTER_2])
        work_shifts.append(INTER_3)
    end
end
```

Algorithm 6: Filter an event log by resource and discard all events that do not match their calendar

```
Data: Event log
      Resource
Result: Filtered event log
Function filter_event_log_resource(event_log, resource):
    resource_eventlog ← filter(event_log, resource)
    role_list ← extract_role(resource_eventlog)
    dict_calendar ← dictionary()
    for ROLE in role_list do
        role_eventlog ← filter(event_log, ROLE)
        dict_calendar[ROLE] ← calendar_discovery(role_eventlog)
    end
    new_event_log ← [ ]          /*Filtered event log starts empty
    for case in resource_eventlog do
        new_case ← [ ]
        for event in case do
            for ROLE in role_list do
                if event["activity"] ∈ ROLE then
                    if event["start"] & event["complete"]
                        ∈ dict_calendar[ROLE] then
                        new_case.append(event)
                    end
                    break
                end
            end
        end
        if new_case.size() > 0 then
            new_event_log.append(new_case)
        end
    end
return new_event_log
```

5 Experiments

In this section are reported all the experiments that we performed to analyze the goodness of our approaches for the discovery of role's working calendars and the discovery of work-shifts for roles and resources.
All algorithms are fully implemented in *Python*[1] and the key packages used are *BPSimpy*[2] for the generation of simulation models compliant with the Business Process Simulation Specification (BPSim) defined in Section 2.5 and *PM4Py*[3] for data manipulation and feature extraction.

The synthetic event logs were generated using the *Lanner simulator (L-sim)*[4] which allows us to define scenarios with calendars for roles or calendars for every single resource. Synthetic event logs with either role or resource's work-shifts are obtained by tweaking the "*rrule*" field in the definition of *iCalendar* [3] for each shift.

To test our approach for role's work-shifts discovery against the methods available in the literature (Section 3) that are instead based on the resource's calendar discovery, we limited our simulation models to have only resources that coincide with the roles i.e. the activities that a resource can perform belong to a single role.
While in the case of resource's work-shifts discovery we were able to define more complex simulation models, even with resources performing activities of every role.

All experiments, in the case of absence or presence of noise, were carried out against the three main methods proposed by the literature: Calendar Expression discovery 3.1 by Estrada-Torre et Al. Direct Sampling 3.3.1 and Cluster-Based Sampling 3.3.2 by Martin et Al.
For all results gathered a brief analysis and explanation were provided in Sections 5.2.2, 5.2.4, 5.3.2 and 5.3.4.
In the case of Direct Sampling 3.3.1 and Cluster-Based Sampling 3.3.2 to avoid the possibility of returning a single skewed result (especially in the case of event logs heavily influenced by noise) we computed the average accuracy by repeating the experiments 100 times and returned the average score of all tests.

All algorithms defined in Section 4 take event logs filtered by a specific day of the week as input. Since we observed, in the case of human resources and no night shifts, that each filtered event log is mutually independent of the other, i.e. does not share any event, it is possible to apply the methods in parallel for each day of the week and obtain a complete set of calendars or work-shifts.
In our experiments, this was viable using environment-specific containers such as dictionaries and paying attention not to overlap the accesses to the segments holding each separate filtered event log.

[1] www.python.org
[2] github.com/claudiafracca/BPSimpyLibrary.git
[3] pm4py.fit.fraunhofer.de
[4] io.witness.cloud

5.1 Synthetic Event Log Over Real-Life Event Log

Apart from the first experiments used to gain domain knowledge and experience with Python libraries and evaluate the implementation of the algorithms in the literature, all subsequent tests were carried out using synthetic event logs that we generated. This is due to three major problems with the real-life event logs currently available:

- The absence of both "timestamp:start" and "timestamp:complete" in the events, as those are necessary to determine intervals of availability and active periods, for this reason, all event logs missing the start and end of activities were discarded.
- In the case of event logs with both start and complete timestamp, we instead found that it was the field specifying the resource that was missing. Given that the record of resources' activities was necessary to compare our Algorithm 5 for resource's work-shifts discovery with the methods proposed in the literature, all event logs missing the information regarding the resource performing the activity were discarded.
- The absence of a "*Ground Truth*", even using the remaining event logs (most of which were found through the literature described in Section 3) carries a fundamental problem, that is, we had no way to check that any result we obtain could mirror the true calendars of the event log. A qualitative analysis could be made, but, for example, it does not allow to check the accuracy of calendars or work-shifts, or even the existence of work-shifts in the event log.

For these reasons, most of the experiments done, and all of the ones analyzed in this section are made using synthetic event logs and the accuracy of the calendars was evaluated by comparing the intervals discovered against the "*Ground Truth*" using the similarity function (2).

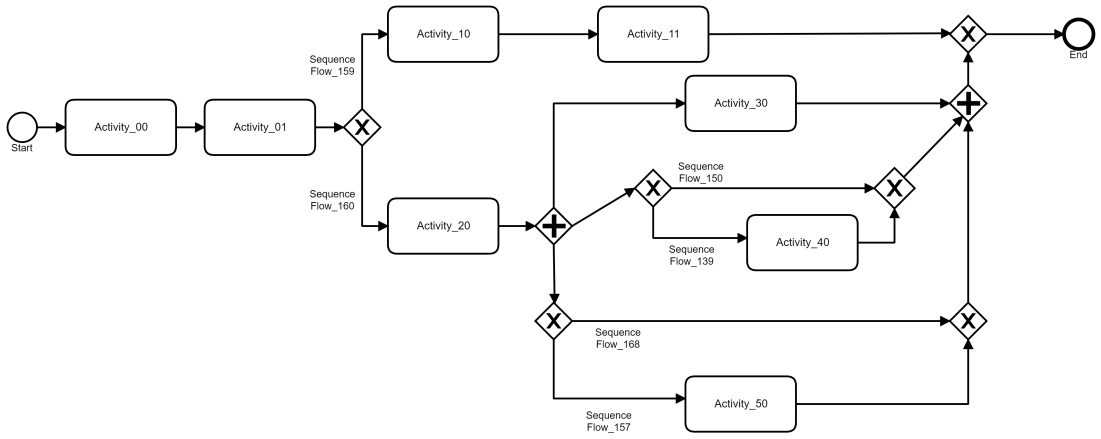

Figure 5: BPMN diagram used to generate event log with no noise

5.2 Experiments with Synthetic Event Log Containing no Noise

The initial simulation models were based on the BPNM in Figure 5. It generates scenarios with no noise and the following roles:

- Role 0: "*Activity_ 00*", "*Activity_ 01*"
- Role 1: "*Activity_ 10*", "*Activity_ 11*"
- Role 2: "*Activity_ 20*"
- Role 3: "*Activity_ 30*"
- Role 4: "*Activity_ 40*"
- Role 5: "*Activity_ 50*"

5.2.1 Experiment for Role's Work-shifts Discovery in the Case of Absence of Noise

For this type of experiment, it was necessary to have the resources' activities coinciding with a single role. Each role was simulated using 5 resources for a total of 30 resources.
Here follows the analysis of three cases:

- Case 1: availability of role 0, all working days (Mon to Fry) from 8:30 to 12:30 and from 14:00 to 18:00
- Case 2: availability of role 1, all weekend (Sat to Sun) from 5:00 to 12:00 and from 14:30 to 22:30
- Case 3: availability of role 2,3,4 and 5, all days (Mon to Sun) with 4 resources working 8:00 to 12:30 and one resource working 12:30 to 13:30

	Calendar Expression	Direct Sampling	Cluster-Based Sampling	Role's Work-shifts Discovery
Case 1	0.9813	0.9813	0.9813	0.9813
Case 2	0.9889	0.9889	0.9889	0.9889
Case 3	0.8187	0.9969	0.9969	0.9969

Table 9: Results of the application of the methods proposed in the literature and our algorithm for role's work-shifts discovery on event logs with no noise

5.2.2 Results of Experiment for Role's Work-shifts Discovery in the Case of Absence of Noise

As we can see from Table 9, in the simplistic cases 1 and 2 with no noise and an equal allocation of resources, all algorithms perform similarly (the difference from the "*Ground Truth*" is given from the fact that in the simulation the resources may not start working exactly when available) returning the same calendars. While in case 3 the algorithm for Calendar Expression discovery 3.1 does not consider the confidence in the calendar expression of the resource working from 12:30 to 13:30, discarding it from the calendar expression used to calculate the intervals of the calendar. In particular, the time-points around 12:30 account for 5 resources (ending of availability for 4 e starting of availability of one), while the time-points at 13:30 are supported only by one resource.

5.2.3 Experiment for Resource's Work-shifts Discovery in the Case of Absence of Noise

In this type of experiment, we analyze three different resources belonging to three separate scenarios that we simulated:

- Case 4: resource_4 belongs to role 0 and is available all working day (Mon to Fry) from 8:30 to 12:30
- Case 5: resource_5 belongs to role 3, role 4 and role 5. It is available all months from 6:30 to 12:30 all working day (Mon to Fry), but on March, April and May its availability on Tuesday becomes from 14:30 to 20:00.
- Case 6: resource_6 belongs to role 0 and role 2. It is available all working days (Mon to Fry), from 8:30 to 14:30 on every month except on June and July when it is available from 11:15 to 17:00.

	Calendar Expression	Direct Sampling	Cluster-Based Sampling	Resource's Work-shifts Discovery
Case 4	0.9790	0.9789	0.9791	0.9791
Case 5	0.9127	0.9260	0.9508	0.9508
Case 6	0.9407	0.9168	0.9251	0.9655

Table 10: Results of the application of the methods proposed in the literature and our algorithm for resource's work-shifts discovery on event logs with no noise

5.2.4 Results of Experiment for Resource's Work-shifts Discovery in the Case of Absence of Noise

Table 10 shows again that in the more simplistic case 4 all algorithms have similar performances.
In case 5 both the Calendar Expression discovery 3.1 and Direct Sampling 3.3.1 have a slightly smaller average accuracy given from the edge case of availability on Tuesday of resource_5.
In case 6 we can observe a drop in accuracy even for the Cluster-Based Sampling 3.3.2, this is because the availability intervals for different months overlap and tricks the algorithm to create medoids of active periods from 8:30 to 17:00, while the algorithm for Calendar Expression discovery 3.1 has enough dates with different intervals to sample for time-points and can create calendar expressions that cover both intervals.

5.3 Experiments with Synthetic Event Log Containing Noise

We then tested how our algorithm could perform against the currently available algorithms in the case of event logs with noise, to create this type of event log we developed a new BPNM, shown in Figure 6, capable of generating new scenarios with noise.
We were also able to regulate the quantity of noise simulated by adjusting the probability for each path in the exclusive gateways.
The simulation model had the following roles:

- Role 0: "*Activity_00*", "*Activity_01*", "*Noise_00*", "*Noise_01*"
- Role 1: "*Activity_10*", "*Activity_11*", "*Noise_10*", "*Noise_11*"
- Role 2: "*Activity_20*", "*Noise_20*"
- Role 3: "*Activity_30*", "*Noise_30*"
- Role 4: "*Activity_40*", "*Noise_40*"
- Role 5: "*Activity_50*", "*Noise_50*"

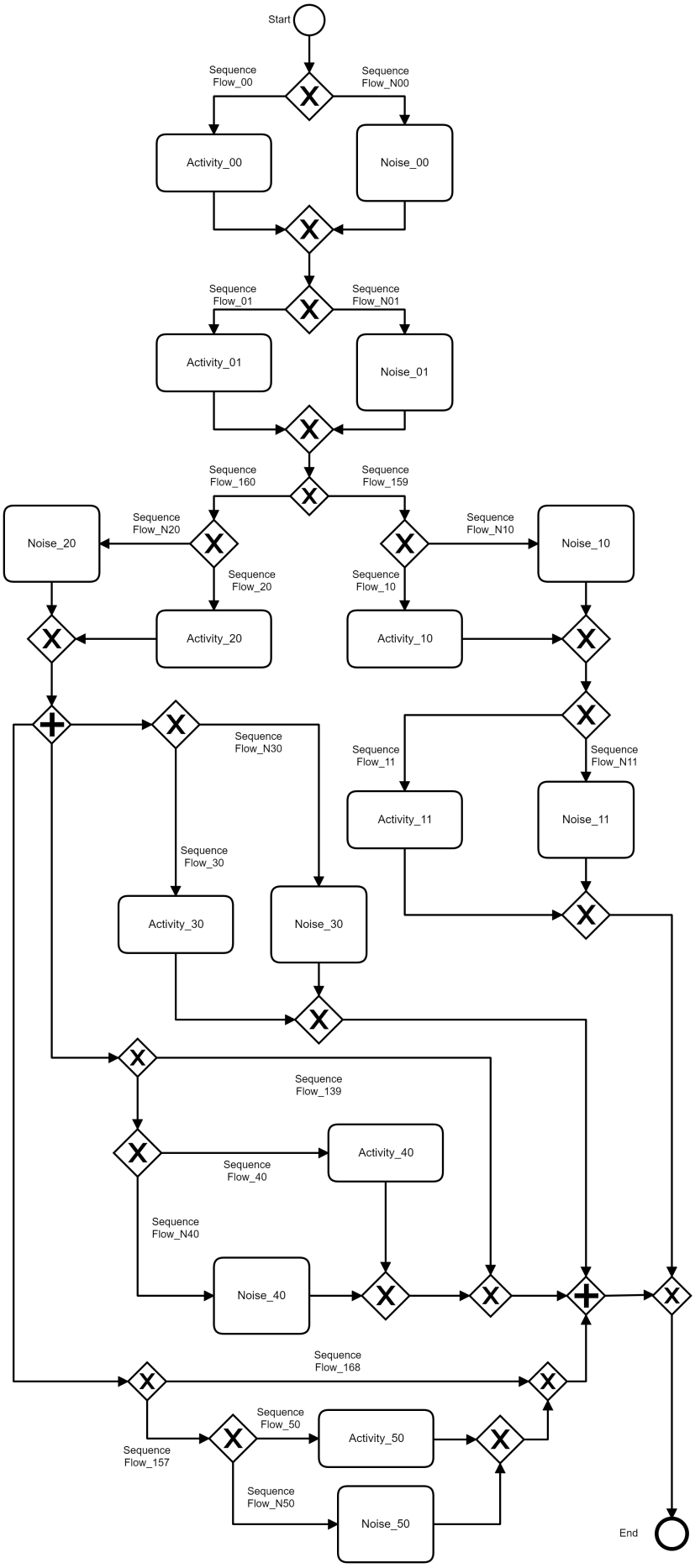

Figure 6: BPMN diagram used to generate event log with noise

	Calendar Expression	Direct Sampling	Cluster-Based Sampling	Role's Work-shifts Discovery
Case 7	0.9449	0.8407	0.9538	0.9507
Case 8	0.9102	0.8620	0.9351	0.9858
Case 9	0.9002	0.7089	0.9561	0.9031

Table 11: Results of the application of the methods proposed in the literature and our algorithm for role's work-shifts discovery on event logs with noise

```
Monday    :    08:04  -  12:31     12:29  -  14:28     14:02  -  15:56
Tuesday   :    08:00  -  12:27     12:31  -  14:28     14:01  -  15:58
Wednesday:     08:02  -  12:31     12:29  -  14:30     14:03  -  15:57
Thursday  :    08:00  -  12:30     12:29  -  14:30     14:02  -  15:58
Friday    :    08:00  -  12:30     12:29  -  14:30     14:03  -  15:57
Saturday  :
Sunday    :
```

Figure 7: Case 8, role 2's work-shifts discovered by our Algorithm 3

5.3.1 Experiment for Role's Work-shifts Discovery in the Case of Presence of Noise

For this type of experiment, it was necessary to have the resources' activities coinciding with a single role. Each role was simulated using 5 resources for a total of 30 resources.
Here follows the analysis of three cases:

- Case 7: availability of role 0, all working days (Mon to Fry) from 8:30 to 12:30, with noise from 7:15 to 7:30 and from 15:00 to 15:30
- Case 8: availability of role 2, all working days (Mon to Fry)
  with 3 resources working 8:00 to 12:30 all months,
  one resource working 12:30 to 14:30 on June
  and one resource working from 14:00 to 16:00 on July.
  Noise present from 6:00 to 7:00 and from 18:00 to 19:00
- Case 9: availability of role 1, all working days (Mon to Fry) from 13:45 to 23:15, with noise from 0:00 to 14:00 and from 23:00 to 24:00

5.3.2 Results of Experiment for Role's Work-shifts Discovery in the Case of Presence of Noise

From Table 11 in case 7 we can observe how almost all algorithms are consistent with their noise detection, only Direct Sampling 3.3.1 suffers from the edge cases e.g. dates with only noisy activity registered that, when sampled by the algorithm, greatly impact the overall average accuracy.
In case 8 we purposely limit the dates on which two of the five resources are available and as we expect on all algorithms except ours those resources are considered noise.

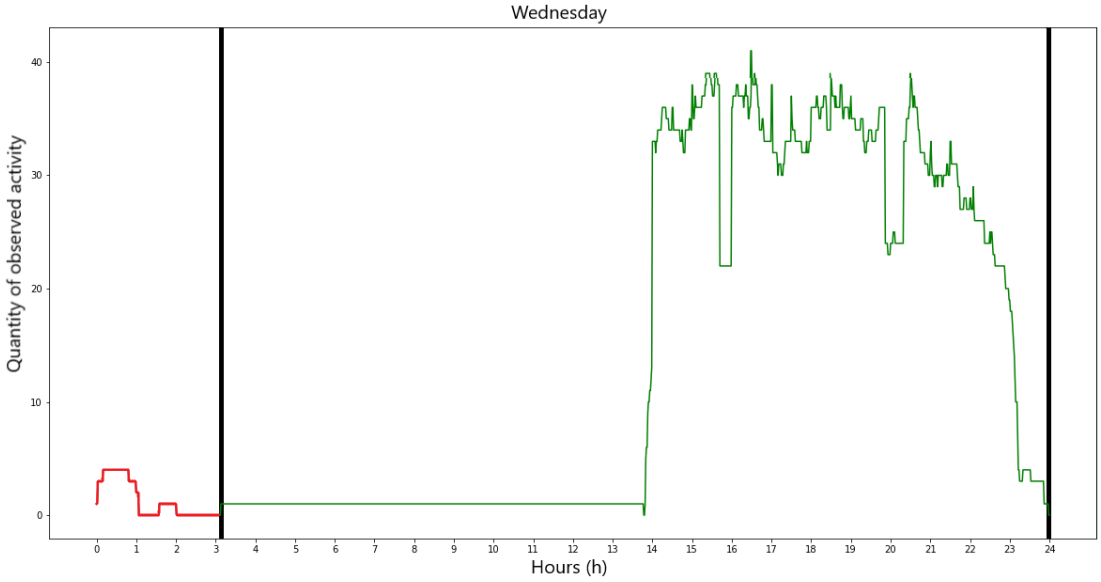

Figure 8: Case 9, role 1's calendar for Wednesday

In particular: the Calendar Expression discovery 3.1 has not enough support for the calendar expression of those two resources as opposed to the other three; Direct Sampling 3.3.1 has only $\frac{1}{12}$ of the possibilities to choose an active period that covers 4 resources, but no chances to choose one that covers all five resources; Cluster-Based Sampling 3.3.2 can not create medoids large enough to cover all three cases and also exclude the boundary pending cases that represent noise i.e. the intervals from 6:00 to 7:00.
In case 9 our algorithm performs worse than Cluster-Based Sampling 3.3.2 and is similar to the Calendar Expression discovery 3.1, this is because the intervals of simulated noise intersect with intervals of real work in the segments located at 13:45 to 14:00 and from 23:00 to 23:15, so as shown on Figure 8 the noise is not removed and influences the resulting work-shifts. While the other algorithms perform similarly to case 7.

5.3.3 Experiment for Resource's Work-shifts Discovery in the Case of Presence of Noise

In this case, we analyze two different resources belonging to two separate scenarios that we simulated:

- Case 10: resource_10 belongs to role 1 and is available all working day (Mon to Fry) from 8:30 to 12:30
  with noise from 4:00 to 6:00, from 14:00 to 14:30, form 15:00 to 15:30, from 17:00 to 18:00 and from 22:00 to 23:30.

- Case 11: resource_11 is available available all working day (Mon to Fry) from 6:30 to 12:30 on odd months for the roles 3, 4 and 5, while on even months it is available from 14:30 to 20:00 for role 0.
  We simulated noise on odd months from 15:00 to 17:00 and on even months from 8:30 to 10:30.

	Calendar Expression	Direct Sampling	Cluster-Based Sampling	Resource's Work-shifts Discovery
Case 10	0.9648	0.8709	0.9747	0.9760
Case 11	0.7591	0.7323	0.7489	0.9726

Table 12: Results of the application of the methods proposed in the literature and our algorithm for resource's work-shifts discovery on event logs with noise

5.3.4 Results of Experiment for Resource's Work-shifts Discovery in the Case of Presence of Noise

Table 12 shows in case 10 that almost all algorithms have similar performances and can remove all instances of noise. the algorithm for Direct Sampling 3.3.1 again suffers from the randomness of its implementation and in cases in which it chooses dates with only noise the average score decreases.
In case 11 our method greatly outperforms the other implementations. This is due to its ability to filter out cases based on the calendar associated with its activity's role. While the other algorithms are not able to differentiate between real activity and noise if they are registered on the same date.

6 Conclusions

This thesis aimed to develop and evaluate a method to automatically discover role's working calendars for simulation models from an event log.
These availability calendars express timeframes during which a resource executing an activity belonging to that specific role can be allocated as available in the model.

At first, we tried to mirror the methods proposed in the literature, and we developed our Algorithms 1 and 2 for the discovery of resource's working calendars, like the methods proposed by Estrada-Torre et Al. in Section 3.1 and by Martin et Al. in Sections 3.3.1 and 3.3.2.
But in our first experiments we noted that when we apply our methods to the distribution of work of a single resource, they usually produce skewed distributions (multiple small intervals of less than 1 hour) that result in calendars that do not represent or offer a good enough generalization of the real availability of the resource. We also observed, as stated in Section 5.1, that it is not unusual to have a real-life event log that only contains the activity and the timestamps while the information regarding the resource is missing.

For those reasons we decided to develop our Algorithms 1 and 2 for the discovery of role's working calendars in the case of event log with noise, and then use the resulting availability constraints in our approaches for the retrieval of work-shifts calendars for roles and resources.

The methods that we have defined provide data-driven support to gather insights into resource and role availability for a particular process. These availability constraints can then be used in the construction of simulation models for the discovery of issues related to the availability of resources playing different roles.
To outline constraints in staff scheduling problems, our methods for work-shifts discovery enable the detection of different rules affecting a resource's availability patterns, e.g. a resource working part-time on call may have different availability timeframes from week to week. The current solutions proposed in the literature for this particular problem either are a simple listing of availability for all days in the event log [7] offering no generalization, or similar to our methods trying to find general patterns by aggregating similar timeframes [6], but they rely too heavily upon a random choice, both when choosing which timeframes to aggregate and also when sampling the intervals to define the resulting work-shift.

A simple example that outlines the shortcoming of those methods is the task of discovering the availability of a part-time resource for two weeks, in the first week they work only in the morning, while in the second week they are available only in the afternoon.
Both approaches: Direct Sampling 3.3.1 and Cluster-Based Sampling 3.3.2, can randomly produce the following combination of availability:

- resource available two weeks in the morning;
- resource available two weeks in the afternoon;
- resource available in the afternoon during the first week and in the morning the second week;
- resource available in the morning for the first week and in the afternoon for the second week.

As we can see, this leads to a chance of $\frac{1}{4}$ to define the exact work-shifts and $\frac{3}{4}$ of defining the wrong work-shifts. While as we have shown our approach can define at once the correct availability.

6.1 Future Works

We observed that in certain cases our method of calendar discovery could suffer from noisy intervals that intersect real work's intervals. In a real case scenario, in which we do not know the real distribution and allocation of work for resources and roles, we should not be heavily penalized if we confuse those intervals of noise for real work.
Nonetheless, if we observe the case of Figure 8 in which the noisy interval (from 3:00 to 14:00) intersects with the real work interval (from 14:00 to 23:00) we can understand how this heavily affects the resulting calendar (interval from 3:00 to 24:00).
The first naive approach that we could think of may be to subtract one or more units from all the distribution until we reduce to zero the intervals of noise. This approach could also lead to changes in the boundaries of the real work intervals that we want to instead maintain, so it is necessary to further study and develop this or other possible approaches that can tackle this particular edge case that our method can not solve at the moment.

References

[1] Wil van der Aalst. "Data Science in Action". In: *Process Mining: Data Science in Action*. Springer Berlin Heidelberg, 2016. ISBN: 978-3-662-49851-4. DOI: 10.1007/978-3-662-49851-4_1. URL: https://doi.org/10.1007/978-3-662-49851-4_1.

[2] Workflow Management Coalition. *BPSim Business Process Simulation Specification*. URL: https://www.bpsim.org/specifications/2.0/WFMC-BPSWG-2016-01.pdf.

[3] Bernard Desruisseaux. *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. 2009. DOI: 10.17487/RFC5545. URL: https://www.rfc-editor.org/info/rfc5545.

[4] Marlon Dumas et al. *Fundamentals of Business Process Management*. Springer. ISBN: 9783662565094.

[5] Bedilia Estrada-Torres et al. "Discovering business process simulation models in the presence of multitasking and availability constraints". In: (2021). ISSN: 0169-023X. DOI: https://doi.org/10.1016/j.datak.2021.101897. URL: https://www.sciencedirect.com/science/article/pii/S0169023X21000240.

[6] Niels Martin et al. "Retrieving Resource Availability Insights from Event Logs". In: (2016). DOI: https://ieeexplore.ieee.org/document/7579385.

[7] Niels Martin et al. "Retrieving the resource availability calendars of a process from an event log". In: (2020). ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2019.101463. URL: https://www.sciencedirect.com/science/article/pii/S0306437919305150.

[8] OMG organization. *ABOUT THE BUSINESS PROCESS MODEL AND NOTATION SPECIFICATION VERSION 2.0.2*. URL: https://www.omg.org/spec/BPMN/2.0.2/.