DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
32

MANUEL CAMARGO

Automated Discovery of Business Process Simulation Models From Event Logs: A Hybrid Process Mining and Deep Learning Approach

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

32

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

32

MANUEL CAMARGO

Automated Discovery of Business Process Simulation Models From Event Logs: A Hybrid Process Mining and Deep Learning Approach

UNIVERSITY OF TARTU
Press

1632

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in Computer Science on December 02, 2021 by the Council of the Institute of Computer Science, University of Tartu.

Supervisor

Prof.	Marlon Dumas University of Tartu Tartu, Estonia
Assoc. Prof.	Oscar González-Rojas Universidad de los Andes Bogotá, Colombia

Opponents

Prof.	Jorge Muñoz-Gama Pontificia Universidad Católica de Chile Santiago de Chile, Chile
Assoc. Prof.	Gabriel Pedraza Ferreira Universidad Industrial de Santander Bucaramanga, Colombia

The public defense will take place on January 11, 2022 at 16:15 in Online.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

ISSN 2613-5906
ISBN 978-9949-03-776-6 (print)
ISBN 978-9949-03-777-3 (PDF)

University of Tartu Press
`http://www.tyk.ee/`

To my family and friends

ABSTRACT

Modern organizations need to constantly adjust their business processes in order to adapt to internal and external changes, such as new competitors, new regulations, changes in customer expectations, or changes in strategic objectives. For example, due to a pandemic, a retailer might experience a 50% increase in their number of online orders while, during the same time, their volume of in-store purchases declines by 30%. To adjust to these changes, the managers may decide to re-deploy employees from the retail stores to the warehouses of the company and the company's online customer service department. To inform their decisions, the managers need to have an accurate estimate of the impact of the above changes on the delivery and customer service response times.

A common approach to make such estimates is to use Business Process Simulation (BPS). BPS refers to the use of computers to explore the dynamics of a business process over time. BPS has long proven to be a useful approach to answer what-if questions in the context of business process redesign. At the same time, the predictions made by BPS models are known to be relatively inaccurate due to the way they are usually applied.

Traditionally, domain experts create simulation models manually by using manual data gathering techniques (e.g. interviews, observations, and sampling). This approach makes the creation of simulation models time-consuming and error-prone. In real-life, business processes tend to be more complex than what domain experts can capture in a manually designed simulation model. Yet, any omission in the simulation model can significantly affect the accuracy and reliability of a simulation. Other limitations of current BPS approaches arise from fundamental assumptions that business process simulation engines make. For example, business process simulation engines assume that human workers work in a robotic (or factory-line) style– meaning that they conduct their work continuously during working hours, without any distractions, without multitasking, and without fatigue. In other words, current business process simulation approaches are not able to capture and reproduce the complexity of human behavior.

In this context, this thesis investigates the following overarching question: How to automatically create accurate business process simulation models based on data extracted from enterprise information systems? Previous research on this question has demonstrated the viability of using a family of techniques for the analysis of business process execution data, known as process mining, to semi-automatically extract BPS models from execution data. Such techniques are fall under the banner of Data-Driven Simulation (DDS). This thesis starts by noting that existing techniques in the field of DDS require manual intervention and fine-tuning to produce accurate simulation models. To address this gap, the thesis presents and evaluates a fully automated technique for DDS capable of discovering and fine-tuning BPS models through process mining techniques. The core idea of the technique is to assess the accuracy of a BPS model automatically us-

ing a similarity measure that considers both the ordering of activities and their execution times. On this basis, the proposed technique employs a Bayesian optimization algorithm to maximize the similarity between the behavior generated by the BPS model and the behavior observed in the execution data.

The thesis, thus, shows that the proposed DDS technique generates models that accurately reflect the ordering of activities. However, the proposed technique often falls short when it comes to predicting the timing of each activity. This phenomenon is due to the assumptions that BPS techniques make about the behavior of resources in the process. To tackle this shortcoming, the thesis combines DDS techniques based on process mining, with generative modeling techniques based on deep learning. In this respect, the thesis makes two contributions. First, it proposes an approach to learn generative deep learning models that are able to produce timestamped sequences of activities (with associated resources) based on historical execution data. Second, it proposes an approach to combine DDS techniques based on process mining, with generative deep learning modeling techniques. The thesis shows that this hybrid approach to learn BPS models leads to simulations that more closely reflect the observed sequences of activities and their timings compared to a DDS technique based purely on process mining and techniques based purely on deep learning.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Acronyms

BPMN	Business Process Model and Notation. 24, 25, 45, 48, 49, 51, 54, 74, 75, 88, 90, 93
BPS	Business Process Simulation. 16–20, 24–26, 35–38, 41, 42, 44, 48, 50–52, 54, 56–59, 61, 74, 76, 94, 106–109, 111
BPTD	Business Process Trace Distance. 53, 54
CFLS	Control-Flow Log Similarity. 76, 83, 84, 86, 93, 100
CNN	Convolutional Neural Network. 28, 29, 38, 40–42
CPN	Colored Petri Nets. 36
DDS	Data-driven Simulation. 18, 20, 21, 35, 37, 38, 42, 74, 75, 78, 80, 81, 84, 85, 87, 88, 90–92, 94, 98, 100, 106–110
DES	Discrete Event Simulation. 24, 51, 107
DFFN	Deep Feed-Forward network. 27–29, 38, 40
DL	Deep Learning. 18–21, 26–28, 31, 32, 34, 35, 38, 41, 42, 62, 74, 75, 77–79, 81, 84, 86, 88, 90–92, 98, 100, 102, 106, 108, 109
DLd	Damerau-Levenshtein distance. 46, 52, 53, 68–71, 83
DTW	Dynamic Time Warping. 103, 104
ELS	Event Log Similarity. 54, 55, 58, 59, 83, 84, 86
EMD	Earth Mover’s Distance. 76, 83, 84, 86, 88, 98–104
GAN	Generative Adversarial Network. 34, 40–42, 77, 79, 80, 85, 86, 89, 100–103
LSTM	Long Short-Term Memory network. 29, 30, 38–42, 62, 63, 65, 66, 68, 69, 73, 74, 77–80, 85–87, 89, 95, 97, 100–103, 108, 110
MAE	Mean Absolute Error. 52, 69–71, 73, 83, 84, 86, 88, 89, 98, 99, 101–106
PDF	Probability Distribution Function. 25, 26, 49–51, 58, 76
PM	Process Mining. 18, 23, 27, 35–38, 44, 45, 54, 77, 92, 106, 109
PPM	Predictive Process Mining. 27–31, 33
RNN	Recurrent Neural Network. 28, 29, 38–40, 43
RO	Resources’ Occupation. 95–97
TPE	Tree-structured Parzen Estimator. 55, 58
WIP	Work-in-progress. 95, 96

Event logs Acronyms

ACR	Academic Credentials Recognition. 57, 59, 81, 82, 89, 99, 103
BPIC	Business Process Intelligence Challenge. 63–65, 67–73, 81, 82, 89, 98, 99, 103, 104
CALL	Call center records of a helpdesk process. 82, 86, 89
CFL	Confidential Large-size. 82, 89, 99, 103, 105, 106
CFS	Confidential Small-size. 82, 89, 98, 99, 103
CVS	CVS retail pharmacy. 82, 89, 99, 103–106
INS	Insurance claims. 82, 89, 99, 103
MP	Manufacturing Production. 57, 59, 81, 82, 89, 98, 99, 103
P2P	Purchase-to-Pay. 44, 57, 59, 81, 82, 85, 89, 98, 99, 103
UB	Undisclosed Banking. 82, 89, 98, 99, 103

1. INTRODUCTION

Organizations are subject to constant change motivated by an innumerable set of internal and external factors, such as downward price pressure, changes in regulations, or new technological opportunities. These changes drive organizations to constantly adapt and update their business processes. For example, the possible automation of tasks will imply the reorganization of resources, or for instance, the addition of a quality assurance stage to comply with a new regulation will require reorganizing the process's sequence flow. Changes like these require resolving a fundamental question: How to assess the impact of a possible change on a process regarding temporal and cost measures?

Traditionally, these decisions are made based on the intuition and expertise of managers and business analysts, which works well if the number of possible changes is small or if the impact of the change is not critical. However, the number of possible changes frequently grows exponentially, and calculating the impact of potential changes is difficult to predict. In these situations, analysts need a tool that allows them to automatically explore hundreds or thousands of hypothetical changes in a short time, thus simplifying the decision-making process. This type of analysis is known as what-if analysis. In this context, process analysts answer this question using a tool called Business Process Simulation (BPS).

BPS is an approach for the quantitative analysis of business processes [25], which are commonly used when deciding how to improve a business process concerning one or more cost and time-related performance measures [91]. The main idea of BPS is to use a computational simulation model that captures essential business process characteristics. With the help of such a simulation model, it is possible to make all kinds of hypothetical changes without taking risks in real life. The analyst can, for example, add two additional resources to a process and compare if it is better than adding only one additional resource or, for example, an analyst can suggest parallelizing multiple activities that are currently performed sequentially and compare cycle times versus current times. In this way, the BPS becomes a kind of super-intuition that supports decision-making related to changes in the processes.

1.1. Problem statement

Despite its great potential, the applicability of BPS is hindered by various limitations. Some of these limitations are methodological, that is, they arise from the methods that are currently used to create simulation models. A key ingredient for BPS is a simulation model (herein, a BPS model) that accurately reflects the actual dynamics of the process. An accurate initial simulation model provides a reliable basis for evaluating and drawing conclusions about implementing possible changes in real life. However, domain experts have traditionally created BPS models using manual data gathering techniques, such as interviews, contextual inquiries, and on-site observation. This approach is time-consuming and error-prone as it requires careful attention to numerous pitfalls [61, 1].

Another design problem is the low precision of manually constructed simulation models. A BPS model comprises a process model enriched with simulation parameters that allow its reproduction over time. Therefore, the accuracy of a BPS model depends mainly on how faithfully these components capture the observed reality. However, process models produced by domain experts often do not capture all possible execution paths (e.g., exceptional paths are left aside). Indeed, since business process models are often designed for documentation and communication purposes, they need to balance completeness and understandability. 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 [76].

Furthermore, even if we put aside the problems of creating simulation models, current BPS techniques would still present a poor representation of the temporal dynamics of business processes. These are due to the difficulty that BPS simulators have in reflecting the complexities of human behavior [1, 3]. Among the limitations that simulation techniques suffer, we can find the following:

- Waiting times are exclusively due to resource contention (i.e., a resource cannot start a task because it is busy with another task).
- They assume that resources exhibit robotic behavior. If a resource is available and may perform an enabled activity instance, the resource will immediately start it (eager resources consume work items in FIFO mode).
- They assume that a resource only works on one task instance simultaneously; this means that a task is strictly performed by one resource.
- Similarly, there is no time-sharing outside the simulated process; in other words, these techniques 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 [64].
- They assume no fatigue, no interruptions, and no distractions beyond "stochastic" ones.
- Oftentimes, they do not consider differences between the performance of resources, meaning that every resource in a pool has the same performance as others.

Considering the complexity of the BPS limitations described above, in this thesis, we will focus on building more accurate BPS models. For this purpose, we will assume that a simulation model must accurately reproduce the current state (AS-IS) of a business process before assessing the impact of possible changes (WHAT-IF scenarios). Similarly, we hypothesize that using data and machine learning models for this purpose might help find answers to some problems of the current simulation technique. Specifically, this thesis addresses two research questions:

RQ1 How to automatically create accurate business process simulation models based on data extracted from enterprise information systems?

RQ2 How to create simulation techniques that more accurately capture the observed temporal dynamics of business processes?

1.2. Previous work and research gaps

Several research groups have studied and partially addressed the problem raised in RQ1. In the literature, the use of data has mainly been explored in conjunction with Process Mining (PM) tools to create more reliable BPS models [54]. One could consider a PM as the family of machine learning techniques that allows users to analyze data extracted from business information systems (i.e., an event log) in order to obtain information that improves business processes [4]. The discovery of simulation models that use PM tools is known as Data-driven Simulation (DDS).

Proposed DDS techniques for discovering simulation models range from semi-automated [40] to automated [74, 68]. In all of them, a DDS model is built by first discovering a process model from an event log and then adjusting many parameters (e.g., arrival rate, processing times, and conditional branching probabilities). DDS approaches significantly decrease the risks of introducing biases due to the intervention of the modeler in the model creation phase. However, the existing proposals have shortcomings not yet addressed, leaving the research question open:

GAP1 Existing studies have not extensible explored the question of measuring the accuracy of BPS models derived from data; and,

GAP2 The fine-tuning of simulation parameters is left to the modeler, thus leaving the door open for the introduction of biases during the creation of BPS models.

Moreover, none of the previous works directly addresses RQ2, as this question implies a more profound intervention in the simulation technique. Face this question forced us to take a step back to have a broader perspective on the panorama of possible solutions to the problems of representation of the temporal dynamics of the current simulation technique. In a more generic way and without delving into too much detail, a BPS model is a generative model of business processes. A generative model of business processes is a statistical model constructed from an event log that can generate traces that resemble those observed in the log and other traces of the process. Generative process models have several applications in PM, including anomaly detection [63], what-if scenario analysis [16], conformance checking [77], and predictive monitoring [84].

Especially in predictive monitoring, the use of Deep Learning (DL) generative models has been explored with great success. DL generative models are machine learning models consisting of interconnected layers of artificial neurons adjusted based on input-output pairs to maximize accuracy. In predictive monitoring, the use of these models has been due to the increase in the computational capacity and the refinement of techniques for processing large volumes of data; the applications of DL techniques in the resolution of prediction, classification, and generation problems have been experiencing constant growth [44]. DL techniques have demonstrated their great potential in the resolution of complex problems in diverse areas such as healthcare, autonomous driving, image and speech recognition, and natural language understanding[33].

Starting from an ongoing case (prefix), these models have been used to predict the next event category and its timestamp, or the most remaining likely sequence of events (suffix) [27, 57, 84]. Moreover, suitably trained DL generative models can also be used to generate entire traces and even entire logs (not just suffixes) [18]. The ability of these models to learn the non-linear relationships between multiple variables accurately forces us to ask ourselves if it is possible to use DL generative models as simulation models, and what their relative precision would be.

Hypothetically, the approach would avoid the difficulties presented in the current simulation technique by capturing the complexities of the human component involved in business processes typically overlooked by traditional techniques (e.g., multitasking, batching). However, before these techniques can be used in BPS, there must be a solution to three important problems:

GAP3 DL generative models must be able to generate not only remaining sequences of events (suffixes) but also complete logs starting from scratch (prefixes of size zero).

GAP4 Similarly, the generated logs must include the category of the event, associated resource, and start and end times of the activities, thus allowing an evaluation of the performance of the scenarios.

GAP5 Furthermore, the deep learning techniques must be able to perform what-if analysis, one of the BPS's key features.

The challenge in this sense resides in the black box structure of neural networks. This structure prevents the direct introduction of changes in the process that is being evaluated. In this thesis, we evaluate approaches to address these gaps, to enable the use of DL techniques for process simulation.

1.3. Research Methodology

In this thesis, we follow the design science research methodology [35], which focuses on the continuous construction and evaluation of artifacts in order to improve their performance. Following this methodology, we identified the problems of the current process simulation technique, defined two research questions w.r.t., RQ1 and RQ2, and identified the gaps in the state-of-the-art related to the solution of the questions. This process allowed us to define a set of requirements for constructing and evaluating two initial artifacts. The first artifact, related to RQ1, is a DDS approach that automatically discovers, evaluates, and performs the tuning of process simulation model parameters.

The second artifact we build is aimed at finding a solution to the RQ2. In a preparatory way, we proposed a deep-learning generative model training technique that allows the generation of complete event logs from input data. Subsequently, as a benchmark, we exhaustively compared the relative precision of both artifacts, which allowed us to define a new set of requirements for the construction of a third artifact. The third artifact proposes a hybrid simulation technique between DDS and DL to provide a solution to RQ2.

1.4. Contributions

Contribution 1: *We propose an automated DDS method to discover an accuracy-optimized BPS model from an event log (Chapter 4).* To solve the GAP1, the proposed method asses the accuracy of the simulation model using a similarity measure that considers both the ordering of activities and their execution times. Likewise, in response to the GAP2, the approach optimizes the generated model applying a Bayesian hyperparameter optimization algorithm able to maximize the similarity between the behavior generated by the BPS model and the behavior observed in the log. The method has been implemented as a tool (namely Simod) that generates process simulation models from event logs. The magnitude of the accuracy enhancements achieved by the proposed method has been evaluated via experiments on one synthetic and two real-life event logs from different domains.

Contribution 2: *We propose new pre- and post-processing methods and architectures for building and using generative models from event logs using LSTM neural networks (Chapter 5).* In response to GAP3 and GAP4, this thesis proposes an approach to learn generative DL models able to generate complete traces (or suffixes of traces starting from a given prefix) consisting of triplets (event type, role, and timestamp). This approach allowed us to obtain a generative deep learning approach comparable to state-of-the-art techniques, but potentially applicable to process simulation. The relative precision of the technique in terms of similarity of the order of activities and their execution times was tested through experiments on nine real-life event logs from different domains.

Contribution 3: In preparation to fill the GAP5, *we compared the relative precision and characteristics of the DDS versus DL generative models under the same conditions*. This work allowed us to define the requirements to create a hybrid approach capable of taking advantage of the strengths of both families of generative models (Chapter 6). We evaluate the generative approaches using eleven real and synthetic event logs, which vary in structural and temporal characteristics.

Contribution 4: Finally, in response to GAP5, *we present a hybrid approach* *to learn process simulation models from event logs wherein a (stochastic) process model is extracted by using DDS techniques and then combined with a DL model to generate timestamped event sequences (traces) (Chapter 7).* An experimental evaluation using synthetic and real event logs against an existing DDS method and two DL methods shows that the resulting hybrid simulation models match the temporal accuracy of pure DL models while retaining the what-if analysis capability of DDS approaches.

The above contributions have been previously documented in publications I-IV, as referenced at the end of the thesis (see "List of original publications").

1.5. Outline

The rest of the thesis is structured as follows. In Chapter 2, we introduce the relevant concepts from business process simulation and deep learning. Chapter 3 presents the systematic literature review and taxonomy for existing DDS and DL methods. In Chapter 4, we develop the Simod tool as representative of the DDS approaches; we also perform an experimental evaluation of the tool. Chapter 5 presents and evaluates new pre- and post-processing methods and architectures for learning accurate deep learning models of business processes. Chapter 6 empirically compares the accuracy of DDS approaches versus DL generative models of business processes and discusses their relative strengths and weaknesses. Chapter 7 proposes and evaluates a new hybrid simulation technique that combines the strengths of DDS and DL generative models. Chapter 8 concludes the thesis and outlines directions for future work. Table 3 summarizes the research questions, the research gaps associated with each research question, the contributions addressing those research gaps, the chapters where these contributions are unfolded, and the publications related to each contribution.

Research Question	Research Gaps	Contributions
RQ1 - How to automatically create accurate business process simulation models based on data extracted from enterprise information systems?	**GAP1** - Existing studies have not extensible explored the question of measuring the accuracy of BPS models derived from data. **GAP2** - The fine-tuning of simulation parameters is left to the modeler, thus leaving the door open for the introduction of biases during the creation of BPS models.	**#1** - Simod: Automated Discovery of Business Process Simulation Models from Event Logs (see Chapter 4) **Publications**: - (2020) Decision Support Systems [16]. - (2019) BPM19 Demo paper [19].
RQ2 - How to create simulation techniques that more accurately capture the observed temporal dynamics of business processes?	**GAP3** - DL generative models must be able to generate not only complete sequences of events, but also complete logs starting from scratch. **GAP4** - The generated logs must include the category of the event, associated resource, and start and end times of the activities, thus allowing an evaluation of the performance of the scenarios.	**#2** - Learning accurate generative models of business processes (see Chapter 5) **Publications**: - (2019) BPM19 Conference paper [18].
	GAP5 - The deep learning techniques must be able to perform what-if analysis, one of the BPS's key features.	**#3** - Data-driven Simulation vs Deep Learning (see Chapter 6) **Publications**: - (2021) PeerJ Computer Science [17].
		#4 - Hybrid learning of Business Process Simulation Models (see Chapter 7) **Unpublished**

Table 3: The first column presents the research questions, the second the gaps associated with the questions, and the third the thesis contributions and publications.

2. BACKGROUND

This section presents background concepts used throughout the thesis.

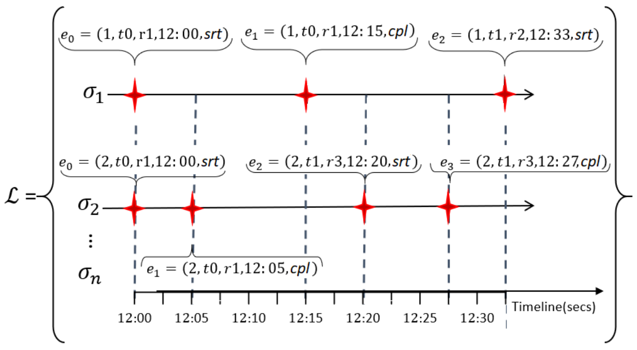
2.1. Event log definition

For their analyses, PM techniques use the event logs generated by the process-aware information systems that support the operation. In practice, the event logs can be very different from each other. However, all event logs contain the records of event occurrences at specific moments in time– where each event refers to a particular process instance. Table 4 presents a basic example of an event log for a production process.

Case ID	Activity	Resource	Timestamp	Transition	Work Order Qty	Part Desc.	Report Type	Qty Completed	Qty Rejected	Qty for MRB
Case 1	Turning & Milling	ID4932	01/29/2012 23:24:00	start	10	Cable Head	S	1	0	0
	Turning & Milling	ID4932	01/30/2012 05:43:00	complete	10	Cable Head	S	1	0	0
	Turning & Milling Q.C.	ID4163	01/31/2012 13:20:00	start	10	Cable Head	D	9	1	0
	Turning & Milling Q.C.	ID4163	01/31/2012 14:50:00	complete	10	Cable Head	D	9	1	0
	Laser Marking	ID0998	02/01/2012 08:18:00	start	10	Cable Head	D	9	0	0
Case 200	Final Inspection Q.C.	ID4618	02/02/2012 12:38:00	start	251	Spur Gear	D	250	1	0
	Final Inspection Q.C.	ID4618	02/02/2012 14:16:00	complete	251	Spur Gear	D	250	1	0
	Packing	ID4820	02/03/2012 00:00:00	start	251	Spur Gear	D	250	0	0
	Packing	ID4820	02/03/2012 01:00:00	complete	251	Spur Gear	D	250	0	0

Table 4: Fragment of an example event log

More formally, an event log $\mathscr{L}$ is a set of traces such as $\mathscr{L} = \{\ \sigma_i \mid \sigma_i \in \mathscr{S}, 1 \leq i \leq \mathscr{K}\ \}$, in which $\mathscr{S}$ represents the set of all process traces, and $\mathscr{K}$ is the number of traces. A trace is a non-empty sequence of events $\sigma = \langle e_1, e_2, \ldots, e_n \rangle$. The set of all possible events is $\mathscr{E}$, and each event e has at least a case identifier i , an activity label l, a resource assigned to execute the task, and t the timestamp in seconds of the task. The timestamp of the event can be associated with a transition in the life cycle of the activity that generates the event y, typically the start or completion of the activity, but other transitions can be included. Additionally, the event can also contain additional data attributes specific to each process $d_{ij} \in \mathscr{D}_j$, $1 \leq j \leq m$, such as $e_i = (i_i, l_i, r_i, t_i, y_i, d_{i1}, \ldots,\ d_{im})$. Fig. 1 illustrates these definitions.



$e_0 = (1, t0, r1, 12{:}00, srt)$
$e_1 = (1, t0, r1, 12{:}15, cpl)$
$e_2 = (1, t1, r2, 12{:}33, srt)$
σ_1
$e_0 = (2, t0, r1, 12{:}00, srt)$
$e_2 = (2, t1, r3, 12{:}20, srt)$
$e_3 = (2, t1, r3, 12{:}27, cpl)$
$\mathcal{L} =$
σ_2
$\vdots$
σ_n
$e_1 = (2, t0, r1, 12{:}05, cpl)$
12:00
12:05
12:10
12:15
12:20
12:25
12:30
Timeline(secs)

Figure 1: Event-log definition

2.2. Business process simulation

A model is a simplification — smaller, less detailed, less complex, or all these together — of some phenomenon, behavior, or system [32]. Science, in general, builds models. These can be mathematical, logical, conceptual, or computational. Simulation, for its part, uses the computer to explore the dynamics of a model over time. In general, it makes sense when looking to represent complex systems. The simulation of simple and complicated systems falls better in the plane of the illustration of phenomena (for educational purposes, for example) or computer graphics. Modeling and simulation are carried out mainly for three purposes:

1. When we seek to understand (and explain) fundamental processes.
2. When we want a phenomenon or system to behave as we wish / would like.
3. When we want to see emergencies, dynamics, processes, elements, and others that we cannot see (= understand) outside of simulation and modeling frequently.

A simulation is a powerful tool and way of thinking that allows us to understand systems and solve problems characterized by high uncertainty, changing environments, symmetry breaks, emergencies, and evolution, among other features. It is also helpful in cases where it is impossible to intervene in the real system due to ethical, economic, temporal, or governmental limitations.

Business processes are ideal for using simulation since all the above conditions are met. BPS is undoubtedly one of the main quantitative process analysis techniques and is used mainly for the evaluation of process design options and possible scenarios (i.e., what-if analysis) [25]. Specifically, in this last application, the objective is deciding how to improve a business process concerning one or more cost and time-related performance measures [91].

Discrete Event Simulation (DES) is the most widely used simulation technique in BPS [1, 25]. DES seeks to represent the changes on a system (in our case, a business process) as a sequence or series of events that occur in discrete moments. In this sequence, an entity is processed and transformed, and these changes are recorded into a simulated log. For example, a purchase order is processed step-by-step until sending the product to the customer in a purchase-to-pay process. The underlying statistical paradigm that supports DES is queuing theory. Traditionally, BPS models are made up of a formal process model enhanced with parameters that enable its execution over time by a DES simulator, as can be seen on Fig. 2.

This thesis considers business process models represented in the Business Process Model and Notation (BPMN). In its basic form, a BPMN process model consists of activity nodes (or *activities* for short) and gateways that are interconnected by sequence flows. A *split gateway* has multiple outgoing sequence flows. An *exclusive decision gateway* is a split gateway that encodes one decision– i.e., when the execution of the process reaches this gateway, only one of its outgoing sequence flows is taken. An *inclusive decision gateway* allows multiple outgoing flows to be taken when the branching conditions are satisfied. Any inclusive deci-

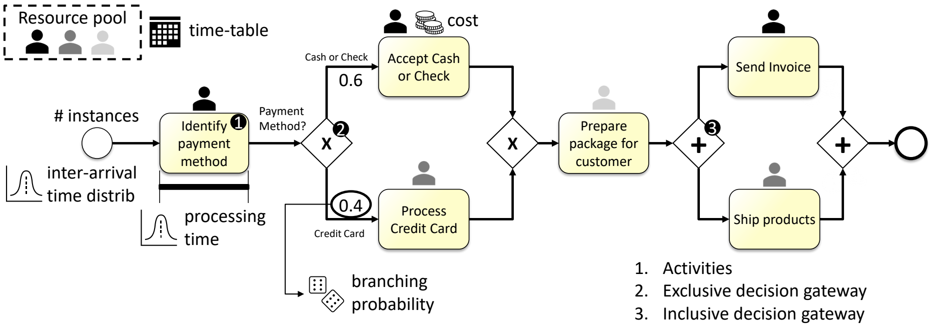


Figure 2: BPS model components example for a purchase-to-pay process

sion gateway can be trivially transformed into a combination of exclusive decision gateways and parallel gateways; hence, we can restrict ourselves to exclusive decision gateways without a loss of generality. The branches coming out of a decision gateway are called *conditional branches*. A BPS model then consists of a BPMN process model plus the following elements [25]:

- The *mean inter-arrival time* of cases and their associated Probability Distribution Function (PDF), e.g., one instance of a case is created every 10 seconds on average with an exponential distribution.
- The PDF of the processing times of each activity. For example, the processing times of an activity may follow a normal distribution with a mean of 20 minutes and a standard deviation of 5 minutes or an exponential distribution with a mean of 10 minutes.
- Optionally, other performance attributes for the task– such as cost and added-value produced by the activity.
- For each conditional branch in the process model, a *branching probability* (i.e., percentage of time the conditional branch in question is taken when the corresponding decision gateway is reached).
- The *resource pool* that is responsible for performing each activity in the process model. For example, in an insurance claims handling process, a possible resource pool would be the *claim handlers*. Each resource pool has a size (e.g., the number of claim handlers or the number of clerks). The instances of a resource pool are the *resources*.
- A *timetable* for each resource pool, indicating the periods during which the resources are available to perform activities (e.g., Monday-Friday from 9:00 to 17:00).
- A function that maps each task in the process model to a resource pool.
- The required number of process instances to be simulated (e.g., 1000).

Once the model is defined, the simulator stochastically creates new process cases according to the inter-arrival time PDF. The simulator constrains the execution of each case according to the control-flow semantics described in the process model and following the next activity execution rules:

i If an activity in a case is enabled, and there is an available resource in the pool associated with this activity, the activity is started and allocated to one of the available resources in the pool;

ii When an activity is completed, the resource allocated to the activity is available again.

Hence, the waiting time of an activity is entirely determined by the availability of a resource. Resources are assumed to be eager– as soon as a resource is assigned to an activity, the activity is started.

The simulation of a BPS model yields a simulation log consisting of events generated during the execution time and a collection of performance metrics. Suppose we created the simulation model to reflect the AS-IS state of the process accurately. In that case, we could use this output to understand internal process dynamics, identify possible improvement opportunities such as bottlenecks, or perform a what-if analysis.

The most used performance metrics in BPS are cycle, processing, and waiting time. The *cycle time* of a process instance (herein called a *case*) is the amount of time between the moment the case starts, and the moment it ends. By extension, we define the cycle time of an instance of an activity as the amount of time between the moment the activity instance is enabled (i.e., ready to be executed) and the moment it completes.

The *processing time* of an activity instance is the time comprehended between the moment a resource start working on it and its completion. Usually, there is a delay between the moment an activity instance is enabled and the moment it starts. This delay is called *waiting time*. We define the processing time of a case as the amount of time when the process instance is *active*, meaning that at least one activity instance of this case has started but not yet been completed. The *waiting time* of a case is the cycle time of the case minus the processing time.

These definitions also apply to a process, which consists of a set of cases. The cycle time of a process is the mean cycle time of its cases. Similarly, the cycle time of an activity is the mean cycle time of its activity instances.

2.3. Deep learning in business processes

DL is a subfield of machine learning concerned with the construction and use of networks composed of multiple interconnected layers of neurons (perceptrons), which perform non-linear transformations of data [34]. These transformations allow the network to learn the behaviors/patterns observed in the data. Theoretically, the more layers of neurons there are in a network, the more it becomes possible to detect higher-level patterns in the data due to the composition of complex functions [44]. DL models have been applied in several subfields of PM, particularly in the context of *Predictive Process Mining (PPM)*. PPM is a class of PM techniques concerned with predicting, at runtime, some properties about the future state of a case, e.g., predicting the next event(s) in an ongoing case or the remaining time until completion of the case.

Fig. 3 illustrates a case in which we have a running case $\sigma = \langle e_1, e_2, e_3 \rangle$ and we want to know its possible future states. For this purpose, we use a generative DL model that has been trained using examples of possible subsequences of execution and expected outcomes. The possible outcomes can be the category of the next event and its respective timestamp (in our example the category and timestamp of e_4), the case's continuation (in our example the full subsequence $\langle e_4, e_5, ..., e_n \rangle$), and the remaining time of the process (in the example the time difference between e_n and e_3). This predictive information can be used for the proactive adaptation of the process. Proactive adaptation can mitigate the impact of execution problems by dynamically re-planning the flow of a running process instance. Proactive process adaptation thereby can avoid contractual penalties or time-consuming roll-back and compensation activities. Below, we describe the most common DL architectures applied to predicting the future of an ongoing case.

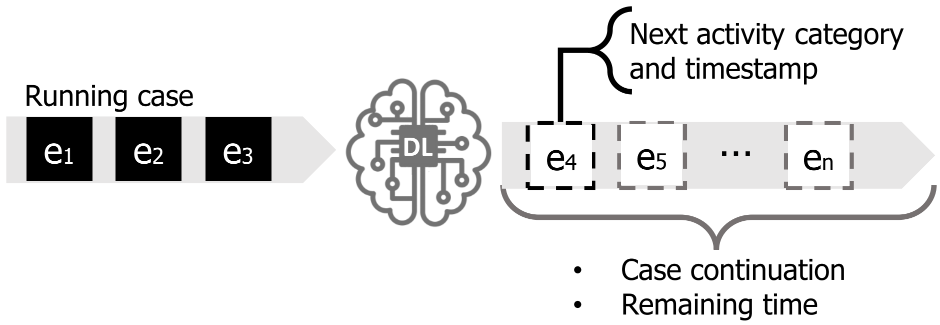


Figure 3: Example of the Deep learning models application in business processes

2.3.1. Deep learning architectures in predictive processes monitoring

Deep Feed-Forward networks (DFFNs). These kinds of networks, also called multi-layer perceptrons, are the most basic DL models. In DFFNs, each neuron of a layer is connected to all the neurons of the next layer, allowing the information to flow without cycles. Commonly these networks are made up of an input layer, followed by intermediate layers of neurons, and an output layer (see Fig. 4). DFFNs uses a back-propagation algorithm to calculate a function that produces the desired output from the input data [33, 78].

DFFNs are well suited for tabular data; however, they are not the most popular kind of architecture in PPM. The scarce application of this type of architecture in

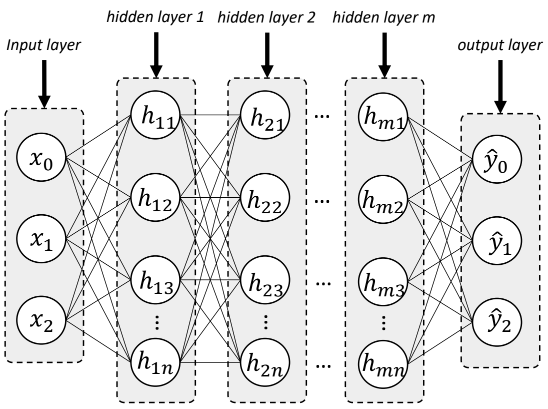


Figure 4: In the context of PPM, the input of the network are prefixes of the traces recorded in the event log, e.g., for the trace $\sigma = <e_1, e_2, e_3, e_4>$ the possible inputs are $(e_1)(e_1, e_2)(e_1, e_2, e_3)$, and the expected outputs $(e_2)(e_3)(e_4)$

PPM is due to the sequential nature of business processes, in which the dependencies between activities play a primary role. Notwithstanding, some authors have applied them in combination with other methods, such as autoencoders [57], or after extracting features from the process model [88].

Convolutional Neural Networks (CNNs). CNNs are specialized DL models for processing grid-like data such as images[45]. CNNs extend DFFNs by introducing three additional concepts: convolutional layers (local filters), pooling layers, and weight sharing. A convolutional layer convolves a kernel with the input to obtain a feature map. Many kernels of different sizes are often applied on the same input, resulting in many feature maps of different sizes. Pooling layers are placed between convolutional layers to reduce the number of parameters to be calculated and speed up the training phase. In the CNNs structure, usually, upper layers use broader filters that work on lower resolution inputs, processing the more complex parts of the input. Finally, fully connected layers combine the inputs from all positions. This hierarchical organization allows CNNs to model local structures in the input using supervised learning algorithms. Fig. 5 shows the general architecture of CNNs.

Although this type of architecture does not specialize in the management of temporal dependencies between observations as observed in event logs, some authors have explored its application in PPM [39, 22]. In fact, CNNs are the second most used technique in these kinds of predictive tasks due to their high performance in terms of training time, as will be discussed in Chapter 3.

RNN and LSTM networks. Recurrent Neural Networks (RNNs) are DL models specialized in processing sequential data in which temporal relations are relevant [78]. RNNs have been applied with great success in problems related to text processing, such as text generation or the prediction of the next word of a sentence. Thanks to the similarities between the structures of event logs and para-

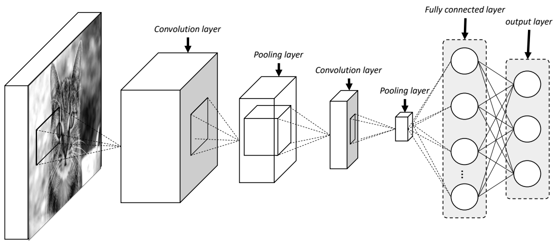


Figure 5: In the same way of DFFNs, the input of the CNNs are prefixes of the traces recorded in the event log in many cases padded in order to create a matricial input, e.g., for the trace $\sigma = <e1, e_2, e_3, e_4>$ the possible inputs are $(e_1,0,0,0),(e_1,e_2,0,0),(e_1,e_2,e_3,0),(e_1,e_2,e_3,e_4)$, and the expected outputs $(e_2),(e_3),(e_4)$

graphs (i.e., traces such as phrases, events such as words), the RNN models are also the most commonly used in PPM[84, 27, 18, 62, 36]. Each RNN neuron, also called a "cell," contains cyclic connections capable of storing representations of events that pass through them. In fact, RNNs can be unfolded, which is equivalent to having multiple copies of the same cell connected sequentially. Each cell takes as input the hidden state — the memory — of the previous cell and the current input to generate a new hidden state in the network. More formally, we can calculate the hidden state of a cell h_t as:

$$h_t = f(Wh_{t-1} + Vx_t + b) \tag{2.1}$$

In this equation, t represents the current moment, and $t-1$ the previous one. The model takes a sequence of the input at the time t x_t and the hidden state of the previous cell h_{t-1}, and computes them using a non-linear activation function (commonly tanh). W and V are matrices of weights, and b the bias vector. Fig. 6 presents the basic RNN cell structure.

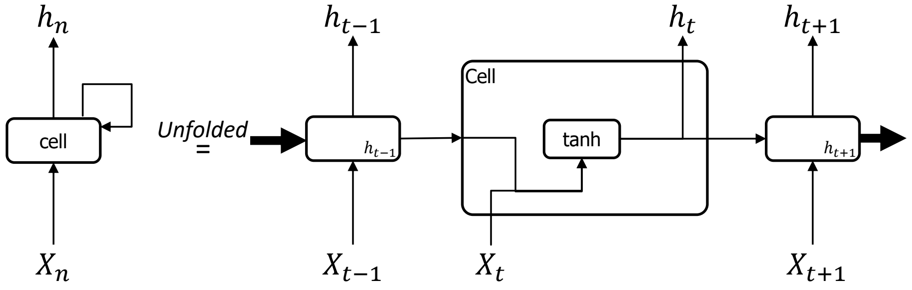


Figure 6: RNN basic structure

Even though RNNs have a good performance when predicting sequences with short-term temporary dependencies, they fail to account for long-term dependencies. Long Short-Term Memory networks (LSTMs) [37] address this problem by using a mechanism of long-term memory called "cell state" c_t. In the long-term memory, the information flows from cell to cell with minimal variation – keeping certain aspects constant during the processing of all inputs – creating coherence in long-term prediction. To decide what information is added or removed, LSTMs using gated structures. The following equations describe the LSTM model:

$$f_t = \sigma(W_f h_{t-1} + V_f x_t + b_f) \tag{2.2a}$$
$$i_t = \sigma(W_i h_{t-1} + V_i x_t + b_i) \tag{2.2b}$$
$$o_t = \sigma(W_o h_{t-1} + V_o x_t + b_o) \tag{2.2c}$$
$$\widetilde{C_t} = tanh(W_c h_{t-1} + V_c x_t + b_c) \tag{2.2d}$$
$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{2.2e}$$
$$h_t = o_t * tanh(C_t) \tag{2.2f}$$

In these equations, f_t is the "forget gate" and filters what information is removed from the cell state c_t; i_t is the "input gate" and controls what information is going to be updated in the cell state. C_t is the combination of the past and current information of the cell. $\widetilde{C_t}$ is the calculation of the cell state for the current time step. Finally, o_t is the "output gate," which computes the new output h_t of the cell. Fig. 7 presents the basic LSTM cell structure.

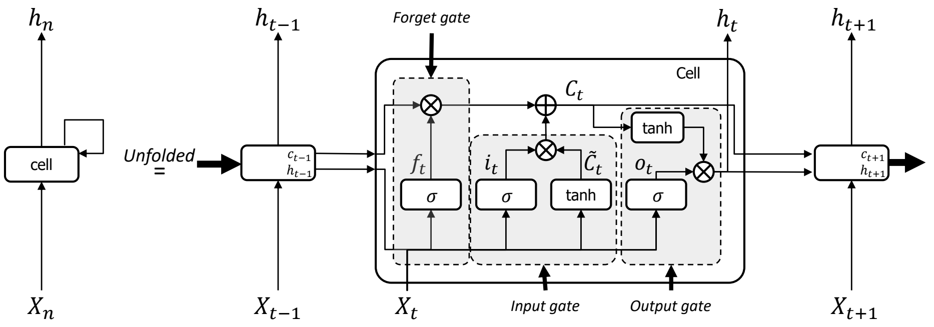
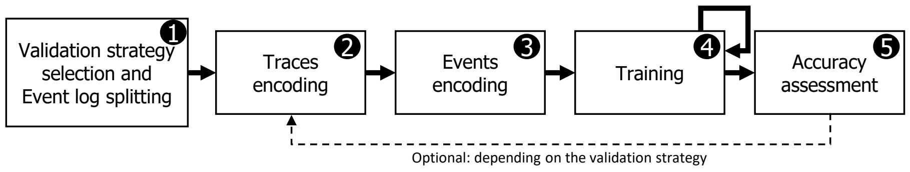


Figure 7: LSTM basic structure

2.3.2. Predictive process monitoring training pipeline

Deep learning models are mainly used, but not exclusively, in supervised learning tasks. Supervised learning seeks to deduce a probability function from labeled training data. The supervised learning approaches applied in PPM follow a series of defined steps (see Fig. 8):

Validation strategy selection and event log splitting step. A widespread problem in machine learning is over-fitting, which occurs when the model fits exactly against its training data, thus implying that the model cannot generalize its results to future data. To avoid this risk, it is necessary to implement validation strategies that ensure the model is relevant to data collected in the future, not only with the



❶ Validation strategy selection and Event log splitting
❷ Traces encoding
❸ Events encoding
❹ Training
❺ Accuracy assessment
Optional: depending on the validation strategy

Figure 8: Supervised training pipeline

model's training data. There are multiple validation strategies– such as Holdout, K-fold cross-validation, and Random Sub-sampling– which can be implemented to avoid this risk. These techniques work with subsets of the data (train and test splits), so that the model is always tested with unobserved data. Fig. 9 presents the validation schemes most commonly used in machine learning.

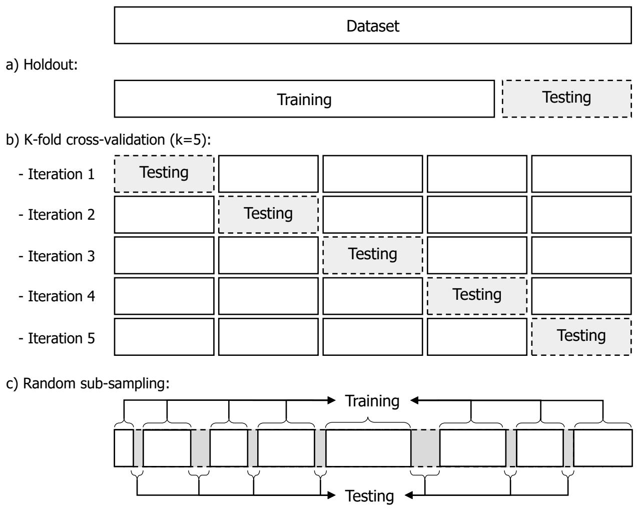


Figure 9: Most common splitting strategies

In PPM and other applied disciplines, the temporal split strategy is commonly used as a validation approach. In this approach, the splits are defined from the selection of a fixed time-point, where any trace after that point is used for testing. This strategy is considered the most strict and realistic setting, which is why it is used in this thesis. However, one limitation of the temporal splitting is that after calculating the intersection between the training and testing sets, the total number of cases and events retained is much smaller than under other strategies, meaning there are fewer cases available for training/validation/testing. Fig. 10 presents the temporal split validation scheme used in this thesis.

Traces encoding step. In DL models, traces must be encoded in fixed-size tensors. However, there is great variability in the length of each trace in a business process, so this step supposes a significant challenge. Furthermore, this step also defines how the training targets are fed into the neural network. Some of the

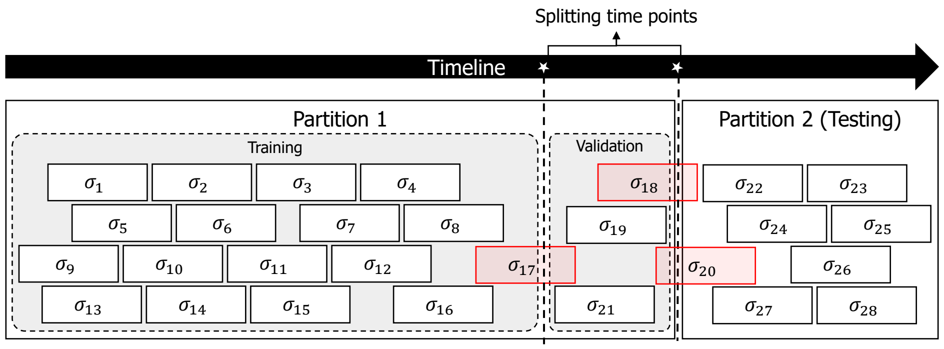

Splitting time points
Timeline
Partition 1
Training
Validation
Partition 2 (Testing)
σ_1 σ_2 σ_3 σ_4
σ_5 σ_6 σ_7 σ_8
σ_9 σ_{10} σ_{11} σ_{12} σ_{17}
σ_{13} σ_{14} σ_{15} σ_{16}
σ_{18} σ_{19} σ_{21}
σ_{20} σ_{22} σ_{23} σ_{24} σ_{25} σ_{26} σ_{27} σ_{28}

Figure 10: In this image, σ_n represents each log trace arranged in the temporal order. We marked in red the traces that resulted incomplete due to the selected temporary cutoff points. We will remove such traces from the event log

methods most used in the literature for this task are the following:

- *Prefixes encoding*: This method seeks to extract incremental sub-sequences from each trace. Each sub-sequence must be right padded with zeroes if they are shorter than the specified vector length, which usually corresponds to the size of the longest trace in the event log. For example, if we have the trace $\sigma_1 = \langle e_1, e_2, e_3, e_4 \rangle$ the set of possible padded prefixes that we can extract is $\{\langle 0,0,0,e_1\rangle, \langle 0,0,e_1,e_2\rangle, \langle 0,e_1,e_2,e_3\rangle, \langle e_1,e_2,e_3,e_4\rangle\}$. The target variable is commonly the next event in the sub-sequence, the time remaining until the end of the case, or the complete suffix of the sequence.
- *N-grams encoding*: In this method, every contiguous sequence of n-items is extracted from a given trace, where n is a parameter defined by the modeler. As in the prefix encoding method, the n-grams must also be padded, allowing the creation of training examples from the first event of the trace. For example, if we have the trace $\sigma_1 = \langle e_1, e_2, e_3, e_4 \rangle$ the set of possible n-grams with n = 3 is $\{\langle 0,0,e_1\rangle, \langle 0,e_1,e_2\rangle, \langle e_1,e_2,e_3\rangle, \langle e_2,e_3,e_4\rangle\}$.
- *Sequential encoding*: In this method, the log is viewed as a text, each trace as a sentence of that text, and each activity as a word. In this type of encoding, a window W of events is moved from the beginning of the record to the end, creating non overlapped samples of the size of the window at each step. In case if a window is incomplete, it can be discarded or filled with zeros. In the same way, every end of case is denoted by a special symbol. For example, if we have a set of traces $\{\langle e_1,e_2,e_3\rangle, \langle e_4,e_5\rangle\}$ the set of possible examples created a window W = 2 is $\{\langle e_1,e_2\rangle, \langle e_3,EOT\rangle, \langle e_4,e_5\rangle\}$.

Events encoding step. Input encoding: DL predictive models require the features to be represented as numerical vectors. This coding is done differently depending on the nature of the variables, i.e., continuous, or categorical. In the case of continuous features, these must be scaled to avoid the magnitude of bias a variable generates in its interpretation by the model. Continuous variables can be scaled using multiple techniques such as log-normalization, min-max normalization, and z-score normalization– this being generally a design decision. In the case of categorical attributes, their transformation is more complex, since the transformation can directly affect the performance of the model. Some of the most used coding strategies in PPM for this task are:

- *One-hot encoding*: In this strategy, each possible value of the category variable is converted into a new attribute that can take the value of 1 if the observation corresponds to that value or 0 otherwise.
- *Frequency-based*: This type of coding seeks to express how many times a category has occurred up to the current position. This encoding is useful when the modeler must add temporal information to the encoding of the sequences.
- *Embedding*: In this technique, each category of the variable is uniquely assigned to a space of n-dimensions. The coordinates in this space are assigned randomly or using the stochastic gradient descent algorithm during the training of a predictive model.

Training step. Depending on the predictive task required, it is possible to train the models in a discriminative or generative way. The training method determines "what" the models learn from the data. Discriminative training seeks to teach the models the decision limit between classes, while generative training seeks to teach the real distribution of each class on the data. In other words, a model that employs discriminative training learns the conditional probability distribution $p(y|x)$ directly or learns a direct map from the x inputs to the class label. Using generative training, the model learns the joint probability distribution $p(x,y)$ of the inputs x and the label Y. The models make their predictions using Bayes' rules to calculate $p(y|x)$ and then select the most probable label Y. The type of training determines how each training step is applied, from creating examples to evaluating accuracy. Fig. 11a and Fig. 11b present an example of the differences in each training step using the same sample data set.

Both generative and discriminative training methods have advantages and disadvantages. For instance, models trained in a generative way (aka, generative models) can handle missing or partially labeled data and can readily handle compositionality (e.g., faces with glasses and/or hats, and/or mustaches), whereas models trained in a discriminative way (aka, discriminative models) need to see all combinations of possibilities during training. However, generative models are also more computationally expensive than discriminative models. On its part, discriminative models are typically fast at making predictions for new (test) data points, while generative ones often require an iterative solution. The selection of one algorithm or another depends on the required predictive task, data availability, and computing capacity. This thesis mainly uses generative models since the predictive task required imply the creation of new data for which learning the joint probability distribution is the natural approach.

Generative Adversarial Networks (GANs). GANs are DL generative models that use a type of training called adversarial that differs somewhat from the classic discriminative and generative training. Adversarial training consists of training, simultaneously, two different networks (one called generator and another called discriminator) to compete with each other. The generator network is trained to produce new data points from some random, uniform distribution. Its goal is to fool the discriminator into believing that the input it sends is real. The discriminator network is trained to identify the fake data produced by the generator from the real data. In each training cycle, the generator becomes better to confuse the discriminator; however, the discriminator becomes better at detecting the fake traces produced by the generator as well, which forces a joint improvement of the models. This process repeats for a time or until the Nash equilibrium is found. Fig 11c presents the general structure of a GAN model using Adversarial training.

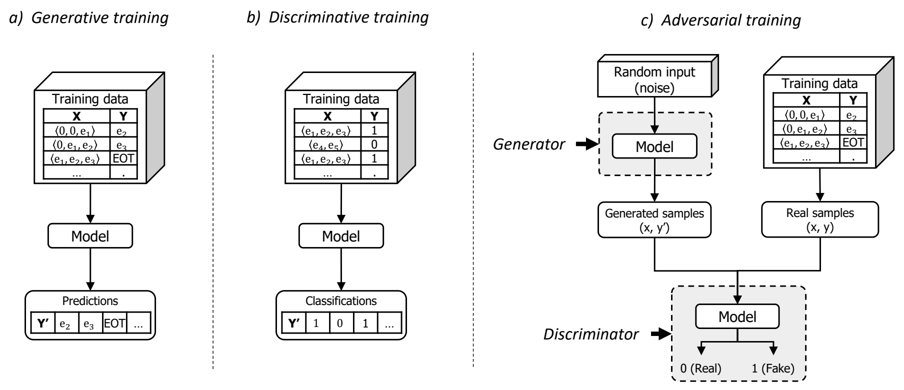


Figure 11: Comparison between different kinds of training methods

3. STATE OF THE ART

In this review of the state-of-the-art, we aim to answer the following questions:

Q1 What approaches of generative models of business processes are based on data-driven simulation?

Q2 What approaches of generative models of business processes are based on deep learning models?

Q3 How could these methods be classified?

Q1 and Q2 are the main research questions that aim to identify the existing approaches of generative models of business processes that we can potentially use in the business process simulation. Q3 aims to define a taxonomy to classify the approaches in groups based on their, specific, characteristics. We filtered and contextualized the works in the systematic literature reviews already existing to answer these questions. In the case of DDS approaches, we review the work of Martin et al. [54], while in the case of DL approaches, we rely on the work of Rama-Maneiro et al. [71].

3.1. Data-driven simulation approaches

Data-driven approaches to BPS can be classified in two categories. The first category consists of approaches that provide conceptual guidance on how to discover BPS models using PM techniques. The second category consists of approaches that seek to automate the discovery of BPS models. Below, we review each of these two categories.

3.1.1. Conceptual guidance for data-driven simulation using PM techniques

These approaches discuss how PM techniques can be used to extract, validate, and tune BPS model parameters without seeking to provide fully automated support.

Martin et al. [53] identify four components of a simulation model– namely entities, activities, resources and gateways. The authors identify BPS modeling tasks related to each of these components (e.g., modeling gateways, modeling activities). In [54], the same authors present a literature review on the use of PM techniques to support each of these modeling tasks. This review sheds insights into the question of how to choose PM techniques for each of the BPS modeling tasks. In this thesis, we use these insights as a basis to design an automated method for the discovery of BPS models from event logs.

In [91], the authors present an approach to enhance a given process model with simulation parameters. This approach differs from the one presented in the present thesis in that it assumes that the process model is given as an input (in addition to the event log). The approach also assumes that the process model perfectly fits the event log. In reality, though, the traces in the event log may deviate with respect to the behavior captured in the process model. Moreover, the approach in [91] does not seek to provide an automated end-to-end approach for discovering BPS models from event logs rather, it focuses on providing guidance for approaching some steps in the discovery of a BPS model.

The authors in [61] present a methodology for process improvement based on data-driven simulation. The authors propose as a first step the discovery of simulation models from data for the representation of the current state of the process. Next, the authors propose the manual evaluation of possible scenarios to lead the process to a desired state. The work is illustrated with three case studies from the gas, government, and agriculture industries. This work is useful for scoping the relevance of using data for discovering simulation models, as well as for identifying the need for automating this task to explore possible scenarios more quickly and efficiently. However, it does not provide concrete guidance for discovering accurate BPS models from process execution data.

3.1.2. Automated discovery of business process simulation models using process mining

The methods in this category seek to automate the discovery of BPS models from event logs by means of PM techniques.

Rozinat et al. [75] propose a semi-automatic approach to discover BPS models based on Colored Petri Nets (CPN). In this work, an event log is used as an input for the discovery of various elements of a BPS model, including the process model, the conditional branching probabilities, and the resource pools. However, the automatic discovery of activity processing times and case inter-arrival times (and their probability distributions) are left aside. In [74], the authors go further by proposing a technique to discover more complete BPS models that include processing times and case inter-arrival times. These simulation parameters are then combined with the process model into a single CPN, which can be simulated using a CPN tool. One limitation of the work of Rozinat et al. [74] is that it does not seek to automatically adjust or fit the probability distributions of the processing times of activities (nor the probability distribution of case inter-arrival times). Also, the step where the multiple model BPS model elements are merged together is not automated. Moreover, Rozinat et al. do not seek to optimize the accuracy of the BPS model. The authors suggest measuring the accuracy of the simulation model by comparing the cycle time produced by the BPS model concerning the input event log, but this only provides a coarse-grained assessment. Two event logs may have similar cycle times, yet the activities in the corresponding traces may occur at very different points in time and order.

Khodyrev et al. [40] propose a PM approach to generate BPS models tailored for the short-term prediction of performance measures. The authors extract the structure of the process as a Petri Net and establish the dependencies between elements and variables using decision trees. A limitation of this approach is that it does not discover the resource perspective (i.e., the resource pools) of the BPS model. Instead, it assumes that an infinite amount of resources may perform each activity. While the authors automatically discover the conditional branching probabilities and of the activity processing times, the integration of these elements into a BPS model is left to the user. Moreover, the approach does not define how to measure and optimize the accuracy of the resulting BPS model.

Gawin et al. [31] combine multiple PM techniques to create a BPS model that reflects actual process behavior. Specifically, PM techniques are employed to extract the process model structure, the resource pools, the activity processing times, and the decision logic of decision gates. Interviews and process documentation techniques are used to elicit the case inter-arrival times, the costs of the use of resources, and the definition of resource schedules. The simulation parameters discovered in this way are then manually linked in the ADONIS tool, leading to a BPS model that is then executed with a capacity analysis algorithm of this latter tool. This latter work differs from the one reported in this thesis in that it does not seek to automate the extraction of all elements of a BPS simulation model (nor their assembly). Also, it does not seek to measure and optimize the accuracy of the BPS model.

Finally, in a more recent study Pourbafrani et al. [69] propose a DDS approach to derive the system dynamics of simulation models from logs for what-if analysis. This type of model allows performing analysis at a high level of abstraction without going into the details of the process. The same authors [68] present an approach for the generation of DDS models based on time-aware process trees by automating the extraction of simulation parameters. However, the responsibility for tuning these parameters lies completely on the user.

Table 5 summarizes the capabilities of the above approaches for BPS model discovery. In this table, the symbol $(+)$ implies that the feature is supported, $(-)$ implies not supported, and $(+/-)$ implies partially supported, e.g., supported but not in an automated manner.

Characteristics	**Rozinat et al. (2006)[75]**	**Rozinat et al. (2009)[74]**	**Khodyrev et al. (2014)[40]**	**Gawin et al. (2015)[31]**	**Pourbafrani et al. (2020)[68]**
Sequence flow discovery	(+)	(+)	(+)	(+)	(+)
Resource pools discovery	(+)	(+)	(-)	(+)	(+)
Branching probabilities discovery	(+)	(+)	(+/-)	(+)	(+)
Probabilities distribution fitting	(-)	(-)	(-)	(-)	(+)
Model assembly	(-)	(-)	(-)	(-)	(+)
Accuracy assessment	(-)	(+/-)	(+/-)	(+/-)	(+)
Accuracy optimization	(-)	(-)	(-)	(-)	(-)

Table 5: Comparison of approaches to discover and/or enhance BPS models

In summary, this review of related works allowed us to identify the characteristics that every DDS approach must meet and the sources of possible biases that the existing works have not addressed. Specifically, most of the works in the current state-of-the-art present two shortcomings that we must manage in order to obtain a fully automatic tool for the discovery of DDS models: (i) Existing studies have not extensible explored the question of measuring the accuracy of BPS models concerning the input data used to discover them, (ii) none of the works perform automatic optimization of the hyperparameters of the PM techniques to obtain the most accurate model possible. In this research, we propose a fully automated simulation model discovery method that addresses these shortcomings. We materialized the method through a tool called Simod [16], that we introduce in Chapter 4.

3.2. Generative neural network models in predictive monitoring tasks

DL models have been widely applied in the field of predictive process monitoring. Predictive process monitoring is a subfield of PM that deals with forecasting how a running case will develop. DL models have been used successfully in tasks such as process outcome prediction, anomaly detection, and prediction of the future of an ongoing case in this area. We focus mainly on those models used in the last task, since it is the one that directly uses DL generative models. Related work in this area presents many combinations between possible architectures, ways to encode traces and events, and the number of predictive tasks that each model can perform. However, it is possible to identify three large groups: those using generative RNNs as a base, those using other types of models such as DFFNs or CNNs, and those who explore complementary methods to improve the precision of the generative models. Next, we review each of these three categories.

3.2.1. Related work based on RNNs

Due to the sequential nature of event logs, the type of neural network most used in predictive process monitoring has been RNNs (especially LSTM networks), since they specialize in processing this type of data.

Tax et al. [84] is one of the first works to use RNNs networks to predict the future state of an ongoing case. Specifically, the authors employed LSTM networks to predict the type of next event and its timestamp. In this approach, the authors encoded the log trace using incremental sub-sequences called prefixes, while the types of events are encoded using one-hot encoding. Supplementary, each event is enhanced with other features related to the event's occurrence time, such as the time of the day, the time since the previous event, and the accumulated duration since the start of the case. The network architecture consisted of a shared LSTM layer that fed two independent LSTM layers– one specialized in predicting the next event category and the other in predicting times. By repeatedly predicting the next event category in a case and its timestamp, the authors also used their approach to predict the remaining sequence of events until case completion and the remaining cycle time. The experiments showed that the LSTM approach outperforms automata-based methods for predicting the remaining sequence of events and the remaining time [14, 67]. In this approach, the event type was one-hot encoded. This design choice is suitable when the number of event types is low, but detrimental for larger numbers of event types.

Evermann et al. [27] also applied LSTM networks to predict the type of the next event of a case. Unlike [84], this approach used the embedded dimension of LSTMs to reduce the input's size and to include additional attributes such as the resource associated with the task execution. The network's architecture comprised two LSTM hidden layers. An empirical evaluation showed that this approach sometimes outperforms the approach of [84] at the task of predicting the next event. However, this approach focused on predicting only event types. It cannot handle numerical variables, and hence it cannot predict the next event's timestamp.

Navarin et al. [62] proposed an adaptation of the [84] approach specialized in predicting the remaining time of a running case. The authors used the same input encoding methods and LSTM architectures as those used by [84]. The main difference between both approaches focused on the definition of the target variable. In this approach, the models were trained to predict the remaining time of the case directly; conversely, in [84] the prediction was performed iteratively. The authors demonstrated that direct training avoids the accumulation of errors, thus improving the accuracy of the overall method. This approach outperformed all the reference works for the predictive task of predicting the remaining time of an ongoing case. However, it cannot be generalized to other predictive tasks, such as predicting the next event or the process suffix.

Multiple works have explored the impact of data attributes to improve the accuracy of LSTM models. For example, in Schonig et al. [79], the authors employed LSTM models for the prediction of the next event type and associated resource. For this task, they used complete sequences of activities, resources, and data attributes. Similarly, in Hinkka et al. [36] the authors used GRU models in conjunction with prefixes as encoding methods, and data attributes to predict the category of the next event. The main contribution of this work was the proposal of a clustering method of data attributes to reduce the dimensionality of the input data.

More elaborately, Lin et al. [49] proposed an RNN-based approach, namely MM-pred, which used data attributes and an encoder-decoder architecture for predicting the next event type and the suffix of an ongoing case. The proposed architecture was composed of encoders, modulators, and decoders. Encoder and decoder layers used LSTM networks to transform the features of each event into, and from, hidden representations. The modulator component infers a variable-length alignment weight vector, in which each weight represents the relevance of the attribute for predicting future events and features.

In the previous approaches, the use of data attributes significantly improved the accuracy of the models, which was very useful in the case of predictive tasks that were carried out directly, e.g., the prediction of the next event. However, predictive tasks that require been carried out iteratively by feeding backed the models– e.g., event logs generation from scratch. The use of data attributes would imply the accumulation of errors in all the features – since they need to be done iteratively— significantly deteriorating the predictions' quality.

Taymouri et al. [86] in a more recent study, proposed to use a GAN method to train an LSTM model capable of predicting the type of the next event and its timestamp. The strategy proposed by the authors consists of two LSTM models, one generative and one discriminative, that were trained simultaneously through a game of adversaries. In this game, the generative model must learn to confuse a discriminative model to avoid distinguishing real examples from fake ones. As the game unfolds, the discriminative model increases its ability to distinguish between fake and real examples, thus forcing the generator to improve the generated examples. The authors showed that this GAN approach outperforms classical training methods for the task of predicting the next event and timestamp on certain datasets.

3.2.2. Related work based on other DL architectures

Arguing some limitations of RNNs, such as the limited handling of long-term dependencies and the high computational cost involved in their training, alternatively, some works have explored the use of other types of networks such as DFFNs or CNNs that have many applications in fields such as image processing.

Mehdiyev et al. [57] proposed another approach to predict the next event using a multi-stage deep learning approach. In this method, each event is mapped to a feature vector, and multiple transformations are applied to reduce the input's dimensionality, i.e., extracting n-grams, using a hash function, and passing the input through two auto-encoder layers. Then the transformed input is then processed by a DFFN responsible for the next-event prediction. This approach did not handle numerical variables, and hence it can not predict timestamps or durations.

In Theis et al. [88] the authors proposed an alternative approach to the RNN and CNN models, in which they used a DFFN network in conjunction with a decay function technique for the prediction of the next event. The method discovers a process model in Petri nets format and performs a token replay of the event log on the model. Then apply a decay function that encodes the time between the current timestamp and the times a token was in each place during replay. The training target is the next event after each replayed activity prefix on the Petri net. The proposed technique had the advantage of potentially capture parallel relations between the activities that were not easily captured by observing the ordering of the events in the log. However, this encoding is not compatible with more robust and specialized models in sequences such as RNNs and CNNs. Similarly, its structure makes it difficult to generalize to other predictive tasks, such as predicting the process suffix.

The works proposed by [39, 22, 65] used CNN networks to predict the next event category. Al-Jebrni et al. [39] employed a neural network composed of five one-dimensional CNN layers, the authors selected this architecture considering the similarity in the data dimensionality and its cheaper computational cost. As inputs, this model used uniquely sequences of event types encoded with embedded dimensions. One of the main problems of CNN networks is defining the size of the kernel used for the convolution operation, which significantly affects the information extracted. This problem was not explored in [39]; however, the work proposed by Di Mauro et al. [22] addressed this problem by using a module called inception for one-dimensional CNN networks. The inception module simultaneously applies many convolutions with different kernel sizes over the same input. In addition to the activity category sequences, this model uses time sequences. The proposed method encodes the log sequences as prefixes and uses embedded dimensions to encode the categorical attributes. Unlike the two previous works, the proposed by Pasquadibisceglie et al. [65] used a traditional two-dimensional CNN network. The method transforms the temporal data enclosed in the event log into spatial data to treat them as images. This work used the prefixes of activities and times and encoded the event types using embedded dimensions.

All the three previous works exceed the accuracy of LSTM-based ones for predicting the next event category. However, they still do not consider the problem of simultaneously predicting the next event and its timestamp, so they cannot be used in BPS, which is the problem at hand in this thesis.

Table 6 summarizes the capabilities of the above DL generative models. In this table, the column "Architecture type" indicates the kind of neuronal network used in the approach: Long Short-Term Memory (LSTM), LSTM trained with Adversarial training (LSTM-GAN), Gated Recurrent Unit (GRU), Convolutional Neural Network (CNN), Autoencoder (AE), and Deep Feedforward Network (DFNN). The "Encoding methods" column describes the trace and event encoding methods applied to transform the event log in vectors. "Trace encoding" methods: Full sequence (SEQ), Prefixes padded (PRFX), N-gram (NGRAM), and Timed state sample (TSS). "Event encoding" methods: Embedding (EMB), One-hot encoding (ONE-HOT), Frequency-based (FB), and Index-based (IDX). The "Input features" column describes the relevant features supported in the model's training (the symbol X indicates supported). Sub-columns names: Event type/Activity (AC), Resource (R), Time features (TF), Data Attributes (DA), Engineered features (EF), and Process Model (PrM). The "Predictive task" column describes the supported predictive tasks (the symbol X indicates supported). Next event prediction sub-columns: Event type/Activity (AC), resource (R), Time features/timestamp (TF), Data Attributes (DA). Sequences prediction sub-columns: Case continuation/Activities sequence (AC), Remaining Time (RT), and Data attributes sequence (DA).

This review allowed us to identify the works in the current state-of-the-art that can potentially be used in business process simulation. Specifically, in this thesis, we present an approach to train DL generative models based on LSTM networks

Approaches	Architecture Type	Encoding methods		Input Features						Predictive task						
		Trace encoding	Events encoding							Next Event				Sequences		
				AC	R	TF	DA	EF	PrM	AC	R	TF	DA	AC	RT	DA
Tax et al. (2017) [84]	LSTM	PRFX	ONE-HOT	X		X				X		X		X	X	
Evermann et al. (2017) [27]	LSTM	SEQ	EMB	X						X						
Navarin et al. (2017) [62]	LSTM	PRFX	ONE-HOT	X			X	X							X	
Schonig et al. (2018) [79]	LSTM	SEQ	ONE-HOT	X	X		X			X	X					
Hinkka et al. (2019) [36]	GRU	PRFX	ONE-HOT	X				X		X						
Lin et al. (2019) [49]	LSTM	SEQ	EMB	X			X			X			X	X		X
Taymouri et.al. (2020) [86]	LSTM-GAN	PRFX	ONE-HOT	X		X				X		X				
Mehdiyev et al. (2017) [57]	AE+DFNN	NGRAM	IDX	X			X			X						
Theis et al. (2019) [88]	DFNN	TSS	FREQ	X		X	X		X	X						
Al-Jebrni et al. (2018) [39]	CNN	SEQ	EMB	X						X						
Pasquadibisceglie et al. (2019) [65]	CNN	PRFX	FREQ	X		X				X						
Di Mauro et al. (2019) [22]	CNN	PRFX	EMB	X		X				X						

Table 6: Comparison of deep learning generative models applied in predictive process monitoring

that can handle large numbers of event types (see Chapter 5), in which we combine the idea of using the embedded dimensions from [27] with the idea of interleaving shared and specialized layers from [84]. We selected these methods because they have the capability of generating both the type of the next event in a trace and its timestamp. This means that if we iteratively apply these methods starting from an empty sequence, via an approach known as *hallucination*, we can generate a sequence of events such that each event has one timestamp (the end timestamp). Hence, these methods can be used to produce entire sequences of timestamped events. Therefore, they can be used to generate event logs that are comparable to those that DDS methods generate, with the difference that the above DL training methods associate only one timestamp to each event. In contrast, DDS methods associate both a start and end timestamp to each event. Accordingly, we needed to adapt the above two DL methods to generate two timestamps per event for full comparability.

3.2.3. Related work in complementary areas

The following works do not directly attack questions Q1 and Q2; however, we decided to include them because they provide a complementary view of some techniques that can be used in BPS.

In one complementary work, Di Francescomarino et al. [21] proposed encoding of rules to determine the next task in a running case as a complementary technique to the works of [84, 27]. The authors identified two types of rules, named no-cycle and a-priori. No-cycle rules intend to detect and avoid the stagnation in high probabilities of LSTM models, a frequent problem in this type of network. Meanwhile, a-priori knowledge rules filter the output of the generative model based on known factors by an analyst that can determine the future path of an ongoing case that, in principle, these factors can change from case to case. The rules were defined in terms of Linear Temporal Logic (LTL) [66] and are applied over the prediction performed by the generative models. The generative models are trained without knowing the rules, thus enabling their redefinition without retraining the models. The results showed that the application of restrictions on the generative model's output helped improve the precision of the models.

In another work, Tax et al. [83] compared the performance of several techniques for predicting the next element in a sequence using real-life datasets. Specifically, the authors compared generative Markov models (including all-k Markov AKOM), RNN, and automata-based models in terms of precision and ability to be interpreted. The results that the AKOM model yields the highest accuracy (outperforming an RNN architecture in some cases) while automata-based models have a higher ability to be interpreted. This latter study addressed the problem of predicting the next event's type. Still, it does not consider the problem of simultaneously predicting the next event and its timestamp, as we do in this thesis.

4. SIMOD: AUTOMATED DISCOVERY OF BUSINESS PROCESS SIMULATION MODELS FROM EVENT LOGS

According to the review of the state-of-the-art works, we identify two research gaps in the current BPS techniques based on data:

GAP1 Existing studies have not extensible explored the question of measuring the accuracy of BPS models derived from data; and,

GAP2 The fine-tuning of simulation parameters is left to the modeler, thus leaving the door open for the introduction of biases during the creation of BPS models.

In response, this section will introduce the simulation model discoverer (Simod) tool specialized in generating simulation models from event logs automatically. Simod uses an automated process discovery technique to extract a process model from an event log and then enhances this model with simulation parameters extracted via a combination of trace alignment, replay, and curve-fitting techniques. The tool incorporates a Bayesian hyperparameter optimization technique to fine-tune the accuracy of the resulting simulation model. The first part of this chapter will describe the tool's execution pipeline and each of its steps. In the same way, we will introduce the Event-Log Similarity metric specialized in measuring the similarity between two event logs, including the sequence and time perspective. Later we will present the evaluation of the tool that was carried out using three event logs (one synthetic and two real-life) from different domains.

4.1. Approach Description

The proposed method takes as input an event log in which every event (corresponding to the execution of an activity instance) has the following attributes: a case identifier, an activity label, a resource that performed the activity, the start timestamp, and the end timestamp[1]. The resource attribute is required to discover the available resource pools, their timetables, and the mapping between activities and resource pools. Equally, the start and end timestamps are required to compute the processing time of activities and their respective probability distributions.

Fig. 12 illustrates the steps of the proposed method for automating the discovery of BPS models. These steps are explained and exemplified in the following subsections using a synthetic event log of a *Purchase-to-Pay (P2P)* process. This event log [2] consists of 21 activities, 27 resources, and 9119 events related to 608 cases.

[1]Alternatively, the event log can also be processed in scenarios where each activity instance is recorded as a start event and an end event, each one with a corresponding timestamp.

[2]Taken from the tutorial of the Disco PM tool, available at `http://fluxicon.com/academic/material/`

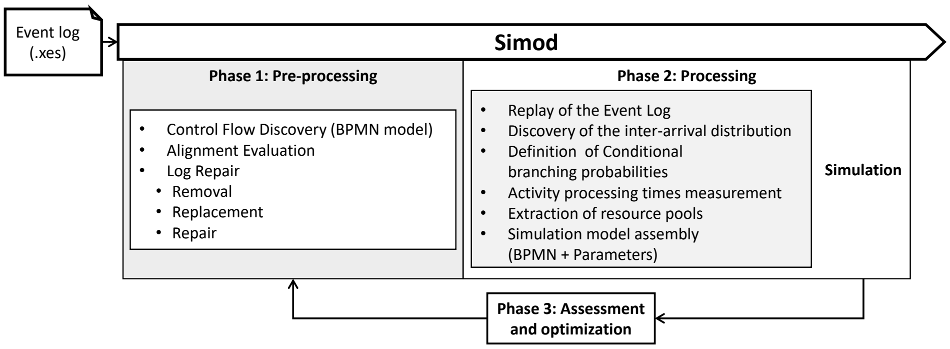

Event log
(.xes)
Simod
Phase 1: Pre-processing
Control Flow Discovery (BPMN model)
Alignment Evaluation
Log Repair
Removal
Replacement
Repair
Phase 2: Processing
Replay of the Event Log
Discovery of the inter-arrival distribution
Definition of Conditional branching probabilities
Activity processing times measurement
Extraction of resource pools
Simulation model assembly (BPMN + Parameters)
Simulation
Phase 3: Assessment and optimization

Figure 12: Steps of the BPS model discovery method

4.1.1. Phase 1: Pre-processing

These steps allow us extracting a BPMN process model from the event log and guaranteeing their conformance[3] [59]. Most of the time, due to the characteristics of the process discovery algorithms, the models do not reflect all the possible paths in a business process (the fitness is not 100%). Therefore, the proposed method allows applying repair actions on the log to improve the fitness between the model and the log.

Control Flow Discovery. We use the Split Miner algorithm [11] to generate BPMN v2.0 models from event logs. We selected this process discovery method since it achieves high levels of accuracy (precision and fitness) while at the same time producing simple process models [10]. However, there is no limitation to use other process discovery methods (e.g., Inductive Miner [46]).

Split Miner allows us the discovery of models with different sensitivity levels, which depend on the epsilon (ε) and eta (η) parameters. The ε parameter refers to the parallelism threshold, which determines the number of concurrent relations captured between events. The η parameter refers to the percentile for frequency threshold, which only the η percentiles most frequent paths between activities. Table 7 outlines the structure of the event log used as input, while Fig. 13 illustrates the resulting model using ε as 0.3 and η as 0.7.

Alignment Evaluation. We measure the degree to which each trace in the log can be aligned with a corresponding trace produced by the process model by using the *fitness measure* proposed in [7]. This alignment is a sequence that has the length of the longest trace, and it consists of three symbols: SM ("synchronous move"), MM ("move-on-model"), and ML ("move-on-log"). An SM indicates that the two traces match (i.e., the current activity is the same in both traces). An MM means that the two current activities do not match and that the algorithm will "skip" the current activity in the model. Thus, the algorithm moves forward in the

[3] *Conformance checking* is the discipline in PM that attempts to align the execution trace of a case with the model to detect deviations. The *fitness* measure quantifies the extent to which the model can reproduce the traces recorded in the log to quantify conformance [15].

Case ID	***Activity***	***Start Timestamp***	***Complete Timestamp***	***Resource***
1	Create Purchase Requisition	2011/01/01 00:00:00	2011/01/01 00:37:00	Kim Passa
2	Create Purchase Requisition	2011/01/01 00:16:00	2011/01/01 00:29:00	Immanuel Karagianni
3	Create Purchase Requisition	2011/01/01 02:23:00	2011/01/01 03:03:00	Kim Passa
1	Create Request for Quotation	2011/01/01 05:37:00	2011/01/01 05:45:00	Kim Passa
1	Analyze Request for Quotation	2011/01/01 06:41:00	2011/01/01 06:55:00	Karel de G root
2	Create Request for Quotation	2011/01/01 08:16:00	2011/01/01 08:26:00	Alberto Duport
4	Create Purchase Requisition	2011/01/01 08:39:00	2011/01/01 09:00:00	Fjodor Kowalski

Table 7: Event-log format example

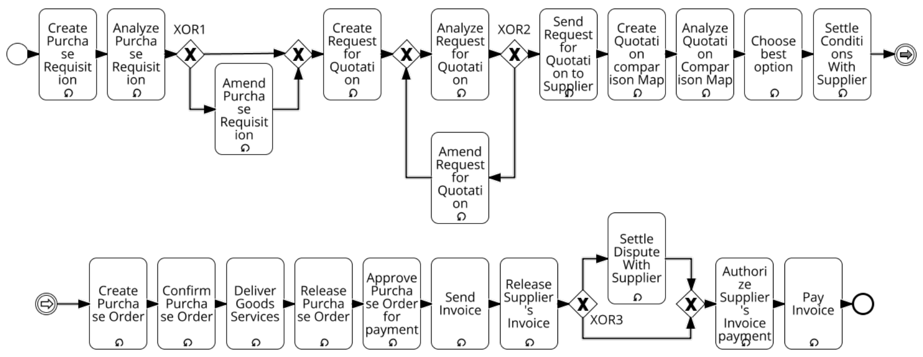

Create Purchase Requisition
Analyze Purchase Requisition
XOR1
Amend Purchase Requisition
Create Request for Quotation
Analyze Request for Quotation
XOR2
Amend Request for Quotation
Send Request for Quotation to Supplier
Create Quotation comparison Map
Analyze Quotation Comparison Map
Choose best option
Settle Conditions With Supplier
Create Purchase Order
Confirm Purchase Order
Deliver Goods Services
Release Purchase Order
Approve Purchase Order for payment
Send Invoice
Release Supplier's Invoice
XOR3
Settle Dispute With Supplier
Authorize Supplier's Invoice payment
Pay Invoice

Figure 13: SplitMiner BPMN output example of the purchasing process

trace of the model, but we stay in the current position in the log trace. Conversely, an ML means that the two current activities do not match; thus, the current activity is *skipped* in the trace of the log to align the two traces (and remain in the same position in the trace of the model). A perfectly aligned pair of traces contains only SM symbols. Otherwise, the number of MM and ML symbols capture the level of misalignment.

Log Repair. Once conformance (fitness) is measured, it can be improved by performing a model repair, an event log repair, or both [73]. We perform a log repair for those traces that do not fully fit a trace in the discovered process model. We propose three methods for this purpose: removal, replacement, and repair.

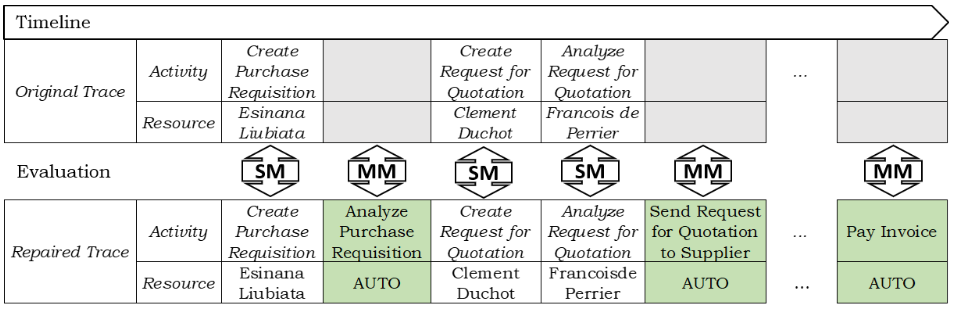
The *Removal* method omits the traces that are not in conformance with the extracted model, leaving only a reduced log composed of conformant traces. This method is the most natural and computationally cheap to avoid the outlier traces in the data source. However, if the event log has low conformance with the model, it could result in a tiny event log that could not sufficiently represent the process dynamics.

The *Replacement* method replaces each non-conformant trace with a copy of the most similar conformant one. This action keeps the amount of traces of the log and globally compensates for the lack of conformance. We define the similarity between traces as one minus the normalized Damerau-Levenshtein distance (DLd) between two strings. We created an alphabet by assigning a unique character to each event in the log to construct words that describe the execution order of the activities. Then, each non-conformant trace is compared with all the conformant ones to find the most similar.

The *Repair* method aligns each process trace of the log with the extracted process model. This action keeps the maximum recorded observations, preserving the recorded time variability. We use the automata-based alignment technique proposed in the ProConformance 2.0 tool[4], which determines the optimal alignments between the log traces and the model and suggests a minimum number of movements to make them conformant. To repair a given trace as a first step, we scan its corresponding alignment from left to right, and then we apply one of three operations:

- When an ML is found, we remove the responsible event in the trace.
- When an MM is found, we annotate the log with zero processing time and a special resource called "AUTO." This annotation means that the activity does not consume resources and hence does not have an impact on the cycle time of the process.
- Finally, When an SM is found, the algorithm advances one step in the trace.

Fig. 14 illustrates a resulting repaired trace from the original trace with case ID 100 of the purchasing process event log mentioned above. This trace has three activities, ending prematurely concerning the process model discovered from the log. In this case, the ProConformance tool returns a fitness value of 0.4/1 and suggests a type of alignment that can be used to repair the event log.



Timeline								
Original Trace	*Activity*	*Create Purchase Requisition*		*Create Request for Quotation*	*Analyze Request for Quotation*		...	
	Resource	*Esinana Liubiata*		*Clement Duchot*	*Francois de Perrier*			
Evaluation		SM	MM	SM	SM	MM		MM
Repaired Trace	*Activity*	*Create Purchase Requisition*	Analyze Purchase Requisition	*Create Request for Quotation*	*Analyze Request for Quotation*	Send Request for Quotation to Supplier	...	Pay Invoice
	Resource	Esinana Liubiata	AUTO	Clement Duchot	Francoisde Perrier	AUTO	...	AUTO

Figure 14: Example: repairing a non-conformant trace

As an example of applying the three methods and their differences, suppose we have an event log with 30% of traces that do not conform to the discovered model. The Removal technique would eliminate these non-conforming traces, leaving only 70% for the extraction of parameters. On the other hand, the Replacement technique would seek in the 70% of conformant traces the most similar to the 30% non-conformant, with the purpose of duplicate them, resulting in a log of the same size as the original with a 30% of duplications. Finally, the alignment technique

[4]http://apromore.org/wp-content/uploads/2017/04/ProConformance2.zip

would adjust each non-conformant trace, resulting in a log with 70% of original traces and 30% of corrected ones.

Alternatively, this step could also be performed using the alignment technique proposed in [8] which is capable of measuring the precision between an event log and a model using alignments. However, for ease of implementation, we used the technique proposed by [7] since it has a command-line version that is easily integrable to the Simod discovery pipeline. Similarly, we were able to use conformance checking techniques in conjunction with partitioning techniques [60] which would allow us to divide large processes into sets of sub-processes that can be easily analyzed. However, this would give us multiple versions of the process model, which would make it difficult to simulate and analyze.

4.1.2. Phase 2: Processing

At this stage, the tool extracts the simulation parameters and assembles them with the process structure to create a BPS model.

Replay of the Event Log. We created a replay algorithm (see Algorithm 1) that takes as input a process model and a repaired trace to calculate the processing time and the enablement time of each activity execution (event) in the trace, as well as the traversal frequency of each conditional flow. We use these measures later to calculate the simulation parameters. The *enablement time* is the moment the activity is allowed to start according to the state of the execution. In the simplest case, the enablement time is equal to the end time of the preceding activity in the trace, but this is not always the case, especially in the presence of parallel activities in the process model. The *traversal frequency* is the number of times that the conditional branch is traversed while replaying the trace in the log. Additionally, the tool calculates the waiting time of each activity by subtracting the start time minus the enablement time. The waiting times calculated in this way are later compared with the waiting times calculated by the simulator to determine the accuracy of the simulation.

This algorithm computes the traversal frequencies for each trace in the log and then sums up them to compute the total traversal frequency of each conditional flow. The algorithm relies on the concept of *marking of a BPMN process model* [23] to capture an execution state concerning a BPMN model. A marking in a semantically correct (sound) process model is a function that maps each sequence flow in the model to a boolean value. A sequence flow is mapped to true if and only if there is a token in that sequence flow in the current state. The current marking of the model initializes where there is a token in the sequence flow coming out of the start event of the model. Then, the algorithm iterates over each event in the input trace, containing start and end timestamps, to calculate the processing time of the activity. Before handling a given event *e*, the algorithm fires every gateway enabled in the current marking until no more gateways can be fired. When an XOR-gateway is fired, the conditional flow that leads to an activity

Algorithm 1: Replay

```
inputs   : A Process Model M, A trace T
output   : processingTime: A map from events in T to Int
output   : enablementTime: A map from events in T to Timestamp
output   : traversalFrequency: A map from sequence flows in M to Int
for each ( $e \in T$ ) {
    $processingTime[e] \leftarrow endTime(e) - startTime(e)$;
    repeat
        $gatewayFired \leftarrow false$;
        for each ( $g \in gateways(M)$ ) {
            if $isEnabled(M, currentMarking, g)$ then
                if $gatewayType(G) = XOR$ then
                    $tcf \leftarrow traversedConditionalFlow(M, currentMarking, g, e)$;
                    $traversalFrequency[tcf]++$;
                    $currentMarking \leftarrow fire(M, currentMarking, g, e)$;
                    $gatewayFired \leftarrow true$;
                end
            end
        }
    until not $gatewayFired$;
    for each ( $t \in Tasks(M)$ where $isEnabled(M, currentMarking, g)$ ) {
        if $(enablementTime[nextOccurrence(t, T)] \neq \emptyset)$ then
            $enablementTime[nextOccurrence(t, T)] \leftarrow currentTime$;
        end
    }
    $currentMarking \leftarrow fire(M, currentMarking, e)$;
    $currentTime \leftarrow endTime(e)$;
}
return $processingTime, enablementTime, traversalFrequency$;
```

corresponding to the event *e* is traversed. Accordingly, the traversal frequency of this conditional flow is increased by one.

The algorithm iterates over the activities enabled in the current marking. If an activity is enabled and the enablement time of the next occurrence of this activity has not yet been initialized (the activity was not enabled before), then the enablement time of this activity is set to be equal to the current execution time. At this point, and given that the model can parse every trace in the repaired input log, the activity corresponding to the event *e* must be enabled. Accordingly, the algorithm fires this activity and updates the current execution time to be equal to the end time of the event *e*. The algorithm relies on two auxiliary functions: the *isEnabled* function determines if a gateway is enabled in the current marking of a BPMN model, whereas the *fire* function computes the marking reached from the current marking when firing a given gateway. These functions implement the semantics of gateways defined in the BPMN standard. Specifically, a split gateway (with a single incoming flow) is enabled when a token is in its incoming flow. When it fires, the token is removed from its incoming flow, and a token is produced in each of its outgoing flows (in case of an AND gateway) or in one of its outgoing sequence flows (in case of an XOR gateway). In the latter case, the conditional flow leading to enablement of the next event *e* in the trace is selected. Conversely, an AND-join gateway is enabled if there is a token in each of its incoming flows, while an XOR-join gateway is enabled when there is a token in any of its incoming flows. When a join gateway fires, the tokens in its incoming flows are removed, and a token is added to its outgoing flow.

Discovery of the inter-arrival distribution. This step determines the PDF of the inter-arrival times for the cases. To this end, the traces in the log are sorted by the start time of their activities. We assume the timestamp of the first event in a trace as the case creation time. Otherwise, a pre-processing task can be used to estimate the actual case creation time as discussed in [55]. Then, we calculate the difference between the subsequent start times of the traces daily based. The resulting data series of inter-arrival times are then analyzed to determine which PDF yields the minimum standard error. Suppose the volume of data is too low to determine a distribution (i.e., less than 100 observations). In that case, we assume an exponential distribution with a mean equal to the mean of the observed data. We used this distribution as it is commonly suggested in the process simulation literature [25]. Our current implementation supports Normal, Exponential, Uniform, Fixed-value, Triangular, Gamma, and Log-normal PDFs. In the running example, the PDF that best fits the observed inter-arrival times is an exponential PDF with a mean of 15455 seconds.

Definition of Conditional branching probabilities. A BPS model requires defining the probabilities of the paths enabled by decisions made in the gateways. These probabilities can be established by assigning equal values to each conditional branch (e.g., if there are two branches, we assign 0.5 probability to each) or by replaying or aligning the traces in the event log against the discovered process model. In the latter case, we normalize the traversal frequencies of the outgoing branches computed during replay so that their sum is one, hence converting these traversal frequencies into (normalized) probabilities. In the case of our example event log, the XOR1 gateway has two possible paths to the activities "Amend Request for Quotation" and "Send Request for Quotation to Supplier." These paths were executed on 563 and 608 occasions, respectively, which means execution probabilities of 0.48 and 0.52 of these paths (see Fig. 15).

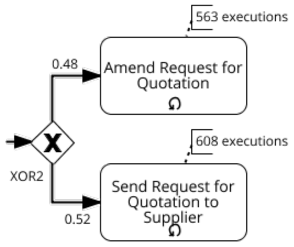

563 executions
0.48
Amend Request for Quotation
608 executions
XOR2
Send Request for Quotation to Supplier
0.52

Figure 15: XOR gateway probabilities definition example

Activity processing times measurement. We determine the PDF of the processing time of a given activity A in the process model in two steps. First, we create a data series consisting of observed processing time for each execution of activity A in the log (computed by the log replay). Next, we fit a collection of possible distribution functions to the data series to select the distribution function that yields the smallest standard error. For example, we analyze each one of the 21 activities in the purchasing process event log. As can be seen in Table 8, most of the processing times follow a Uniform distribution with a mean of 3600 seconds.

Extraction of resource pools. Resource pools defining organizational roles and groups are discovered by using the algorithm proposed in [81]. This algo-

	PDF						
	Uniform	Normal	Exponential	Gamma	Lognorm	Fixed	Triangular
# of Activities	9	2	–	2	3	5	–
Mean	3600	1285.25	–	1027.55	764.66	24	–
StdDev	0	136.82	–	548.92	704.54	32.86	–

Table 8: Purchasing process activities probability distribution functions summary

rithm defines activity execution profiles for each resource by creating a graph using the correlation of profiles, considering only the relations that overpass a user-defined *similarity threshold*. The resulting graph is a set of unconnected components (clusters) that correspond to groups of resources (resource pool) that generally perform the same type of activities. The algorithm in [81] then assigns each activity to one or more resource pools. Therefore, we post-process this output to assign each activity to the resource pool that most frequently performs it, as required by a BPS model. In our running example, 26 resources were grouped in 5 resource pools, each one assigned to one activity when using the algorithm mentioned above[5].

Simulation model assembly. Once we have compiled all the simulation parameters, we put them together with the BPMN model into a single data structure. This step is dependent on the target DES simulator (e.g., BIMP or Scylla). In BIMP, for example, this step involves embedding the simulation parameters inside the BPMN model, using proprietary XML tags.

Simulate Process. In this last step, the BPS model is given as input to a DES model simulator to generate a simulated event log. It is important to note that this step can be performed using another DES simulator such as those evaluated in Jansen-Vullers et al.[38]; However, we use BIMP and Scylla for ease of implementation since they have command line versions that allow their easy integration into the Simod pipeline. Below, we discuss how the accuracy of the resulting BPS model is assessed and optimized.

4.1.3. Phase 3: Assessment and optimization

This stage aims to assess the accuracy of the event log generated by the simulator concerning the input event log (i.e., ground truth) and automatically combine the discovery parameters (i.e., the set of parameters used by the discovery algorithms used in phases 1 and 2) to obtain the most accurate BPS model.

Assess BPS model accuracy. In order to tune the BPS models produced by Simod, we need to have a way of measuring the accuracy of a BPS model. We propose to measure the accuracy of a BPS model by simulating it and then measuring the similarity (or conversely, the distance) between, on the one hand, the

[5]It turns out that this event log contains information about roles and the mapping from activities to roles. We found that the algorithm in [81] re-discovered the roles already present in the event log (without using this information) with 100% accuracy

traces in the event log generated by the simulation and, on the other hand, the traces in the log from which the BPS model was discovered (the *ground truth*). To make the log generated by the BPS model comparable to ground truth, we simulate precisely the same number of traces as in the original log, e.g., if the original log has 500 traces, we ask the simulator to simulate 500 traces.

We need to define a similarity (or a distance) measure between the simulated log traces and the ground truth traces to apply the above idea. A straightforward way of doing so is by calculating the Mean Absolute Error (MAE) between the cycle times of the simulated traces and the real traces. While simple to compute, this measure is coarse-grained. Two traces may have the same cycle time yet consist of very different sets of events.

Another approach to comparing pairs of traces is through the DLd: the minimum number of operations (e.g., adding, deleting, replacing, or transposing symbols) required to transform a given string into another. For example, given two traces represented as strings: "abcd" and "acbd", their DLd is two. It is easy to normalize this measure to return a number between zero and one by dividing the absolute DLd by the maximum length of the two traces.

The DLd captures the differences in the activity occurrences and in the ordering of activities. However, it does not take into account two requirements that arise when comparing two business process execution traces:

1. The DLd penalizes transposed activities, even if these activities are parallel and hence may be completed in any order. For example, if activities *b* and *c* are parallel activities, the distance between traces "abcd" and "acbd" should be zero. The difference is accidental: in one trace *b* occurs before *c*, but it could have been vice-versa.
2. The DLd does not consider the waiting times and processing times of the activity occurrences represented by the events in a trace. Nevertheless, the ability to faithfully capture activities' waiting and processing times is a critical requirement in business process simulation.

To address the first requirement, we propose to modify the cost function used in the DLd so that, if two activities are parallel, we do not penalize transposed occurrences of these activities in the compared traces. To this end, we first analyze the input event log in order to discover pairs of parallel activities using the so-called *alpha concurrency oracle*. The alpha concurrency oracle states that in an event log, two activities (a, b) are parallel if sometimes b directly follows a and sometimes a directly follows b. The alpha algorithm uses these heuristics for process discovery [5]. Note that these heuristics is not fail-proof. More fine-grained concurrency oracles have been proposed in the literature [9], but they are more complex to calculate, and they are not fail-proof either. In other words, we could refine this idea by using more fine-grained concurrency oracles at the price of higher computational cost.

Given the concurrency relation || between activities returned by the alpha concurrency oracle, we modify the cost function used in the DLd so that an occurrence of an activity b can be replaced by an occurrence of a parallel activity c without penalty, provided that both activities b and c occur in both input traces. In other words, if these activities co-occur in both input traces, their occurrences are interchangeable.

To address the second requirement, we draw inspiration from [24], which defined a variant of the DLd for timed words. The idea is that if two events in the compared traces have the same activity label, but their waiting and processing times do not match, we assign a penalty when matching this pair of events proportional to the difference between their timestamps. This penalty is normalized so that it is between zero and one.

Based on the above ideas, we propose a modified version of the DLd, namely the *Business Process Trace Distance (BPTD)*. To define the BPTD measure, we first introduce some notations.

We define an event as a tuple $e = (l, p, w)$, where l is a symbol taken from the alphabet of all possible activity labels L, p is the processing time and w the event waiting time of the activities – $p, w \in \mathbb{R}+$. Moreover, let $\mathscr{E}$ is the set of all possible events, i.e.

$$\mathscr{E} = \{(l, p, w) |\ l \in L;\ p, w \in \mathbb{R}+\} \tag{4.1}$$

A trace is a non-empty sequence of events $\sigma = \langle e_1, e_2, \ldots, e_n \rangle$ such that $e_i = (l_i, p_i, w_i) \in \mathscr{E}, 1 \leq i \leq n$. The set of all process traces is $\mathscr{S}$. An event log $\mathscr{L}$ is a set of traces from $\mathscr{S}$ and $\mathscr{K}$ is the number of traces in the event log.

$$\mathscr{L} = \{\sigma_i | \sigma_i \in \mathscr{S}, 1 \leq i \leq \mathscr{K}\} \tag{4.2}$$

Given two traces $\sigma, \sigma' \in \mathscr{S}$, the DLd between σ and σ' is the output of following recursive function when initially invoked with $i = |\sigma|$ and $j = |\sigma'|$:

$$d(\sigma, \sigma', i, j) = min \begin{cases} 0 & if\ i = j = 0 \\ d(\sigma, \sigma', i-1, j) + c(\sigma\langle i \rangle, \sigma'\langle j \rangle) & if\ i\ >\ 0 \\ d(\sigma, \sigma', i, j-1) + c(\sigma\langle i \rangle, \sigma'\langle j \rangle) & if\ j\ >\ 0 \\ d(\sigma, \sigma', i-1, j-1) + c(\sigma\langle i \rangle, \sigma'\langle j \rangle) & if\ i, j\ >\ 0 \\ d(\sigma, \sigma', i-2, j-2) + c(\sigma\langle i \rangle, \sigma'\langle j \rangle) & if\ i, j > 1 \\ & \&\ \sigma\langle i \rangle = \sigma'\langle j-1 \rangle \\ & \&\ \sigma\langle i-1 \rangle = \sigma'\langle j \rangle \end{cases} \tag{4.3}$$

In the classical definition of the DLd, the cost function c returns one for each deletion, insertion, replacement (mismatch), or transposition. BPTD is defined in the same way as the DLd, but it uses a cost function that (i) does not penalize the replacement of one activity by another parallel activity if both of these activities co-occur in the input traces; (ii) introduces a penalty in case two events match (i.e., they have same activity label or correspond to parallel activities) but they have different waiting times or processing times. Formally, the cost function used by BPTD is the following one.

$$c(e,e') = \begin{cases} \beta|p-p'|+(1-\beta)\,|w-w'| & \text{if } l=l' \vee (l \parallel l' \wedge l,l' \in \sigma \wedge l,l' \in \sigma') \\ 1 & otherwise \end{cases} \tag{4.4}$$

...where $|p-p'|$ is the absolute error of the normalized processing time and $|w-w'|$ is the absolute error of the normalized waiting time. The coefficient β represents the weight given to the processing time and 1 - β the weight given to the waiting time (we take $\beta = 0.5$ by default).

Iter.	Sec.	Events				Cost with concurrency	Cost without concurrency
0	σ	$e_1=(a,0.3,0.4)$	$e_2=(b,0.5,0.1)$	$e_3=(c,0.4,0.1)$		0	0
	σ'	$e'_1=(a,0.2,0.4)$	$e'_2=(c,0.5,0.2)$	$e'_3=(b,0.5,0.1)$	$e'_4=(d,0.1,0.1)$		
1	σ	$e_1=(a,0.3,0.4)$	$e_2=(b,0.5,0.1)$	$e_3=(c,0.4,0.1)$		0.042	0.042
	σ'	$e'_1=(a,0.2,0.4)$	$e'_2=(c,0.5,0.2)$	$e'_3=(b,0.5,0.1)$	$e'_4=(d,0.1,0.1)$		
2	σ	$e_1=(a,0.3,0.4)$	$e_3=(c,0.4,0.1)$	$e_2=(b,0.5,0.1)$		0.1	1
	σ'	$e'_1=(a,0.2,0.4)$	$e'_2=(c,0.5,0.2)$	$e'_3=(b,0.5,0.1)$	$e'_4=(d,0.1,0.1)$		
3	σ	$e_1=(a,0.3,0.4)$	$e_3=(c,0.4,0.1)$	$e_2=(b,0.5,0.1)$		0	0
	σ'	$e'_1=(a,0.2,0.4)$	$e'_2=(c,0.5,0.2)$	$e'_3=(b,0.5,0.1)$	$e'_4=(d,0.1,0.1)$		
4	σ	$e_1=(a,0.3,0.4)$	$e_3=(c,0.4,0.1)$	$e_2=(b,0.5,0.1)$	$e_4=(d,0.1,0.1)$	1	1
	σ'	$e'_1=(a,0.2,0.4)$	$e'_2=(c,0.5,0.2)$	$e'_3=(b,0.5,0.1)$	$e'_4=(d,0.1,0.1)$		
					Total cost	**1.142**	**2.042**

Table 9: Exemplification of BPTD measure

The BPTD measure allows us to compare two traces. However, the problem we initially posed was that of comparing two logs. To compare two event logs (the simulated log against the ground-truth log), we define a similarity measure, namely *Event Log Similarity (ELS)*, as the mean between the pairing of each trace in one log with a trace in the other log. Specifically, we search for the pairing of traces that minimizes the sum of the Business Process Trace Distances (BPTDs) between the paired traces. We map the problem of pairing the traces of the two logs to the *assignment problem*, and we use the well-known Hungarian algorithm[42] to find the minimal-distance pairing. Given two logs L_1 and L_2 and given a minimal-distance pairing $P = \{(\sigma_1,\sigma_2) \mid \sigma_1 \in L_1 and \sigma_2 \in L_2\}$ between the traces in these logs:

$$ELS(L_1,L_2) = \frac{1}{|P|}(\Sigma_{(t_1,t_2)\in P} BPTD(\sigma_1,\sigma_2))$$

For the optimization and evaluation phases, we use the *ELS* similarity function with L_1 being the simulated log and L_2 being the ground truth.

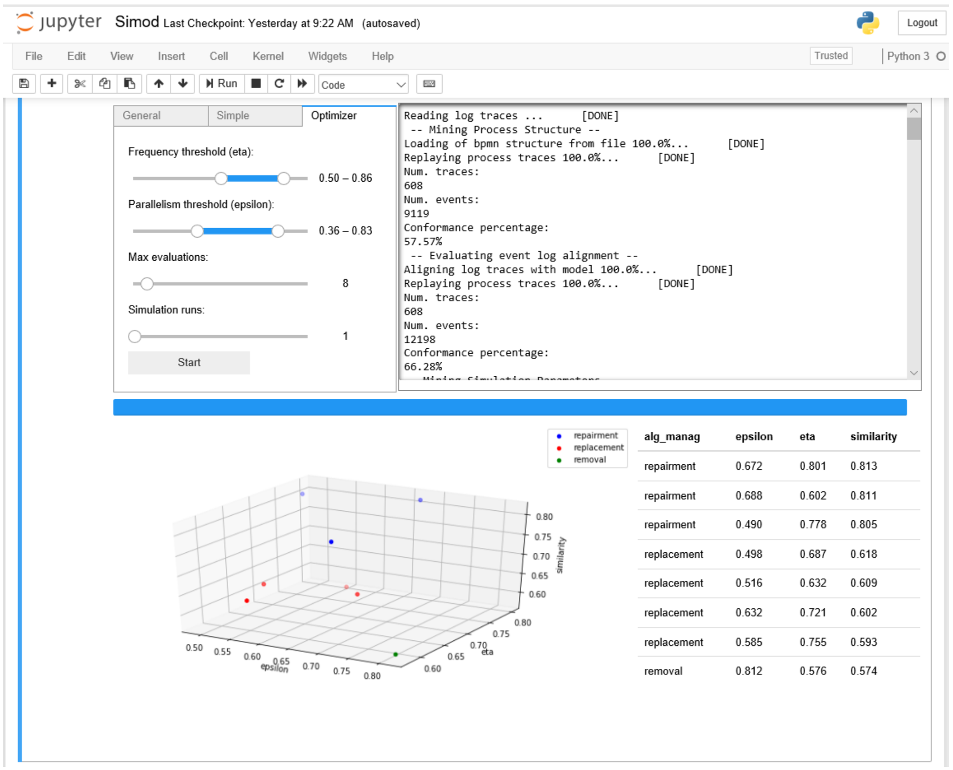
Hyperparameter optimization. The previous subsections described the automatic creation of a BPS model that integrates multiple perspectives of the process. These perspectives are discovered by using one or more Process Mining algorithms. However, critical decisions must be made in each step, either on the techniques to be used or on the parameters' values. For example, low filter values in the BPMN miner can drastically affect the simulation accuracy, creating spaghetti models impossible to reproduce by the simulator. The same can happen when choosing how similar the resources of the discovered pools should be, or when it is required to select the best way to calculate the probabilities of the decision gateways.

In a traditional approach, an expert would manually search for the best combinations by modifying the values of the parameters based on his expertise and intuition. However, this is a time-consuming approach that often leads to far-from-optimal results [90]. Therefore, we propose to use a Tree-structured Parzen Estimator (TPE) as a hyperparameter optimizer [13] to find the best settings based on the historical accuracy of the executed models. TPE is a sequential algorithm that defines on each trial the following parameter configuration. The decision is based on past results and nested functions that select the parameters' values based on a probability distribution and ranges specified for each one. The objective function seeks to minimize the loss, which is calculated as the inverse ELS measure. Table 10 defines the used search space.

Category	**Variable**	**Distribution**	**Range**
Control flow discovery	Parallelism threshold (ε)	Uniform	[0...1]
	Percentile for frequency threshold (η)	Uniform	[0...1]
Log repair technique	Repair	n/a	
	Removal	n/a	
	Replace	n/a	
Conditional branching probabilities	Random	n/a	
	Equiprobable	n/a	
	Discovered	n/a	
Resource pools	Similarity threshold	Uniform	[0...1]

Table 10: Search space definition

Simod Interface. Simod was developed in Python 3.6 and offers a user interface for Jupyter Notebooks. The user interface allows to select an event-log in XES or CSV format and to decide how to generate and analyze the model (see Fig. 16). From the interface, it is possible to define the preprocessing parameters manually or use the hyperparameter optimizer. In both cases, Simod provides information on the execution of the discovery steps and the results obtained from the similarity evaluation.



Frequency threshold (eta): 0.50 – 0.86

Parallelism threshold (epsilon): 0.36 – 0.83

Max evaluations: 8

Simulation runs: 1

```
Reading log traces ...      [DONE]
 -- Mining Process Structure --
Loading of bpmn structure from file 100.0%...      [DONE]
Replaying process traces 100.0%...      [DONE]
Num. traces:
608
Num. events:
9119
Conformance percentage:
57.57%
 -- Evaluating event log alignment --
Aligning log traces with model 100.0%...      [DONE]
Replaying process traces 100.0%...      [DONE]
Num. traces:
608
Num. events:
12198
Conformance percentage:
66.28%
```

alg_manag	epsilon	eta	similarity
repairment	0.672	0.801	0.813
repairment	0.688	0.602	0.811
repairment	0.490	0.778	0.805
replacement	0.498	0.687	0.618
replacement	0.516	0.632	0.609
replacement	0.632	0.721	0.602
replacement	0.585	0.755	0.593
removal	0.812	0.576	0.574

Figure 16: Simod interface

4.2. Evaluation

The proposed method has been implemented in an open-source tool, namely Simod, which is packaged as a Python application with Jupiter Notebook interface.[6] Simod takes as input an event log in XES format and produces a BPS model ready to be simulated using the BIMP simulator [6][7]. The source code of Simod can also be configured to produce models for the Scylla simulator [70], but BIMP is used as the default simulator because Scylla only supports a restricted set of probability distributions, thus restricting the space of configuration options.

Using Simod, we conducted an evaluation to address the following specific research questions:

SQ1 What is the accuracy of the BPS models generated by the proposed method?

SQ2 To what extent the hyperparameter optimization step improves the accuracy of the BPS models?

[6]Tool and datasets available at https://github.com/AutomatedProcessImprovement/Simod/tree/v2.0.0

[7]Available at http://bimp.cs.ut.ee

4.2.1. Datasets

A pre-requisite to discover a BPS model from an event log is that the events in the log should have both start and end timestamps. Unfortunately, this pre-requisite is not fulfilled by publicly available real-life logs such as those in the 4TU Collection of event logs.[8] As an alternative, we validate the proposed approach using one synthetic event log and two real-life event logs that satisfy the above requirement.

The first event log is a synthetic log generated from a model unavailable to the authors of a *P2P* process[9]. This log is the same log used as a running example in Sec. 4.1. The second log stems from an *Academic Credentials Recognition (ACR)* process at the University of Los Andes in Colombia. The log comes from a deployment of a Business Process Management System (BPMS), specifically Bizagi. The model corresponding to this log was not available to the authors of this thesis. The third log is that of a *Manufacturing Production (MP)* process, exported from an Enterprise Resource Planning system [48]. The tasks in this process refer to steps (or "stations';) in the manufacturing process. The characteristics of these logs are given in Table 11.

Event log	**Num. traces**	**Num. events**	**Num. activities**	**Avg. activities per trace**	**Max. activities per trace**	**Mean duration**	**Max. duration**
P2P	608	9119	21	14.9	44.0	21.5 days	108 days 7 hours
ACR	954	4962	16	5.2	23.0	14.9 days	135 days 19 hours
MP	225	4503	24	20.0	177.0	20.6 days	87 days 10 hours

Table 11: Statistics of the event logs

We choose these three event logs because they have distinct characteristics regarding the number of activities, traces, resources, and times, and they come from different domains. The *ACR* event log is the one that contains the highest number of traces and the least number of average activities per trace, while *MP* is the log with the least traces and the highest number of average activities per trace. On the other hand, the *P2P* log corresponds to the scenario in which the event log is generated from a process model defined in ideal conditions, so a 100% fit between the two and high accuracy in the simulation is expected. The *ACR* log corresponds to a self-service process executed on a BPMS. It is a relatively complex process, which delivers a service to hundreds of users and involves over a dozen workers. Finally, the *MP* event log is the scenario in which the process structure is unknown and where the behavior of the resources can significantly affect the structure of the process.

[8] `https://data.4tu.nl/repository/collection:event_logs_real`

[9] The log is part of the academic material of the Fluxicon Disco tool –`https://fluxicon.com`

4.2.2. Experiment setup

To address the research questions, we compared a baseline variant of our method against the hyperparameter optimized variant. The hyperparameter values used for the baseline are given in the Baseline column in Table 12. We define the values of the hyperparameters in this scenario from the default values suggested by the authors of the discovery algorithms and from values suggested in the literature about the construction of simulation models [25]. The hyperparameter-optimized variant explores 100 hyperparameter combinations per log using the TPE optimization method mentioned above. The range of hyperparameter values given to the TPE optimizer is shown in Table 12.

Parameter	Baseline values	Optimizer ranges
Parallelism threshold (ε)	0.1	[0...1]
Percentile for frequency threshold (η)	0.4	[0...1]
Log repair technique	Removal	Repair, Removal, Replace
Conditional branching probabilities	Equiprobable	Random, Equiprobable, Discovered
Similarity threshold	0.5	[0...1]
Inter-arrival times	Exponential PDF	PDF Discovered from data
Processing times	Exponential PDF	PDF Discovered from data

Table 12: Values of the configuration scenarios.

We discovered BPS models from each event log using both the baseline and the optimized method. We then simulated each BPS model ten times, and each time, we measured the similarity between the simulated log and the ground-truth using the ELS measure introduced earlier. A total of 3000 simulated event logs were generated: 1000 per input event log (10 simulation runs for each of 100 parameter combinations tested by the hyperparameter optimizer).

The results of a simulation may vary from one run to another due to its stochastic nature. To ensure that these stochastic variations are not responsible for the conclusions drawn from the experiments, we applied the single-queue Mann-Whitney U test to validate the significance of the optimization accuracy improvement. The alternative hypothesis was that the accuracy of the best configuration is higher than the accuracy of the baseline scenario. In this evaluation, we compared the ten simulation runs of the best BPS found by the optimizer against the ten simulation runs of the baseline model.

4.2.3. Results

Fig. 17 shows the accuracy results in terms of ELS when simulating the baseline model and the BPS models generated by the optimizer for each of the event logs. Since the search space has six dimensions, we organize the results by grouping the hyperparameters according to the stage to which they belong, i.e., preprocessing or processing. Sub-figures 17a, 17c and 17e contrast the accuracy of the BPS models with the η and ε values used for discovering the process model and with the non-conformances handling technique. Sub-figures 17b, 17d, and 17f plots the accuracy against the similarity threshold for discovering resource pools and against the method for determining the conditional branching probabilities.

In the *P2P* event log, we observe a linear association between using the Removal technique and model accuracy (see Figure 17a). In the processing stage (see Fig. 17b), the branching probability discovery technique, in conjunction with similarity threshold values of around 0.85, has a strong linear association with accuracy. We also observe that the hyperparameter-optimized method outperforms the baseline scenario.

The *MP* log led to different results. In the preprocessing stage (see Fig. 17c), we observe an association between using the Repair technique and higher accuracy values. In the processing stage (see Fig. 17d), there is a weak positive association between using the Equiprobable allocation of probabilities and model accuracy. The similarity threshold does not appear to be a determining factor. Compared to the baseline model, the optimizer could find better configurations; however, the choice of non-conformance handling technique played a more critical role in this log.

In the *ACR* event log, in the preprocessing stage (see Fig. 17e), the Removal technique and the Repair technique have a strong association with higher model accuracy values, especially with eta values greater than 0.4. However, none of these two non-conformance handling techniques shows clear superiority. On the other hand, and surprisingly, the Equiprobable allocation of branching probabilities leads to higher accuracy (see Fig. 17f). As was the case in the *MP* log, the similarity threshold does not appear to be a determining factor. For this log, the baseline method led to slightly lower accuracy than the hyperparameter-optimized method.

As shown in Table 13, the Mann-Whitney U test found that the differences in accuracy between the simulation runs of the optimized BPS model and the runs of the baseline BPS model are statistically significant for each of the three logs (i.e., null hypothesis rejected with p-values$<$ 0.5).

log	**Baseline (ELS)**	**Optimizer (ELS)**	**p-values**
P2P	0.8175	0.8622	9.13E-05
MP	0.2741	0.3118	9.13E-05
ACR	0.8699	0.8800	0.01882

Table 13: One tail Mann-Whitney U test results

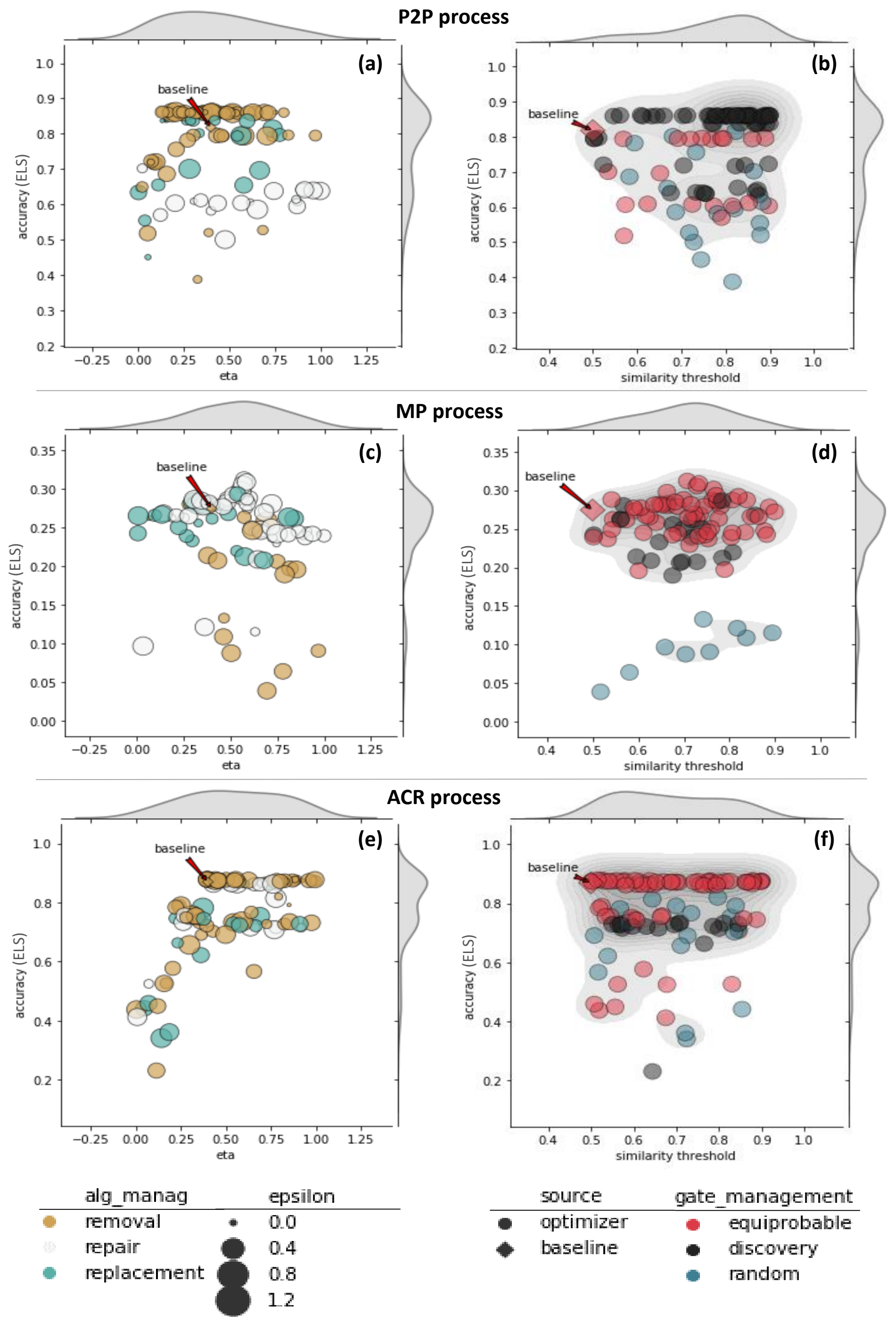


Figure 17: Accuracy results of the hyperparameter optimizer execution vs. baseline scenario in terms of ELS (the bigger the better)

4.3. Conclusions

In this chapter, we presented the first version of Simod, a method for automated discovery of business process simulation models from event logs. In response to the GAPs in the state-of-the-art, we also defined a measure for assessing the accuracy of a BPS model relative to an event log and used a hyperparameter optimizer to maximize the accuracy of the final BPS model.

The empirical evaluation of the method has shown that the hyperparameter optimization method significantly improves the accuracy of the resulting BPS model relative to an approach where default parameters were used. Also, it was observed that the best configuration found varies from one event log to another, further emphasizing the need for automated hyperparameter optimization in this setting. A threat to the validity of this study is that each parameter is extracted using a particular algorithm because our focus was on the automatic discovery of simulation models and the search for greater precision concerning the process model used as the basis. One possible extension of this tool could include multiple extraction options for each parameter.

Another limitation of the present study is that the evaluation is restricted to one synthetic and two real-life event logs. As such, the generalizability of the results is limited: The results might be different for other event logs, and as shown in the evaluation, particularly for event logs for which the automated process discovery technique does not manage to discover an accurate process model. This is because a prerequisite to discover a BPS model from an event log is that the events in the log should have both start and end timestamps. Unfortunately, this prerequisite is not easily fulfilled by publicly available real-life logs. As such, the generalizability of the results is limited: The results might be different for other event logs, and as shown in the evaluation, particularly for event logs for which the automated process discovery technique does not manage to discover an accurate process model. We lift this limitation in the following chapters, in which we extended the number of event logs used in the evaluation.

It should be noted that further research has been developed over this tool, including the discovery of other perspectives that were not included in this evaluation. In [26] we extended the discovery method to include calendars that restrict the availability of resources in running time as occurs in real life. This extension and others carried out on the discovery pipeline aimed at improving its accuracy are explained and evaluated in Chapter 6.

5. LEARNING ACCURATE GENERATIVE MODELS OF BUSINESS PROCESSES

Deep learning techniques have recently found applications in the field of predictive process monitoring. These techniques allow us to predict, among other things, what will be the subsequent events in a case, when they will occur, and which resources will trigger them. They also allow us to generate entire execution traces of a business process or even entire event logs, which opens up the possibility of using such models for process simulation. However, to enable its use in this simulation, it is necessary to close some gaps in the state-of-the-art works (see Sec. 3). Specifically in this chapter, we focus on:

GAP3 DL generative models must be able to generate not only remaining sequences of events (suffixes) but also complete logs starting from scratch (prefixes of size zero), and

GAP4 the generated logs must include, as a minimum, the category of the event, associated resource, and start and end times of the activities allowing the evaluation of the performance of the scenarios.

The first part of this chapter will describe the steps of the method and the evaluated LSTM architectures. In the second part of the chapter, we will present two evaluations; the first one compares different instantiations of the proposed architectures in terms of preprocessing and post-processing choices, and the second compares the proposed approach with three comparable baselines in predictive process monitoring, showing that the proposed approach outperforms previously proposed LSTM architectures targeted at this problem.

5.1. Approach Description

This section describes the method we propose to build predictive models from business process event logs. The method uses LSTM networks to predict sequences of successive events, their timestamp, and associated resource pools. Three LSTM architectures are proposed that seek to improve the network's learning concerning the different events logs characteristics. These architectures can accurately reproduce the behavior observed in the log. Fig. 18 summarizes the phases and steps for building predictive models with our method.

5.1.1. Phase 1: Pre-processing

Data transformation.. We carried out specific preprocessing tasks according to the features' nature (i.e., categorical or continuous) to improve the data quality for feeding the models.

In the case of the *categorical features*, our main concern was their transformation into numerical values to be interpreted by the LSTM network without increase the dimensionality of the features. In contrast with previous approaches that only

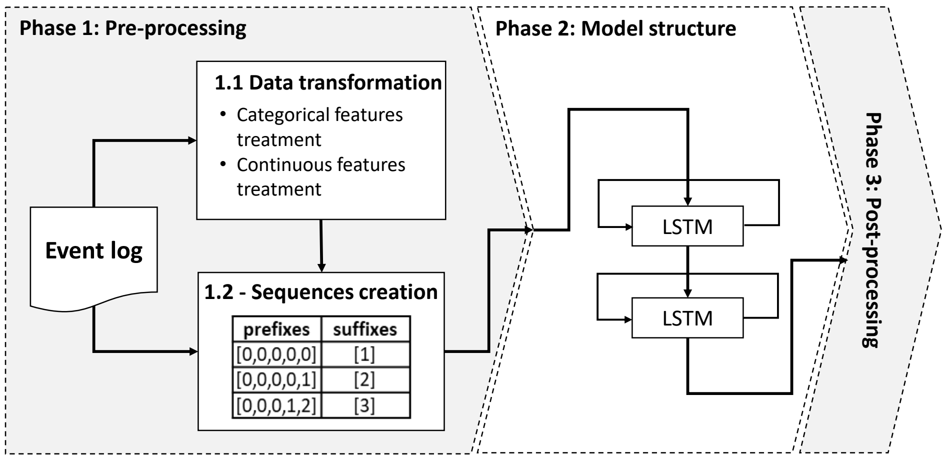


Figure 18: Phases and steps for building predictive models

encode the sequence of activities of the process, our method uses activities and resources as categorical features. The inclusion of multiple categorical features seeks to use more information about the process behavior to improve the prediction accuracy. However, this multiplicity increases the number of potential categories exponentially. To deal with this problem, we propose grouping resources into roles and embedded dimensions for encoding the activities categories.

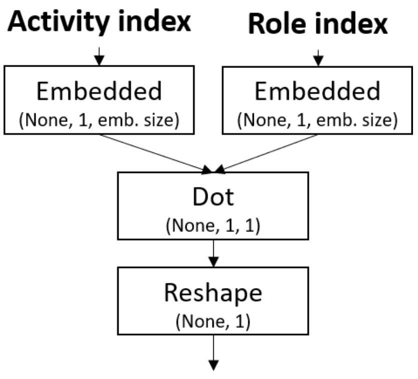
We group resources into roles using the algorithm described by Song and Van der Aalst [81]. This algorithm seeks to discover resource pools (called roles in [81]) based on the definition of activity execution profiles for each resource and the creation of a correlation matrix of similarity of those profiles. This algorithm allowed us to reduce the number of categories of this feature but keep enough information to help the LSTM network make the differences between events clearer.

The use of embedded dimensions helps control the exponential features growth while providing more detailed information about the associations between features. To exemplify its advantages, let us take the event log *BPIC2012*[1], which has 36 activities and five roles. If we use one-hot encoding to represent each unique pair activity-role in the event log, 180 new features composed of 179 zeroes are needed. This huge increment in dimensionality is mainly composed of useless information. In contrast, using embedded dimensions to map the categories into an n-dimensional space, only four dimensions are needed to encode the log, in which each coordinate corresponds to a unique category. In this dimensional space, the distances between points represent how close one activity performed by one role is concerning the same activity performed by another role. This additional information can help the network to understand the associations between events and differentiate them among similar ones.

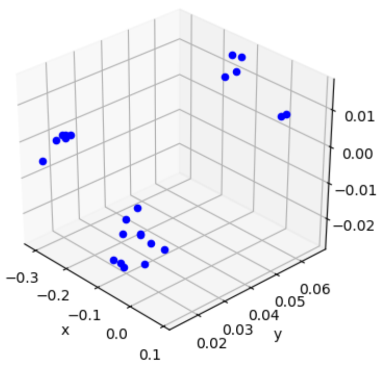
We train an independent network to coordinate the embedded dimensions. The training network was fed with positive and negative examples of association be-

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

tween features, allowing the network to identify and locate near features with similar characteristics. We determine the number of embedded dimensions as the fourth root of the number of categories to avoid a possible collision between them, according to a standard recommendation used in the NLP community[2]. The generated values were exported and reused in all the experiments as non-trainable parameters, which allowed not to increase the complexity of the models. Fig. 19a presents the architecture of the network used for training the embedded layers, whereas Fig. 19b shows an example of a generated embedding 3d space for activities.



(a) Embedded layers



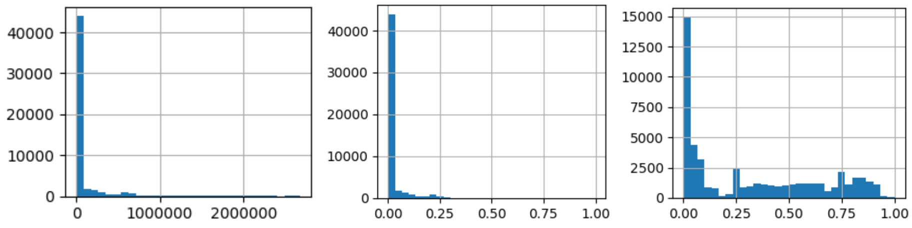
(b) Generated space

Figure 19: (a) The architecture of the model we used to encode the activities and roles categorical features. (b) Each point corresponds to one activity or role; the distances between the points were determined based on the relations the embedding model found in the data.

In the case of *continuous features*, our primary concern was the scaling of the values in a $[0, 1]$ range to avoid interpretation errors in the features' importance by our predictive models. Our model uses the relative time between activities as categorical input, calculated as the time elapsed between the complete timestamp of one event and the complete timestamp of the previous one. The relative time is easier to interpret by the models and helps calculate the events' timestamp in a trace. However, due to the nature of each event log, the relative time may have high variability. If the feature scaling is performed without care, helpful information about the process behavior, such as time bottlenecks or abnormal behaviors, can be lost. For example, we can observe this problem when scaling enabling times. If the times in the event log present high variability or distribution different from the normal distribution, normalization could distort the perception of data; however, the use of log-normalization makes variations in relative times observable. Fig. 20 illustrates the results of scaling the enabling times in the event log *BPIC2012*. Therefore, we consider multiple scaling techniques (i.e., log normal-

[2]`https://www.tensorflow.org/guide/feature_columns`

ization, normalization, z-transform, dividing over the max) as one of the hyper-parameters to be evaluated to determine which best fits the characteristics of the relative times.



(a) Original (b) Maximum value (c) Log-normalized

Figure 20: Scaling of relative times using different methods

Sequences creation.. To create the input sequences and expected events used to train the predictive network, we decided to extract n-grams of fixed sizes from each event log trace. N-grams allow control of the temporal dimensionality of the input and bring clear patterns of sub-sequences describing the execution order of activities, roles, or relative times, regardless of the length of the traces. One n-gram is extracted for each time-step of the process execution and is done for each attribute independently. Meaning that we have three independent inputs for our models: activities, roles, and relative times. Table 14 presents five n-grams extracted from the case id 174770 of the *BPIC2012* event log. The numbers used to represent the activities, roles, and times correspond to the indexes and scaled values in the data transformation step.

Time Step	**Activities**	**Roles**	**Relative times**
0	[0 0 0 0 0]	[0 0 0 0 0]	[0. 0. 0. 0. 0.]
1	[0 0 0 0 10]	[0 0 0 0 5]	[0. 0. 0. 0. 0.]
2	[0 0 0 10 7]	[0 0 0 5 5]	[0. 0. 0. 0. 4.73e-05]
3	[0 0 10 7 18]	[0 0 5 5 1]	[0. 0. 0. 4.73e-05 5.51e-01]
4	[0 10 7 18 5]	[0 5 5 1 1]	[0. 0. 4.73e-05 5.51e-01 1]
5	[10 7 18 5 18]	[5 5 1 1 1]	[0. 4.73e-05 5.51e-01 1 7.48e-04]

Table 14: N-grams for case number 174770 of the BPIC2012 event log

5.1.2. Phase 2: Model Structure Definition

LSTM networks were used as the core of our predictive models since they are a well-known and proven technology to handle sequences, as the nature of a business process event log. Fig. 21 illustrates the basic architecture of our network consisted of an input layer for each attribute, two stacked LSTM layers, and a dense output layer. The first LSTM layer provides a sequence output rather than a single value output to feed the second LSTM layer. Additionally, the categorical attributes have an embedded layer for their coding.

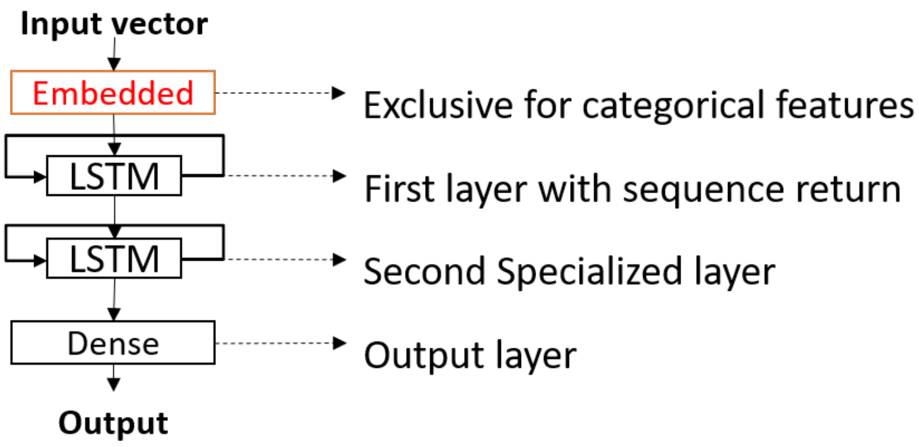
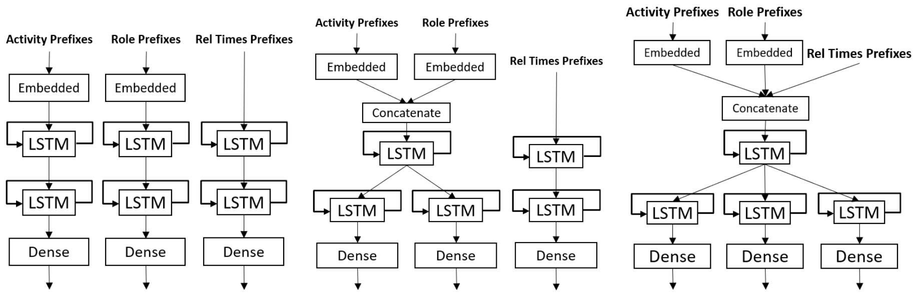


Figure 21: Baseline architecture

Likewise, we tested three variants of the baseline architecture, as is shown in Fig. 22. The hypothesis behind these approaches is that sharing information between the layers can help to differentiate execution patterns. However, the changes produced by the different nature of the variables could interfere with the clear identification of patterns in a log, generating noise in learning. The specialized architecture (see Fig. 22a) does not share any information; in fact, it can be understood as three independent models. The shared categorical architecture (see Fig. 22b) concatenates the activities and roles inputs and shares the first LSTM layer. This architecture is expected to avoid the possible noises introduced by sharing information between features of different nature (i.e., categorical or continuous). The full-shared architecture (see Fig. 22c) concatenates all the inputs and completely shares the first LSTM layer.



(a) Specialized (b) Shared categorical (c) Full shared

Figure 22: Tested architectures

5.1.3. Phase 3: Post-processing

Our technique is capable of generating complete traces of business processes starting from a zero prefix size. This is done by using continuous feedback of the model with each newly generated event until the generation of a finalization event (hallucination). Although previous approaches have used this technique, we also incorporate arguments of the maxima (arg-max) and random choice as techniques for selecting the next predicted event category. Arg-max is the technique commonly used to select the next category of a prediction and consists of selecting the one with the highest predicted probability. In theory, this technique should work well for prediction tasks, such as the most likely category of the next event, given an incomplete case.

However, if the model is used generatively, arg-max is inconvenient because it reduces the variability in the kind of sequences generated. Arg-max provokes the model to get trapped in the highest probabilities. To avoid this bias in the model, we use the random selection (following the predicted probability distribution) method for the category selection of the next predicted event, increasing the variability in the generated traces. This method also helped us reveal what the neural network has learned from the dynamics observed in the event log. Of course, introducing a random element forces us to perform multiple repetitions to reduce the stochastic variability finding the convergence in the measurements. We evaluate both approaches (arg-max and random selection) in the evaluation of results.

5.2. Evaluation

This section describes two evaluations. The first experiment compares different instantiations of the three proposed architectures in terms of preprocessing and post-processing choices. The second experiment compares the proposed approach with three comparable baselines [84, 27, 49] for the next event, suffix, and remaining time prediction tasks (see Sec. 3.2).

5.2.1. Datasets

For this evaluation, we use nine real-life event logs from different domains and with diverse characteristics. Most of the event logs are public and were taken from the *Business Process Intelligence Challenge (BPIC)*, below the description of their characteristics:

- The *Helpdesk*[3] event log contains records from a ticketing management process of the helpdesk of an Italian software company.
- The two event-logs within *BPIC2012*[4] are related to a loan application process from a German financial institution. This process is composed of three

[3]https://doi:10.17632/39bp3vv62t.1

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

sub-processes from which we used the W sub-process to allow the comparison with the existing approaches [83, 27].

- The event log within *BPIC2013*[5] is related to a Volvo's IT incident and problem management. We used the complete cases to learn generative models.
- The five event-logs within *BPIC2015*[6] contain data on building permit applications provided by five Dutch municipalities during four years. We subdivided the original event log into five parts (one per municipality). Initially, all the event logs were specified at a sub-processes level, including more than 345 activities. Therefore, we preprocessed the logs to manage them at a phases level by following the steps described in [87].

Each event log's Sequence Flow (SF) was classified as simple, medium, and complex according to its composition in terms of the number of traces, events, activities, and length of the sequences. In the same way, we classify the Time Variability (TV) as steady or variable according to the relation between the mean and max duration of each event log (see Table 15).

Event log	**Num. traces**	**Num. events**	**Num. activities**	**Avg. activities per trace**	**Max. activities per trace**	**Mean duration**	**Max. duration**	**SF**	**TV**
Helpdesk	4580	21348	14	4.6	15	40.9 days	59.2 days	simple	stedy
BPIC2012	13087	262200	36	20	175	8.6 days	137.5 days	complex	stedy
BPIC2012W	9658	170107	7	17.6	156	8.8 days	137.5 days	complex	stedy
BPIC2013	1487	6660	7	4.47	35	179.2 days	6 years, 64 days	simple	irregular
BPIC2015-1	1199	27409	38	22.8	61	95.9 days	4 years, 26 days	medium	irregular
BPIC2015-2	832	25344	44	30.4	78	160.3 days	2 years, 341 days	medium	irregular
BPIC2015-3	1409	31574	40	22.4	69	62.2 days	4 years, 52 days	medium	irregular
BPIC2015-4	1053	27679	43	26.2	83	116.9 days	2 years, 196 days	medium	irregular
BPIC2015-5	1156	36.234	41	31.3	109	98 days	3 years, 248 days	medium	irregular

Table 15: Event logs description

5.2.2. Experiment 1: Comparison of LSTM architectures and processing options

This experiment compares different instantiations of our approach in terms of their ability to learn execution patterns and reproduce the behavior registered in the event log reliably. Accordingly, we use the LSTM models to generate full event logs starting from size zero prefixes, and we then compare the generated traces against those in the original log.

Experiment setup. We used two metrics to assess the similarity of the generated event logs. The Damerau-Levenshtein distance (DLd) algorithm measures

[5]https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07

[6]https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1

the distance between sequences, regarding the number of editions necessary for one string character to be equal to another. This algorithm penalizes each action, such as insertion, deletion, substitution, and transposition. Their measurements are commonly scaled by using the maximum size between the two sequences that are compared. Therefore, we use its inverse to measure the similarity between a generated sequence of activities or roles and a sequence observed in the actual event log. Then, a higher value implies a higher similarity among the sequences.

We trivially lift the DLd (which applies to pairs of strings or traces) to measure the difference between two event logs by pairing each generated trace with the most similar trace (w.r.t. DLd) of the ground-truth log. Once the pairs are formed (generated trace, ground-truth trace), we calculate the mean DLd. We use the MAE metric to measure the error in predicting timestamps. This measure is calculated by measure the absolute error value of the distance between an observation and the predicted value and then calculating the average value of these magnitudes. We apply this for each pair (generated trace, ground-truth trace).

We used holdout as the validation technique by splitting the event logs into two folds: 70% for training and 30% for validation. We used the first fold to train 2000 models (approximately 220 models per event log). These models were configured with different preprocessing techniques and architectures. The configurations' values were selected by randomly sampling the complete search space of 972 combinations. With each trained model, we generated 15 new event logs using the two techniques for the selection of the next activity described in Sec. 5.1.3. In total, we evaluate more than 32000 generated event logs.

Results. Table 16 summarizes the similarity results of the event logs generated from different model instantiations. The *Pre-processing, model definition, and Post-processing* columns describe the configuration used in each phase for building the evaluated models. The *DLd act* and *DLd roles* columns measure the similarity in the predicted categorical attributes. The *MAE* column corresponds to the mean absolute error of the cycle time of the predicted traces.

These results indicate that it is possible to train models that learn and reliably reproduce the observed behavior patterns of the original logs using our approach. Additionally, the results suggest that for the LSTM models, it is more challenging to learn sequences with a more extensive vocabulary than longer sequences. The models require more examples to learn these patterns, as seen in the *BPIC2012* and *BPIC2015* results. Both logs have more than 30 activities, but there is a significant difference in the amount of traces (see Table 15). The high degree of similarity of the *BPIC2012* also suggests that using embedded dimensions to handle many event types improves the accuracy.

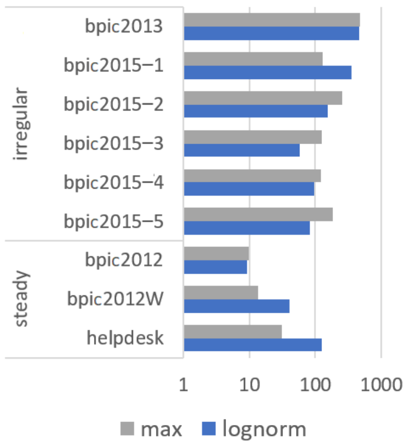
Concerning the architectural components evaluated in this experiment, we analyze them according to the phases to build generative models: preprocessing, model structure and hyper-parameters selection, and prediction. Regarding the *pre-processing phase*, Fig. 23a illustrates how logs with little time variability

Event log	Pre-processing		Model definition	Post-processing	DLd act.	DLd roles	MAE (days)
	Scaling	N-gram size	Architecture	Selection method			
BPIC2012	max	15	specialized	random	**0.8929**	0.7888	9
	max	15	shared cat.	random	0.885	**0.8998**	9
	lognorm	15	concatenated	random	0.8426	0.856	**4**
BPIC2012W	max	15	specialized	random	**0.8742**	0.8245	11.8
	max	15	concatenated	random	0.7902	**0.8552**	7.3
	max	10	concatenated	random	0.7855	0.8329	**5.9**
BPIC2013	lognorm	10	joint	arg max	0.5442	0.698	**242.6**
	max	15	shared cat.	random	**0.7209**	0.8139	471.5
	lognorm	15	shared cat.	random	0.4416	**0.8475**	472.5
BPIC2015-1	max	10	concatenated	random	**0.4397**	0.8048	76.6
	lognorm	10	specialized	random	0.4228	**0.8498**	79.3
	lognorm	10	concatenated	arg in ax	0.3642	0.5922	**40.1**
BPIC2015-2	lognorm	10	shared cat.	arg in ax	**0.3737**	0.6228	159.4
	max	15	concatenated	random	0.3462	**0.8612**	158.3
	max	10	shared cat.	arg max	0.0431	0.1691	**89**
BPIC2015-3	lognorm	10	concatenated	random	**0.4616**	0.8501	53.2
	lognorm	5	concatenated	random	0.4456	**0.8729**	54.4
	lognorm	15	concatenated	arg max	0.4255	0.7786	**39.6**
BPIC2015-4	lognorm	5	concatenated	arg max	**0.4034**	0.7188	96
	lognorm	5	specialized	random	0.3609	**0.8248**	98.8
	max	5	shared cat.	arg max	0.0581	0.0968	**71.1**
BPIC2015-5	lognorm	15	specialized	random	**0.3633**	0.8653	84.1
	max	5	shared cat.	random	0.3323	**0.9019**	82.5
	lognorm	10	concatenated	arg max	0.3228	0.6547	**49.6**
Helpdesk	max	5	shared cat.	random	**0.9568**	**0.9869**	42.1
	max	5	joint	arg max	0.5773	0.7368	**7.3**

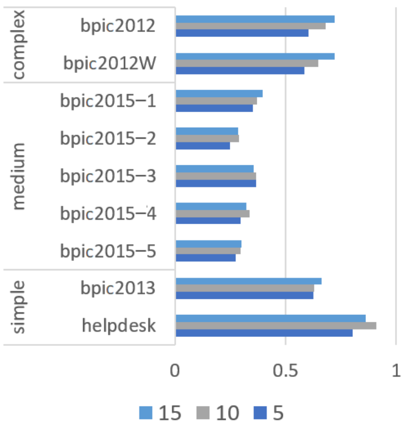
Table 16: Similarity results in event logs for different configurations

present better results using max value as scaling technique. In contrast, logs that have an irregular structure have lower MAE using log-normalization. Additionally, Fig. 23b presents the results of DLd similarity in the use of n-grams of different sizes concerning the structure of event logs. We can observe that longer n-grams have better results for logs with longer traces, showing a steady increasing trend. In contrast, it is not clear a trend for the event logs with medium and simple structures. Therefore, the use of long n-grams should be reserved for logs with very long traces.

Regarding the *model structure definition phase*, Fig. 24 illustrates that the concatenated architecture has the lowest overall similarity. In contrast, the model architecture that only shares information between attributes of the same nature has the median best performance. However, it is not very distant from the specialized architecture, albeit a wider spread. This result implies that sharing information between attributes of different nature can generate noise in the network's patterns



(a) Scaling of relative times results



(b) N-gram size selection

Figure 23: Preprocessing phase components comparison

identification, thus hindering the learning process.

Regarding the *prediction phase*, Fig. 25 shows how random choice outperforms arg-max in all the event logs. This behavior is even more clear in the event logs with longer and complex traces. The results suggest that random choice is advisable to assess the learning process despite the event log structure.

5.2.3. Experiment 2: Comparison against baselines

This experiment aims to assess the relative performance of our approach at the task of predicting the next event, the remaining sequence of events (i.e., suffixes), and the remaining time, for trace prefixes of varying lengths.

Experiment setup. For *next event prediction*, we feed each model with trace prefixes of increasing length, from 1 up to the length of each trace. We predict the next event for each prefix and measure the accuracy (percentage of correct predictions). For suffix and remaining time prediction, we also feed the models with prefixes of increasing lengths. However, this time, we allow the models to hallucinate until the end of the case is reached. The remaining time is then computed by subtracting the timestamp of the last event in the prefix from the timestamp of the last hallucinated event. As in [84], we use DLd as a measure of similarity for suffix prediction and MAE for remaining time prediction. For next event and suffix prediction, we use [84, 27, 49] as baselines while for remaining time prediction, we only use [84], since [27] and [49] cannot handle this prediction task. We only use the *Helpdesk*, *BPIC2012W*, and *BPIC2012* event logs because these are the only logs for which results are reported in the three baselines. The results reported for [27] for the *Helpdesk* and *BPIC2012* event logs correspond to the re-implementation of this technique reported in [49].

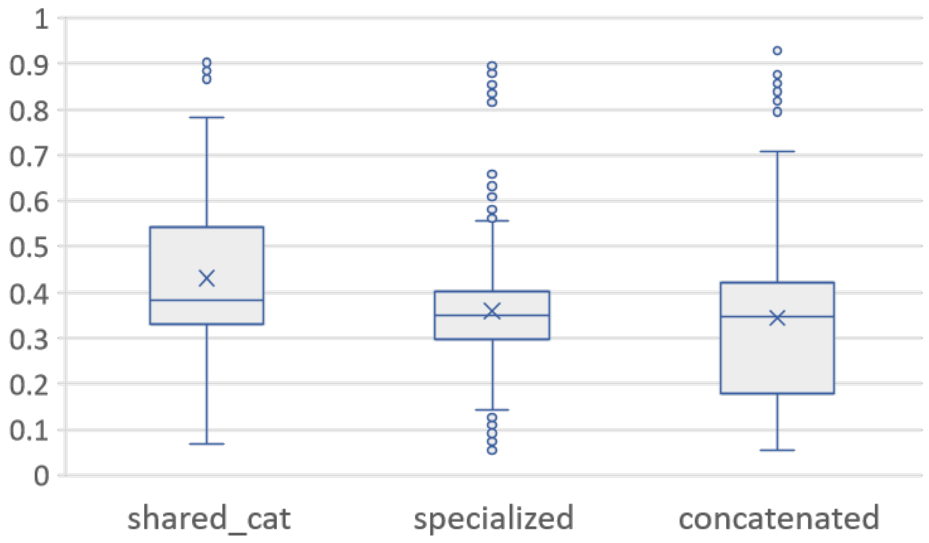
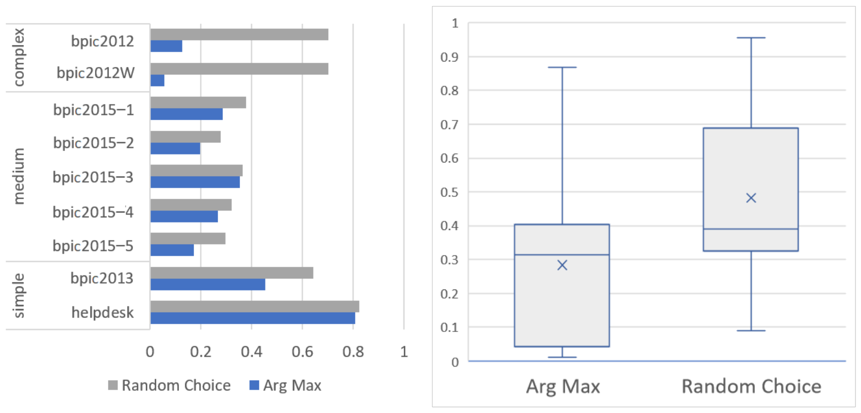


Figure 24: Shared layers' overall similarity



(a) Similarity per structure type (b) Overall similarity

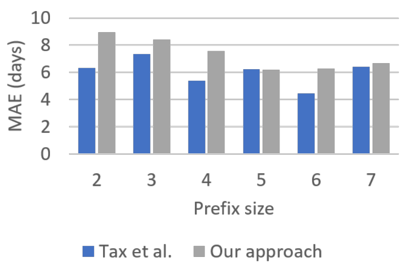
Figure 25: Comparison of next-event selection methods

Results. Table 17 summarizes the average accuracy for the next-event prediction task and the average similarity between the predicted suffixes and the actual suffixes. For the task of next-event prediction, our approach performs similar to that of Evermann et al. [27] and Tax et al. [84] while slightly outperforming them for the *BPIC2012W* event log. However, it underperforms the approach by Lin et al. [49] For the task of suffix prediction, our approach outperforms all baselines, including that of Lin et al. These results suggest that the measures adopted for the dimensionality control of the categorical features allow our approach to achieve consistently good performance even for long sequences.

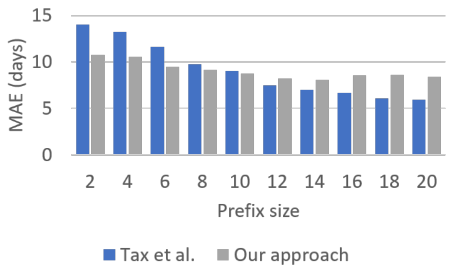
	Next event accuracy			Suffix prediction distance		
Implementation	**Helpdesk**	**BPIC2012W**	**BPIC2012**	**Helpdesk**	**BPIC2012W**	**BPIC2012**
Our approach	0.789	*0.778*	0.786	*0.917*	*0.525*	*0.632*
Tax et al.	0.712	0.760		0.767	0.353	
Everman et al.	0.798	0.623	0.780	0.742	0.297	0.110
Lin et al.	*0.916*		*0.974*	0.874		0.281

Table 17: Next event and suffix prediction results

Fig. 26 presents the MAE for remaining cycle time prediction. Even though the objective of our technique is not to predict the remaining time, it achieves similar performance at this task relative to Tax et al. – slightly underperforming it in one log and slightly outperforming it for long suffixes in the other log.



(a) Helpdesk



(b) *BPIC2012W*

Figure 26: Results of remaining cycle-time MAE in days

5.3. Conclusions

This chapter outlined an approach to train LSTM networks to predict the next event type in a case, its timestamp, and the role associated with the event. By iteratively predicting the next event, the approach can also predict the remaining sequence of events of a case (the suffix), and it can also generate entire traces from scratch. The method consists of a preprocessing phase (scaling and n-gram encoding), an LSTM training phase, and a post-processing phase (selecting the predicted next event among the likely ones). The chapter compared several options for each of these phases concerning generating full traces that closely match the traces in the original log. The evaluation shows that using longer n-grams increases the accuracy in logs with long traces; log-normalization is a suitable scaling method for logs with high variability, and randomly select the next event using the probabilities produced by the LSTM leads to a most comprehensive variety of traces and higher accuracy, relative to always choosing the most likely next event.

The evaluation also showed that the proposed approach outperforms existing

LSTM-based approaches for predicting the remaining sequence of events and their timestamps starting from a given prefix of a trace.

This work helped lay the foundations for applying generative deep learning models in BPS. Indeed, in its essence, a process simulator is a generative model that produces sets of traces consisting of event types, resources, and timestamps. From that set of traces, it is possible to calculate performance measures such as waiting times, cycle times, and resource utilization. While process simulators rely on interpretable process models (e.g., BPMN models), any model capable of generating traces of events composed of an event type (activity label), a timestamp, and a resource, can, in principle, be used to simulate a process.

The generative models trained with the method that we present in this chapter can generate complete traces of processes starting from prefixes of zero size, allowing the generation of event records composed of multiple traces, providing a solution to the research GAP3. Furthermore, in response to the research GAP4, this technique can generate events composed of event types, resources, and timestamps resembling those produced by traditional simulation models. These logs can potentially be used to analyze the behavior of a process –as it could be done using more conventional simulation models—, with the advantage that DL models do not require the definition of an explicit process model or the distributions of times and resources as a basis. In the next chapter, we will evaluate and compare in a more detailed way the characteristics of the conventional DDS generative models vs. the DL generative models introduced in this chapter.

6. DATA-DRIVEN SIMULATION VS DEEP LEARNING

A generative model is a statistical model capable of generating new data instances from previously observed ones. In the context of business processes, a generative model creates new execution traces from a set of historical traces, also known as an event log. Up to this point in this thesis, we introduce two types of generative business process models: In Chapter 4 we proposed a data-driven simulation approach, and in Chapter 5 we proposed an approach to train deep learning generative models. Until now, these two approaches have evolved independently, and their relative performance has not been studied. This chapter fills this gap by empirically comparing a data-driven simulation approach with multiple deep learning approaches for building generative business process models. The study sheds light on the relative strengths of these two approaches and raises the prospect of developing hybrid approaches that combine these in preparation to solve the research GAP5 (cf. Sec. 1.2).

In the Sections 6.1 and 6.2, we describe the adaptations made to the generative models to compare them under the same conditions. Later, in Section 6.3 we exhaustively compared the relative precision and characteristics of the DDS versus DL generative models under the same conditions, including the number of event logs (eleven in total).

6.1. Generative data-driven process simulation models

In this research, we use an updated version of the Simod tool that we introduce in the Chapter 4 as a representative DDS method because, to the best of our knowledge, it is the only fully automated method for discovering and tuning business process simulation models from event logs. The use of methods with automated tuning steps, such as that of [74, 68], would introduce two sources of bias in the evaluation: (i) a bias stemming from the manual tuning of simulation parameters, which would have to be done separately for each event log using limited domain knowledge; and (ii) a bias stemming from the fact that the DDS model would be manually tuned while the deep learning models are automatically tuned as part of the model training phase.

By using Simod, we ensure a fair comparison, insofar as we compare a DDS method that performs automatic data-driven tuning of model parameters with deep learning methods that, likewise, tune their parameters (weights) to fit the data. Fig. 27 depicts the updated steps of the Simod method, namely "Structure discovery" and "Time-related parameters discovery".

In the **structure discovery** stage, Simod extracts a BPMN model from data and guarantees its quality and coherence with the event log. The first step is the *Control Flow Discovery*, using the SplitMiner algorithm [11], which is known for being one of the fastest, simple, and accurate discovery algorithms. Next, Simod applies *Trace alignment* to assess the conformance between the discovered pro-

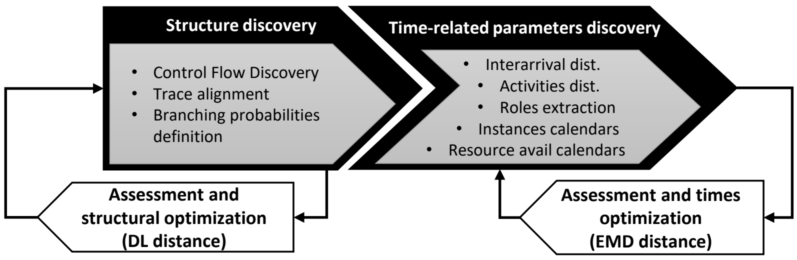

Structure discovery
Control Flow Discovery
Trace alignment
Branching probabilities definition
Time-related parameters discovery
Interarrival dist.
Activities dist.
Roles extraction
Instances calendars
Resource avail calendars
Assessment and structural optimization (DL distance)
Assessment and times optimization (EMD distance)

Figure 27: Pipeline of Simod to generate process models.

cess model and each trace in the input log. The tool provides options for handling non-conformant traces via removal, replacement, or repair to ensure full conformance, which is needed in the following stages. Then Simod discovers the model *branching probabilities*, offering two options: assign equal values to each conditional branch or computing the conditional branches' traversal frequencies by replaying the event log over the process model. Once all the structural components are extracted, they are assembled into a single data structure that a discrete event simulator can interpret (e.g., Bimp). The simulator is responsible for reproducing the model at discrete moments, generating an event log as a result. Then, Simod uses a hyperparameter optimization technique to discover the configuration that maximizes the Control-Flow Log Similarity (CFLS) between the produced log and the ground truth.

In the **time-related parameters discovery** stage, Simod takes as input the structure of the optimized model, extracts all the simulation parameters related to the perspective of times, and assembles them in a single BPS model. The extracted parameters correspond to the PDF of *Inter-arrival times*, the *Resource pools* involved in the process, the *Activities durations*, the *instances' generation calendars* and the *resources' availability calendars*. The PDFs of inter-arrival times and activities durations are discovered by fitting a collection of possible distribution functions to the data series, selecting the one that yields the smallest standard error. The evaluated PDFs correspond to those supported by the BIMP simulator (i.e., normal, log-normal, gamma, exponential, uniform, and triangular distributions). The resource pool is discovered using the algorithm proposed by Song and Van der Aalst [81]; likewise, the resources are assigned to the different activities according to the frequency of execution. Finally, Simod discovers calendar expressions that capture the resources' time availability, restricting the hours they can execute tasks. Similarly, the tool discovers case creation timetables that limit when the process instances can be created. Once all these simulation parameters are compiled, Simod again uses the hyperparameter optimization technique to discover the configuration that minimizes the Earth Mover's Distance (EMD) distance between the produced log and the ground truth.

The final product of the two optimization cycles is a model that reflects the structure and the simulation parameters that best represent the time dynamics observed in the ground truth log.

6.2. Generative deep learning models of business processes

DL models have been applied in several subfields of PM, particularly in the context of *predictive process monitoring*. Predictive process monitoring is a class of PM techniques that are concerned with predicting, at runtime, some properties about the future state of a case, e.g., predicting the next event(s) in an ongoing case or the remaining time until completion of the case.

Fig. 28 depicts the main phases for the construction and evaluation of DL models for predictive process monitoring. In the first phase (preprocessing), the events in the log are transformed into (numerical) feature vectors and grouped into sequences, each sequence corresponding to the execution of a case in the process (a trace). Next, a model architecture is selected depending on the prediction target. In this respect, different architectures may be used for predicting the type of the next event, its timestamp, or both (see Chapter 3).

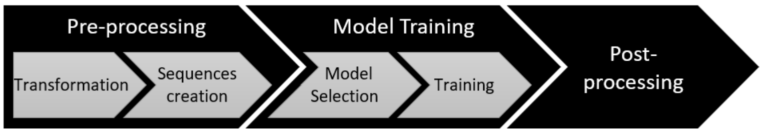


Figure 28: Phases and steps for building DL models

In this chapter, we tackle the problem of generating traces consisting of event types (i.e., activity labels) and timestamps. As we reviewed in the Chapter 2, one of the earliest studies to tackle this problem in predictive process monitoring was [84], which proposed an approach to predict the type of the next event in an ongoing case, as well as its timestamp using LSTM architectures. The same study showed that this approach could be effectively used to generate the remaining sequence of timestamped events, starting from a given case prefix. However, this approach cannot handle high-dimensional inputs due to its reliance on the one-hot encoding of categorical features. As a result, its accuracy deteriorates as the number of categorical features increases.

We lift this limitation in the LSTM approach that we introduce in Chapter 5, which extends the approach of [84] with two mechanisms to handle high-dimensional input, namely n-grams and embeddings, and integrates a mechanism for avoiding temporal instability, namely Random Choice next-event selection. A more recent study [86] proposes to use a GAN method to train an LSTM model capable of predicting the type of the next event and its timestamp. The authors show that this GAN approach outperforms classical training methods (for predicting the next event and timestamp) on specific datasets.

In the empirical evaluation reported in this chapter, we retain our LSTM approach (see Chapter 5) and the GAN approach of [86] as representative methods for training generative DL models from event logs. We selected these methods because they can generate both the type of the next event in a trace and its timestamp. This means that if we iteratively apply these methods starting from an empty sequence, via an approach known as *hallucination*, we can generate a sequence of events such that each event has one timestamp (the end timestamp). Hence, these methods can be used to produce entire sequences of timestamped events. Therefore, they can be used to generate event logs comparable to those that DDS methods generate, with the difference is that the above DL training methods associate only one timestamp to each event, whereas DDS methods associate both a start and end timestamp to each event. Accordingly, we need to adapt the above two DL methods to generate two timestamps per event for full comparability. In the following subsections, we describe these approaches and how we adapted them to fit this requirement.

6.2.1. DeepGenerator approach

Next, we describe the adaptation of the LSTM approach that we proposed in Chapter 5 from now on DeepGenerator. The *DeepGenerator* approach trains a generative model by using attributes extracted from the original event log. Specifically, DeepGenerator uses activities, roles, relative times (start and end timestamps), and contextual times (day of the week, time during the day). The original approach can produce traces consisting of triplets (event type, role, timestamp), where a *role* refers to a group of resources who are able to perform a given activity (e.g., "Clerk" or "Sales Representative"). This chapter adapts DeepGenerator to generate sequences of triplets of the form (event-type, start-timestamp, end-timestamp). Each triplet captures the execution of an activity of a given type (event-type) together with the timeframe during which the activity was executed. In this chapter, we do not attach roles to events in order to make the DeepGenerator method fully comparable to Simod as we discussed in Sect. 6.4.

In the **preprocessing phase** (cf. Fig. 28), DeepGenerator applies encoding and scaling techniques to transform the event log depending on the data type of each event attribute (categorical vs. continuous). Categorical attributes (activities and roles) are encoded using embeddings to keep the data dimensionality low, as this property enhances the neural network's performance. Meantime, start and end timestamps are relativized and scaled over a range of $[0,1]$. The relativization is carried out by first calculating two features: the activities duration and the time-between-activities. The duration of an activity (a.k.a. the processing time) is the difference between its complete timestamp and its start timestamp. The time-between-activities (a.k.a. the waiting time) is the difference between the start timestamp of an activity and the end timestamp of the immediately preceding activity in the same trace. All relative times are scaled using normalization or log-normalization, depending on the variability of the times in the event log. Once the features are encoded, DeepGenerator executes the *sequences' creation* step to extract n-grams, which allow better handling of long sequences. One n-gram is generated for each step of the process execution, and this is done for each attribute independently. Hence, DeepGenerator uses four independent inputs: activity prefixes, role prefixes, relativized durations, and relativized time-between-activities.

In the **model training phase**, one of two possible architectures is selected for training. These architectures, depicted in Fig. 29, vary depending on whether they share intermediate layers. The use of shared layers sometimes helps to better differentiate between execution patterns. These architectures were defined and explained in more detail in Section 5.1.2. DeepGenerator uses LSTM layers or GRU layers. Both of these types of layers are suitable for handling sequential data, with GRU layers sometimes outperforming LSTM layers [51, 20].

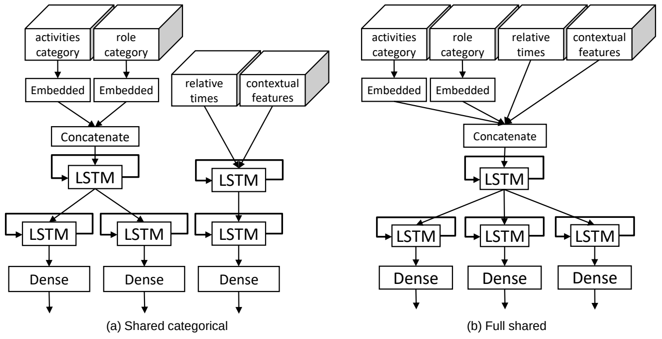


Figure 29: (a) this architecture concatenates the inputs related with activities and roles, and shares the first layer, (b) this architecture completely shares the first layer. Despite the role's prefixes are encoded and predicted, their accuracy is not evaluated.

Finally, the **post-processing phase** uses the resulting DL model in order to generate a set of traces (i.e., an event log). DeepGenerator takes each generated trace and uses the classical hallucination method to repeatedly ask the DL model to predict the next event given the events observed so far (or given the empty trace in the case of the first event). This step is repeated until we observe the "end of trace" event. At each step, the DL model predicts multiple possible "next events", each one with a certain probability. DeepGenerator selects among these possible events randomly weighted by the associated probabilities. This mechanism turns out to be the most suitable for the task of generating complete event logs by avoiding getting stuck in the higher probabilities [18].

6.2.2. LSTM-GAN approach

The approach proposed by [86] trains LSTM generative models using the GAN strategy. The strategy proposed by the authors consists of two LSTM models, one generative and one discriminative, that are trained simultaneously through a game of adversaries. In this game, the generative model has to learn how to confuse a discriminative model to avoid distinguishing real examples from fake ones. As the game unfolds, the discriminative model becomes more capable of distinguishing between fake and real examples, thus forcing the generator to improve the generated examples. Fig. 30 presents the general architecture of the GAN strategy proposed in [86]. We performed modifications in every phase of this approach to be able to generate full traces and entire event logs to make it fully comparable with DDS methods.

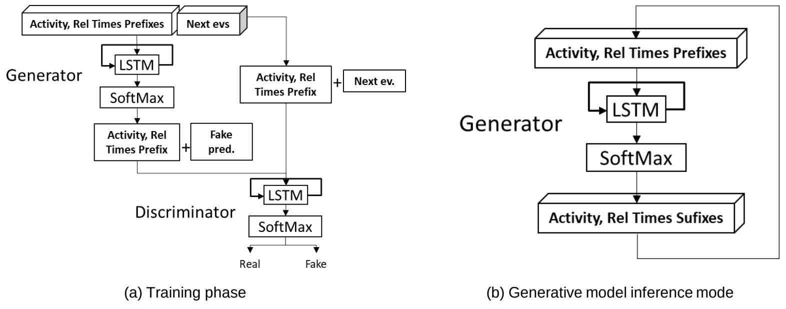

Activity, Rel Times Prefixes
Next evs
Generator
LSTM
SoftMax
Activity, Rel Times Prefix
Next ev.
Activity, Rel Times Prefix
Fake pred.
Discriminator
LSTM
SoftMax
Real
Fake
(a) Training phase
Activity, Rel Times Prefixes
Generator
LSTM
SoftMax
Activity, Rel Times Sufixes
(b) Generative model inference mode

Figure 30: (a) Training strategy, (b) Inference strategy.

In the **preprocessing phase** (cf. Fig. 28), the features corresponding to the activity's category and relative times are encoded and transformed. The model uses one-hot encoding for creating a binary column for each activity and returning a sparse matrix. We adapted the model to enable the prediction of two timestamps instead of one. The original method by [86] only handles one continuous attribute per event (the end timestamp). We added another continuous attribute to capture the time (in seconds) between the end of the previous event in the sequence and the start of the current one. This additional attribute is herein called the *inter-activity times*. Next, the inter-activity times are then rounded up to the granularity of days to create a so-called design matrix composed of the one-hot encoded activities and the scaled inter-activity times. Then, we create the prefixes and the expected events in order to train the models. Since the original model was intended to train models starting from a k-sized prefix, all the smaller prefixes were discarded, and the prediction of the first event of a trace was not considered. We also adapted the model to be trained to predict zero-size prefixes. For this purpose, we extended the number of prefixes considered by including a dummy start event before each trace and by applying right-padding to the prefixes. This modification of the input implied updating the loss functions in order to consider the additional attribute.

In the **model training phase**, [86] trained specialized models to predict the next event from prefixes of a predefined size. While this approach is suitable for predicting the next event, it is not suitable for predicting entire traces of unknown size. Therefore, we train a single model with a prefix of size five. This strategy is grounded on the results of the evaluation reported by [80], from which the authors concluded that increasing the size of the prefix used by the LSTM models (beyond a size of five events) does not substantially improve the model's predictive accuracy.

Finally, in the **post-processing phase** we take the complete predicted suffix to feedback the model instead of considering only the first event predicted by the model (see Fig. 30). We carry out this operation to take advantage of the fact that the original generative model is a sequence-to-sequence model, which receives a sequence of size k and predicts a sequence of size k. The empirical evidence reported by [18] shows that concatenating only the last event predicted by the model generates a rapid degradation in the model's long-term precision, as the model gets trapped in always predicting the most probable events. Accordingly, we use random selection to select the next type of event.

6.3. Evaluation

This section presents an empirical comparison of DDS and DL generative process models. The evaluation aims at addressing the following questions: what is the relative accuracy of these approaches when it comes to generating traces of events without timestamps? and what is their relative accuracy when it comes to generating traces of events with timestamps?

6.3.1. Datasets

We evaluated the selected approaches using eleven event logs that contain both start and end timestamps. In this evaluation, we use real logs from public and private sources and synthetic logs generated from simulation models of real processes:

- The event log of a manufacturing production *Manufacturing Production (MP)* process is a public log that contains the steps exported from an Enterprise Resource Planning system [48].
- The event log of a *Purchase-to-Pay (P2P)* process is a public synthetic log generated from a model not available to the authors.[1]
- The event log from an *Academic Credentials Recognition (ACR)* process of a Colombian University was gathered from its BPM system (Bizagi).
- The W subset of the *BPIC2012*[2] event log, which is a public log of a loan application process from a Dutch financial institution. The W subset of this log is composed of the events corresponding to activities performed by human resources (i.e. only activities that have a duration).
- The W subset of the *BPIC2017*[3] event log, which is an updated version of the *BPIC2012* log. We carried out the extraction of the W-subset by following the recommendations reported by the winning teams participating in the *BPIC* 2017 challenge [4].

[1]The log is provided as part of the Fluxicon Disco tool – `https://fluxicon.com/`
[2]`https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`
[3]`https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b`
[4]`https://www.win.tue.nl/bpi/doku.php?id=2017:challenge`

- We used three private logs of real-life processes, each corresponding to a scenario of different sizes of data for training. The *UB* log belongs to an undisclosed banking process, and the *CALL* log belongs to the call center records of a helpdesk process. Both of them correspond to large-size training data scenarios. The *INS* logs belong to an insurance claims process, corresponding to a small size of training data scenario. For confidentiality reasons, only the detailed results of these three event logs will be provided.
- We used three synthetic logs generated from simulation models of real-life processes [5]. The selected models are complex enough to represent scenarios in which occur parallelism, resource contention, or scheduled waiting times. From these models, we generate event logs varying the number of instances representing greater or lesser availability of training data. The *CVS retail pharmacy (CVS)* event-log is a large-size training data scenario from a simulation model of an exercise described in the book Fundamentals of Business Process Management [25]. We generated the *Confidential Large-size (CFL)* and *Confidential Small-size (CFS)* event logs from an anonymized confidential process. They were used to representing scenarios of large and small size training data.

Table 18 characterizes these logs according to the number of traces and events. The *BPIC2017W* and *BPIC2012W* logs have the largest number of traces and events, while the *MP*, *CFS* and *P2P* have fewer traces but a higher average number of events per trace.

Size	Type of source	Event log	Num. traces	Num. events	Num. activities	Avg. activities per trace	Avg. duration	Max. duration
LARGE	REAL	*UB*	70512	415261	8	5.89	15.21 days	269.23 days
LARGE	REAL	*BPIC2017W*	30276	240854	8	7.96	12.66 days	286.07 days
LARGE	REAL	*BPIC2012W*	8616	59302	6	6.88	8.91 days	85.87 days
LARGE	REAL	*CALL*	3885	7548	6	1.94	2.39 days	59.1 days
LARGE	SYNTHETIC	*CVS*	10000	103906	15	10.39	7.58 days	21.0 days
LARGE	SYNTHETIC	*CFL*	2000	44373	29	26.57	0.76 days	5.83 days
SMALL	REAL	*INS*	1182	23141	9	19.58	70.93 days	599.9 days
SMALL	REAL	*ACR*	954	4962	16	5.2	14.89 days	135.84 days
SMALL	REAL	*MP*	225	4503	24	20.01	20.63 days	87.5 days
SMALL	SYNTHETIC	*CFS*	1000	21221	29	26.53	0.83 days	4.09 days
SMALL	SYNTHETIC	*P2P*	608	9119	21	15	21.46 days	108.31 days

Table 18: Event logs description

6.3.2. Evaluation measures

We use a generative process model to generate an event log (multiple times) and then we measure the average similarity between the generated logs and a ground-

[5] `https://zenodo.org/record/4264885`

truth event log. To this end, we define four measures of similarity between pairs of logs: Event Log Similarity (ELS), Mean Absolute Error (MAE) of cycle times, the Earth Mover's Distance (EMD) of the histograms of activity processing times, and Control-Flow Log Similarity (CFLS). It is important to clarify that the generation of time and activity sequences is not a classification task. Therefore, the precision and recall metrics traditionally used for predicting the next event do not apply. Instead, we use symmetric distance metrics (i.e., that penalize the differences between a and b in the same way as from b to a) that measure both precision and recall at the same time, as explained in [47].

ELS and **MAE** metrics measure the differences between two event logs at trace level as we already defined in Chapter 4. ELS is a measure that combines the control-flow and the temporal perspective. Meanwhile, MAE is focused on the temporal similarity between two logs. To complement them, we include two metrics, one that focuses solely on the differences in terms of control-flow at the trace level (CFLS) and the other that gives us a perspective of the global differences in log times (ELS).

CFLS is defined based on a measure of distance between pairs of traces: one trace coming from the original event log and the other from the generated log. We first convert each trace into a sequence of activities (i.e. we drop the timestamps and other attributes). In this way, a trace becomes a sequence of symbols (i.e. a string). We then measure the difference between two traces using the DLd, which is the minimum number of edit operations necessary to transform one string (a trace in our context) into another. The supported edit operations are insertion, deletion, substitution, and transposition. Transpositions are allowed without penalty when two activities are concurrent, meaning that they appear in any order, i.e. given two activities, we observe both AB and BA in the log. Next, we normalize the resulting DLd distance by dividing the number of edit operations by the length of the longest sequence. We then define the *control-flow trace similarity* as the one minus the normalized DLd distance. Given this trace similarity notion, we pair each trace in the generated log with a trace in the original log, in such a way that the sum of the trace similarities between the paired traces is maximal. This pairing is done using the Hungarian algorithm for computing optimal alignments [42]. Finally, we define the **CFLS** between the real and the generated log as the average similarity of the optimally paired traces.

The cycle time MAE is a rough measure of the temporal similarity between the traces in the original and the generated log. It does not take into account the timing of the events in a trace – only the cycle time of the full trace. To complement the cycle time MAE, we use the EMD between the normalized histograms of the mean durations of the activities in the ground-truth log vs the same histogram computed from the generated log. The EMD between two histograms H1 and H2 is the minimum number of units that need to be added to, removed to, or transferred across columns in H1 in order to transform it into H2. The EMD is zero if the observed mean activity durations in the two logs are identical, and it tends to one the more they differ.

The above leaves us with a metric that measures the control-flow differences at trace level (CFLS), one that measures the time differences at trace level (MAE), one that integrates both perspectives (ELS), and finally, one that provides a global perspective on the temporal differences at event log level (EMD).

6.3.3. Experiment setup

The aim of the evaluation is to compare the accuracy of DDS models vs DL models discovered from event logs. Fig. 31 presents the pipeline we followed.

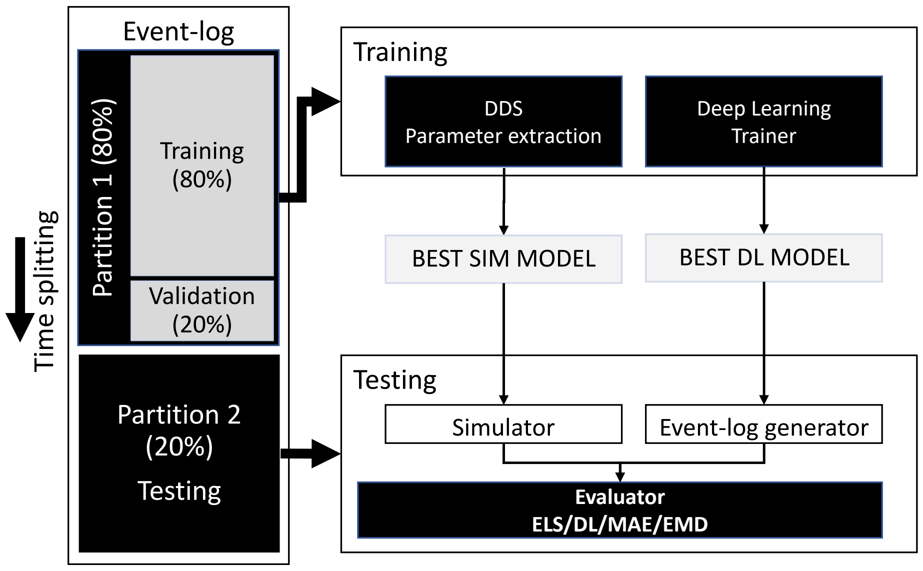


Figure 31: Experiment pipeline

We used the hold-out method with a temporal split criterion to divide the event logs into two folds: 80% for training and 20% for testing. Next, we use the training fold to train the DDS and the DL models. The use of temporal splits is common in the field of predictive process monitoring (from which the DL techniques included in this study are drawn from) as it prevents information leakage [18, 86].

We use the first 80% of the training fold to construct candidate DDS models and the remaining 20% for validation. We use Simod's hyperparameter optimizer to tune the DDS model (see the tool's two discovery stages in Sec. 6.1). First, the optimizer in the structure discovery stage was set to explore 15 parameter configurations, with five simulation runs per configuration. At this stage, we kept the DDS model that gave the best results on the validation sub-fold in terms of CFLS averaged across the five runs. Second, the optimizer in the time-related parameters discovery stage was set to explore 20 parameter configurations with five simulation runs per configuration. Then, we hold the DDS model that gave the best results on the validation sub-fold in terms of EMD averaged across the five runs. As a result of the two stages, Simod found the best model in both structure and time dynamics. We defined the number of optimizer trials in each stage by considering the differences in the search space's size in each stage (see Simod's model parameters in Table 19).

The results showed that the best possible value is reached in fewer attempts than expected. Fig 32 shows the log *P2P* in which the best model was found in the first optimization stage at trial 10 and in the second stage at trial 13.

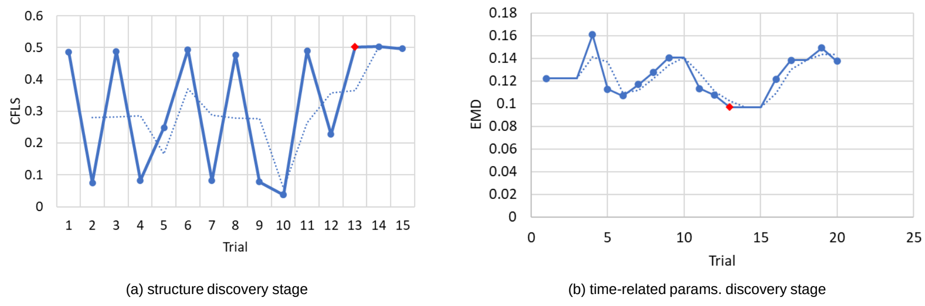


Figure 32: (a) In CFLS units the higher the best, (b) in EMD units the lower the best

Next, we apply the random search for hyperparameter optimization for each family of generative models (LSTM and GRU). Similar to the DDS approach, we explore 40 random configurations with five runs each, using 80% of the training fold for model construction and 20% for validation. This sample size was chosen to ensure a confidence level of 95 % with a confidence interval of 6 (see LSTM/GRU's model parameters in Table 19).

Model	Stage	Parameter	Distribution	Values
Simod	Structure discovery	Parallelism threshold (ε)	Uniform	[0...1]
		Percentile for frequency threshold (η)	Uniform	[0...1]
		Conditional branching probabilities	Categorical	{Equiprobable, Discovered}
	Time-related parameters discovery	Log repair technique	Categorical	{Repair, Removal, Replace}
		Resource pools similarity threshold	Uniform	[0...1]
		Resource availability calendar support	Uniform	[0...1]
		Resource availability calendar confidence	Uniform	[0...1]
		Instances creation calendar support	Uniform	[0...1]
		Instances creation calendars confidence	Uniform	[0...1]
LSTM/GRU	Training	N-gram size	Categorical	{5, 10, 15}
		Input scaling method	Categorical	{Max, Lognormal}
		# units in hidden layer	Categorical	{50, 100}
		Activation function for hidden layers	Categorical	{Selu, Tanh}
		Model type	Categorical	{Shared Categorical, Full Shared}

Table 19: Parameter ranges and distributions used for hyperparameter optimization

In the case of the LSTM-GAN implementation, as proposed by the authors [86], we dynamically adjust the size of hidden units in each layer being twice the input's size. Additionally, we use 25 training epochs, a batch of size five, and a prefix size of five.

The above led us to one DDS, one LSTM, one GRU, and one LSTM-GAN model per log. We then generated five logs per retained model. To ensure comparability, we create each generated log of the same size (number of traces) as the testing fold of the original log. We then compare each generated log with the testing fold using the ELS, CFLS, EMD, and MAE measures defined above. We report the mean of each of these measures across the 5 logs generated from each model to smooth out stochastic variations.

6.3.4. Findings

Fig. 33 presents the evaluation results of CFLS, MAE and ELS measures grouped by event log size and source type. Table 3 presents the exact values of all metrics sorted by metric, event log size, and source type. The Event-log column identifies the evaluated log; meanwhile, the GRU, LSTM, LSTM (GAN), and SIMOD columns present the accuracy measures. Note that ELS and CFLS are similarity measures (the higher, the better), whereas MAE and EMD are error/distance measures (lower is better).

The results show a clear dependence of training data size on the models' accuracy. Simod presents a greater similarity in the control flow generation for small logs in three of the five evaluated logs, as shown by the CFLS results. In the remaining two logs, the measure is not far from the best-reported values. Simod obtains the smallest errors in four of the five logs in terms of MAE, which leads to higher ELS similarity in four of the five logs. However, the LSTM model presents the best CFLS results in five of the six evaluated large logs, where the GRU model approaches better in the remaining one. The LSTM model obtains the lowest errors in the MAE in four of the six logs, whereas the LSTM-GAN model approaches better in the remaining two. The difference between the DL and Simod models for the MAE measure is constant and dramatic in some cases, such as the *CALL* log. In this log, Simod generates a difference almost four times greater than that reported by the DL models. This result can be due to a contention of resources that is non-existent in the ground truth.

When analyzing the ELS measure, which joins the two perspectives of control flow and time distance, the LSTM model obtains the greatest similarity in five out of six models and the GRU model in the remaining one. The LSTM-GAN model does not obtain a better result in this metric due to its poor performance in control flow similarity. The LSTM-GAN model's low performance is because the temporal stability of the models' predictions declines rapidly, despite having a higher precision in predicting the next event as demonstrated in [86]. This result also indicates overfitting on the models, preventing the generalization of this approach for this predictive task.

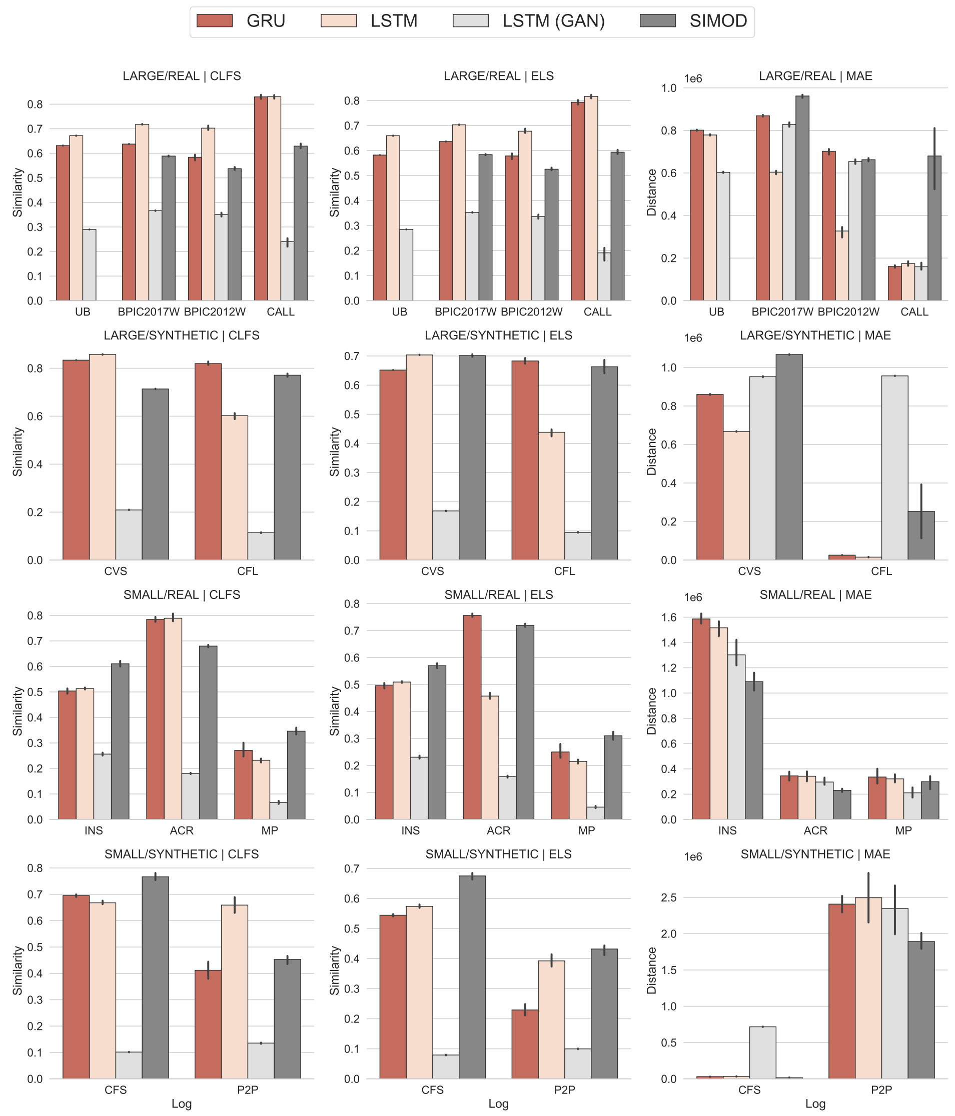


Figure 33: In the first column the CLFS are presented in similarity units (the higher, the better), the second column presents the MAE results in distance units, and the third column presents the ELS results (the higher, the better)

On the one hand, the results indicate that DDS models perform well when capturing the occurrence and order of activities (control-flow similarity) and that this behavior is independent of the training dataset size. A possible explanation for this result is that event logs of business processes (at least those included in this evaluation) follow certain normative pathways captured sufficiently by automatically discovered simulation models. However, Deep Learning models, especially LSTM models, outperform the DDS models if a sufficiently large training dataset is available.

On the other hand, Deep Learning models are more accurate when capturing the cycle times of the cases in the large logs (cf. the lower MAE for DL models vs. DDS models). Here, we observe that both DDS and DL models achieve similar EMD values, which entails that both models predict activities' processing times with similar accuracy. Therefore, we conclude that the differences in temporal accuracy (cycle time MAE) between DL and DDS models come from the fact that DL models can better predict the waiting times of activities rather than the processing times. [6]

The inability for DDS models to accurately capture the waiting times can be attributed to the fact that these models rely on the assumption that the waiting times can be fully explained by the availability of resources. In other words, DDS models assume that resource contention is the sole cause of waiting times. Furthermore, DDS models operate under the assumption of *eager resources* as discussed in Sec. 1.1 (i.e., resources start an activity as soon as they are allocated). Conversely, DL models try to find the best fit for observed waiting times without any assumptions about the behavior of the resources involved in the process.

6.3.5. Discussion

The results of the empirical evaluation reflect the trade-offs between DDS models and deep learning models. Indeed, these two families of models strike different trade-offs between modeling capabilities (expressive power) on the one hand, and interpretability on the other.

The results specifically put into evidence the limitations in modeling capabilities of DDS models. Such limitations arise both along the control-flow level perspective (sequences of events) and along the temporal perspective (timestamps associated to each event).

From a control-flow perspective, DDS models can only generate sequences that can be fully parsed by a business process model. In the case of Simod, this model is a BPMN model. The choice of modeling notation naturally introduces a representational bias [2]. For example, free-choice workflow nets – which have the expressive power of BPMN models with XOR and AND gateways [28] – have limitations that prevent them from capturing certain synchronization constructs [41]. Adopting a more expressive notation may reduce this representational bias, possibly at the expense of interpretability. Furthermore, any DDS approach relies on an underlying automated process discovery algorithm. For example, Simod relies on the Split Miner algorithm [11] to discover BPMN models. Every such algorithm is limited in terms of the class of process models that it can generate. For example the Split Miner and other algorithms based on directly-follows graphs (e.g. Fodina) cannot capture process models with duplicate activity labels (i.e. multiple activity nodes in the model sharing the same label). Mean-

[6]The cycle time of a process instance adds the processing times (activity durations) and the waiting times [25]

Metric	**Size**	Type of source	**Event Log**	GRU	LSTM	LSTM-GAN	SIMOD
CLFS	**LARGE**	REAL	***UB***	0.63141	***0.67176***	0.28998	
			BPIC2017W	0.63751	***0.71798***	0.36629	0.58861
			BPIC2012W	0.58375	***0.70228***	0.35073	0.53744
			CALL	0.82995	***0.83043***	0.24055	0.62911
		SYNTHETIC	***CVS***	0.83369	***0.85752***	0.20898	0.71359
			CFL	***0.81956***	0.60224	0.11412	0.77094
	SMALL	REAL	***INS***	0.50365	0.51299	0.25619	***0.61034***
			ACR	0.78413	***0.78879***	0.18073	0.67959
			MP	0.27094	0.23197	0.06691	***0.34596***
		SYNTHETIC	***CFS***	0.69543	0.66782	0.10157	***0.76648***
			P2P	0.41179	***0.65904***	0.13556	0.45297
MAE	**LARGE**	REAL	***UB***	801147	778608	***603105***	
			BPIC2017W	868766	***603688***	828165	961727
			BPIC2012W	701892	***327350***	653656	662333
			CALL	160485	174343	***159424***	679847
		SYNTHETIC	***CVS***	859926	***667715***	952004	1067258
			CFL	25346	***15078***	956289	252458
	SMALL	REAL	***INS***	1586323	1516368	1302337	***1090179***
			ACR	344811	341694	296094	***230363***
			MP	335553	321147	***210714***	298641
		SYNTHETIC	***CFS***	30327	33016	717266	***15297***
			P2P	2407551	2495593	2347070	***1892415***
ELS	**LARGE**	REAL	***UB***	0.58215	***0.65961***	0.28503	
			BPIC2017W	0.63643	***0.70317***	0.35282	0.58412
			BPIC2012W	0.57862	***0.67751***	0.33649	0.52555
			CALL	0.79336	***0.81645***	0.19123	0.59371
		SYNTHETIC	***CVS***	0.65160	***0.70355***	0.16854	0.70154
			CFL	***0.68292***	0.43825	0.09505	0.66301
	SMALL	REAL	***INS***	0.49625	0.50939	0.23070	***0.57017***
			ACR	***0.75635***	0.45737	0.15884	0.71977
			MP	0.25019	0.21508	0.04570	***0.31024***
		SYNTHETIC	***CFS***	0.54433	0.57392	0.07930	***0.67526***
			P2P	0.22923	0.39249	0.09968	***0.43202***
EMD	**LARGE**	REAL	***UB***	0.00036	0.00011	***0.00001***	
			BPIC2017W	0.00060	0.01010	0.00072	***0.00057***
			BPIC2012W	0.00077	0.00061	0.00006	***0.00002***
			CALL	0.00084	0.15794	0.00090	***0.00072***
		SYNTHETIC	***CVS***	0.61521	0.57217	0.40006	***0.13509***
			CFL	***0.00472***	0.00828	0.03529	0.06848
	SMALL	REAL	***INS***	0.03343	0.00308	0.33336	***0.00001***
			ACR	0.49996	0.68837	***0.25012***	0.58674
			MP	***0.12609***	0.33375	0.28577	0.31411
		SYNTHETIC	***CFS***	0.08253	0.10784	0.06924	***0.03461***
			P2P	0.25306	0.33747	0.23898	***0.03888***

Table 20: This table presents the detailed results of the DDS vs DL generative models evaluation, the underlined values correspond to the best values reported by the models

while, the Inductive Miner algorithm cannot capture non-block-structured process models [10]. In contrast, deep learning models for sequence generation rely on non-linear functions that model the probability that a given activity occurs after a given sequence prefix. Depending on the type of architecture used and the parameters (e.g. the number of layers, the type of activation function, learning rate), these models may be able to learn dependencies that cannot be captured by the class of BPMN models generated by a given process discovery algorithm such as Split Miner.

Along the temporal perspective, DDS models make assumptions about the sources of waiting times of activities. Chiefly, DDS models assume that waiting times are caused exclusively by resource contention, and they assume that as soon as a resource is available and assigned to an activity, the resource will start the activity in question (robotic behavior) [1]. Furthermore, DDS models generally fail to capture interdependencies between multiple concurrent cases (besides resource contention) such as batching or prioritization between cases (some cases having a higher priority than others) [1]. Another limitation relates to the assumption that resources perform one activity at a time, i.e. no multitasking [26]. In contrast, deep learning models simply try to learn the time to the next-activity in a trace based on observed patterns in the data. As such, they may learn to predict delays associated with inter-case dependencies as well as delays caused by exogenous factors such as workers being busy performing work not related to the simulated process. These observations explain why deep learning models outperform DDS models when it comes to capturing the time between consecutive activities (and thus the total case duration). DDS models are prone to underestimating waiting times, and hence cycle times, because they only take into account waiting times due to resource contention. Meanwhile, deep learning models learn to replicate the distributions of waiting times regardless of their origin.

On the other hand, that DDS models are arguably more interpretable than deep learning models, insofar as they rely on a white-box representation of the process that analysts typically use in practice. This property implies that DDS models can be modified by business analysts to capture what-if scenarios, such as what would happen is a task was removed from the model. Also, DDS models explicitly capture one of the possible causes of waiting times, specifically resource contention, while deep learning models do not explicitly capture any such mechanism. As such, DDS models are more amenable to capture increases or reductions in waiting or processing times that arise when a change is applied to a process. Specifically, DDS models are capable of capturing the additional waiting time (or the reduction in waiting time) that result from higher or lower resource contention, for example due to an increase in the number of cases created per time unit.

6.4. Conclusions

In this chapter, we compared the accuracy of two approaches to discover generative models from event logs: Data-Driven Simulation and Deep Learning. The results suggest that DDS models are suitable for capturing the sequence of activities of a process. On the other hand, DL models outperform DDS models when predicting the timing of activities, specifically the waiting times between activities. This observation can be explained by the fact that the simulation models used by DDS approaches assume that waiting times are entirely attributable to resource contention, i.e. to the fact that all resources that are able to perform an enabled activity instance are busy performing other activity instances. In other words, these approaches do not take into account the multitude of sources of waiting times that may arise in practice, such as waiting times caused by batching, prioritization of some cases relative to others, resources being involved in other business processes, or fatigue effects.

A natural direction for future work is to extend existing DDS approaches in order to take into account a wider range of mechanisms affecting waiting times, to increase their temporal accuracy. However, the causes of waiting times in business processes may ultimately prove to be so diverse, that no DDS approach would be able to capture them in their entirety. An alternative approach would be to combine DDS approaches with DL approaches to take advantage of their relative strengths. In such a hybrid approach, the DDS model would capture the control-flow perspective, while the DL model would capture the temporal dynamics, particularly waiting times. The DSS model would also provide an interpretable model that users can change in order to define "what-if" scenarios, e.g. a what-if scenario where an activity is removed, or a new activity is added.

Two challenges need to be overcome to design such a hybrid DDS-DL approach: (i) how to integrate the DDS model with the DL model; and (ii) how to incorporate the information of a what-if scenario into the DL model. These two requirements for the development of a hybrid technique are addressed in Chapter 7.

7. HYBRID LEARNING OF BUSINESS PROCESS SIMULATION MODELS

Taking as input the analysis performed, and the requirements defined in Chapter 6, in this chapter, we propose the DeepSimulator method for learning generative models for business process simulation by combining DDS and DL models. This technique seeks to fill in the research GAP 5 (cf. Sec. 1.2) and answer the research question RQ2 (cf. Sec. 1.1) concerning how to create simulation techniques that more accurately capture the observed temporal dynamics of business processes.

The first part of this chapter will describe the steps of the approach. Sec. 7.1.1 presents the first phase of the method that uses the DDS model, or more specifically, a stochastic model trained to generate distributions of sequences of activities [47]. Sec. 7.1.2 presents the time series prediction method used to generate process instances. Finally, Sec. 7.1.3 presents the deep learning approach in charge of predicting and incorporating the timestamps into the sequences generated by the stochastic model. In the second part of the chapter, we will present two evaluations; the first one exhaustively compares the relative accuracy of the proposed method concerning pure DDS and DL approaches. Additionally, the second evaluation assesses the approach's ability to simulate a process after a change is introduced (what-if analysis).

7.1. Approach Description

Fig. 34 depicts the architecture of the DeepSimulator approach. The architecture is a pipeline with three phases. The first phase uses PM techniques to learn a model to generate sequences of (non-timestampted) events. The second and third phases enrich these sequences with case start times and activity start and end times. Below, we discuss each phase in turn.

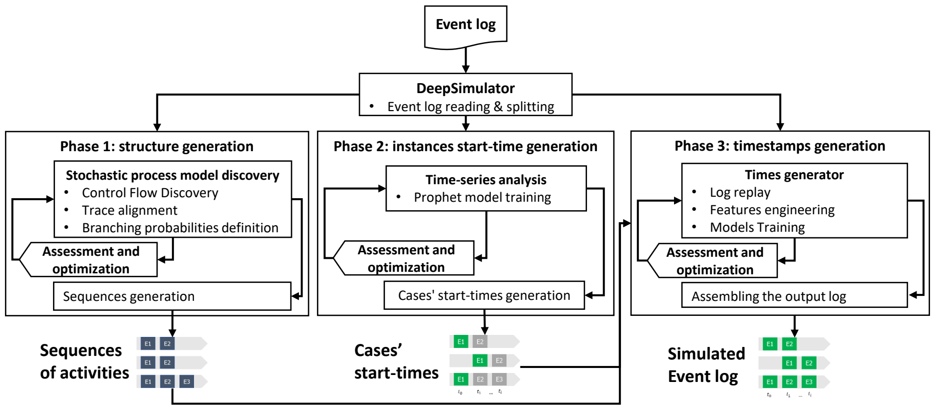

Event log
DeepSimulator
• Event log reading & splitting
Phase 1: structure generation
Stochastic process model discovery
• Control Flow Discovery
• Trace alignment
• Branching probabilities definition
Assessment and optimization
Sequences generation
Phase 2: instances start-time generation
Time-series analysis
• Prophet model training
Assessment and optimization
Cases' start-times generation
Phase 3: timestamps generation
Times generator
• Log replay
• Features engineering
• Models Training
Assessment and optimization
Assembling the output log
Sequences of activities
Cases' start-times
Simulated Event log

Figure 34: Overview of the proposed BPS model discovery method

7.1.1. Phase 1: Activity sequences generation

This phase aims to extract a stochastic process model [47] from the log and to use it to generate sequences of activities that resemble those in the log. A stochastic process model is a process model with branching probabilities assigned to each branch of a decision point. In this thesis, we represent process models using the standard BPMN notation. Phase 1 starts with a *control-flow discovery* step, where we first discover a plain (non-stochastic) process model using the Split Miner algorithm [11]. This algorithm relies on two parameters: the sensitivity of the parallelism oracle (η) and the level of filtering of directly-follow relations (ε). The former parameter determines how likely the algorithm will discover parallel structures, while the latter determines the percentage of directly-follows relations between activity types are captured in the resulting model. Like other automated process discovery algorithms, the Split Miner discovers a process model that does not perfectly fit the log. The discovered process model cannot parse some traces in the log. This hinders the calculation of the branching probabilities. Accordingly, we apply the *trace alignment* algorithm in [72] to compute an alignment for each trace in the log that the model cannot parse. An alignment describes how a trace can be modified to be turned into a trace that can be parsed by the model (via "skip" operations). Based on the alignments, we either *repair* each non-conformant trace, or we *replace* it with a copy of the most similar conformant trace (w.r.t. string-edit distance). The choice between the repair and the replacement approaches is a parameter of the method.

Next, DeepSimulator uses the (conformant) event log to discover the *branching probabilities* for each branching point in the model. Here, DeepSimulator offers two options: (i) assign equal values to each conditional branch; or (ii) compute the branching branches by replaying the aligned event against the process model. The first approach may perform better for smaller logs, where the probabilities computed via replay are not always reliable, while the latter may be preferable for larger logs.

The DeepSimulator combines the process model and the branching probabilities to assemble a stochastic process model. In this step, the DeepSimulator uses a Bayesian optimization technique to discover the hyperparameter settings (i.e., values of ε, η, replace-vs-repair, and equal-vs-computed probabilities) that maximize the similarity between the generated and the ground truth sequences in terms of activity sequences. The optimizer uses a holdout method, and as a loss function, it uses the CFLS metric described in [16]. The CFLS metric is the mean string-edit distance between the activity sequences generated by the stochastic process model and the traces in the ground-truth log after their optimal alignment.[1]

Finally, in the *sequences' generation* step, DeepSimulator uses the resulting

[1]We did not use the stochastic conformance checking metrics over Petri nets defined in [47] because our method handles BPMN models with inclusive join gateways, which cannot be directly transformed to Petri nets (without exponential blowout) as demonstrated in [29]

stochastic process model to generate a bag of activity sequences without timestamps. This bag of sequences is used as the log's base structure in Phase 3.

7.1.2. Phase 2: Case start-times generation

In this phase, we generate each process instance's start time in the output log. Traditionally, DDS models generate the start-time of cases by randomly drawing from an unimodal distribution of the interarrival times between consecutive cases. A typical BPS model captures the interarrival times using a negative exponential distribution (i.e., it models the creation of cases as a Poisson process). However, a single distribution is not realistic enough to capture real scenarios. For example, cases might be created more frequently on Mondays than on Thursdays in a claims handling process.

Instead of fitting an interarrival distribution, the DeepSimulator models the case generation as a time series prediction problem as the number of cases generated per hour of the day. This type of modeling allows us to use robust techniques such as ARIMA or ETS tested successfully in several contexts such as Stock Market Analysis or Workload Projections. DeepSimulator uses the Prophet [85] model proposed by Facebook because it is one of the simplest but, at the same time, more accurate predictive models for this type of task. Prophet starts from the time series decomposition into four main components (i.e., trend, seasonality, holidays, and error) and applies specialized techniques to model each component.

The trend component decomposes those non-periodic changes in the time series values, which are modeled using logistic growth models or Piecewise linear models. The seasonality component decomposes the periodic changes repeated at fixed intervals (hours, weeks, months, or years), which are modeled by using the Fourier series. The holidays component represents the effects of holidays that occur on potentially irregular schedules over one or more days. This component is optionally modeled and is defined manually by a domain expert, since it is specific to each time series. The model automatically calculates the error, corresponding to all those unforeseen changes that the model cannot fit.

In the *time-series analysis* step, we use a saturated logistic growth model to fit the case generation trend. We chose this model, considering that the time series is limited by a lower and upper bound. The lower bound corresponds to 0, which is the minimum number of cases attended in the process, and the upper bound is theoretically limited by the capacity of the process. The parameter that most significantly affects data trend capture is changepoint-prior-scale that determines how much the trend changes at the trend change points. The smaller the value of this parameter, the less flexible the trend running the risk of under-fitting; on the contrary, the greater the risk of over-fitting. Therefore, for this parameter, DeepSimulator tests different values between [0.001, 0.5]. Analogously, the parameter that most directly affects the seasonality capture is the seasonality-prior-scale. This parameter affects the flexibility of seasonality, the smaller this value, the model tends to focus on small fluctuations, and the larger the model is fixed on large fluctuations. For this parameter, DeepSimulator tests different values between [0.01, 10].

We do not define the Holidays component in a specific way since for each log, it is independent, and in principle, it is unknown by the simulator. However, this component possibly can be discovered by a calendar discovery technique such as the one proposed in [26]. We used grid search for selecting the best hyperparameters of the model because the search space consists of only sixteen possible configurations (see Sec. 7.2.3). For validation, we used the internal mechanism of Prophet based on cross-validation and the selection of cutoff points.

In the *case start-times generation* step, we use the Prophet time series model to determine the number of cases to be created at each hour of the simulation. We then generate the start-times, for each simulation hour, by modeling the intercase arrival times via a normal distribution.

7.1.3. Phase 3: Activity timestamps generation

We enhance the activity sequences generated in Phase 1 to capture waiting times and processing times in this phase. The DeepSimulator trains two LSTM models to perform two predictive tasks: the processing time of a given activity (herein called the *current activity*) and the waiting time until the start of the next activity. This task differs in two ways from approaches used to predict the next event and its timestamp [18, 86]. First, we do not seek to predict the next event since the sequences of activities are generated by the stochastic process model (cf. Phase 1). Second, we need to support changes in the process model (e.g., adding or removing tasks) for enabling what-if analysis.

Therefore, we train one model specialized in predicting the processing time of the current activity and another specialized in predicting the waiting time until the next activity. Both models differ in the set of features since they act at different moments in the predictive phase, as shown in Fig. 35.

The processing time predictive model uses the following features as inputs: the label and processing time of the current activity, the time of day of the current activity's start timestamp, the day of the week, and inter-case features such as the Work-in-progress (WIP) of the process and the activity and Resources' Occupation (RO) at the start of the activity.

The waiting time predictive model uses the following features as inputs: the next activity's label, the time of day of the current activity's end timestamp, the day of the week, and inter-case features such as the WIP of the process and the RO at the end of the current activity.

In the *log replay* step, we calculate the waiting and processing times of each activity by replaying each trace in the input log (or in a training subset thereof) against the process model discovered in Phase 1. As explained in Sec. 2.2 an activity's processing time is the difference between its end and start timestamps.

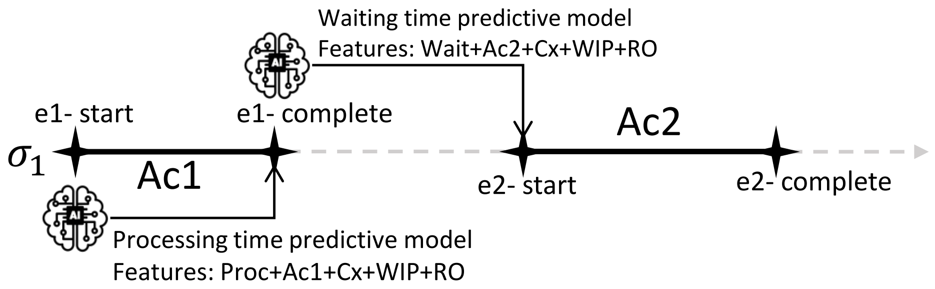

Waiting time predictive model
Features: Wait+Ac2+Cx+WIP+RO
e1- start
e1- complete
Ac2
σ1
Ac1
e2- start
e2- complete
Processing time predictive model
Features: Proc+Ac1+Cx+WIP+RO

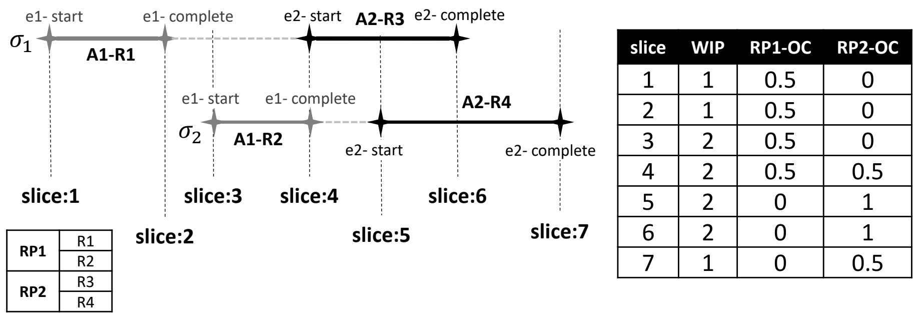
Figure 35: Predictive models timeline and features

An activity's waiting time is the difference between its start time and enablement time, i.e., when it was ready to be executed according to the process model. All waiting and processing times are scaled to the range [0...1] by dividing them by the largest observed values.

In the *feature engineering* step, we compute and encode all the remaining features used by the models. We calculate the time of the day as the elapsed seconds from the closest midnight until the event timestamp; this feature is scale over 86400 seconds. The day of the week is modeled as a categorical attribute and encoded using one-hot encoding. We include these latter features since they provide contextual information, allowing the model to find seasonal patterns in the data that may affect waiting and processing times. In the same way, considering that the overall process performance is affected by the process' WIP and the RO [43], we use two inter-case features that measure these variations.

The WIP of the process measures the number of active tasks at each moment in the log transversally. The RO measures each resource pool's percentage occupancy in the log, implying that a new feature is created for each pool to record the occupation-specific variations. Since the information about the size and composition of the resource pools is not always included in the logs, we grouped resources into roles by using the algorithm described in [81]. This algorithm discovers resource pools based on the definition of activity execution profiles for each resource and the creation of a correlation matrix of similarity of those profiles. WIP and RO are calculated by replaying over time the log events, recording the variations in both features at every time point. Fig. 36 presents an example of the WIP and RO calculation at each time point or slice. In this example, we have two traces that overlap at execution time σ_1 and σ_2, each composed of four events that correspond to the execution of two activities, A1 and A2. In the example, we have two resource pools RP1 and RP2, each one with two resources. Likewise, each activity is associated with a pool and is executed by a resource of said pool.

Finally, we encode the current activity's label using pre-trained embedded dimensions. We use embeddings for two reasons. First, embeddings help prevent exponential feature growth associated with one-hot encoding [18]. Second, embedded dimensions allow adding new categories (i.e., activity labels) without



RP1	R1
	R2
RP2	R3
	R4

slice	WIP	RP1-OC	RP2-OC
1	1	0.5	0
2	1	0.5	0
3	2	0.5	0
4	2	0.5	0.5
5	2	0	1
6	2	0	1
7	1	0	0.5

Figure 36: Example of WIP and RO inter-case features encoding

altering the predictive model's structure. These embedded dimensions are an n-dimensional space, where each category (each activity level) is encoded as a point in that space. An independent network fed with positive and negative examples of associations between activities is used to map the activity labels to points. The network maps activities that co-occur or occur close to each other to nearby points. This mechanism also allows adding a new point in that space by updating the encoding model without altering the predictive model's input size. Each time a new activity is added to the process model for what-if analysis, we generate examples of traces involving this new activity and use these examples to determine the coordinates of the new activity label to be encoded in the embedded space. Then, we update the predictive model's embedded layers with the new definition, and the predictive model can handle the new activity label from that point on.

Once encoded the features, we extract n-grams of fixed sizes from each trace to create the input sequences to train the model. As shown in Fig. 37, both models are composed of two stacked LSTM layers and a dense output. A model receives the sequences as inputs and the expected processing and waiting times as a target. The user can vary the number of units in the LSTM layers (50 or 100), the activation function (tanh or selu), the size of the n-gram (5, 10, or 15 events), and the use of all the RO inter-cases or just the one of the resource pool associated to the execution of the activity. The activation function of the dense layer is linear.

7.1.4. Assembling the output log

The output log is generated by assembling each generated sequence (see Phase 1), with the generated case start time (see Phase 2) and the processing and waiting times predicted iteratively (see Phase 3). In each iteration, the trained model predicts times relative to the current activity in seconds, which are transformed into absolute times by adding them to the start time of the case. Then, the DeepSimulator generates a simulated log composed of a bag of traces, each trace consisting of a sequence of triplets (activity label, start-timestamp, end timestamp).

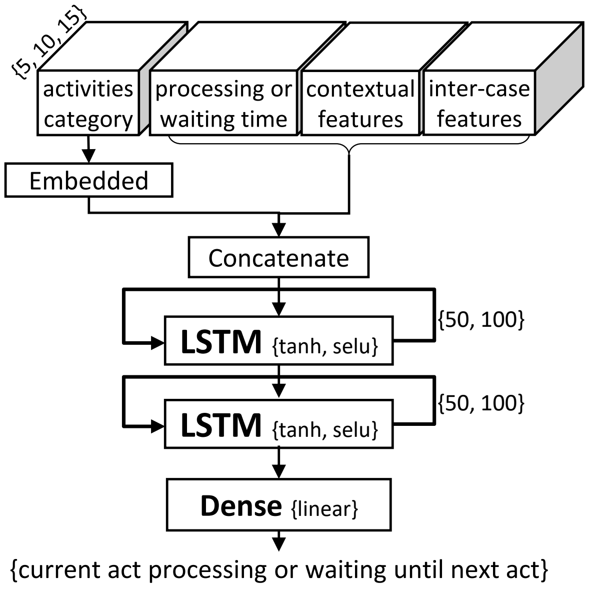


Figure 37: Deep learning models architectures

7.2. Evaluation

We empirically compare the DeepSimulator (DSIM) vs. DDS and DL approaches in terms of the similarity of the simulated logs they generate relative to (a fold of) the original log. We also assess the accuracy of the DSIM models for the "what-if" analysis task of adding new process activities.

7.2.1. Datasets

We evaluated the approaches using 10 logs that contain both start and end timestamps. We use real-life logs (R) from public and private sources and synthetic logs (S) generated from simulation models of real processes.[2] Table 21 provides descriptive statistics of the logs. The *UB* and *BPIC2017W* logs have the largest number of traces and events, while *MP*, *CFS* and *P2P* have fewer traces, but more events per trace.

7.2.2. Evaluation measures

To evaluate the temporal accuracy of a model M produced by one of the methods under evaluation, we compute a distance measure between a log generated by model M and a ground-truth log (a testing subset of the original log). We use two distance measures: the MAE of cycle times and the EMD of the normalized histograms of activity timestamps grouped by day/hour. Both of the were introduced in the Chapter 6.

[2]Logs and models available at `https://doi.org/10.5281/zenodo.5080502`

Size	Source	Log	#Traces	#Events	#Act.	Avg. activities per trace	Avg. duration	Max. duration	Description
LARGE	R	*UB*	70512	415261	8	5.89	15.21 days	269.23 days	Undisclosed banking process*
	R	*BPIC2017W*	30276	240854	8	7.96	12.66 days	286.07 days	Dutch financial institution updated
	R	*BPIC2012W*	8616	59302	6	6.88	8.91 days	85.87 days	Dutch financial institution
	S	*CVS*	10000	103906	15	10.39	7.58 days	21.0 days	*CVS* retail pharmacy**
	S	*CFL*	2000	44373	29	26.57	0.76 days	5.83 days	Anonymized confidential process**
SMALL	R	*INS*	1182	23141	9	19.58	70.93 days	599.9 days	Insurance claims process*
	R	*ACR*	954	4962	16	5.2	14.89 days	135.84 days	Academic Credential Recognition
	R	*MP*	225	4503	24	20.01	20.63 days	87.5 days	Manufacturing Production
	S	*CFS*	1000	21221	29	26.53	0.83 days	4.09 days	Anonymized confidential process**
	S	*P2P*	608	9119	21	15	21.46 days	108.31 days	Purchase-to-Pay process

Table 21: (*) Private logs, (**) Generated from simulation models of real processes

The *cycle time MAE* measures the temporal similarity between two logs at the *trace level*. The absolute error of a pair of traces T1 and T2 is the absolute value of the difference between the cycle time of T1 and T2. The cycle time MAE is the mean of the absolute errors over a collection of paired traces. Given this trace distance notion, we pair each trace in the generated log with a trace in the original log so that the sum of the trace errors between the paired traces is minimal. This pairing is done using the Hungarian algorithm for computing optimal alignments [42].

The cycle time MAE is a rough measure of the temporal similarity between the ground-truth and the simulated traces. But it does not consider the start time of each case, nor the start and end timestamps of each activity. To complement MAE, we use the *EMD* between the normalized histograms of the timestamps grouped by day and hour in the ground-truth log vs. the same histogram computed from the generated log. The EMD between two histograms H1 and H2 is the minimum number of units that need to be added, removed, or transferred across columns in H1 to transform it into H2. The EMD is zero if the observed distributions in the two logs are identical, and it tends to one the more they differ.

7.2.3. Experiment 1: Accuracy of generated models

The evaluation aims to compare the accuracy of DSIM models vs. DDS and DL models discovered from logs.

Experiment Setup. For this, in this section, we start from the same experiment setup as in Chapter 6 i.e., we use Simod as a baseline DDS approach since it is fully automated for parameters setting. As DL baselines, we used an adaptation of the LSTM approach proposed by Camargo et al. [18] – herein labeled the LSTM method – as well as the GAN approach by Taymouri et al. [86] – herein labeled LSTM(GAN). Both DL approaches attain high accuracy concerning the task of generating timestamped trace suffixes [71]. Fig. 38 summarizes the experiment setup.

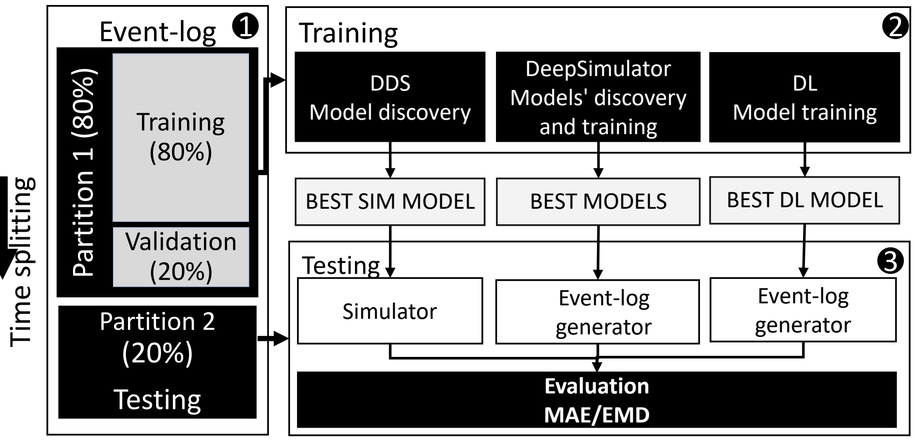


Figure 38: Setup of experiment 1

We used the hold-out method with a temporal split criterion to divide the logs into two main folds: 80% for training-validation and 20% for testing. From the first fold, we took the first 80% for training and 20% for validation. We use temporal splits to prevent information leakage [18, 86].

The DDS technique (Simod) was set to explore 15 parameter configurations to tune the stochastic process model. For each configuration, we executed five simulation runs and computed the CFLS measure (cf. Sec. 6.1, Phase 1) between each simulated log and the validation fold. We selected the stochastic model that gave the lowest average CFLS w.r.t. the validation fold. Then, the optimizer was set to explore 20 simulation parameter configurations (i.e., the parameters that Simod uses to model resources and processing times) again using five simulation runs per configuration. We then selected the configuration with the lowest average EMD (cf. Sec. 6.1, Phase 2) between the simulated log and the validation fold. We used the parameter ranges given in Table 22 for tuning.

The LSTM technique was hyperparameter-optimized using grid search over a space of 48 possible configurations (see Table 22). For LSTM model training, we used 200 epochs, the cycle time MAE as the model's loss function, Nadam as the optimizer, and early stopping and dropout to avoid model over-training. The LSTM(GAN) technique was configured to dynamically adjust the size of the hidden units in each layer so that their size is twice the input's size, as proposed by the authors [86]. We used 25 training epochs, a batch of size five, and a prefix size of five.

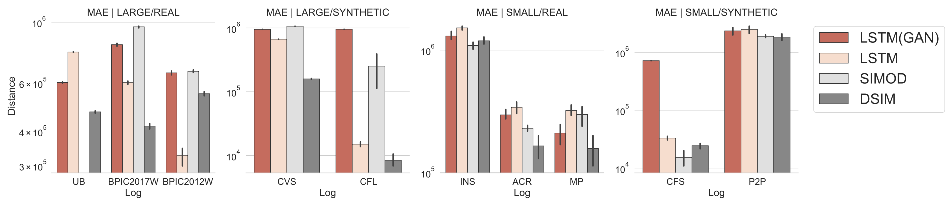
DSIM was tuned by randomly exploring 15 parameter configurations with five simulation runs per configuration in the stochastic model discovery phase (cf. Sec. 7.1, Phase 1). In Phases 2 and 3, we used grid search to explore the space of hyperparameter configurations specified in Table 22.

In the end, we generated four models per log: one SIMOD, one LSTM, one LSTM(GAN), and one DSIM. We then generated five logs per retained model, each of the same size (number of traces) as the original log's testing fold to ensure comparability. Each generated log was compared with the testing fold using the MAE and EMD measures. We report the mean of each of these measures across five runs.

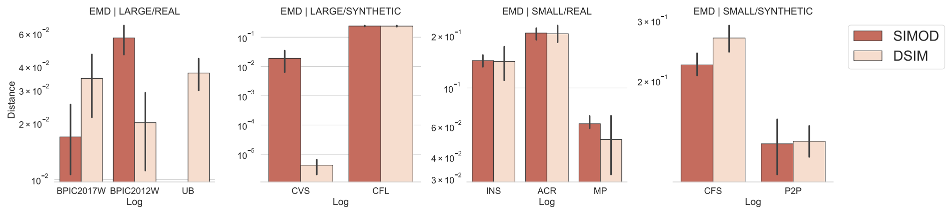
Model	Stage	Parameter	Distribution	Values
SIMOD	Structure discovery	Parallelism threshold (ε)	Uniform	[0...1]
		Percentile for frequency threshold (η)	Uniform	[0...1]
		Conditional branching probabilities	Categorical	{Equiprobable, Discovered}
	Time-related parameters discovery	Log repair technique	Categorical	{Repair, Removal, Replace}
		Resource pools similarity threshold	Uniform	[0...1]
		Resource availability calendar support	Uniform	[0...1]
		Resource availability calendar confidence	Uniform	[0...1]
		Instances creation calendar support	Uniform	[0...1]
		Instances creation calendars confidence	Uniform	[0...1]
LSTM	Training	N-gram size	Categorical	[5, 10, 15]
		Input scaling method	Categorical	{Max, Lognormal}
		# units in hidden layer	Categorical	{50, 100}
		Activation function for hidden layers	Categorical	{selu, tanh}
		Model type	Categorical	{shared_cat, concatenated}
DSIM	Structure generation	Parallelism threshold (ε)	Uniform	[0...1]
		Percentile for frequency threshold (η)	Uniform	[0...1]
		Conditional branching probabilities	Categorical	{Equiprobable, Discovered}
	Cases start-times generation	changepoint-prior-scale	Categorical	{0.001, 0.01, 0.1, 0.5}
		seasonality-prior-scale	Categorical	{0.01, 0.1, 1.0, 10.0}
	Timestamps generation	N-gram size	Categorical	{5, 10, 15}
		# units in hidden layer	Categorical	{50, 100}
		Activation function for hidden layers	Categorical	{selu, tanh}
		Single resource-pool intercase feature	Boolean	{True, False}

Table 22: Hyperparameters used by optimization techniques

Results. Fig. 39 and Table 23 present the accuracy results grouped by metrics, log size and source type. Cycle time MAE and EMD are error/distance measures (lower is better).



(a) Cycle time MAE results



(b) Earth Mover's Distance (EMD) results

Figure 39: Evaluation results at trace and log levels

At the traces level (see Fig. 39a) in the large logs (real-life and synthetic), the DSIM models outperform the LSTM and SIMOD approaches. The DSIM models obtain by far the best result in four out of five logs. In small logs, DSIM models obtain the lower MAE values in three out of five logs and SIMOD by a few in the remaining. As expected, the pure DL model results show a significant dependence on the volume of data available for learning the log's complex temporal dynamics. However, although the DSIM approach also uses DL models for time prediction, it is insensitive to the data volume. The LSTM(GAN) models present a low performance in general terms due to the temporal stability of the models' predictions declines rapidly, despite having a high precision in predicting the next event [86]. These results also indicate overfitting on the LSTM(GAN) models, preventing the generalization of this approach for this predictive task.

At the global representation of times, DSIM obtains as well the best results in seven of the ten logs evaluated, surpassing the SIMOD results (see Fig. 39b). Despite the minor differences between both approaches, the Prophet model used by DSIM can precisely capture the dynamics of instance generation without defining calendars. The calendar definition can be included in the Prophet model as part of the holidays component, potentially improving its performance. We did not evaluate the LSTM and LSTM(GAN) techniques concerning the global representation of times because these models do not handle the generation of process instances, operating exclusively at the level of individual process traces.

Size	Type	Log	MAE				EMD	
			LSTM(GAN)	LSTM	SIMOD	DSIM	SIMOD	DSIM
LARGE	R	*UB*	603105	778608	N/A	*471239*	N/A	*0.036932*
		BPIC2017W	828165	603688	961727	*418422*	*0.016873*	0.034584
		BPIC2012W	653656	*327350*	662333	548813	0.056789	*0.020098*
	S	*CVS*	952004	667715	1067258	*158902*	0.018955	*0.000004*
		CFL	956289	15078	252458	*8441*	0.240567	*0.239087*
SMALL	R	*INS*	1302337	1516368	*1090179*	1190019	0.143675	*0.142097*
		ACR	296094	341694	230363	*165411*	0.207050	*0.205106*
		MP	210714	321147	298641	*157453*	0.062227	*0.050479*
	S	*CFS*	717266	33016	*15297*	24326	*0.222515*	0.266749
		P2P	2347070	2495593	1892415	*1836863*	*0.130655*	0.132898

Table 23: Evaluation results (lower values are better)

7.2.4. Experiment 2: Process demand modification

In this experiment, we compared the ability of DSIM to reproduce a what-if scenario in which the process arrival rate contains high and low periods of case influxes.

Experiment setup. AS a first step to evaluate this scenario, we created a modified version of the three largest logs (i.e., BPIC2012W, BPIC2017W, and CVS). We first divide each log into six batches of the same number of cases and create two groups, interleaving them. The first group is composed of batches 1, 3, and 5 and represents the high influx of cases for which we did not make any changes. The second group comprises batches 2, 4, and 6, representing the low influx of cases; therefore, we reduced the arrival rate by 1/3, randomly eliminating two out of three cases. Fig. 40 presents the modification made in the BPIC2017W log.

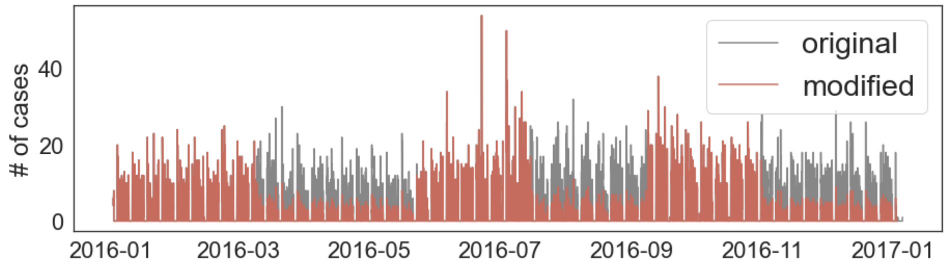


Figure 40: Original and modified cases creation

Later we trained the DSIM and SIMOD models in the same way as we did in Experiment 1 using these modified logs. Later, we generated five logs, each of the same size (number of traces) as the original log's testing fold. Each generated log was compared with the testing fold using the MAE and EMD measures. In a complementary way, we report the Dynamic Time Warping (DTW) distance between the time series of the number of cases generated per hour of the day between the testing partition and the logs generated. DTW gives a non-linear (elastic) alignment between two-time series, producing an intuitive similarity measure. We report the mean of each of these measures across five runs.

Results. Table 24 presents the results of cycle time MAE, EMD of the log timestamps grouped by day and hour and DTW of the interarrival times. Fig. 41 shows the generation of cases of the testing partitions vs. the logs generated by each of the models per input event log. DSIM has a lower error in cycle times in all cases. We can explain this result because the DL models in charge of predicting waiting and processing times consider inter-case attributes that describe the Workload of the process. The use of these attributes implies that the models can adjust to variations in the load of the process.

Regarding the results of EMD and DTW, both metrics follow the same trends. In two of the three logs, DSIM obtains the best results - In some cases much better w.r.t. CVS log -, and in the rest, the results remain close to those of SIMOD. In the case of the CVS log, SIMOD, when using a single distribution, generated the cases in a very short interval of time, which caused a contention of resources that triggered errors in cycle times. On the contrary, DSIM distributed the creation in a much greater interval than expected. In the case of BPIC2012W, the opposite occurred. SIMOD generated the cases in a much longer interval than expected. In the BPIC2017W log, DSIM and SIMOD tended to a medium generation of cases, for which both approaches obtained similar results.

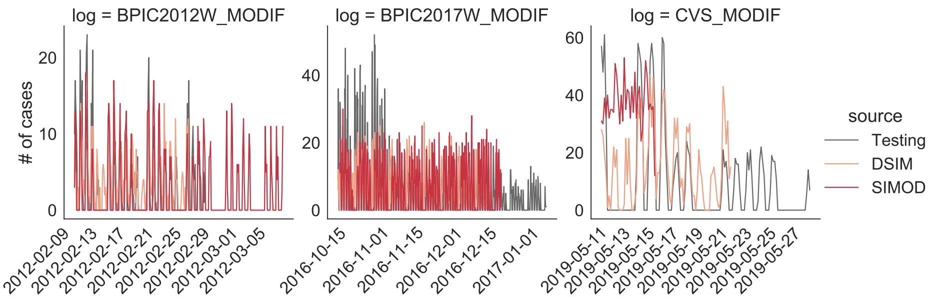


Figure 41: Cases creation in generated logs versus cases creation in the test partitions

Log	MAE		EMD		DTW	
	SIMOD	DSIM	SIMOD	DSIM	SIMOD	DSIM
BPIC2017W	971151	***417572***	***0.02222***	0.03593	***3185***	3647
BPIC2012W	660211	***534341***	0.11295	***0.04853***	515	***458***
CVS	1489252	***467572***	0.03213	***0.00001***	3380	***849***

Table 24: Detailed results

7.2.5. Experiment 3: Addition of a new activity

We evaluate the ability of DSIM to simulate a process after a change (what-if analysis). We consider the scenario where one activity is added to a process. This scenario is challenging because it adds an element to the process that was not known when the DSIM model was trained. Other possible changes, such as altering the branching probabilities or removing activities, do not add unknown elements.

Experiment setup. For each of the two synthetic logs (*CVS* and *CFL*), we selected a random activity A, and we eliminated all occurrences of A from the log. We then trained a DSIM simulation model using this modified log (cf. left-hand side of Fig. 42). Next, we generated synthetic data consisting of positive and negative examples of pairs composed by the activity label and associated resource (see Sec. 7.1.3). Using these synthetic samples, we updated the embedded dimensions to include activity A (without modifying the embedding of the remaining activities). We then plugged the updated embedding into the previously trained DSIM model (cf. right-hand side of Fig. 42). We then calculated the errors of the DSIM model of the "as-is" process (before a change) and of the DSIM model of the "what-if" process (after adding an activity).

We measured the error using the MAE metric. Additionally, we calculated the RMSE and SMAPE metrics to confirm that the results do not depend on the chosen error metric.

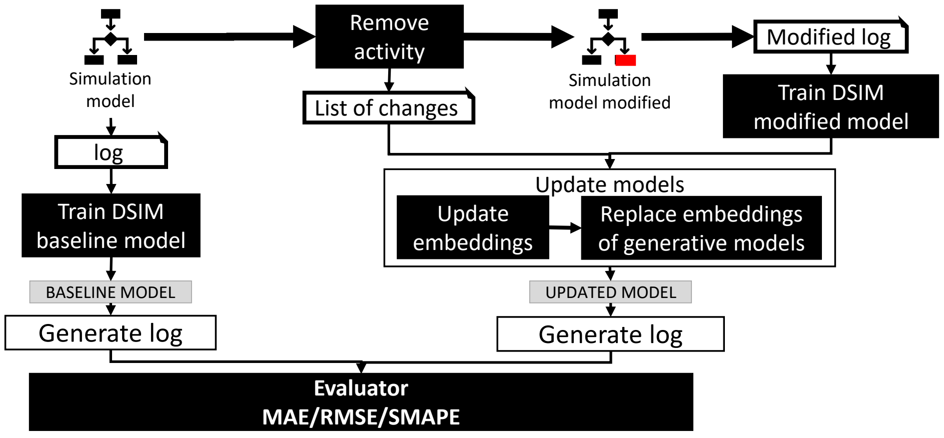


Figure 42: Setup of experiment 2

Results. Table 25 presents results of cycle time MAE, RSME, and SMAPE grouped for each log, both for the simulation model of the initial process (as-is model) and the model after the modification (what-if model). In general terms, the what-if model has the higher prediction error at the trace level concerning the baseline model in both event logs. These high error values are evident in the *CVS* log, where the SMAPE of the updated model is 184% compared to 31.97% of the baseline model. However, these results also demonstrate that embedded dimensions enable deep-learning models to incorporate and predict activities that were not present in the training log without model retraining. These results open the discussion on using more specialized embedding techniques such as word2vec or transformer models that allow coding activities in a more precise way. Using these techniques would improve the precision of deep-learning models in predicting both as-is scenarios and possible what-if scenarios.

Log	Source	MAE	RMSE	SMAPE
CFL	as-is	7155	22006	0.1563
	what-if	17546	33137	0.2876
CVS	as-if	283061	357717	0.3197
	what-if	1040344	1052255	1.8460

Table 25: Experiment 2 evaluation results

7.3. Conclusions

This chapter presented a method, namely DSIM, to learn BPS models from event logs using a combination of PM and DL techniques. The design of this method is driven by the observation that traditional methods for discovering BPS models from event logs – a.k.a. DDS methods – do not capture delays between activities caused by factors other than resource contention within the simulated process, such as fatigue effects, batching effects, and inter-process dependencies. In contrast, DL techniques can learn complex temporal patterns present in event logs. Accordingly, the proposed method discovers a stochastic process model from an event log using automated process discovery and trace alignment techniques, and it then uses DL models to add timestamps to the traces produced by the stochastic model. The method is designed in such a way that the stochastic process model can be later modified (activities may be added or removed, and branching probabilities may be altered), thus enabling some forms of what-if analysis.

The chapter reported on an empirical comparison of the proposed method against an existing DDS method and two DL methods. The evaluation shows that the DeepSimulator method outperforms the DDS and DL methods, while retaining the what-if analysis capabilities of DDS methods. These results fill the research GAP5 (cf. Sec. 1.2) and answer the research question RQ2 (cf. Sec. 1.1) concerning how to create simulation techniques that more accurately capture the observed temporal dynamics of business processes.

The evaluation in a what-if analysis setting (adding an activity to the process) shows that the accuracy of the method may considerably degrade in this setting. We foresee that this drawback can be addressed by experimenting with embedding techniques that take into account the context in which an activity occurs, such as word2vec or transformer models – as opposed to computing the embedding based on the activity-resource pairings as reported in this thesis.

8. CONCLUSION AND FUTURE WORK

8.1. Summary of contributions

This thesis addressed the question of how to create more accurate business process simulation models derived from data. More specifically, we are interested in how to improve the temporal accuracy of the current BPS technique. Accordingly, we proposed two research questions:

RQ1 How to automatically create accurate business process simulation models based on data extracted from enterprise information systems?

RQ2 How to create simulation techniques that more accurately capture the observed temporal dynamics of business processes?

To answer the research question RQ1, we filter and contextualize the existing literature reviews in the area. We identified two research gaps in the current state-of-the-art in the field of DDS:

GAP1 Existing studies have not extensible explored the question of measuring the accuracy of BPS models derived from data; and,

GAP2 The fine-tuning of simulation parameters is left to the modeler, thus leaving the door open for the introduction of biases during the creation of BPS models.

To address these gaps, in contribution 1, we proposed a method to automatically discover BPS models from event logs and to fine-tune the accuracy of the discovered BPS model. The proposed method takes an event log as the input, automatically discovers a process model, aligns the log to the model (and repairs it accordingly), and applies replay and organizational mining techniques to extract all the parameters required for simulation.

Each of the steps in the method relies on a number of hyperparameters. The proposed method uses a Bayesian optimization technique to fine-tune these hyperparameters to maximize the accuracy of the resulting BPS model, which is measured in terms of a timed string-edit distance between the log(s) the simulation model generates and the original log.

The proposed method was implemented as an open-source tool named Simod. Simod was evaluated using three real-life event logs from different domains. The evaluation showed that the hyperparameter optimization method significantly improved the accuracy of the resulting BPS model relative to an approach built using standard default parameters. Furthermore, the evaluation revealed that the best configuration of hyperparameters varies significantly from one event log to another– further emphasizing the need for automated hyperparameter optimization in this setting.

On the other hand, the evaluation of the Simod method put into evidence its limitations– in particular, temporal accuracy. These limitations can be explained by the fact that most of the existing DES engines used in BPS, including the one used by Simod, assume that the only source of waiting time in a process is resource contention, i.e. the fact that a task instance cannot be started if the resource is busy with other task instances. In doing so, these techniques fail to account for the multitude of sources of waiting times that may arise in practice, such as waiting times caused by batching, prioritization, resources being involved in other business processes besides the one under analysis, or fatigue.

Considering that the factors that can determine waiting times are numerous and diverse, proposing an answer to the RQ2 research question required approaching the problem from a different angle. This thesis investigated the hypothesis that machine learning techniques, specifically deep learning techniques, can be used to increase the accuracy of BPS models extracted from data. After reviewing existing deep learning techniques applied in business processes, this thesis identified three further gaps that must be addressed to make these techniques applicable in this context:

GAP3 DL generative models must be able to generate not only remaining sequences of events (suffixes) but also complete logs starting from scratch (prefixes of size zero).

GAP4 Similarly, the generated logs must include the category of the event, associated resource, and start and end times of the activities, thus allowing an evaluation of the performance of the scenarios.

GAP5 Furthermore, deep learning techniques must be able to perform what-if analysis, one of the BPS's key features.

In response to research gaps 3 and 4, in our contribution 2, we proposed a method to train deep-learning generative models capable of generating complete event logs from input data composed of activities, roles, and timestamps as done in BPS. The approach consisted of a pre-processing phase (scaling and n-gram encoding), an LSTM training phase, and a post-processing phase (selecting the predicted next event among the likely ones). In a first experiment, this thesis compared and analyzed several options for each phase concerning accurately generating full traces from scratch– thus enabling its use in simulation. In the second experiment, the evaluation showed that the proposed approach outperformed existing DL approaches for predicting the remaining sequence of events and their timestamps starting from a given prefix of a trace.

Subsequently, as a benchmark and in preparation to solve research gap 5, we compared the relative precision and characteristics of the DDS versus DL generative models under the same conditions, including the number of event logs (eleven in total). The results suggested that DDS models are suitable for capturing the sequence of activities of a process. In contrast, DL models outperform DDS models when predicting the timing of activities– specifically, the waiting times between activities. This is because simulation models used by DDS approaches assume that waiting times are entirely attributable to resource contention, i.e., all resources that can perform an enabled activity instance are busy performing other activities. Conversely, DL models try to find the function that best fits the observed waiting times without any assumptions about the behavior of the resources involved in the process. Furthermore, this study clarified how to construct a hybrid technique between DDS and DL to improve the temporal accuracy of current DDS techniques while retaining the what-if capabilities.

Finally, to address gap 5, in the contribution 4 in contribution 3, we presented a method - namely DeepSimulator - to learn BPS models from event logs using a combination of PM and DL techniques. The proposed method discovers a stochastic process model from an event log using automated process discovery and trace alignment techniques. The DeepSimulator method then uses DL models to add timestamps to the traces produced by the stochastic model. The method was designed so that the stochastic process model can be later modified (activities may be added or removed, and branching probabilities may be altered); therefore, the method enables some forms of what-if analysis. The proposed approach was compared empirically against the previously developed DDS method and two DL methods. The evaluation showed that the DeepSimulator method outperforms the DDS and DL methods while retaining the what-if analysis capabilities of DDS methods.

In summary, to answer the research questions, we proposed two approaches. The first one – Simod —focuses on answering the RQ1; the tool automatically discovers BPS models from an event log, thus responding to the problems of creating simulation models of the current technique. Furthermore, to our knowledge, this is the only tool capable of measuring and optimizing the precision of a simulation model concerning the input log– thus, significantly improving its precision. However, due to the intrinsic limitations of the simulation technique employed in BPS, Simod has not yet fully responded to the challenge of improving the temporal accuracy of the simulation models. Due to this, to answer the RQ2, we proposed a second tool – DeepSimulator – that discovers hybrid simulation models with deep learning techniques capable of representing the temporal dynamics of the process in a more precise way, thus answering the research question.

To support the reproducibility of the research results, the source code of all the artifacts and experiments performed in this thesis is made available in code repositories (see Appendix A).

8.2. Future work

The research presented in this thesis opens up several directions for future work, described in the following paragraphs.

Extending the capabilities of data-driven simulation. A direction for future work to enhance the accuracy of DDS is to take into account the fact that resources are not always available to perform tasks in a process. In recent work, conducted in parallel with this PhD research, we extended the Simod method (see Sec. 4) to discover calendars that capture the availability periods of resources based on the data available in an event log [26]. However, the discovered availability calendars are hard to interpret and are too granular, since it is assumed that all resources within a resource pool share the same availability calendar. Future work can refine this technique by working at a more granular level– like resource pool or individual resources (i.e., differentiated availability calendars for each resource)– so that the discovered calendars reflect the constraints observed in the process more realistically.

As an alternative to fitting an inter-arrival distribution for the generation of cases in this thesis, we tested the use of time series prediction models [85]. The integration and extension of this type of technique in DDS models could improve the precision of the temporal representation of DDS models. This type of modeling allows the representation of the more complex dynamics of case generation– for example, the constant growth of the demand of the process or the inclusion of seasonal patterns not currently captured by fitting of one single distribution.

A direction to enhance the capabilities of DDS is to include techniques for discovering perspectives from event logs that have not yet been included in the simulation models, which can make the simulation models more realistic. For example, [52, 12] propose techniques for the discovery of decision rules that can be used as an alternative to the use of probabilities in decision gateways. Martin et al. [56] propose techniques for the discovery of batching, and Suriadi et al. [82] propose techniques for the discovery of task prioritization. The use of these, and other perspectives [54], may require the design of simulation engines that can integrate them into execution.

One of the main limitations of data-driven simulation is the constraint of requiring event logs to contain start and complete timestamps of process activities. In practice, this type of event log is rare. A possible direction of future work is in the development of discovery techniques that allow approximating the processing time of the tasks in cases in which there is only one timestamp available. Similar to these examples, each simulation model component is susceptible to improvement and is a rich extension source.

Extending the capabilities of hybrid simulation techniques. A key challenge to use deep-learning models for process simulation is how to capture "what-if" scenarios (e.g., the effect of removing a task or removing a resource). The evaluation of the DeepSimulator technique in a what-if analysis setting (adding an activity to the process) shows that the method's accuracy may considerably degrade. We foresee that this drawback can be addressed by experimenting with embedding techniques that consider the context in which an activity occurs, such as word2vec [58] or transformer models [89]. These techniques can be used instead of computing the embedding based on the activity-resource pairings, as reported in this thesis. Another possible way to meet this challenge is to apply techniques to guide the generation of event sequences from LSTM models using constraints along the lines of [21].

The problem of handling what-if scenarios is closely related to the problem of adapting a predictive model to handle concept drifts. A concept drift refers to a scenario in which the relationship between the input data and the target variable changes over time– affecting the precision of a trained predictive model [30]. In both cases (i.e., what-if and concept drift), the predictive model must adapt to dynamics that were not part of the training data. Some of the techniques that have already been evaluated in the management of the conceptual drift in predictive process monitoring, such as incremental learning, can be explored in the case of process simulation [58]. This research topic is still open in the deep learning community and is a potential source of research in business processes.

Given that the proposed approach generates logs consisting of sequences of timestamped activity instances, another avenue for future work is extending the approach to generate events that include resource and domain-specific attributes. A related avenue is to extend the approach to support a broader range of changes, such as changes in the resource perspective (adding or removing resources). Another avenue is to validate the proposed method via case studies to complement the "postmortem" evaluation reported in this thesis.

Automatically creating and evaluating improvement scenarios. In this thesis, we explored the problem of discovering simulation models from data. However, to support effective decision-making, it is necessary to take a step further and to investigate the effect of difference change scenarios on the performance of a process. Along this direction Lopez-Pintado et al. [50] explore how to optimize the allocation– using BPS to evaluate the possible configurations. The same idea can be extended to explore other possible changes in the process, such as changes in the sequence flow or in the availability of resources. Building process optimization techniques on top of the simulation techniques developed in this thesis is another avenue for future work.

BIBLIOGRAPHY

[1] Wil M. P. van der Aalst. “Business Process Simulation Survival Guide”. In: *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*. Ed. by Jan vom Brocke and Michael Rosemann. Springer, 2015, pp. 337–370.

[2] Wil M. P. van der Aalst. “On the representational bias in process mining”. In: *Proceedings of WETICE 2011*. IEEE, 2011, pp. 2–7.

[3] Wil M. P. van der Aalst. “Process Modeling and Analysis”. In: *Process Mining: Data Science in Action*. Springer, 2016, pp. 55–88.

[4] Wil M. P. van der Aalst and et al. “Process Mining Manifesto”. In: *Proceedings of BPM Workshops 2012*. LNBIP. Springer, 2012, pp. 169–194.

[5] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. “Workflow mining: Discovering process models from event logs”. In: *IEEE Transactions on Knowledge and Data Engineering* 16.9 (2004), pp. 1128–1142.

[6] Madis Abel. “Lightning Fast Business Process Simulator”. MA thesis. University of Tartu, 2011.

[7] A. Adriansyah, B. van Dongen, and W. van der Aalst. “Conformance checking using cost-based fitness analysis”. In: *Proceedings of EDOC 2011*. IEEE, 2011, pp. 55–64.

[8] Arya Adriansyah et al. “Alignment based precision checking”. In: *Proceedings of BPM Workshops 2012*. LNBIP. Springer, 2013, pp. 137–149.

[9] Abel Armas-Cervantes et al. “Local Concurrency Detection in Business Process Event Logs”. In: *ACM Transactions on Internet Technology* 19.1 (2019), pp. 1–23.

[10] Adriano Augusto et al. “Automated Discovery of Process Models from Event Logs: Review and Benchmark”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.4 (2018), pp. 686–705.

[11] Adriano Augusto et al. “Split miner: automated discovery of accurate and simple business process models from event logs”. In: *Knowledge and Information Systems* 59.2 (2019), pp. 251–284.

[12] Ekaterina Bazhenova, Susanne Buelow, and Mathias Weske. “Discovering Decision Models from Event Logs”. In: *Proceedings of BIS 2016*. LNBIP. Springer, 2016, pp. 237–251.

[13] James Bergstra et al. “Algorithms for Hyper-parameter Optimization”. In: *Proceedings of NIPS 2011*. Curran Associates Inc., 2011, pp. 2546–2554.

[14] Dominic Breuker et al. “Comprehensible Predictive Models for Business Processes”. In: *MIS Quarterly* 40.4 (2016), pp. 1009–1034.

[15] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. “Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity”. In: *International Journal of Cooperative Information Systems* 23.1 (2014), p. 1440001.

[16] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. “Automated discovery of business process simulation models from event logs”. In: *Decision Support Systems* 134 (2020), p. 113284.

[17] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. “Discovering generative models from event logs: data-driven simulation vs deep learning”. In: *PeerJ Computer Science* 7 (2021), e577.

[18] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. “Learning Accurate LSTM Models of Business Processes”. In: *Proceedings of BPM 2019*. Vol. 168. LNCS. Springer, 2019, pp. 286–302.

[19] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. “Simod : A Tool for Automated Discovery of Business Process Simulation Models”. In: *Proceedings of BPM Dissertation Award, Doctoral Consortium, and Demonstration Track 2019*. CEUR, 2019, pp. 139–143.

[20] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *Proceedings of NIPS Workshops 2014*. 2014, pp. 1–9.

[21] Chiara Di Francescomarino et al. “An eye into the future: Leveraging a-priori knowledge in predictive business process monitoring”. In: *Proceedings of BPM 2017*. LNCS. Springer, 2017, pp. 252–268.

[22] Nicola Di Mauro, Annalisa Appice, and Teresa M. A. Basile. “Activity Prediction of Business Process Instances with Inception CNN Models”. In: *Proceedings of AI*IA 2019*. LNCS. Springer, 2019, pp. 348–361.

[23] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. “Semantics and analysis of business process models in BPMN”. In: *Information and Software Technology* 50.12 (2008), pp. 1281–1294.

[24] Simon Dobrišek et al. “An edit-distance model for the approximate matching of timed strings”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 31.4 (2009), pp. 736–741.

[25] Marlon Dumas et al. *Fundamentals of Business Process Management*. Second Edition. Springer, 2018.

[26] Bedilia EstradaTorres et al. “Discovering business process simulation models in the presence of multitasking and availability constraints”. In: *Data & Knowledge Engineering* 134 (2021), p. 101897.

[27] Joerg Evermann, Jana Rebecca Rehse, and Peter Fettke. “Predicting process behaviour using deep learning”. In: *Decision Support Systems* 100 (2017), pp. 129–140.

[28] Cédric Favre, Dirk Fahland, and Hagen Völzer. “The relationship between workflow graphs and free-choice workflow nets”. In: *Information Systems* 47 (2015), pp. 197–219.

[29] Cédric Favre and Hagen Völzer. “The Difficulty of Replacing an Inclusive OR-Join”. In: *Proceedings of BPM 2012*. LNCS. Springer, 2012, pp. 156–171.

[30] João Gama et al. "A survey on concept drift adaptation". In: *ACM Computing Surveys* 46.4 (2014), pp. 1–37.

[31] Bartlomiej Gawin and Bartosz Marcinkowski. "How Close to Reality is the 'as-is' Business Process Simulation Model?" In: *Organizacija* 48.3 (2015), pp. 155–176.

[32] N. Gilbert and K. Troitzsch. *Simulation For The Social Scientist*. Open University Press, 2005.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[34] Xing Hao, Guigang Zhang, and Shang Ma. "Deep Learning". In: *International Journal of Semantic Computing* 10.03 (2016), pp. 417–439.

[35] Alan R. Hevner et al. "Design science in information systems research". In: *MIS quarterly* (2004), pp. 75–105.

[36] Markku Hinkka, Teemu Lehto, and Keijo Heljanko. "Exploiting event log event attributes in RNN based prediction". In: *Proceedings of ADBIS 2019*. CCIS. Springer, 2020, pp. 405–416.

[37] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[38] Monique Jansen-Vullers and Mariska Netjes. "Business process simulation–a tool survey". In: *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark* 38 (2006).

[39] Abdulrhman Al-Jebrni, Hongming Cai, and Lihong Jiang. "Predicting the Next Process Event Using Convolutional Neural Networks". In: *Proceedings of PIC 2018*. IEEE, 2018, pp. 332–338.

[40] Ivan Khodyrev and Svetlana Popova. "Discrete modeling and simulation of business processes using event logs". In: *Procedia Computer Science* 29 (2014), pp. 322–331.

[41] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. "Fundamentals of control flow in workflows". In: *Acta Informatica* 39.3 (2003), pp. 143–209.

[42] Harold W. Kuhn. "The Hungarian Method for the assignment problem". In: *Naval Research Logistics Quarterly* 2 (1955), pp. 83–97.

[43] M. Laguna and J. Marklund. *Business Process Modeling, Simulation and Design*. CRC Press, 2018.

[44] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[45] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2323.

[46] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. "Scalable process discovery and conformance checking". In: *Software and Systems Modeling* 17.2 (2018), pp. 599–631.

[47] Sander J. J. Leemans et al. “Stochastic process mining: Earth movers’ stochastic conformance”. In: *Information Systems* 102 (2021), p. 101724.

[48] Dafna Levy. *Production Analysis with Process Mining Technology*. 2014.

[49] Li Lin, Lijie Wen, and Jianmin Wang. “MM-Pred: A Deep Predictive Model for Multi-attribute Event Sequence”. In: *Proceedings of SIAM 2019*. Society for Industrial and Applied Mathematics, 2019, pp. 118–126.

[50] Orlenys López-Pintado et al. “Silhouetting the Cost-Time Front: Multi-objective Resource Optimization in Business Processes”. In: *Proceedings of BPM Forum 2021*. LNBIP. Springer, 2021, pp. 92–108.

[51] Sanidhya Mangal, Poorva Joshi, and Rahul Modak. *LSTM vs. GRU vs. Bidirectional RNN for script generation*. 2019. arXiv: `1908.04332`. URL: `http://arxiv.org/abs/1908.04332`.

[52] Felix Mannhardt et al. “Decision mining revisited - Discovering overlapping rules”. In: *Proceedings of CAiSE 2016*. LNCS. Springer, 2016, pp. 377–392.

[53] Niels Martin, Benoît Depaire, and An Caris. “The use of process mining in a business process simulation context: Overview and challenges”. In: *Proceedings of CIDM Symposium 2014*. IEEE, 2014, pp. 381–388.

[54] Niels Martin, Benoît Depaire, and An Caris. “The Use of Process Mining in Business Process Simulation Model Construction”. In: *Business & Information Systems Engineering* 58.1 (2016), pp. 73–87.

[55] Niels Martin, Benoît Depaire, and An Caris. “Using Event Logs to Model Interarrival Times in Business Process Simulation”. In: *Proceedings of BPM Workshops 2015*. LNBIP. Springer, 2015, pp. 255–267.

[56] Niels Martin, Luise Pufahl, and Felix Mannhardt. “Detection of batch activities from event logs”. In: *Information Systems* 95 (2021), p. 101642.

[57] Nijat Mehdiyev, Joerg Evermann, and Peter Fettke. “A Multi-stage Deep Learning Approach for Business Process Event Prediction”. In: *Proceedings of CBI 2017*. IEEE, 2017, pp. 119–128.

[58] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *Proceedings of ICLR Workshops 2013*. 2013, pp. 1–12.

[59] Jorge Munoz-Gama. *Conformance Checking and Diagnosis in Process Mining*. LNBIP. Springer International Publishing, 2016.

[60] Jorge Munoz-Gama, Josep Carmona, and Wil M. P. Van Der Aalst. “Conformance checking in the large: Partitioning and topology”. In: *Business Process Management*. LNCS. Springer, 2013, pp. 130–145.

[61] Laura Mŭruşter and Nick R. T. P. van Beest. “Redesigning business processes: a methodology based on simulation and process mining techniques”. In: *Knowledge and Information Systems* 21.3 (2009), pp. 267–297.

[62] Nicolò Navarin et al. “LSTM networks for data-aware remaining time prediction of business process instances”. In: *Proceedings of SSCI Symposium 2017*. IEEE, 2017, pp. 1–7.

[63] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. “BINet: Multivariate Business Process Anomaly Detection Using Deep Learning”. In: *Proceedings of BPM 2018*. LNCS. Springer, 2018, pp. 271–287.

[64] Chun Ouyang et al. “Modelling complex resource requirements in Business Process Management Systems”. In: *Proceedings of ACIS 2010*. 2010, pp. 1–11.

[65] Vincenzo Pasquadibisceglie et al. “Using convolutional neural networks for predictive process analytics”. In: *Proceedings of ICPM 2019*. IEEE, 2019, pp. 129–136.

[66] A. Pnueli. “The temporal logic of programs”. In: *Proceedings of SFCS 1977*. IEEE, 1977, pp. 46–57.

[67] Mirko Polato et al. “Time and activity sequence prediction of business process instances”. In: *Computing* 100.9 (2018), pp. 1005–1031.

[68] Mahsa Pourbafrani, Shuai Jiao, and Wil M. P. van der Aalst. “SIMPT: Process Improvement Using Interactive Simulation of Time-Aware Process Trees”. In: *Proceedings of RCIS 2021*. LNBIP. 2021, pp. 588–594.

[69] Mahsa Pourbafrani, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. “Supporting Automatic System Dynamics Model Generation for Simulation in the Context of Process Mining”. In: *Proceedings of BIS 2020*. LNBIP. 2020, pp. 249–263.

[70] Luise Pufahl, Tsun Yin Wong, and Mathias Weske. “Design of an Extensible BPMN Process Simulator”. In: *Proceedings of BPM Workshops 2017*. LNBIP. Springer, 2017, pp. 782–795.

[71] Efrén Rama-Maneiro, Juan C. Vidal, and Manuel Lama. *Deep Learning for Predictive Business Process Monitoring: Review and Benchmark*. 2021. arXiv: `2009.13251 [cs.LG]`. URL: `https://arxiv.org/abs/2009.13251`.

[72] Daniel Reißner et al. “Scalable alignment of process models and event logs: An approach based on automata and S-components”. In: *Information Systems* 94 (2020), p. 101561.

[73] Andreas Rogge-Solti et al. “In Log and Model We Trust? A Generalized Conformance Checking Framework”. In: *Proceedings of BPM 2016*. LNCS. Springer, 2016, pp. 179–196.

[74] Anne Rozinat, Ronny S. Mans, and Wil. M. P. van der Aalst. “Discovering simulation models”. In: *Information Systems* 34.3 (2009), pp. 305–327.

[75] Anne Rozinat, Ronny S. Mans, and Wil. M. P. van der Aalst. “Mining CPN Models: Discovering Process Models with Data from Event Logs”. In: *In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN*. 2006, pp. 57–76.

[76] Toma Rusinaite et al. “An approach for allocation of shared resources in the rule-based business process simulation”. In: *Proceedings of CompSysTech 2016*. ACM, 2016, pp. 25–32.

[77] Mohammadreza Fani Sani et al. “Conformance Checking Approximation Using Simulation”. In: *Proceedings of ICPM 2020*. IEEE, 2020, pp. 105–112.

[78] Jürgen Schmidhuber. “Deep Learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), pp. 85–117.

[79] Stefan Schönig et al. “Deep learning process prediction with discrete and continuous data features”. In: *Proceedings of ENASE 2018*. SCITEPRESS, 2018, pp. 314–327.

[80] Renuka Sindhgatta et al. “Exploring Interpretable Predictive Models for Business Processes”. In: *Proceedings of BPM 2020*. LNCS. Springer, 2020, pp. 257–272.

[81] Minseok Song and W. M. P. van der Aalst. “Towards comprehensive support for organizational mining”. In: *Decision Support Systems* 46.1 (2008), pp. 300–317.

[82] Suriadi Suriadi et al. “Discovering work prioritisation patterns from event logs”. In: *Decision Support Systems* 100 (2017), pp. 77–92.

[83] Niek Tax, Irene Teinemaa, and Sebastiaan J. van Zelst. “An interdisciplinary comparison of sequence modeling methods for next-element prediction”. In: *Software and Systems Modeling* 19.6 (2020), pp. 1345–1365.

[84] Niek Tax et al. “Predictive Business Process Monitoring with LSTM Neural Networks”. In: *Proceedings of CAiSE 2017*. LNCS. Springer, 2017, pp. 477–492.

[85] Sean J. Taylor and Benjamin Letham. “Forecasting at Scale”. In: *American Statistician* 72.1 (2018), pp. 37–45.

[86] Farbod Taymouri et al. “Predictive Business Process Monitoring via Generative Adversarial Nets: The Case of Next Event Prediction”. In: *Proceedings of BPM 2020*. LNCS. Springer, 2020, pp. 237–256.

[87] Irene Teinemaa, Anna Leontjeva, and Karl-Oskar Masing. “BPIC 2015: Diagnostics of building permit application process in dutch municipalities”. In: *BPI Challenge Report*. 2015.

[88] Julian Theis and Houshang Darabi. “Decay Replay Mining to Predict Next Process Events”. In: *IEEE Access* 7 (2019), pp. 119787–119803.

[89] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of NIPS 2017*. Curran Associates Inc., 2017, pp. 5998–6008.

[90] Nils Witt and Christin Seifert. “Understanding the Influence of Hyperparameters on Text Embeddings for Text Classification Tasks”. In: *Proceedings of TPDL 2017*. LNCS. Springer, 2017, pp. 193–204.

[91] Moe Thandar Wynn et al. “Business process simulation for operational decision support”. In: *Proceedings of BPM Workshops 2007*. LNBIP. Springer, 2008, pp. 66–77.

9. CODE REPOSITORIES

The implementations and the code required to run the experiments reported in the thesis can be found in the following code repositories:

I Simod tool: `https://github.com/AutomatedProcessImprovement/Simod`

II DeepGenerator approach: `https://github.com/AdaptiveBProcess/GenerativeLSTM`

III DeepSimulator tool: `https://github.com/AdaptiveBProcess/DeepSimulator`

IV Adapted LSTM(GAN) approach: `https://github.com/AdaptiveBProcess/LSTM-GAN`

A screencast of the Simod tool is available at `https://youtu.be/i9X5jwjuipk`. This video illustrates two typical scenarios. In the first scenario, the user manually explores the different preprocessing options of the tool to generate a simulation model. In the second scenario, the user defines a search space, and the tool automatically explores the combination looking for the optimal one.

ACKNOWLEDGEMENTS

I want to thank my supervisors Marlon Dumas and Oscar González-Rojas for their constant support and feedback throughout my process. Furthermore, their example taught me the ethics and rigor that must always be present in science.

I also want to thank my family, especially my parents Blanca Chávez and Manuel Camargo, because their unconditional support allowed me to face this challenge even from a distance and during a pandemic.

I would like to especially thank all my friends who have played the role of confidants and constant support throughout this process and with whom I have shared so many good memories over the years.

I want to thank Anti Alman for his help with the translation of my abstract. Finally, I want to thank my girlfriend, Maria Jürgen, for her help in translating some sections of my thesis and for her support and affection in which I have found the peace to complete this last part of the journey.

I am very thankful to David Eickhoff and the Signavio team, who welcomed me in Berlin and made me feel like part of the group during all my internship.

I would also like to acknowledge the European Research Council (PIX project) for funding my studies. The experiments provided in the thesis were performed using the resources from the University of Tartu High-Performance Computing Center.

SISUKOKKUVÕTE

Äriprotsesside simulatsioonimudelite automatiseeritud tuvastamine sündmuslogidest: Protsessikaevel ja süvaõppel põhinev hübriidlähenemine

Kaasaegsed organisatsioonid peavad oma äriprotsesse pidevalt muutma, et kohaneda erinevate sisemiste ja välimiste muutustega nagu näiteks uued konkurendid, uued regulatsioonid, muutused klientide ootustes või muutused strateegilistes eesmärkides. Näiteks pandeemia oludes võib jaemüüja internetikaubanduse maht suureneda 50%, samas kui kohapeal sooritatud ostude maht langeb näiteks 30%. Sellise muutunud olukorraga kohanemiseks võib jaemüüja otsustada töötajate ümberpaigutamise jaekauplustest ettevõtte ladudesse ja veebipõhise klienditeeninduse osakonda. Seda tüüpi otsuste teadlikuks vastuvõtmiseks on jaemüüjal vaja täpset hinnangut selle kohta, millist mõju antud otsus avaldaks kaupade kohaletoimetamise ja klientide päringutele vastamise aegadele.

Tavapärane lähenemine selliste hinnangute andmiseks on kasutada äriprotsesside simuleerimist. Äriprotsesside simuleerimine viitab äriprotsesside ajalise dünaamika arvuti abil uurimisele ja tegemist on kasuliku lähenemisega vastamaks „mis-oleks-kui“ tüüpi küsimustele äriprotsesside ümberdisainimise kontekstis. Samas, tulenevalt sellest kuidas äriprotsesside simuleerimist tavaliselt rakendatakse, on selle lähenemisega saadud ennustused teadaolevalt suhteliselt ebatäpsed.

Äriprotsesside simuleerimisel kasutatavad simulatsioonimudelid luuakse tavaliselt valdkonna ekspertide poolt käsitsi, kasutades manuaalseid andmekogumismeetodeid (intervjuud, vaatlused, valikulised andmete väljavõtted), mis omakorda muudab simulatsioonimudelite loomise ajamahukaks ja veaaltiks. Reaalsuses on äriprotsesside käitumine sageli oluliselt keerukam sellest, mida valdkonna eksperdid suudaksid käsitsi koostatud simulatsioonimudelites kajastada. Samas iga simulatsioonimudelist välja jäänud detail võib oluliselt mõjutada äriprotsesside simuleerimise täpsust ja usaldusväärsust. Teised olemasolevate äriprotsesside simuleerimise lähenemiste puudujäägid tulenevad äriprotsesside simulatsioonimootorite poolt tehtavatest põhimõttelistest eeldustest. Näiteks eeldus et inimesed töötavad robotitele (või tehase tööliinidele) sarnasel viisil, ehk et tööd tehakse töötundide jooksul järjepidevalt, püsiva tähelepanuga, kõrvalistele töödele aega kulutamata ja väsimatult. Ehk teisisõnu, olemasolevad äriprotsesside simuleerimise lähenemised ei ole võimelised kajastama ja seega ka taaslooma inimkäitumise keerukust.

Ülaltoodust lähtuvalt uurib käesolev doktoritöö järgnevat üleüldist küsimust: Kuidas automatiseeritult luua täpseid äriprotsesside simulatsioonimudeleid tuginedes ettevõttete infosüsteemidest kogutud andmetele? Antud küsimusega seonduv varasem teadustöö on näidanud, et äriprotsessi käitlemisandmete analüüsitehnikaid, mida tervikuna nimetatakse protsessikaeveks, on võimalik edukalt kasutada äriprotsesside simulatsioonimudelite pool-automatiseeritult loomiseks ning

vastavate tehnikate kohta kasutatakse üldnimetust andmepõhine simuleerimine. Käesolev doktoritöö juhib kõigepealt tähelepanu tõsiasjale, et täpsete simulatsioonimudelite loomine, kasutades olemasolevaid andmepõhise simuleerimise tehnikaid, nõuab käsitsi sekkumist ja peenhäälestamist. Selle puudujäägi lahendamiseks esitatakse ja hinnatakse käesolevas doktoritöös andmepõhise simuleerimise täielikult automatiseeritud lahendus, mis suudab tuvastada ja peenhäälestada simulatsioonimudeleid rakendades protsessikaeve tehnikaid. Lahenduse tuumikidee on automatiseeritult hinnata simulatsioonimudeli täpsust, arvestades nii tegevuste järjekorda kui ka tegevuste kestuseid. Täpsuse hinnangule tuginedes rakendatakse antud lähenemises Bayesi optimeerimisalgoritmi eesmärgiga saavutada maksimaalne sarnasus simulatsioonimudeli poolt genereeritud käitumise ja äriprotsessi käitlemisandmete vahel.

Seejärel näitab käesolev doktoritöö, et esitatud andmepõhise simuleerimise tehnika loob simulatsioonimudeleid, mis peegeldavad tegevuste järgnevusi täpselt, aga samas ei suuda sageli täpselt ennustada tegevuste kestust. Antud puudujääk on põhjustatud andmepõhise simuleerimise tehnikates tehtavatest eeldustest seoses ressursside käitumisega äriprotsessides. Antud puudujäägi lahendamiseks kombineeritakse käesolevas doktoritöös protsessikaevel tuginevaid andmepõhise simuleerimise tehnikaid ja generatiivseid süvaõppepõhiseid modelleerimise tehnikaid. Selles osas esitab käesolev doktoritöö kaks teaduslikku panust. Esiteks, lähenemine generatiivsete süvaõppe mudelite loomiseks, mis võimaldavad protsessi ajalooliste käitlemisandmete põhjal genereerida ajatembeldatud sündmuste järgnevusi koos sündmustele vastavate ressurssidega. Teiseks, lähenemine protsessikaevel tuginevate andmepõhise simuleerimise tehnikate ja generatiivsete süvaõppepõhiste modelleerimise tehnikate kombineerimiseks. Käesolev doktoritöö näitab, et sellise hübriidlähenemisega loodud simulatsioonimudelid võimaldavad luua simulatsioone, mis peegeldavad protsessi käitlemisandmetes sisalduvaid sündmuste järgnevusi ja kestuseid täpsemalt kui ainult andmepõhisele simulatsioonile tuginevad tehnikad ja täpsemalt kui ainult süvaõppele tuginevad tehnikad.

CURRICULUM VITAE

Personal data

Name:	Manuel Camargo
Date of Birth:	18.01.1985
Citizenship:	Colombian

Education

2018–2022	University of Tartu, Faculty of Science and Technology, doctoral studies, specialty: Computer Science
2018–2022	Universidad de los Andes, Department of Computer and Systems Engineering, doctoral studies, specialty: Engineering
2014–2016	Universidad de los Andes, Computer and Systems Engineering Department, master's studies, specialty: Architectures of Information Technologies.
2002–2012	District University Francisco José de Caldas, Faculty of Engineering, bachelor's studies, specialty: Computer Systems Engineer

Employment

2019–2021	University of Tartu, Junior Research Fellow
2017–2018	Universidad de los Andes, Doctoral Assistant
2016–2017	Paris School of Economics - PSE, Consultant
2014–2016	Universidad de los Andes, Graduate Research Assistant
2013–2014	4Sight Technologies, Software Testing Coordinator Mapfre USA

Scientific work

Main fields of interest:

- process mining
- business process simulation
- predictive process monitoring
- deep learning

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Manuel Camargo
Sünniaeg::	18.01.1985
Kodakondsus:	Colombia

Haridus

2018–2022	Tartu Ülikool, Loodus- ja täppisteaduste valdkond, doktoriõpe, eriala: Informaatika.
2018–2022	Los Andes Ülikool, Arvuti- ja süsteemitehnika teaduskond, doktoriõpe, eriala: Inseneriteadus.
2014–2016	Los Andes Ülikool, Arvuti- ja süsteemitehnika teaduskond, magistriõpe, eriala: Tarkvaraarhitektuur.
2002–2012	Francisco José de Caldas Piirkondlik Ülikool, Inseneriteaduse valdkond, bakalaureuseõpe, eriala: Arvutisüsteemide Inseneriteadus.

Teenistuskäik

2019–2021	Tartu Ülikool, infosüsteemide nooremteadur
2017–2018	Los Andes Ülikool, doktoriõppe abiõpetaja
2016–2017	Pariisi Majanduskool - PSE, konsultant
2014–2016	Los Andes Ülikool, teadusassistent
2013–2014	4Sight Technologies, tarkvaratestimiskoordinaator Mapfre USA projektis

Teadustegevus

Peamised uurimisvaldkonnad:

- protsessikaeve
- äriprotsessi simulatsioon
- ennustav protsesside jälgimise süsteem
- süvaõpe

LIST OF ORIGINAL PUBLICATIONS

Publications in the scope of the thesis

I Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. Learning Accurate LSTM Models of Business Processes. **In International Conference on Business Process Management**, Volume 168, 286–302, 2019.
Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.

II Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. Simod: A Tool for Automated Discovery of Business Process Simulation Models. **Proc. Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019**, 2019.
Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.

III Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. Automated discovery of business process simulation models from event logs. **Decision Support Systems**, Volume 134, 113284, 2020.
Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.

IV Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. Discovering generative models from event logs: data-driven simulation vs deep learning. **PeerJ Computer Science**, Volume 7, 577, 2021.
Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.

Publications out of the scope of the thesis

V Bedilia Estrada-Torres, **Manuel Camargo**, Marlon Dumas, and Luciano García-Bañuelos, and Ibrahim Mahdy, and Maksym Yerokhin. Discovering business process simulation models in the presence of multitasking and availability constraints. **Data & Knowledge Engineering**, Volume 134, 101897, 2021.

VI Bedilia Estrada-Torres, **Manuel Camargo**, Marlon Dumas, and Maksym Yerokhin. Discovering Business Process Simulation Models in the Presence of Multitasking. **In International Conference on Research Challenges in Information Science 2020**, 381—397, 2020.

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

113. **Jaak Randmets.** Programming Languages for Secure Multi-party Computation Application Development. Tartu, 2017, 172 p.
114. **Alisa Pankova.** Efficient Multiparty Computation Secure against Covert and Active Adversaries. Tartu, 2017, 316 p.
116. **Toomas Saarsen.** On the Structure and Use of Process Models and Their Interplay. Tartu, 2017, 123 p.
121. **Kristjan Korjus.** Analyzing EEG Data and Improving Data Partitioning for Machine Learning Algorithms. Tartu, 2017, 106 p.
122. **Eno Tõnisson.** Differences between Expected Answers and the Answers Offered by Computer Algebra Systems to School Mathematics Equations. Tartu, 2017, 195 p.

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

1. **Abdullah Makkeh**. Applications of Optimization in Some Complex Systems. Tartu 2018, 179 p.
2. **Riivo Kikas**. Analysis of Issue and Dependency Management in Open-Source Software Projects. Tartu 2018, 115 p.
3. **Ehsan Ebrahimi**. Post-Quantum Security in the Presence of Superposition Queries. Tartu 2018, 200 p.
4. **Ilya Verenich**. Explainable Predictive Monitoring of Temporal Measures of Business Processes. Tartu 2019, 151 p.
5. **Yauhen Yakimenka**. Failure Structures of Message-Passing Algorithms in Erasure Decoding and Compressed Sensing. Tartu 2019, 134 p.
6. **Irene Teinemaa**. Predictive and Prescriptive Monitoring of Business Process Outcomes. Tartu 2019, 196 p.
7. **Mohan Liyanage.** A Framework for Mobile Web of Things. Tartu 2019, 131 p.
8. **Toomas Krips.** Improving performance of secure real-number operations. Tartu 2019, 146 p.
9. **Vijayachitra Modhukur.** Profiling of DNA methylation patterns as biomarkers of human disease. Tartu 2019, 134 p.
10. **Elena Sügis.** Integration Methods for Heterogeneous Biological Data. Tartu 2019, 250 p.
11. **Tõnis Tasa.** Bioinformatics Approaches in Personalised Pharmacotherapy. Tartu 2019, 150 p.
12. **Sulev Reisberg.** Developing Computational Solutions for Personalized Medicine. Tartu 2019, 126 p.
13. **Huishi Yin.** Using a Kano-like Model to Facilitate Open Innovation in Requirements Engineering. Tartu 2019, 129 p.
14. **Faiz Ali Shah.** Extracting Information from App Reviews to Facilitate Software Development Activities. Tartu 2020, 149 p.
15. **Adriano Augusto**. Accurate and Efficient Discovery of Process Models from Event Logs. Tartu 2020, 194 p.
16. **Karim Baghery.** Reducing Trust and Improving Security in zk-SNARKs and Commitments. Tartu 2020, 245 p.
17. **Behzad Abdolmaleki.** On Succinct Non-Interactive Zero-Knowledge Protocols Under Weaker Trust Assumptions. Tartu 2020, 209 p.
18. **Janno Siim.** Non-Interactive Shuffle Arguments. Tartu 2020, 154 p.
19. **Ilya Kuzovkin.** Understanding Information Processing in Human Brain by Interpreting Machine Learning Models. Tartu 2020, 149 p.
20. **Orlenys López Pintado.** Collaborative Business Process Execution on the Blockchain: The Caterpillar System. Tartu 2020, 170 p.
21. **Ardi Tampuu.** Neural Networks for Analyzing Biological Data. Tartu 2020, 152 p.