BIRMINGHAM CITY UNIVERSITY

DOCTORAL THESIS

A Deep Learning Approach to Business Process Mining

Author:
Khadijah Muzzammil HANGA

Supervisors:
Dr. Yevgeniya KOVALCHUK
Prof. Mohamed GABER

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Data Analytics and Artificial Intelligence
School of Computing and Digital Technology

February 24, 2023

Declaration of Authorship

I, Khadijah Muzzammil HANGA, declare that this thesis titled, "A Deep Learning Approach to Business Process Mining" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: [illegible]

Date: 07/10/22

"We are what we repeatedly do. Excellence, then, is not an act but a habit."

Aristotle

BIRMINGHAM CITY UNIVERSITY

Abstract

Faculty of Computing, Engineering and the Built Environment
School of Computing and Digital Technology

Doctor of Philosophy

A Deep Learning Approach to Business Process Mining

by Khadijah Muzzammil HANGA

Competing and evolving markets force organisations to continuously monitor, evaluate, and optimise their business processes. To do the task at scale, organisations often turn to automatic mining of process execution logs constantly generated by various information systems. Many open-source and commercial tools have been developed in recent years to help organisations perform various process mining tasks using process execution logs (often called event logs), such as process discovery, conformance checking, and detecting drifts in processes. Compared to traditional process mining techniques such as Petri nets and Business Process Model and Notation (BPMN), deep learning methods such as Recurrent Neural Networks and Long Short-Term Memory (LSTM) in particular have proven to achieve better performance in terms of accuracy and generalising ability when predicting sequences of activities performed as part of business processes based on event logs. However, unlike traditional network-based process mining techniques that can be used to visually present all activity sequences of the discovered business process, existing deep learning-based methods for process mining lack a mechanism explaining how the activity sequence predictions are made. To address this limitation, this thesis proposes an extensible process mining solution that combines the benefits of interpretable graph-based methods and more accurate but implicit deep learning methods. The main contributions of this research are: (i) building an LSTM model for predicting business process activity sequences from event logs that outperforms existing state-of-the-art deep learning solutions; (ii) proposing a graph-based approach to explaining the decision-making process of the LSTM model when predicting business process activity sequences; and (iii) developing methods for detecting and localising sudden concept drift in event logs (i.e., offline) and event streams (i.e., online) using deep learning and graph-based approaches. The proposed methods have been extensively evaluated by conducting experiments using real-life and artificial event logs and have been demonstrated to outperform existing state-of-the-art solutions in many cases.

Acknowledgements

The research presented in this thesis was funded by the Petroleum Technology Development Fund (PTDF). Without this financial support, completing this degree would not have been possible. Words can not express how grateful I am for this opportunity. A big thank you to the PTDF team. First and foremost, I would like to express my sincere gratitude to my supervisory team, including, Dr Yevgeniya Kovalchuk, and Prof. Mohamed Medhat Gaber, for their continuous support throughout my PhD study. I am indebted to both of them for their insight, ideas, guidance, and patience throughout my research. Their feedback, advice, and expertise have allowed me to shape this thesis. It has indeed been a great pleasure working with both of them. I am grateful to Dr Atif Azad, Dr Shadi Basurra, and Dr Jason Hockman, the members of my PhD progression panel for three consecutive years, for their constructive feedback. I also thank the Doctoral Research College (DRC) staff and the Faculty of Computing, Engineering, and the Built Environment (CEBE) of Birmingham City University for their administrative support during my research. A special thank you goes out to Dr Mariam Adedoyin-Olowe for her encouraging words and kindness. I must appreciate and thank my colleagues in the Data Analytics and Artificial Intelligence (DAAI) research group; Amna, Hansi, Zakaria, Aliyu, Hossein, Besher, and Lorraine, for their friendship and kindness, which has made my PhD journey at BCU extraordinary. I have to acknowledge Elias Akin, who has been very supportive. He was my coding wizard, always willing to spare me some minutes despite his tight schedule, and spotted those bugging bugs in my codes in the nick of time. My special appreciations go to my parents, Muzzammil Sani Hanga, and Fatima Batul Mukhtar. No words can truly express my gratitude for their unrepayable love and support. I sincerely appreciate my husband, Kabiru Mahmud Yabo, for his constant support and encouragement throughout my studies. To Muhammad and Fatimah-Batool for sharing my PhD journey accordingly. Thank you for being such good children and making it possible for me to complete what I started. Another special appreciation goes to my sisters, Humaira, Nusaiba, Zainab, and Tayseer, I can't thank you enough for your support. A big thank you to Rahina Muhammad for leaving home to offer me support whenever I needed it. I appreciate my mother and father-in-law, Aisha Mahmud Yabo, Mahmud Shehu Yabo and my sisters-in-law for their prayers and love. I am very grateful to all my aunties, uncles, cousins, nieces, and nephews. Thank you all for your prayers, well wishes, visits, and encouragement. I pray for my aunty, Maryam Musa Suleiman, who passed away without watching me complete my PhD. She would have been delighted to see the day. I am incredibly grateful to my cousins Ya. Nabilah, Ubaidullah, his wife Zainab, and Abdul-Ilah for their moral support, generosity, advice, and willingness to help in whatever way. Finally, I owe a great thank you to my long-term friends, Maryam Awal, Hadiza Aminu, Fati Imam, Zeezah, Zambuk, Anfa, Hadiza Mahmud, Fathiyyah, Waheeda, Anty Mash. They were of great emotional support during my journey. I will definitely not forget, Uncle Umaru, his wife Aunty Hawwa and Aunty Amina, thank you very much for being there, and making my stay in Birmingham amazing. I must say, I am a person of people. Their love has surrounded me since the beginning of my PhD. If I have to thank everyone, my list will be endless. All of them have touched me in one way or another. A massive thank you to you all.

Contents

List of Figures

List of Tables

List of Abbreviations

BPIC	**B**usiness **P**rocess **I**ntelligence **C**hallenge
BPM	**B**usiness **P**rocess **M**anagement
CSV	**C**omma **S**eparated **V**alues
DFG	**D**irectly **F**ollows **G**raph
DL	**D**eep **L**earning
D-L distance	**D**amerau–**L**evenshtein distance
ETM	**E**volutionary **T**ree **M**iner
GPS	**G**lobal **P**ositioning **S**ystem
HM	**H**euristic **M**iner
IM	**I**nductive **M**iner
IoT	**I**nternet **o**f **T**hings
LSTM	**L**ong **S**hort **T**erm **M**emory
MAE	**M**ean **A**bsolute **E**rror
ML	**M**achine **L**earning
NLP	**N**atural **L**aguage **P**rocessing
PGraphDD-QM	**P**rocess **G**raph **D**rift **D**etection **Q**uality **M**etrics
PGraphDD-SS	**P**rocess **G**raph **D**rift **D**etection **S**imilarity **S**core
PGraphDL	**P**rocess **G**raph **D**rift **L**ocalisation
PM	**P**rocess **M**ining
PPM	**P**redictive **P**rocess **M**ining
RFID	**R**adio **F**requency **ID**dentification
RNN	**R**ecurrent **N**eural **N**etwork
SM	**S**plit **M**iner
XES	e**X**tensible **E**vent **S**tream

List of Symbols

e	event
a	activity
P	transition probabilities
N	set of nodes
E	set of edges
A_ij	adjacency matrix
P_ij	prediction probability matrix
σ	trace
μ	trained LSTM model
ϕ	probability threshold
L	event log
M	process model
T	set of traces
$\mathcal{S}$	event stream
$\mathcal{G}$	process graph
$\mathcal{E}$	set of events
$\mathcal{P}$	set of attributes
$\mathcal{T}$	a complete trace
$\mathcal{R}$	reference window
$\mathcal{D}$	detection window

This thesis is dedicated to the five people without whom this journey wouldn't have been possible. To my loving parents, Muzzammil Sani Hanga and Fatima Batul Mukhtar, my husband, Kabiru Mahmud Yabo, and my son and daughter, Muhammad and Fatimah-Batool, whose unconditional love, affection, support, and prayers are my strength in everything I do.

S

Chapter 1

Introduction

1.1 Preamble

Businesses go through daily processes to accomplish their mission; the better their processes, the more profitable the organisation. Due to the rapid emergence and evolution of new business models and the increasing complexity of global operations, management of business processes has become more important. Factors such as the increase in the number of orders of goods, the need for faster information transfer, better decision making, international competitors, and the need to adapt to changes in demand challenge the profitability of small and large companies (Ko, Lee, and Lee, 2009). To address these challenges, many companies started using information systems to perform their operations and interact with customers. Some of these include supply chain management (SCM), Enterprise Resource Planning (ERP), and Relationship Management system (CRM) systems. SCM systems collect information about multiple individuals and tasks related to the production and transport of goods and services. ERP systems record all transactions that happen in a business process; they help manage the various steps of a company's operations, such as purchasing, sales, and marketing. CRM systems keep track of the details of a customer's interactions with a company. These information systems collect a huge amount of event data. Although these data can be used for accounting, auditing, and business analytics, the challenge is to derive valuable insights from it and to have the ability to oversee and improve workflows.

This chapter introduces the research problem addressed in this thesis, detailed in Chapter 2. It also presents the research gaps, the research aim and objectives. Afterwards, it highlights the main contributions and concludes by setting the thesis outline.

1.2 Business Process Management

Many modern businesses realise that keeping up with emerging and competing markets requires developing and using novel ways of managing business processes (Jokonowo et al., 2018). Business Process Management (BPM) comprises techniques and tools to identify, analyse and monitor business processes to optimise their performance (Dumas et al., 2013). BPM is considered a continuous cycle comprising a number of phases (Dumas et al., 2013). With BPM, organisations can ensure that their processes are efficient and effective, resulting in a profitable organisation.

1.3 Process Mining

Process mining (PM) emerged from BPM to help organisations discover, evaluate and improve workflows using processes recorded in an event log. Since then, PM has made the BPM cycle more effective and efficient by eliminating manual processes and allowing higher-level modelling and reengineering work.

Organisations often use PM to audit their business processes, derive valuable insight about business operations, and improve their services and customer relations. In particular, PM is used to automatically extract process models from event logs to analyse how processes are implemented in reality, providing the avenue for monitoring and improvement (Figure 1.1). PM exposes how processes are being executed instead of how they should be executed. A PM model differs from a manually drafted process model because it can accurately account for what is happening in reality (Lau et al., 2009). It brings about transparency and delivers timely factual evidence and insights from transaction records. More specifically, PM extracts useful information by answering the following questions.

- What process are people following?
- Where are the obstacles in the process?
- Where do people/machines drift from the outlined process?

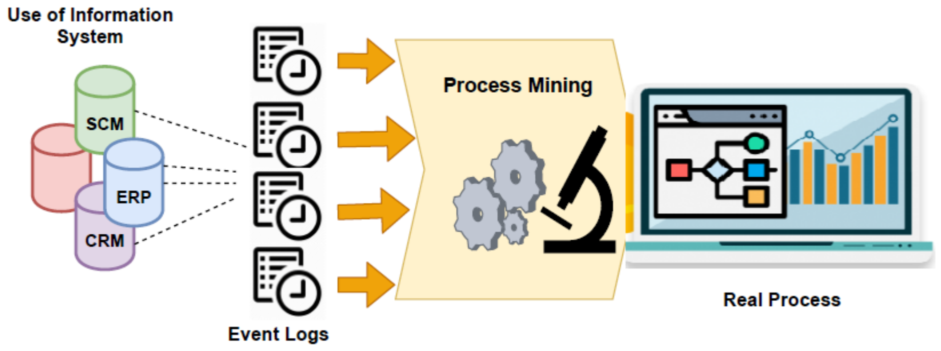


FIGURE 1.1: Process Mining Framework

An example use case is retail, where collected data from all stages, i.e., from order placement to delivery, can be used by PM techniques to find gaps in the ongoing process, which can then be addressed based on organisational goals. These goals can be customer satisfaction, productivity improvement, or maximising profit. Another use case is healthcare; PM can improve the accuracy of clinical investigation results collected worldwide. Other industries where PM is applicable include manufacturing, logistics, accounting, oil and gas, food processing, and distribution. Commonly performed PM tasks are *Process discovery*, *conformance checking*, and *process model enhancement*.

1.4 Concept Drift Detection

Business processes are also prone to constant changes, which are motivated by numerous internal and external factors. Changes can be in response to seasonal trends, constantly evolving market conditions and customer preferences, and introducing new rules and regulations. Changes push organisations to adapt and update their business processes constantly. For example, adding a quality assurance stage to comply with a new regulation will require reorganising the process's sequence flow. Concept drift occurs when a change is observed in the process while it is being analysed (Bose et al., 2011; Bose et al., 2013). Concept drifts are planned (for example, regulatory changes) or unexpected (for example, change in resource capacity) (Seeliger, Nolle, and Mühlhäuser, 2017) Detecting such changes can help answer a question like how to explore the impact of a change on a process regarding temporal and cost measures?. Traditionally, this question is answered based on the foreknowledge and expertise of managers and business analysts. However, this only works well if the number of changes is small or if the impact of the change is not critical. When the number of changes frequently increases, it is difficult to calculate the impact of potential changes. In these situations, analysts need a tool that allows them to automatically explore multiple changes in a short time, making the decision-making process more manageable.

1.5 Problem Statement

Despite its great potential, PM is hampered by several limitations. *Process discovery* is the most studied type of PM. It automatically constructs a process model appropriately displaying the observed behaviour as it is captured in an event log without any inferred information. These models enable businesses to evaluate performance, check compliance, spot anomalies and suggest improvements. Most process discovery algorithms use frequency-based heuristics, genetic-based heuristics, probabilistic-based approaches, and the theory of regions to produce a process model. The applicability and effectiveness of these approaches depend on the event log features and structure of processes. When applied to real-life event logs, the majority of existing process discovery methods demonstrate such limitations as:

- Producing broad and spaghetti-like models or models with poor fitness and precision (i.e., these methods cannot discover process models that would express the observed behaviour in the best possible way) (Augusto et al., 2018).
- It has not proven easy to achieve a compromise between the four process discovery quality metrics, *fitness, precision, generalisation* and *complexity* (Augusto et al., 2018).

Also, traditional PM techniques assume processes are stable; thus, these techniques are not suitable for analysing modern-day dynamic business processes. State-of-the-art process drift detection methods are predominantly based on classical statistical analysis. While many of them were reported to perform well, they have some limitations. For example,

- The windowing technique used in many concept drift detection algorithms is highly dependent on the right choice of window size; a wrong window size can result in a high number of false negatives and false positives.
- Some methods are unable to locate the exact moment of a drift.
- Some methods are not automated; they require human involvement in feature selection and change point recognition, making them impractical.

- Most of the existing methods for detecting concept drifts in business processes are designed to work offline (requiring the entire event logs featuring cases before and after a drift has occurred). Some online methods detect drifts with a long delay and some do not perform well on processes whose logs display many distinct executions. Thus, detecting drifts in the online scenario (i.e., as they happen) remains challenging.

Taking into account the limitations of PM described above, this thesis will focus on building more accurate PM models. For this purpose, it will be assumed that a model must accurately reproduce the current state of a business process (AS-IS) before it is applied to solve other PM tasks. We hypothesise that using high-performing deep learning models for this purpose might help counter several limitations.

1.6 Deep Learning

Deep Learning (DL) is a type of machine learning based on Artificial Neural Networks (ANN). DL achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts. Each concept is defined with more precise concepts and abstract representations determined with respect to less abstract ones (Goodfellow, Bengio, and Courville, 2016; Wani et al., 2020). DL makes use of several non-linear processing units to perform feature extraction as well as transformation. The output from each layer is taken as input by each successive layer (Brownlee, 2016).

DL approaches have recently gained popularity in the PM field. In particular, several Recurrent Neural Network (RNN) architectures, such as Long Short-Term Memory (LSTM), have been applied to predict subsequent events in business processes, time of occurrence and completion, and resources that trigger the events. Due to its ability to retain information over a long period, LSTM can learn long-term dependencies in a sequence, preventing prior information from being lost (Hochreiter and Schmidhuber, 1997). This quality enables an LSTM model to consider all process instances when predicting.

1.7 Research Gaps

Compared to the earlier PM techniques, LSTM-based methods have shown better performance in terms of accuracy and generalising ability. However, some of the existing proposals have shortcomings, including the following: the models not performing well when making long-term predictions and when used on event logs with many repeated events (Tax et al., 2017; Tello-Leal et al., 2018); inability to handle numerical variables, which made generating sequences of timestamped events impossible (Evermann, Rehse, and Fettke, 2017).

There are debates in the PM community on which methods to use. Recent studies such as (Evermann, Rehse, and Fettke, 2017) and (Tax et al., 2017) consider the DL's ability to generalise and yield high accuracy more imperative than generating explicit process models represented using graphical notation such as Petri-nets, BPMN, and Event-driven Process Chains or UML activity diagrams (Tello-Leal et al., 2018). Unlike these studies, we argue that accuracy and explainability are crucial in the field of PM. Process structures are implicitly reflected in models generated by DL-based methods, and no visually explainable process graphs are produced, which limits the value of such models.

Several techniques have been proposed for detecting business process drifts, e.g. (Bose et al., 2013; Carmona and Gavalda, 2012; Hassani, 2019; Maaradji et al., 2017; Manoj Kumar, Thomas, and Annappa, 2015; Martjushev, Bose, and Aalst, 2015; Zheng, Wen, and Wang, 2017; Seeliger, Nolle, and Mühlhäuser, 2017). The main objective of these techniques is to extract features, such as patterns from the process behaviour recorded in event logs, and perform certain analyses to detect drifts. Although some of these methods have been reported to perform well, they have several limitations. In particular, most existing methods are designed to detect drifts that occur in event logs only (i.e. complete process execution); they do not work in online settings, where streams of events incrementally record the executions of a business process.Some of the approaches that work in online settings detect drift with a long delay, as they need to wait for the trace to complete. Furthermore, since many methods rely on statistical tests over trace distributions, which may not have sufficient data samples when there is high variability in the log, they tend not to perform well on unpredictable processes whose logs contain a high number of distinct traces compared to the total number of traces.

This research was motivated by the importance of analysing and tracking processes in the oil and gas industry supply chain. Detected concept drifts in oil and gas supply chain processes can indicate fraud or inefficiency, and subsequently are of paramount importance. However, as a proof of concept, the case studies reported in this thesis were based on other industries due to the absence and confidentiality of process data from the oil and gas industry.

1.8 Research Aim and Objectives

1.8.1 Aim

The aim of the thesis is to develop a new approach to PM by combining the benefits of interpretable graph-based methods and more accurate but implicit DL methods.

1.8.2 Objectives

The main objectives of the thesis are the following:

1. Build an accurate LSTM model for predicting business process event sequences from event logs;
2. Construct a directly-follows graph (DFG) explaining the decision-making process of the LSTM model when predicting business process event sequences;
3. Detect and localise concept drift in business processes;
4. Validate the research using real-life event logs.

1.9 Contributions

This thesis provides the following contributions to the state-of-the-art of the PM field:

1. *Better performing LSTM models for PM.* Two LSTM models that achieve higher accuracy when predicting subsequent events in business processes than that

achieved by state-of-the-art LSTM models were proposed. The improved performance of the proposed LSTM models can be attributed to the different approaches used to generate inputs for the models and the model network architectures, which employ an embedding, a dense layer, and an LSTM layer.

2. *A graph-based approach to interpreting LSTM models.* The idea of generating graphs of different complexity to visually explain the decision-making process of an LSTM model when predicting the subsequent events in business processes was proposed. The graphs can explore the LSTM model's performance and identify complex cases as per the model's decision so that measures could be taken to improve the model performance in such cases. Furthermore, the graphs can be used to perform various PM tasks such as model discovery, conformance check, and investigation of non-compliance cases.

3. *Novel methods for detecting sudden concept drift.* Two new methods, PGraphDD-QM and PGraphDD-SS, that can detect sudden concept drift in business processes from a control-flow perspective in offline and online scenarios were proposed.

   For offline drift detection, process graphs are generated based on two event logs covering different time periods to detect process drifts that may have occurred between the two logs (or time periods). In particular, an LSTM model trained on an event log covering one period of time is applied to another event log covering a different time period. Process graphs representing the behaviour of the different time periods (i.e. the two different time periods) are generated using the decisions of the LSTM model about the most probable business process flow. According to PGraphDD-QM, the model performance is then separately estimated over the previous and new process graphs using the F-score metric, and the two sets of measures are compared. The change in values of the two sets of measures is assumed to indicate a concept drift. According to PGraphDD-SS, the DFGs generated based on the LSTM model decisions for two different time periods are used to verify the drift, both visually by detecting structural changes and by measuring the similarity score between the adjacency matrices of the two different graphs to estimate the amount of changes observed in the business process after the drift has occurred.

   For online drift detection, it is assumed that individual graph objects are received continuously over time in a graph stream. In particular, the LSTM model trained on a stream of logged events covering a previous period of time is applied to a newly generated stream of events as they occur. Graph streams representing the process behaviour of different time periods (i.e., the previous and new time periods) are generated using the decisions of the LSTM model about the most probable business process flow. According to PGraphDD-QM, an LSTM model trained on a stream of logged events covering a previous period of time is applied to a newly generated stream of events as they occur. Graph streams representing the process behaviour of different time periods (i.e., the previous and new time periods) are generated using the decisions of the LSTM model about the most probable business process flow. The model performance is then separately estimated over the previous and new graph streams using the F-score metric, and the two sets of measures are compared. The change in values of the two sets of measures is assumed to indicate a concept drift. According to PGraphDD-SS, the DFGs generated based on the LSTM model decisions for two different time periods are used to verify the drift, both visually by detecting structural changes and by measuring the similarity score between the adjacency matrices of the two different graphs to estimate the amount of changes observed in the business process after the drift has occurred.

4. *A method for localising concept drifts.* In addition to the drift detection, a method called PGraphDL, which localises drift and correctly identifies the activities or fragments involved in each detected concept drift from the graph, thus providing additional information about the drift, such as which activities have been added or removed was proposed. This information supports further analysis. According to the proposed method, two process graphs (graph A and graph B) from two different windows (i.e. reference and detection) are taken as input. A user selects any path of interest from the base model by specifying an index. The best matching path is searched in graph B for each possible path in graph A by computing the positional score for each candidate path in graph B. The positional score is calculated as the number of activities in graph A located in the same position in graph B divided by the length of the selected path in graph A. The candidate path in graph B with the maximum positional score is selected as the best-matching path in graph A. A maximum positional score of 1 indicates that the paths in the two graphs are identical (i.e. there is no drift). At the same time, each activity in graph B that is not found in the same position as the activity in question in graph A is declared as a positional drift.

1.10 Publications

The following journal papers have been published while working towards this thesis:

- Hanga, K.M. and Kovalchuk, Y., 2019. Machine learning and multi-agent systems in oil and gas industry applications: A survey. Computer Science Review, 34, p.100191.
- Hanga, K.M., Kovalchuk, Y. and Gaber, M.M., 2020. A graph-based approach to interpreting recurrent neural networks in process mining. IEEE Access, 8, pp.172923-172938.
- Hanga, K.M., Kovalchuk, Y. and Gaber, M.M., 2022. PGraphD*: methods for drift detection and localisation using deep learning modelling of business processes. Entropy, 24(7), p.910.

1.11 Data Availability

The datasets used in this thesis can be found as follows:

- Loan Application Process logs Maaradji et al., 2017: https://data.4tu.nl/articles/dataset/Business_Process_Drift/12712436
- Dutch Municipality logs (BPIC 2015) Dongen, 2015: https://data.4tu.nl/collections/_/5065424/1
- Helpdesk logs Polato, 2017: https://data.4tu.nl/articles/dataset/Dataset_belonging_to_the_help_desk_log_of_an_Italian_Company/12675977

- Road Traffic Fine Management Process Leoni and Mannhardt, 2015: https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249
- BPIC 2012 Dongen, 2012: https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204
- BPIC 2013I Steeman, 2013b: https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_incidents/12693914
- BPIC 2013C Steeman, 2013a: https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_closed_problems/12714476

1.12 Code Base

The source code of the methods presented in this thesis can be found in the following repository:
https://github.com/dijahanga/DL_Approach_To_Process_Mining.git

1.13 Overview of the Thesis

The rest of this thesis is organised as follows.

Chapter 2: State of the Art and Related Work. This chapter presents the state-of-the-art related to the core research area of this thesis. It provides a background on BPM, PM, process discovery, DL in PM (including details on LSTM), and concept drifts with its different classes and perspectives.

Chapter 3: Predicting Next Activities in Business Processes. This chapter introduces the first contribution of this thesis. It presents the proposed LSTM models used to achieve higher accuracy when predicting the next events in business processes and a method for generating graphs representing each model's decision-making process. The graphs can be used to understand the performance of the LSTM models and perform a variety of PM tasks such as process discovery, conformance checking, and investigating cases of non-compliance.

Chapter 4: Concept Drift Detection. This chapter describes the second contribution of this thesis. It introduces two new methods called PGraphDD-QM and PGraphDD-SS (where 'P' stands for *process*, 'DD' for *drift detection*, 'QM' for *quality metrics* and 'SS' for *similarity score*) for detecting sudden concept drifts in business processes from a control-flow perspective in both offline and online scenarios.

Chapter 5: Drift Localisation. This chapter describes the third contribution of this thesis. It introduces a new method called PGraphDL (where 'P' stands for *process*, 'DL' for *deep learning*) for localising sudden concept drifts in dynamic business processes based on streams of logged events.

Chapter 6: Conclusion and Future Directions. This chapter concludes the research work presented in the thesis. It summarises the experimental results and discusses possible avenues for future work.

Chapter 2

State of the Art and Related Work

This chapter presents the state of the art and background of this study necessary to understand the proposed solutions. This includes understanding the basic theory of BPM, PM, process discovery methods and their limitations (see a summary of the review protocol in Figure 2.1). It also introduces DL, its place in the PM field, and its common drawback. Lastly, it discusses business concept drift detection, localisation in PM, and related topics. The Chapter positions the research presented in this thesis.

TABLE 2.1: Review protocol

	Review phase	**Detail**
1.	KEYWORDS IDENTIFICATION	The research purpose guided the keyword identification.
2.	SEARCH STRINGS CONSTRUCTION	The following search strings: “process mining”, “business process management”, “process discovery”, “deep learning”, “RNN”, “LSTM”, “DFG”, “concept drift detection”, and "concept drift localisation" were used and combined to obtain a wider overview of the specific concept and techniques.
3.	IDENTIFICATION OF ADDITIONAL KEYWORDS	The search strings led to additional keywords (e.g., “event-stream”, “petri-nets”, “BPMN”, “business process”, “process model”), which are a conceptual subset of the main ones.
4.	DATABASE SELECTION	The following databases were used to gather articles related to our query: Google Scholar, ScienceDirect, ACM Digital Library, Scopus, IEEE Xplore Digital Library and Web of Science.
5.	EXPLOITATION OF THE SEARCH STRINGS	The query was orientated towards the highest comprehensiveness to guarantee a wide coverage of the results.
6.	REVIEW OF THE ARTICLES	The papers’ title, abstract and keywords were carefully read to assess if they fit our research topic and purpose. The selected articles were analysed.
7.	ELICITATION OF THE SCIENTIFIC GAPS	The results from the analysis led to the identification of the scientific gaps.

2.1 Business Process Management

A process is a sequence of activities that are completed to achieve a goal. A business process is thus a process dedicated to achieving a goal for a business. A business process consists of a series of events and activities, where events represent things that happen continuously (Dumas et al., 2013). Businesses go through daily processes to accomplish their mission. The better their processes, the more influential the business will be. Humans carry out many processes without knowing, e.g. getting ready for school, using an ATM, reading e-mail, etc. However, as processes become more complex, the need to document arises. Businesses need to document because it offers them control over how activities are carried out in their organisation, allowing standardisation. The simplest way to document a process is to create a list. The list outlines each step of the process, which can be easily checked after completion. For example, Figure 2.1 shows a simple process of how opening a bank account steps may look.

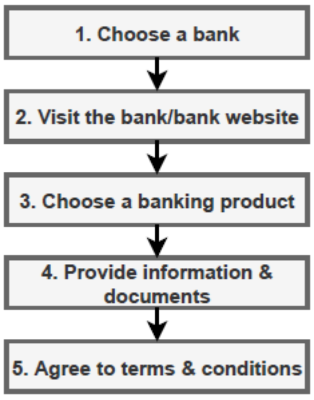

1. Choose a bank
2. Visit the bank/bank website
3. Choose a banking product
4. Provide information & documents
5. Agree to terms & conditions

FIGURE 2.1: Example of a process: how to open a bank account

As organisations begin to document their processes, tracking them becomes an administrative task. As processes change and improve, it is vital to keep track of them and manage them so that they can be easily updated. BPM comprises techniques and tools to identify, analyse and monitor business processes to optimise their performance (Dumas et al., 2013). With BPM, organisations can ensure that their processes are efficient and effective, resulting in a profitable organisation.

BPM is considered as a continuous cycle consisting of several phases (see Figure 2.2. The first phase of the life cycle is *Process identification*, where a collection of interrelated processes is identified. The second phase is *Process discovery*, here the identified processes are depicted as one or several as-is process models. In the third phase, which is *Process analysis*, issues related to the as-is process models are gathered and possibly quantified using performance measures. To address the issues collected in the previous phase (i.e. third phase), the *Process redesign* which is the fourth phase, modifies the as-is process model to the to-be process model to meet the organisation's desired way of functioning. In the fifth phase, *Process implementation*, changes required to transition from the as-is process to the to-be process are prepared and performed. This involves organisational change management and automation. The last phase in the life-cycle is the *Process monitoring*, where relevant data are collected and analysed from the redesigned to-be process to determine how well the process performs in terms of its performance measures and objectives. Any identified issues must be addressed. The BPM cycle may be repeated until no issues are found (Dumas et al., 2013).

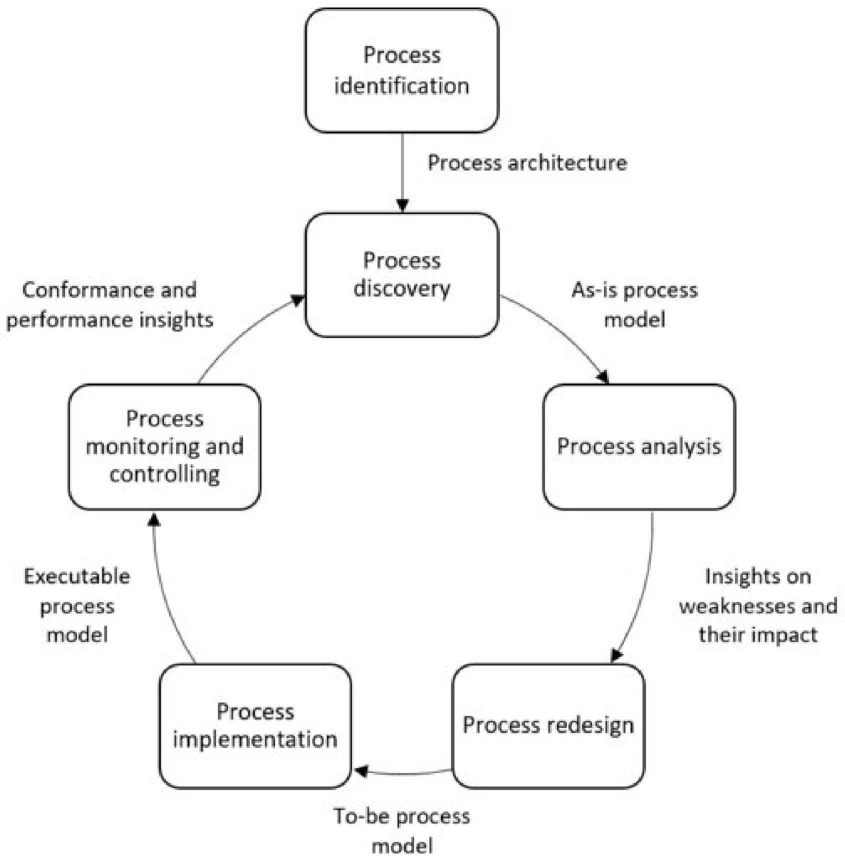


FIGURE 2.2: BPM life-cycle, reconstructed from (Dumas et al., 2013)

2.2 Process Mining

PM is an emerging research domain that is gaining much interest due to its great benefits. PM emerged from BPM to help organisations discover, assess, and improve workflows using event logs of processes. Since then, PM has become a cutting-edge part of BPM. It has made the BPM life-cycle more effective and efficient by dismissing manual processes and allowing higher-level modelling and re-engineering work. It gives organisations a factual picture of their processes to achieve the best outcomes. The goal of PM is to discover, monitor, and improve real processes from event data constantly collected by organisations and rarely used in the analysis of processes. It enables the discovery of sequences of tasks performed in a business process and also the interplay between participants in that process. Mueller and Ali in (Mueller and Ali, 2021) likened the PM scenario to a doctor-patient relationship. The doctor fully examines a patient to become clear about the patient's current state, and this helps the doctor provide the best diagnosis or solution.

PM focuses on establishing a solid relationship between *a process model* and *reality* captured from an event log. The terms *Discovery/Play-In*, *Play-Out*, and *Replay* are used to reflect on this relation. Figure 2.3 illustrates these three concepts on a graph-based process model representation. *Play-Out* refers to the general use of process models, i.e., to generate behaviour. Play-In, referred to as *Discovery* or *inference*, is the opposite of Play-Out, i.e., an example behaviour is taken as input and the goal is to construct a model. *Replay* takes an event log and a process model as input, and the event log is "replayed" on top of the process model (Leemans, 2015).

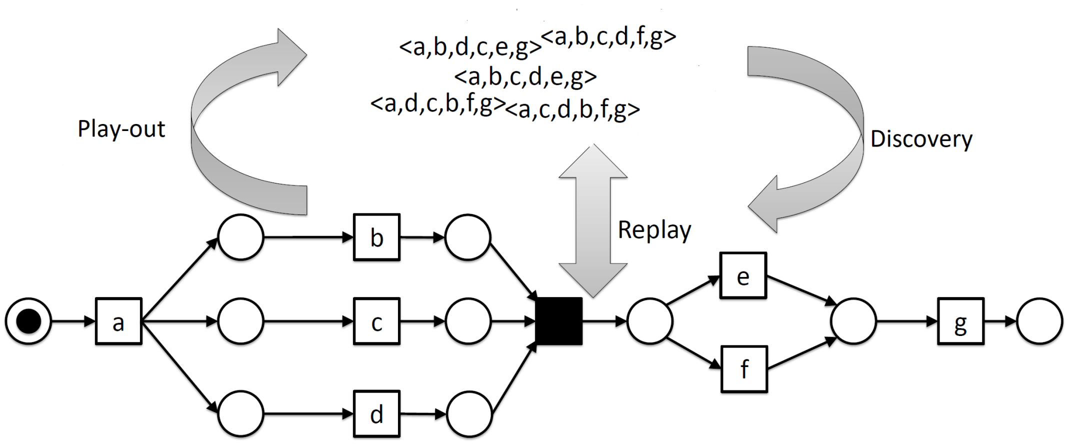


FIGURE 2.3: Graph-based representation of a process model, adopted from (Leemans, 2015)

2.2.1 Process Mining Types

PM may be used for different objectives (see Figure 2.4). The three commonly used PM types are *process discovery*, *conformance checking*, and *enhancement*. Therefore, PM not only aids process model discovery but ensures conformance with business rules and regulations and identifies improvement opportunities (Rojas et al., 2016).

Process discovery. This is the first type of PM. A discovery technique constructs a process model that appropriately imparts the discovered behaviour from an event log or stream containing cases, activities, timestamps, resources, and other data. Several process discovery algorithms have been proposed to produce models for a number of domains, such as healthcare, banking, logistics, government and insurance. (Aalst, 2011b).

Conformance checking. Process models can show what happens in reality and what should happen. Conformance checking is all about checking compliance by comparing the extracted model with reality as recorded in the log or vice versa. The techniques used for the conformance check allow the discovery of dissimilarities between the modelled process and the discovered behaviour (Aalst, 2011b).

Model enhancement. The enhancement allows analysts to add value to or extend the original process models using information about the actual process recorded in event logs (Aalst, 2011b). The information gained from PM can be used to detect obstacles, predict run times, discover roles, relationships, work allocation processes, etc.

2.2.2 Process Mining Perspectives

Four main perspectives can be adopted in PM analyses. These perspectives can either be combined or used alone (Aalst, 2016).

- *The control-flow perspective* is concerned with the control-flow, i.e., discovering sequences of activities in a process. The goal of the control-flow perspective is

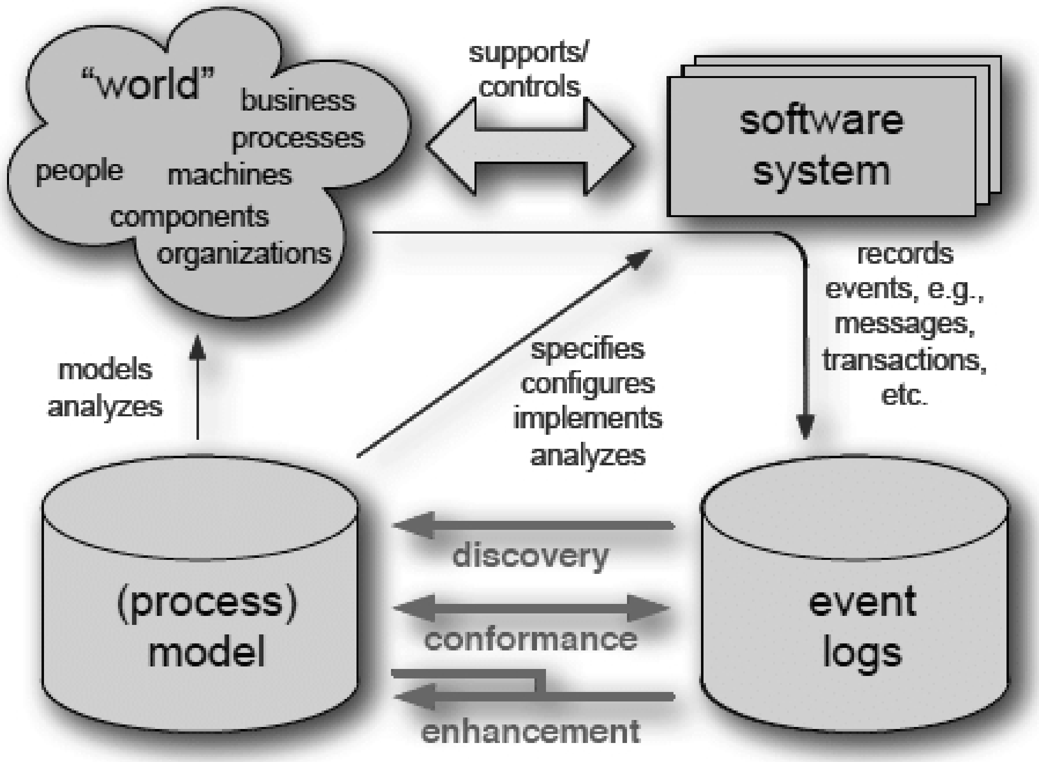


FIGURE 2.4: Three common types of process mining: (a) discovery; (b) conformance; (c) enhancement (Van Der Aalst, 2012)

to find a good characterisation of possible paths expressed in terms of a Petri net or other notation (e.g. BPMN, EPCs or UML ADs).

- *Organisational perspective* is concerned with information about resources mentioned in logs (e.g., people, systems, roles, and departments) and the interplay and collaborations between them. The goal is to structure the organisation by classifying people in roles and organisational units or showing the social network.
- *Case perspective* is concerned with the properties of cases. A case can be distinguished by its path in the process or by the creators working on it. Cases can also be distinguished by the values of the related data elements. For example, suppose that a case represents an order renewal. In that case, it may be interesting to know the supplier or the number of products ordered.
- *Time perspective* is concerned with the timing and frequency of events. If events carry timestamps, it is possible to discover bottlenecks, monitor resource use, and predict the remaining processing time of running cases.

The information gained from PM can be used to analyse and gauge the performance of an organisation, detect obstacles, predict run times, spot anomalies such as control flow deviations, discover roles and relationships, and improve the usage of resources (Ferreira, 2017). PM is applicable to operational processes. Examples of such applications include analysing treatment processes in hospitals, improving customer service processes in companies, understanding customer behaviour in certain processes, analysing failures, and improving interfaces.

2.2.3 Process Mining Applications

PM has been highlighted as a potential way of managing organisational challenges. It has been applied in several sectors such as healthcare, industrial, financial, energy and materials, consumer goods, consumer services, utilities, and technology (Zerbino, Stefanini, and Aloini, 2021). Different perspectives of PM have been explored and, in some cases, combined. Here are some real applications of PM.

Van Der Aalst et al. (Van Der Aalst et al., 2007) used an event log from the Work Flow Management System (WFMS) of a department in charge of construction and maintenance of water and road infrastructure in the Netherlands to identify the process (i.e., how), organisational (i.e., who) and case (i.e., what) perspectives. They used the ProM PM framework and a heuristic algorithm to discover the main flow in invoice handling, which led to the identification of exceptional loops that greatly affected process performance. The outcome of their analysis supported the management team in planning and aiming for certain organisational benchmarks. The authors were also able to spot abnormal behaviours. They got a better picture of the whole process due to combining different mining perspectives.

Lau et al. (Lau et al., 2009) proposed and developed an iterative PM algorithm called "i-PM" to support knowledge discovery in a supply chain network and identify improvement actions using fuzzy rules to maximise customer satisfaction. The approach was evaluated based on historical process data collected from a few departments in a manufacturing company and customer satisfaction reports collected to set up some indices stored in a central warehouse. The algorithm effectively identified the factors that significantly affect customer satisfaction in a supply chain. However, the authors indicated that relying on experts to determine a fuzzy set is very slow and subjective. They suggested adding ways to automatically determine fuzzy sets, such as self-learning AI techniques.

A survey by Rojas et al. (Rojas et al., 2016) shows that PM has gained increased interest and acceptance in healthcare due to its ability to significantly use stored data. Some major applications of PM in the health care sector include discovering process models from event logs, checking for conformity, and analysing social networks. Alvarez et al. (Alvarez et al., 2018) combined the time and organisational perspectives to discover role interaction and collaboration models in the emergency department of a university hospital. Stefanini et al. (Stefanini et al., 2018) used the Fuzzy Miner algorithm on an event log retrieved from hospital information systems to estimate the service time in an emergency department. Ferreira and Alves (Ferreira and Alves, 2011) discovered the social networks at different levels of abstraction within the emergency department of a mid-size hospital.

Roldan et al. (Roldán et al., 2018) used the process, time, and organisational perspectives to detect bottlenecks and inefficiencies in multi-robot missions, such as using an unmanned aerial vehicle fleet in an emergency. Syamsiyah et al. (Syamsiyah et al., 2017) analysed the form handling process within Xerox Services to compare different process variants and to deduce actionable insights regarding the most common process behaviours. Paszkiewicz (Paszkiewicz, 2013) applied conformance checking to check the compliance of a warehouse management system of a manufacturing company with the picking rules. Repta et al. (Repta et al., 2018) analysed the event streams from radio-frequency identification (RFID) readers, global positioning system (GPS) devices, transport vehicles and contact sensors placed on the exit and entry ramp of a warehouse to rebuild the process model depicting the dynamics of the monitored environment.

Below, are some key concepts such as event log, event stream and trace, which are later used as the basis for defining notions related to each method presented in the next chapters.

2.2.4 Event Log

Event logs are the foundation of PM. An event refers to a case/process instance, a task, or a point in time. An event log is thus a collection of cases. The case or process instance is a specific occurrence or execution of a business process, and the activity is an operation, part of a case, executed (Aalst, 2016). An event log stores information about cases and activities, event performers (i.e., the person or device executing the activity), event timestamps (i.e., the moment when the event is triggered), or data elements recorded with the event. Event data are available everywhere, e.g. transaction logs, ERP systems, message logs, hospital database systems, websites, social networks, etc. There are three possible types of event logs (Dakic et al., 2018):

1. Real-life logs containing behaviour recorded from a real-life process.
2. Synthetic logs containing behaviour produced automatically from a real-life process model.
3. Artificial logs containing behaviour automatically extracted from a non-real-life process model or manually creating events.

A typical event log contains a case identifier (ID), which identifies the process instance; a task name, which specifies the activity that has been performed; a user name, which indicates the participant who performed the task; and a timestamp, which indicates the date and time when the task was completed (Ferreira, 2017). Table 2.2 shows a brief example of a simple event log. Each row of the log corresponds to an event, while each column describes the attribute of the event. Events belonging to the same process instance are grouped using a case ID. Formally, given the set of all possible activity names A, the set of all possible case identifiers C and the set of timestamps T, it is possible to define:

Definition 1: Event. An event e is a triplet $e = (c, a, t) \in C \times A \times T$ that describes the occurrence of activity a for the case c at time t. The set of all possible events is called event universe $\mathcal{E} = C \times A \times T$.

Definition 2: Trace. Let $A+$ be a set of all non-empty finite sequences of activities from A. $\sigma \in \mathcal{A}+$ is called a trace when σ represents a firing activity sequence of a process model.

TABLE 2.2: Example of an event log

Case ID	Time-stamp	Activity	Resource	...
01	30-12-2010 11:02	register request	Pete	-
01	31-12-2010 11:32	check ticket	Sue	-
01	05-01-2011 12:00	make decision	Sarah	-
01	07-01-2011 12:50	reject request	Pete	-
02	30-12-2010 11:15	register request	Mike	-
02	30-12-2010 11:30	check ticket	Mike	-
02	05-01-2011 11:55	make decision	Sean	-
02	08-01-2011 12:55	pay compensation	Ellen	-
...	...	...	...	...

Definition 3: Event stream. Given an event set $\mathcal{E}$, $\mathcal{S} \in (\mathcal{C} \times \mathcal{A} \times \mathbb{N})^*$ is an event stream, i.e., a sequence of events that are observed one after the other.

An event stream is an indefinite sequence of events, where each event represents an occurrence of interest at a time. Figure 2.5 shows a simple graphical representation of an event stream. Note that two subsequent events of an event stream may belong to different process instances.

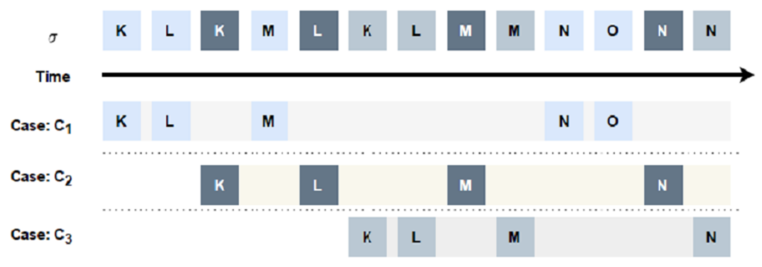


FIGURE 2.5: Graphical representation of a small portion of an event stream. Each box represents an event: the background colours represent the case id, and the letters inside the boxes indicate the activity names. The first row depicts the entire stream portion; the following rows are the single cases in the stream.

2.2.5 Process Discovery

Process discovery is considered the most crucial PM type because all other PM operations depend on it. Process discovery automatically generates a process model that correctly describes a business process based on its event data (Syring, Tax, and Aalst, 2019). Process discovery algorithms take an event log as input to output a process model that should satisfy some properties, which are referred to as the four quality measures or dimensions of PM (see Figure 2.6):

- *Recall:* the discovered model should display all the behaviour observed in the event log (i.e., avoiding *"non-fitting"* behaviour).
- *Precision:* the discovered model should not display behaviour not observed in the event log (i.e. avoiding *"under-fitting"*).
- *Generalisation:* the discovered model should generalise the example behaviour observed in the event log (i.e. avoiding *"over-fitting"*).
- *Simplicity:* the discovered model should not be overly complex. *Occam's Razor* refers to the simplicity dimension: "one should not increase, beyond what is necessary, the number of entities required to explain anything". In the PM field, this is often implemented by quantifying the complexity of the model using the number of nodes, number of arcs, comprehensiveness, etc.

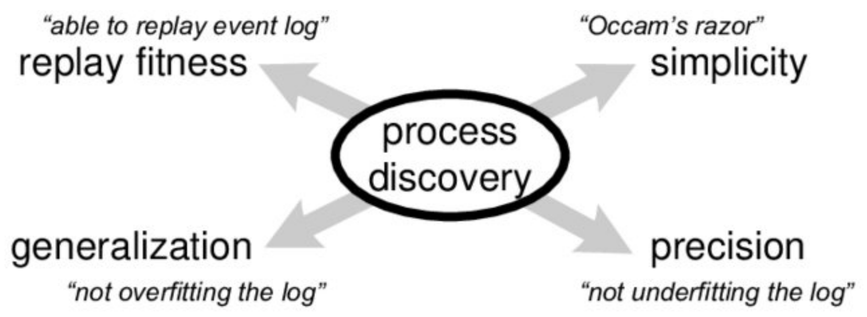


FIGURE 2.6: Different quality dimensions for process model discovery adopted from (Aalst, 2011a)

Most process models show only the control-flow of a process; however, additional perspectives can be added when extending the process model.

The control-flow perspective is an analysis that focuses on discovering the sequence of activities in a business process. The output of this type of analysis is a model that describes the overall behaviour of the process. There exist several algorithms to discover process models such as alpha-algorithm (Aalst, Weijters, and Maruster, 2004), heuristics miner (Weijters and Ribeiro, 2011), fodina (Broucke and De Weerdt, 2017), evolutionary tree miner (ETM) (Conforti et al., 2016), fuzzy miner (Günther and Van Der Aalst, 2007) and split miner (Augusto et al., 2018). These algorithms use different approaches to achieve the same goal, a model that shows the transitions between tasks.

The organisational perspective is an analysis that focuses on discovering interplay and collaborations among the participants in a business process. For this purpose, the data to be analysed are the case id and user columns in the event log.

The time or performance perspective is an analysis that focuses mainly on time. E.g. the average time it takes to perform an activity, the maximum time it takes for the process to reach a certain point, the average end-to-end duration of each process instance, etc. It is common to start with a control-flow analysis of the event log and proceed to performance analysis, the results of which can be displayed on the control-flow graph.

Most algorithms for process discovery use frequency-based heuristics, genetic-based heuristics, probabilistic-based approaches, and theory of regions or hybrid methods to produce a process model (Augusto et al., 2018). The applicability and effectiveness of these algorithms depend on the event log features and structure of the processes. Event logs are often incomplete, making process discovery and quality assessment of the process model to the log challenging. On real-life event logs, the state-of-the-art process discovery methods are subject to two reoccurring drawbacks: they produce broad and spaghetti-like models (as shown in Figure 2.7) or models with poor fitness and precision (i.e. unable to discover process models that will express observed behaviour in the best possible form). They can not trade-off the four process discovery quality metrics (i.e. fitness, precision, generalisation, and complexity). Figure 2.3 lists some popular state-of-the-art process discovery methods with their pros and cons. This is because many of the algorithms are not effective against the following challenges (Aalst, 2010):

- Dealing with complex control-flow constructs. Choosing between concurrent and choice execution cannot be handled by many algorithms.

- Handling duplicates. For instance, if multiple activities co-occur, the records of these activities may be inconsistent.
- Generating consistent models. For example, $\alpha - algorithm$ may yield models with deadlocks when the log shows certain types of behaviour. The modelling language has to be severely limited to achieve a sound model using a modelling language such as Petri nets (i.e., free from deadlocks and other anomalies).
- Balancing "*overfitting*" and *underfitting*. Overfitting is the problem where a particular model is generated when it is obvious that the log only holds example behaviour, i.e. the model explains a particular sample log, while a next sample log of the same process may produce a completely different process model. Underfitting is the problem, where the model over-generalises the example behaviour in the log, i.e. the model allows for very different behaviours from what was seen in the log. As such, these models cannot reliably predict future observations.

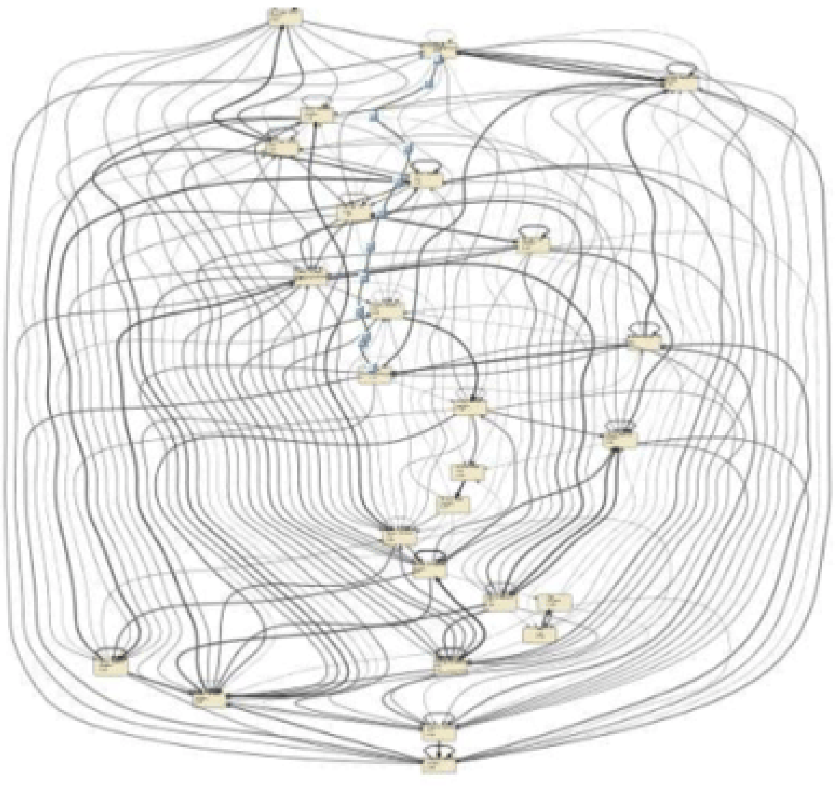


FIGURE 2.7: An example spaghetti-like model adopted from (Suriadi et al., 2013)

TABLE 2.3: Some popular state-of-the-art process discovery methods with their pros and cons

Method	Model Language	Pros	Cons
α − Miner (Aalst, Weijters, and Maruster, 2004)	Petri nets	Simple models Good fitness	Not sound Limited Not robust
Heuristic Miner (HM) (De Weerdt et al., 2012)	BPMN	Good F-score	Complex models Semantic errors Not sound
Inductive Miner (IM) (Leemans, Fahland, and Van Der Aalst, 2013)	Process trees	High fitness Simple models No semantic errors	Low precision
Evolutionary Tree Miner (ETM) (Buijs, Dongen, and Der Aalst, 2012)	Process trees	High precision Simple models	Long execution time Favours certain structures
Split Miner (SM) (Augusto et al., 2019)	BPMN	High fitness Low complexity No semantic errors	Moderate precision Not structured

2.3 Deep Learning

DL is a type of machine learning that involves constructing and using networks composed of multiple interconnected layers of neurons to perform non-linear transformations of data (Hao, Zhang, and Ma, 2016). The essence is to allow the network to learn the patterns or behaviours observed in the data. In theory, the more layers of neurons in a network, the more likely it is to detect higher-level patterns in the data due to the composition of complex functions (LeCun, Bengio, and Hinton, 2015). DL achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts. Each concept is defined with more precise concepts and abstract representations determined with respect to less abstract ones (Goodfellow, Bengio, and Courville, 2016), (Wani et al., 2020). DL has achieved ground-breaking results in various tasks, including image recognition, speech recognition, anomaly detection, disease diagnosis, and natural language processing. DL models have also been applied in several sub-fields of PM, particularly in Predictive Process Mining (PPM). PPM is a class of PM techniques concerned with predicting some properties about the future state of a case. For example, predicting the next event in an ongoing case or predicting the remaining time until a case is completed.

2.3.1 Recurrent Neural Network

Being one of the popular DL architectures, a recurrent neural network (RNN) is acknowledged for its ability to learn and generalise over sequences of inputs rather than individual patterns (Goodfellow, Bengio, and Courville, 2016). It contains recurrent or repeated connections with hidden states distributed over time, allowing the network to keep past information effectively. In RNN, the output at the current time step is the input in the next time step (Figure 2.8). As a result, at each point in the sequence, an RNN model considers the current input and what it remembers about the previous points, thus learning the temporal dependence and contextual information in the input data. This memory quality allows the network to learn long-term dependencies in a sequence, preventing prior information from being lost (Lewis, 2016). This quality suggests the RNN model's ability to consider the context when making a prediction, e.g. when predicting the next word in a sentence, classifying sentiments, and estimating temperature measurements. At the same time, the length of contextual information that a typical RNN model can capture is limited, which can negatively affect the model's performance. This drawback is due to the problem of vanishing or exploding gradients during RNN model training using back-propagation (Hochreiter and Schmidhuber, 1997).

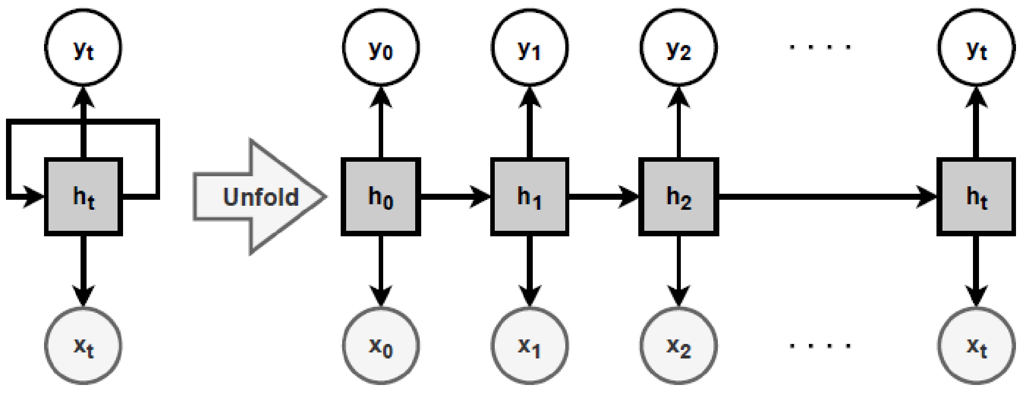


FIGURE 2.8: The basic recurrent neural network (RNN) architecture

2.3.2 Long Short-term Memory

A variation of the RNN architecture called LSTM has been proposed to overcome the problem of vanishing or exploding gradients. The control flow of LSTM is the same as that of RNN; the main distinction is in operations within LSTM cells. A cell, also called a memory block, is the memory of the network that retains information over random time intervals (Fig. 2.9). Information is updated to or removed from a cell through three gate units. In particular, *Forget gate*(f) decides which information to keep or discard, the *Input gate* (i) decides what relevant information to update the cell state, and *Output gate*(o) determines the output. In each LSTM cell, the activation function *sigmoid* (σ) is used to decide which information to keep or ignore, while the hyperbolic tangent function *tanh* is used to help regulate the network.

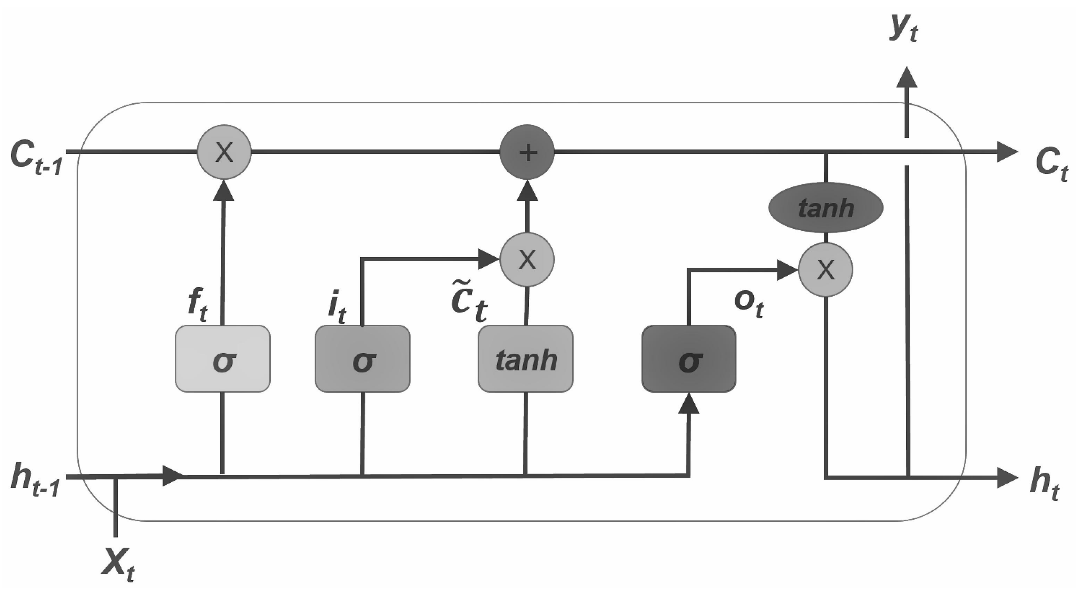


FIGURE 2.9: The long short-term memory (LSTM) cell

The LSTM cell operations can be described using the following formulae:

$$
\begin{aligned}
f_t &= \sigma(W_f.[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i.[h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_c.[h_{t-1}, x_t] + b_c) \\
C_t &= f_t * C_t - 1 + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o.[h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

Using information from the previous hidden state h_{t-1} and the current input x_t, f_t calculates the output of the forget gate, i_t calculates the output of the input gate, $\tilde{C}_t$ calculates the eligible candidate, C_t calculates the new cell state, o_t calculates the output of the output gate, and h_t calculates the new hidden state. W and b represent the weight and bias parameters for each of the *Forget*(f), *Input* (i) *and Output gates* (o) learnt during training, respectively.

2.3.3 Bidirectional Long Short-term Memory

Bidirectional LSTM (BLSTM) (Schuster and Paliwal, 1997) splits the state neurons of regular LSTM into two, the forward state (responsible for positive time direction) and backward state (responsible for negative time direction), both of which are connected to a common output layer. This architecture makes it possible to train a model using all past and future information of a specific time frame. Figure 2.10 shows a basic BLSTM structure, with the LSTM net at the bottom indicating the forward state and the LSTM net at the top indicating the backward state. Without the backward state, it becomes a regular unidirectional LSTM.

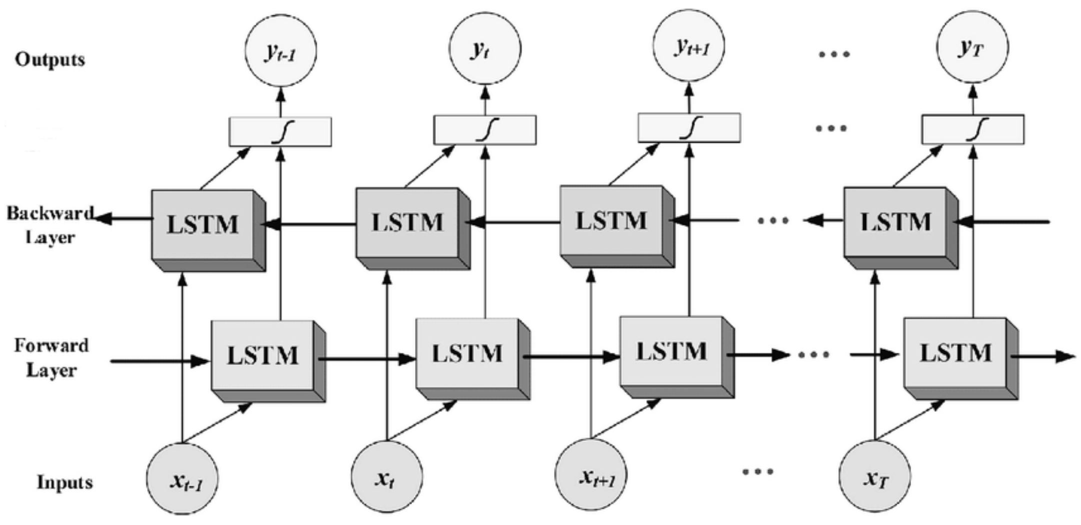


FIGURE 2.10: Basic unfolded bidirectional LSTM (BLSTM) structure

Depending on the problem domain, the BLSTM structure can give better results than other network structures. For example, a BLSTM model was reported to achieve remarkable performance in speech processing tasks, where content is crucial (Graves and Schmidhuber, 2005).

2.3.4 Word Embedding

The idea of word embedding was first introduced in (Xu and Rudnicky, 2000) and (Bengio, 2008) with the neural network language model. The motivation behind it is that words are likely to share similar meanings in the same context (Harris, 1954). According to the Bayes rule, the probability that a word sequence W occurs is:

$$P(W) = \prod_{t=1}^{N} P(w_t \mid w_1, ..., w_{t-1}) = \prod_{t=1}^{N} P(w_t \mid h_t),$$

where $P(W)$ is the joint distribution of the sequence W, and h_t represents the context words around the word w_t. The goal is to evaluate the probability that the word w_t appears given its context information. Because there are many possible combinations of a word and its context, it is impossible to specify all $P(w_t \mid h_t)$. As such, a function Φ is used to map the context into equivalent classes, where all words are statistically independent (Srinivasan, 2017):

$$P(w_t \mid h_t) = P(w_t \mid \Phi(h_t)).$$

Other text representation techniques, such as one-hot, generate high-dimensional vectors, whose length depends on the volume of the vocabulary. In addition, semantics and context are not captured, i.e. the meaning is not modelled effectively. Embeddings are a more robust representation of corpus and can be used to leverage massive DL models. An embedding is a dense vector of floating-point values (Li and Yang, 2018). The values for the embedding are not manually specified. They are learnt by the model during training, just as a model learns the weights for a dense layer. When trained on massive datasets, embeddings capture the context of a word by considering a few other words around it and their order in a sentence.

Embeddings greatly benefit natural language processing (NLP) problems for two reasons. First, they help reduce dimensionality as the number of features can be controlled. Second, the context of a word can be understood since similar words have similar embeddings. Pre-trained word embeddings such as Word2Vec, GloVe, and FastText (Brownlee, 2017) can be used when dealing with NLP problems. Alternatively, the embeddings can be trained using the embedding layer of the Keras library (Gulli and Pal, 2017). The embedding layer is an essential part of neural networks, which learn the relationships between words. For example, in a large corpus that contains all possible English words, the vectors for words like "king", "queen", "man", and "woman" will show some similarity in the multidimensional embedding space. When building a word embedding space, the goal is to capture a relationship in that space, such as meaning, morphology, context, or another relationship. By encoding word embeddings in a densely populated space, words can be represented as vectors with tens or hundreds of dimensions instead of millions (which is the case with one-hot encoded vectors) (Li and Yang, 2018).

Today, word embedding approaches, represented by DL, have gained attraction and have been applied with satisfactory results in many tasks, such as text classification, semantic similarity estimation between words of different languages, question answering, parsing and sentiment analysis, recommender systems, knowledge mining (Gutiérrez and Keith, 2018), etc.

2.4 Deep Learning in Process Mining

Due to its popularity, several recent studies have employed DL techniques to monitor and predict business processes. These DL approaches are based on different types of neural networks. Most of them use LSTM Neural Networks (Tax et al., 2017; Evermann, Rehse, and Fettke, 2017; Camargo, Dumas, and González-Rojas, 2019; Lin, Wen, and Wang, 2019; Tello-Leal et al., 2018; Di Francescomarino et al., 2017), Gated Recurrent Units (GRUs) (Hinkka, Lehto, and Heljanko, 2018), or Convolutional Neural Networks (CNN) (Pasquadibisceglie et al., 2019; Mauro, Appice, and Basile, 2019). These approaches use different encoding techniques for sequences and consider different input data to make predictions.

Tax et al. (Tax et al., 2017) used the LSTM approach to build models that could achieve multi-task learning, namely, predicting the next event of a running case and its timestamp. The authors used one-hot encoding to transform input events into feature vectors. The authors presented two methods: one for activity prediction and the other for timestamp prediction. The Helpdesk and BPIC12 Subprocess W datasets are used for evaluation. The models could predict the continuation of a case up to completion and the remaining cycle time of the running cases. However, the models did not perform well when making long-term predictions and when used in event logs with many repeated events. They predict long sequences of the same event when dealing with traces that have similar activities multiple times.

Motivated by the application of RNN to predict the next word in a sentence, Evermann et al. (Evermann, Rehse, and Fettke, 2017) proposed LSTM networks to predict the next event of a case from an event log. Unlike (Tax et al., 2017), their approach uses the embedded dimension of the LSTMs to reduce the input size. The authors considered the event log as text, traces as sentences, and events in a trace as words in a sentence. The authors used an RNN architecture with a single hidden layer of LSTM cells. They evaluated the approach using the datasets of BPIC 2012 and BPIC 2013. The model could predict the most probable remaining sequence of events in an ongoing case. This study acknowledges that the inability of the model to handle numerical variables made generating sequences of time-stamped events impossible. Another limitation is to interpret the knowledge encoded by the model. The authors further present two mechanisms to interpret the model's results; (i) Hallucinations to predict the suffix and (ii) t-SNE plots to visualise the embedding matrix and the hidden states. The interpretations supplied information about the layers and cells activated for an input sequence. However, they were still limited in their ability to explain a prediction.

Francescomarino et al. (Di Francescomarino et al., 2017) also employed the LSTM architecture proposed by Tax et al. (Tax et al., 2017) and improved its performance by accounting for a-priori knowledge using *A-PRIORI algorithm* on the traces of the test set and dealing with cycles using *NOCYCLE algorithm* to overcome the problems encountered when dealing with traces containing a high number of cycles.

Tello-Leal et al. (Tello-Leal et al., 2018) presented an LSTM network applied to event logs from the Internet of Things (IoT) domain within the context of industry 4.0. Although the network could successfully predict the next activity in a business process, its performance in predicting continuous traces was not always good.

Lin et al. (Lin, Wen, and Wang, 2019) presented an RNN-based modulated model for multi-task prediction of event sequences and event attributes. The model (called MM-Pred) includes an LSTM encoder-decoder and a modulator capable of learning inter-dependencies among event attributes. The modulator combines the hidden representations of the LSTM-based encoder to infer the weight vector. The weighted sum of the encoded vectors is used as input to the decoder layer comprising a two-layer LSTM network to predict the next activity. The authors compared the MM-Pred model with two other models. S-Pred for predicting the next events based on event sequences alone (using the same LSTM encoder-decoder architecture, but without the modulator) and M-Pred for predicting next events and their attributes together (employing the modulator). This work suffers the same limitation as (Evermann, Rehse, and Fettke, 2017): It does not support predicting attributes with numerical domains, including timestamps and durations.

To overcome the limitations, Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) proposed an LSTM architecture with embedded dimensions to predict traces of events, time-stamps and the role associated with each event. The phases in their approach included scaling and extracting n-grams of fixed sizes for each event log trace to create input and target sequences for training the predictive model and a post-processing phase, where predicted next events were randomly selected from the likely ones to generate a greater number of different traces. For the next-event prediction task, their approach performs similarly to Evermann et al. (Evermann, Rehse, and Fettke, 2017) and Tax et al. (Tax et al., 2017), but underperforms Lin et al. (Lin, Wen, and Wang, 2019). However, for suffix prediction, their approach outperforms (Evermann, Rehse, and Fettke, 2017), (Tax et al., 2017) and (Lin, Wen, and Wang, 2019). They attribute the good performance to the measures they adopted for the dimensionality control of the categorical attributes. Table 2.4 summarises some LSTM-based models for monitoring and predicting business processes.

While many of the methods reviewed above have been reported to have performed well, they have some limitations. The approaches did not perform satisfactorily when dealing with multiple instances of the same activity. In most approaches, regression problems, such as the remaining time to complete the case, are not addressed. Also, concept drift and hyper-parameter optimisation are not addressed in existing approaches. It is also worth noting that the proposed approaches can be event log-specific when tackling prediction problems. For example, Rama et al. observed when comparing different approaches for next activity prediction that using the whole set of available attributes is beneficial, as long as the amount of data is sufficient and the log is not too complex. However, when the log is too complex, a more straightforward approach, such as (Tax et al., 2017), may capture more effectively the relations among the activities in the trace than more complex approaches, such as (Hinkka, Lehto, and Heljanko, 2018) and (Camargo, Dumas, and González-Rojas, 2019). Finally, Neu et al. (Neu, Lahann, and Fettke, 2021) have reported the following factors that contributed to the lack of quantitative comparability of state-of-the-art DL methods for process prediction:

- absence of common performance measurement;
- imbalanced datasets (which can affect accuracy scores);
- the choice of inputs (which plays a significant role in the maximal attainable performance scores).

TABLE 2.4: Summary of RNN- and LSTM-based models for monitoring and predicting business processes.

Study	**Dataset**	**Model Architecture**	**Prediction Type**	**Quality Metric**
Tax et al. (Tax et al., 2017)	Helpdesk BPIC12w	2 LSTM Layers A Dense Output Layer	Next activity, Remaining Time	Accuracy, Mean Absolute Error (MAE)
Camargo et al. (Camargo, Dumas, and González-Rojas, 2019)	BPIC12 BPIC13 Helpdesk BPIC15	Input Layer 2 LSTM Layers A Dense Output Layer	Next event, Suffixes	Damerau–Levenshtein (D-L) distance, Accuracy, Mean Absolute Error (MAE)
Evermann et al. (Evermann, Rehse, and Fettke, 2017)	BPIC12 BPIC13	Embedding Layer 1 LSTM Layer	Next event	Precision
Lin et al. (Lin, Wen, and Wang, 2019)	BPIC12 BPIC14 BPIC17 Helpdesk	Encoder-Decoder LSTM Modulator A Dense Output Layer	Next activity, Attributes	Damerau–Levenshtein (D-L) distance Accuracy

2.5 Process Discovery versus Process Prediction Methods

Process discovery methods are used to improve the comprehensibility of control-flow relations between tasks, as observed in event logs. The discovered process models can help to understand how a process should be performed or is performed in reality. More importantly, they provide a foundation for further analysis. Transition diagrams such as Petri-nets, BPMN diagrams, causal nets, state machines, and directed acyclic graphs are often used to represent the outcomes of process discovery methods (Augusto et al., 2018). Using process modelling notation for the visualisation of business process models ensures that domain experts, who may not have the technical knowledge to comprehend how different PM methods work, can understand the generated graphs and decide whether the structure of the model reflects or contradicts the domain knowledge of experts. However, these strategies may lead to the discovery of complex and incomprehensible process models that conceal the correct behaviour of the underlying process.

More recently, DL-based sequence modelling techniques have gained popularity across the PM community in the context of predicting next activities in business processes. Comparative studies such as (Rama-Maneiro, Vidal, and Lama, 2021; Kratsch et al., 2021; Teinemaa et al., 2019; Di Francescomarino et al., 2018; Tax, Teinemaa, and Zelst, 2020) have shown that DL approaches for next activity prediction outperform classical prediction techniques, which use an explicit model representation such as Hidden Markov Models (Lakshmanan et al., 2015), probabilistic finite automatons or state-transition (Breuker et al., 2016; Unuvar, Lakshmanan, and Doganata, 2016). Since then, next activity predictions produced by DL models have been demonstrated to be more accurate than those made through process models discovered using event logs (Tax et al., 2017; Camargo, Dumas, and González-Rojas, 2019; Evermann, Rehse, and Fettke, 2017). At the same time, the reverse interaction, i.e. employing DL to produce accurate process models, is yet to be evaluated using appropriate metrics and comparative studies. As a first step towards this goal, this thesis proposes to combine visually expressive graphs with DL approach to generate the most probable sequences of activities from event logs that can potentially be used in place of process models.

2.6 Concept Drift in Process Mining

Another important use of PM techniques is detecting process drifts from business process executions in event logs. Business processes are subject to unplanned changes and disruptions that can adversely affect the performance of processes. Many factors such as seasonal effects, legislation changes, new rules and regulations, technological advances, and unexpected events (e.g. the coronavirus pandemic) lead to changes (or drifts) in business processes over time. Event logs recorded over a certain period can be expected to differ from event logs recorded over another period. This behavioural change in process execution occurring at some point in time is called concept drift. Concept drift is said to occur when a change is observed in the process while it is being analysed (Bose et al., 2011; Bose et al., 2013).

Process drifts are either planned (e.g. regulatory changes) or unexpected (e.g. change in resource capacity) (Seeliger, Nolle, and Mühlhäuser, 2017). Analysing such changes is of great importance to developing operational processes and obtaining accurate insight into process executions at any given time. The following main challenges arise when dealing with concept drifts in PM:

- *Change point detection:* This involves first and foremost identifying if drift has occurred in a process. If a change is detected, the next thing to do is identify the time periods in which the changes occurred. For example, drift should be detected at the beginning of a season for changes related to seasonal effects.

- *Change localisation:* After detecting a change point, the next step is to identify the region(s) of change in the process model. The change localisation method should identify the exact point in the model where the drift occurs, e.g., between activities 'A' and 'B', without requiring a process model as input.
- *Change characterisation:* Involves defining the perspective of change and determining the type of drift, e.g. sudden, gradual, or incremental. This challenge is rarely reported. Most state-of-the-art approaches focus on drift detection and localisation.
- *Change process discovery:* After identifying, localising, and characterising the drifts, it is essential to use them in a broader context. Change process discovery presents the complete change process based on change detection, localisation, and characterisation. It is done using tools that exploit and relate these discoveries. This will lead to unravelling the evolution of the process change and how it affects the model over time. For example, identifying if a process reoccurs every season. Also, visually showing how the process evolved over a period with annotations showing several perspectives, such as the performance of a process at different time instances.

2.6.1 Approaches to Dealing with Drifts

There are two common approaches to dealing with concept drifts when analysing event logs:

- *Offline analysis* refers to a scenario where changes are discovered using historical data. The entire event log is made available to the analyst in this case. This is appropriate for future analysis, e.g. designing or improving processes for later deployment (Zheng, Wen, and Wang, 2017). For example, offline concept drift analysis can take proactive measures, such as hiring more staff or skipping checks weeks before Christmas.
- *Online analysis* refers to a scenario, where changes need to be discovered near-real-time. In this case, the analyst must deal with the incoming data continuously. This type of analysis can be helpful to organisations interested in learning behavioural changes regarding their customers or changes in demand as they happen. Such real-time prompts can enable organisations to take immediate corrective measures, thus avoiding undesirable consequences (Zheng, Wen, and Wang, 2017).

2.6.2 Perspectives of Drift

There are three critical perspectives in business processes, and one or more of these perspectives may change over time:

- *Control-flow/behavioural perspective:* This change perspective deals with the behavioural and structural changes of a process model. Weber et al. (Weber, Reichert, and Rinderle-Ma, 2008) outline a list of common control-flow change patterns, which are classified into three categories: insertion (I), e.g. inserting or deleting a fragment, resequentialisation (R), e.g. sequentialising two parallel fragments, and optionalisation (O), e.g. placing an existing fragment in a loop in (Maaradji et al., 2016). Table 2.5 lists the common control-flow change patterns and their categories adopted from (Maaradji et al., 2017).

For example, an accident insurance company that used to collect payment after processing and accepting an application can change their process to impose payment before processing. Here, the "reordering" change pattern was applied to the fragments of the payment and application processing process. Sometimes, the control-flow structure of a process may remain unchanged, but its behavioural aspects will change. For example, a mortgage rate of £1,000 classified as "high" last year is classified as "low" this year due to an increase in the mortgage rate to £3,000. In such a situation, the structure of the process remains unchanged, but the routing of cases changes.

- *Data perspective:* This change perspective refers to the changes in the production and consumption of data and the effect on the routing of cases. For example, having a particular document when applying for car insurance may no longer be required.
- *Resource perspective:* This perspective deals with changes in resources, their roles, organisational structure, and their influence on the execution of a process. For example, a person who executes a certain activity could have been changed, roles may change, and people may change roles. In addition, resources are bound to work in a particular manner, and such working patterns may change over time.

TABLE 2.5: Common control-flow change patterns in business processes: I, R, O stand for Insertion, Resequentialisation, and Optionalisation.

Change pattern	**Category**
Insert/delete a fragment between two fragments	I
Insert/delete a fragment in/from parallel branch	I
Insert/delete a fragment in/from conditional branch	I
Duplicate a fragment	I
Substitute a fragment	I
Swap two fragments	I
Move a fragment to between two fragments	I
Move a fragment into/out of conditional branch	I
Move a fragment into/out of parallel branch	I
Make fragments mutually exclusive/sequential	R
Make fragments parallel/sequential	R
Synchronize two fragments	R
Make a fragment loopable/non-loopable	O
Make a fragment skippable/non-skippable	O
Change branching frequency	O

2.6.3 Types of Drifts

Process changes can be classified into momentary and permanent based on the duration for which the change is active. Momentary changes are short-lived and affect only a few cases, whereas permanent changes are persistent and stay for a while (Schonenberg et al., 2008). Permanent changes are studied more, as momentary changes often cannot be discovered due to insufficient data. Permanent changes can be further classified as follows (Fig. 2.11):

- *Sudden drift:* The sudden concept drift occurs when dynamic changes occur during process execution, i.e., a current process is replaced with a new process, and the new process takes over in subsequent process executions. This class of drift can occur due to a change in the law, an emergency, etc. In Figure 2.11a, an existing process P_1 is replaced by a new process P_2. P_1 ceases to exist at the moment of substitution.

- *Gradual drift:* The gradual concept drift occurs when two or more versions of a process coexist, i.e. a new process exists along with an old process over a certain period of time, making it possible to execute both process versions until the old process is gradually discontinued. For example, an organisation might introduce a new delivery process. However, the process is set to apply to future orders only. All previous orders must still follow the previous delivery process. In Figure 2.11b, a current process P_1 is replaced with a new process P_2. Unlike sudden drift, both processes co-exist for some time, with P_1 discontinued gradually.

- *Recurring drift:* The recurring concept drift (Figure 2.11c) occurs when a set of processes is changed back and forth between each other. These types of drifts are either periodic or non-periodic and are often induced by changes in the external environment in which a business process operates. An example is sales that happen in stores during specific periods of the year.

- *Incremental drift:* The incremental concept drift (Figure 2.11d) occurs when a change is introduced incrementally into the running process until the process reaches the desired version. This drift class is more common in organisations that follow the agile business process management methodology (Bose et al., 2013).

2.7 Concept Drift Detection

Most state-of-the-art methods for concept drift detection in business processes use the windowing technique to select traces from an event log to consider for drift analysis, together with statistical hypothesis testing as a solution (Maaradji et al., 2017; Ostovar et al., 2016; Bose et al., 2013; Seeliger, Nolle, and Mühlhäuser, 2017; Carmona and Gavalda, 2012; Martjushev, Bose, and Aalst, 2015). Some studies used clustering-based techniques to find groups of traces that share similar characteristics that can be generalised and used to detect drifts (Yeshchenko et al., 2019; Zheng, Wen, and Wang, 2017; Hompes et al., 2015a; Bolt, Aalst, and De Leoni, 2017; Sousa et al., 2021). Other studies used graph-based analysis techniques (Lu et al., 2016; Seeliger, Nolle, and Mühlhäuser, 2017; Nguyen et al., 2018) or model-to-log alignment (Buijs and Reijers, 2014; Aalst, 2012).

Bose et al. (Bose et al., 2013) presented a method to detect changes in business processes and identify change regions. First, the authors extracted feature sets from event logs and compared their values over different windows. Then, they applied statistical hypothesis testing to investigate a significant difference between two successive windows. Martjushev et al. (Martjushev, Bose, and Aalst, 2015) extended the method proposed by Bose et al. (Bose et al., 2013) by using an adaptive window strategy and presented an approach to automatically detect change points by comparing significant values of two windows produced using hypothesis analysis against a predefined threshold.

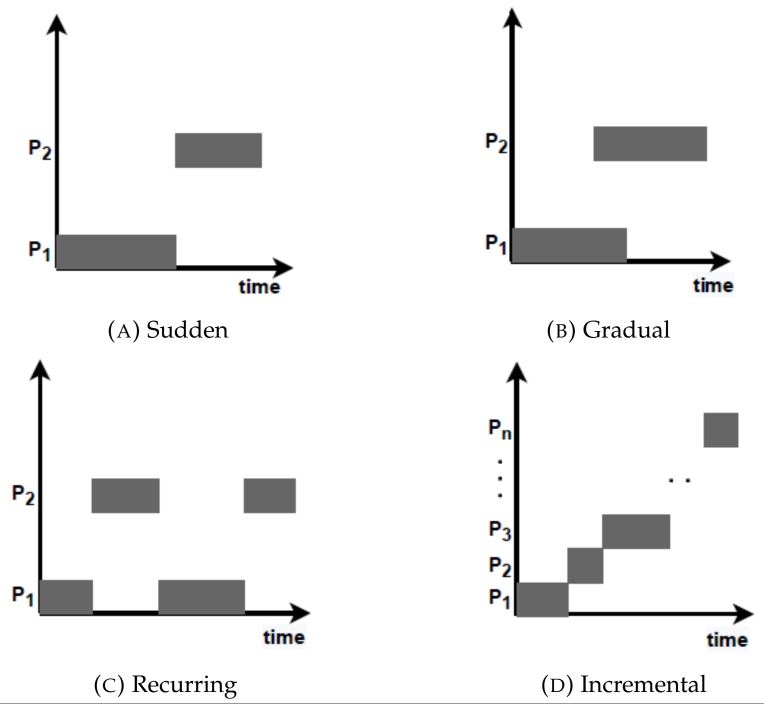


FIGURE 2.11: Different types of drifts. X-axis: time. Y-axis: process variants. Grey rectangles: process instances

Manoj Kumar et al. (Manoj Kumar, Thomas, and Annappa, 2015) proposed a similar method for capturing sudden concept drifts in business processes. In particular, the authors assumed that the representative appearance of feature values changes before and after the occurrence of drift. They applied a windowing strategy to select the instances for detecting and localising drifts, taking note of the sequential order of process instances in the log. The authors used statistical hypothesis testing to examine differences between successive feature values obtained using the event class correlation determined by scanning the entire log, beginning with a matrix set with zero values and then updating for every next relation found while traversing the log. The size of the look-forward window was used to calculate each event that followed the reference event.

Maaradj et al. (Maaradji et al., 2017) proposed a method to detect sudden and gradual drifts from execution traces. The authors tested the statistical hypothesis over the run distribution in two consecutive time windows. They presumed that if a drift occurred at a given time, the distribution of runs before and after would statistically differ and the statistical hypothesis testing could expose the difference. The authors used an adaptive window technique to adjust the sliding window's size, striking a good trade-off between accuracy and drift detection delay.

Similar to Maaradj et al. (Maaradji et al., 2017), Ostovar et al. (Ostovar et al., 2016) also used an adaptive window technique but in an online setting. Their approach involves dividing new observed events into reference and detection windows. The set of events within each window is used to build the corresponding sublog. A contingency matrix is constructed using relations and frequencies extracted from the sublog. The G-test of independence (Harremoës and Tusnády, 2012) is applied to the matrix to obtain the significance probability (p-value). A p-value below a predefined threshold suggests a drift.

Carmona et al. (Carmona and Gavalda, 2012) presented an online technique for dealing with concept drifts. First, the authors applied the theory of abstract interpretation to learn an internal representation of an event log based on a polyhedron. Then, they estimated the soundness of the representation using an adaptive window technique to detect concept drifts automatically.

Li et al. (Li et al., 2017) proposed an extensible feature that uses the sliding-window technique and the heuristic miner to detect and locate concept drifts in incomplete event logs. The authors further improved the Genetic Process Mining (GPM) method (Medeiros, Weijters, and Aalst, 2007) using Weight Heuristic Miner (WHM) (Kurniati, Kusuma, and Wisudiawan, 2016) and Differential Evolution (DE) (Storn and Price, 1997) to discover the new process model of evolving processes.

Zheng et al.(Zheng, Wen, and Wang, 2017) presented a three-stage method for detecting process drifts from event logs. First, the authors represented each trace of an event log as multiple relations such as direct succession and weak order. Then, for each relation, they inspected and partitioned the variation trend. Finally, they clustered all change points revealed by each relation to get the final result.

Yeshchenko et al. (Yeshchenko et al., 2019) proposed a method that involves the following three steps: (i) splitting an event log into sub-logs based on predefined window size and mining declarative (DECLARE) process constraints, (ii) extracting the time-series of the characteristics of the discovered constraints, (iii) clustering the series and detecting change point over them. The final step involves visualising drifts using drift maps and charts.

Seeliger et al. (Seeliger, Nolle, and Mühlhäuser, 2017) used an adaptive window technique to split an event log into a reference window and a detection window. The authors discovered process models for both windows using the Heuristic Miner algorithm (Weijters, Aalst, and Ana Karla, 2006). Then, they applied a statistical significance test on different graph metrics to determine the deviation of both observed process models. Using graph metrics, the authors described changes in the process model to identify process drifts in the event log. They also performed the statistical G-test to determine whether the detection window's process model differs significantly from the reference window's.

Hassani et al. (Hassani, 2019) used an adaptive window method and a modular set of reasonable distance measures to detect drifts in event streams. The authors proposed the StrProMCDD algorithm that collects a batch of events in a pruning period, computes the frequency list for these events, and includes the new frequency list in a temporally ordered list used by ADWIN (Bifet and Gavalda, 2007). The window increases in size for steady process behaviour and shrinks for diverting processes, thus indicating a drift.

De Sousa et al. (Sousa et al., 2021) proposed an online trace clustering approach to detect and localise drifts in online trace streams. According to this approach, traces are mapped onto a vector representation that is used as input to a trace clustering algorithm. The resulting cluster information is used for drift detection and localisation. The authors assumed that each feature representing a significant group of traces should remain stable according to the traces' process behaviour. Hence, they verify whether the current value has changed significantly to detect drift. The method iterates over any of the clustering evolution features over time. The feature value is compared to an estimated tolerance limit in each iteration. A value outside the tolerance limit represents a significant change in behaviour from earlier measurements. After each detection, the drift localisation procedure is started, and a list of the clustering indexes where drifts were detected is returned.

Liu Na et al.(Liu, Huang, and Cui, 2018) presented an online framework to detect concept drift in an event stream based on the relationship between each pair of activities in the process. First, the framework involves initialising the current model, which is used as a benchmark to compare every event trace of the upcoming event streams. Then, adjacency and footprint matrices are extracted for each new trace. The matrix of the new trace is then compared with the matrix of the current model to identify differences. A metric named process model precision is calculated. The difference between the matrices indicates a drift. The method returns the activities and the difference to localise and characterise the drift, indicating whether the drift is sudden or recurring.

Although many of the methods reviewed above performed well, they have limitations. For example, the windowing technique used in many concept drift detection algorithms is highly dependent on the right choice of window size; a wrong window size can result in many false negatives and false positives. Some methods, such as the one proposed by (Carmona and Gavalda, 2012) cannot locate the exact moment of a drift. The method reported in (Bose et al., 2013) is not automated; it requires human involvement in feature selection and recognition of changes, which makes it impractical. (Sousa et al., 2021) approach is capable of dealing with drifts only presentable through trace clustering. Thus, the approach is subject to the limitations of the trace vector representation and clustering algorithms. Finally, most of the existing methods for detecting concept drifts in a business process are designed to work offline, i.e. they require the entire event logs featuring cases from both before and after a drift. Some online methods detect drifts with a long delay and some do not perform well on processes whose logs display many distinct executions. Thus, detecting drifts in online scenarios (i.e. as they happen) remains challenging. Table 2.6 summarises the existing business process drift detection solutions and their limitations.

2.8 Concept Drift Localisation

Concept drift localisation is concerned with identifying the parts of control flow changes that explain the concept drift. Unlike drift detection, whose goal is clear and common in related work, drift localisation approaches do not have universal output requirements.

State-of-the-art drift localisation methods generally compare the process behaviour before and after the drift to report differences. In (Nguyen et al., 2018), relationships between entities of two variant business processes are used to construct two perspective graphs. The common nodes and edges of the two graphs are compared in terms of weight. The output are statistically significant differences using a differential graph or a matrix.

Bolt et al. (Bolt, Leoni, and Aalst, 2018) proposed a technique to compare the behaviour of different variants of the same process based on their executions in event logs. First, the authors build a transition system from two event logs corresponding to a process variant, annotating each of its states and transitions with the measurements of the variants for a particular process metric. Then, they perform a statistical test between every two sets of metric measurements on each state or transition to identify the statistically significant differences. Finally, they distinguish the differences identified by thickening the arc corresponding to statistically significant activity in the transition system. However, the technique can only identify that a particular activity occurs after a sequence of activities in one process but not in the other, thus missing the structural differences between the processes. Furthermore, the technique does not work with event streams.

Van Beest et al. (Beest et al., 2016) used two prime event structures to encode the process behaviour captured by two event logs and compare them. The outcome is a report of their differences as natural language statements. A downside of the technique is that it reports many differences, especially when the changes are applied to process fragments or when they occur in a nested way.

Armas-Cervantes et al. (Armas-Cervantes et al., 2014) proposed a method based on canonically reduced event structures for diagnosing behavioural differences between two process models. The idea is similar to (Beest et al., 2016). However, here, two event structures are built from the process models and compared with each other to extract a set of natural language statements that report their differences. The technique trades off between fully capturing the behaviour of the log and overgeneralising that behaviour. As a result, some differences may be lost and some may be added.

Ostovar et al. (Ostovar, Leemans, and Rosa, 2020) discovered process trees from a portion of the event stream. They used a process tree transformation technique to find a minimum-cost sequence of edit operations that transform the process tree before and after. The results are natural language statements based on common business process change patterns (Weber, Reichert, and Rinderle-Ma, 2008). The method performs well for complex changes with multiple activities, but not so well with localising simple drifts that involve individual activities in the presence of noise.

Gaspar et al. (Sousa et al., 2021) used trace clustering to analyse differences in process behaviour. Their method compares the clustering before and after the detected drift to identify the specific entities involved in the control-flow change. They compare the centroid positions' variation using MSE for each pair of matching clusters from before and after. A high MSE value in a given dimension of the centroid indicates that the corresponding activity behaves differently before and after the drift.

TABLE 2.6: Existing approaches to business process drift detection

Approach	**Studies**	**Techniques Used**	**Limitations**
Model-to-Log Alignment- this group of approaches is based on the presence of a process model and an event log as input to the analysis process.	(Buijs and Reijers, 2014), (Aalst, 2012)	• Alignment matrix (Buijs and Reijers, 2014) • Vertical & horizontal partitioning (Aalst, 2012).	• The approaches resemble conformance-checking tasks, and conformance-checking is not suitable for detecting concept drift as it requires the presence of a precise normative model.
Graph-Based- visualisations are created as an intermediate step in analysing drifts for this group of approaches.	(Lu et al., 2016), (Nguyen et al., 2018)	• Perspective graphs and finding statistical differences between them (Nguyen et al., 2018). • Backtracking, heuristic function, greedy algorithm, cost function REGs (Lu et al., 2016).	• The approach in (Lu et al., 2016) becomes inaccurate when detecting an increasing number of deviations and when applied to real-life logs. • The approach in (Nguyen et al., 2018) assumes that an event log follows a basic structure that allows the proposed investigation.

Clustering-Based- This group of approaches clusters traces to detect changing behaviour in an event log.	(Yeshchenko et al., 2019), (Zheng, Wen, and Wang, 2017), (Hompes et al., 2015a), (Bolt, Aalst, and De Leoni, 2017), (Sousa et al., 2021)	• Sliding window, Hierarchical clustering, Change point detection using PELT and Declarative constraints discovery using MINERFUL(Yeshchenko et al., 2019). • Recursive Partitioning by Conditional Inference (RPCI), points of interest & event augmentation(Bolt, Aalst, and De Leoni, 2017). • MCL & cosine similarity (Hompes et al., 2015a; Hompes et al., 2015b). • DBSCAN clustering (Zheng, Wen, and Wang, 2017).	• The approach in (Hompes et al., 2015a) relies on the distinctions of the cluster sizes and densities to determine the exceptional behaviour from the standard. However, these considerations depend on the similarity matrix, whose sensitivity to changes depends on the similarity measurement and thresholds used. • Users must define a threshold on which the relevance of a point of interest can be decided. • Generally, the approaches presented in this category have in common that they start with defining a behavioural profile or a feature set for the activities executed in a process. Most of these profiles neither represent any notion of equivalence nor provide a diagnosis of differences between pairs of models. They also can not capture certain process behavioural patterns, such as skipping a task, concurrency, or looping.

Windowing and Statistical Analysis - This group of approaches rely on statistical approaches to detect changes in event logs divided into windows of fixed or changeable sizes.	(Maaradji et al., 2017), (Ostovar et al., 2016), (Bose et al., 2013), (Seeliger, Nolle, and Mühlhäuser, 2017), (Carmona and Gavalda, 2012), (Martjushev, Bose, and Aalst, 2015)	• Adaptive windowing, statistical testing (Chi-Square test) and oscillation filter(Maaradji et al., 2017). • Adaptive windowing, statistical testing (G–test of independence)(Ostovar et al., 2016) • Statistical hypotheses testing using Kolmogorov–Smirnov (KS) test, Mann–Whitney test, Hotelling T2 test & windowing(Bose et al., 2013) • Adaptive windowing, Heuristic mining & G-Tests(Seeliger, Nolle, and Mühlhäuser, 2017). • Adaptation of ADWIN approach(Martjushev, Bose, and Aalst, 2015). • Abstract interpretation, Parikh vectors and Adaptive windowing(Carmona and Gavalda, 2012).	• The approach in (Carmona and Gavalda, 2012) is unable to detect multiple drifts at the same time and is unable to pinpoint the exact moment of the drift. • In (Bose et al., 2013), there is a need for user interventions by specifying the features used for drift detection. The approach is also unsuitable for identifying certain types of drift, for example, the insertion of conditional branching in a process model.

2.9 Discussion

The chapter comprehensively reviews the literature on PM and other related works. It outlined problems that can be found in the domain. It also introduced the DL techniques used in PM, including the limitation of their application. Lastly, it introduced drift detection and localisation in PM.

PM aims to discover, monitor and improve real processes (Van Der Aalst, 2012). According to the literature review, research in the PM domain is predominantly focused on process discovery. In (Zerbino, Stefanini, and Aloini, 2021), the following possible two reasons for this are suggested:

1. Process discovery enables conformance and enhancement. In fact, awareness of how business processes are actually performed provides reliable process models to check or improve.

2. Enhancement, unlike discovery, suffers from a shortage of methodologies, algorithms, and plug-ins. This is because (i) its scope is potentially more extensive as it includes resource allocation, business process and information flow

redesign, scenario analysis, and so on; and (ii) it is context-bound and not limited to diagnostic activities.

Process discovery relies on data collected from an information system over a period of time or in real-time data output by running processes, commonly known as an event log. Process discovery methods can automatically construct a process model that appropriately displays the observed behaviour as it is captured in an event log without any inferred information. These models enable businesses to evaluate performance, check compliance, spot anomalies and suggest improvements.

DL methods such as RNN and LSTM have proven to achieve a better performance in terms of accuracy and generalising ability when predicting next events in business processes compared to traditional process discovery techniques such as Petri nets and the BPMN. However, existing DL-based PM methods lack a mechanism that explains how the predictions of the next events are made, unlike traditional network-based PM techniques that can be used to visually present the entire discovered process.

The topic of drift detection and localisation is also introduced in this chapter. Several methods have been proposed for concept drift detection in business processes. Most methods use the windowing technique to select traces from an event log to consider for drift analysis, along with statistical hypothesis testing as a solution. Some studies used clustering-based techniques to find groups of traces that share similar characteristics that can be generalised and used to detect drifts. Other studies used graph-based analysis techniques or model-to-log alignment.

While many of the methods were reported to perform well, they have limitations. For example, the windowing technique used in many concept drift detection algorithms is highly dependent on the right choice of window size; a wrong window size can result in many false negatives and false positives. Some methods are unable to locate the exact moment of a drift (e.g (Carmona and Gavalda, 2012; Sousa et al., 2021)). Some methods are not automated (e.g (Bose et al., 2013)), they require human involvement in feature selection and change point recognition, making them impractical. Finally, most of the existing methods for detecting concept drifts in business processes are designed to work offline; i.e. they require the entire event logs featuring cases from both before and after a drift. Some online methods detect drifts with a long delay, while others do not perform well on processes whose logs display many distinct executions. Thus, detecting drifts in the online scenario (i.e. as they happen) remains challenging.

It is also worth noting that most drift detection approaches concentrate on detecting drifts through their effect on the resulting process instances, i.e., cases in the event log, rather than detecting them by comparing process models depicting different process instances before and after the drifts (Elkhawaga et al., 2020).

2.10 Summary

This chapter discussed the limitations of existing solutions for predicting the next activities in business processes, as well as the detection and localisation of drift in business processes. The following chapters will present the proposed solutions to tackling these challenges with the aim of addressing the discussed limitations. In particular, Chapter 3 addresses the problem of predicting the next activities in business processes by proposing a new approach to PM that combines the benefits of visually explainable graph-based methods and accurate but implicit DL methods.

The following chapters present the proposed methods for detecting and localising drifts in business processes.

Chapter 3

Predicting Next Activities in Business Processes

Chapter 2 discussed that the existing DL-based methods for PM lack a mechanism explaining how the predictions of the next activities in business processes are made. To address this limitation, this chapter presents a graph-based approach to explaining the decision-making process of an LSTM model when generating a sequence of events representing a business process. According to this approach, an LSTM model is trained on an event log first. This model is then employed to find the probabilities for each event present in the log to appear in the business process next. Finally, these probabilities are used to generate a visually explainable process model graph that represents the decision-making process of the LSTM model. A probability threshold is introduced as a parameter to manage the graph complexity and thus enable faster and more directed business process analysis, including discovering the most common event sequences and unusual or suspicious behaviours.

The chapter is structured as follows. Section 3.1 details the proposed approach. Section 3.2 describes the experimental setup, while Section 3.3 presents and discusses the results. Finally, Section 3.4 summarises the chapter. The work described in this chapter has been published in (Hanga, Kovalchuk, and Gaber, 2020).

3.1 Proposed Approach to Process Mining

The proposed PM approach consists of two stages: building an accurate LSTM model to predict the sequences of events in the business process based on event logs and generating a DFG explaining the decision-making process of the LSTM model for predicting the sequences of events in the business process. Figure 3.1 summarises the steps of the two stages of the proposed approach.

3.1.1 Data Preparation

The proposed approach takes as input an event log defined as follows.

Definition 1 (Event log). An event log **L** is a set of traces $\mathcal{T}$ that contain a set of events $\mathcal{E}$ recorded as the result of each execution of a process instance. Each trace t contains a sequence of events $t = e_1, e_2, ..., e_n$, where $e_i \in \mathcal{E}$ and $1 \leq i \leq n$. Each event can have a set of attributes $\mathcal{P}$ that provide additional event details, e.g. timestamp or a resource that executes the specified activity.

Event Log Pre-processing

The concept of NLP was used to parse event logs and train the proposed LSTM model. The model was trained to establish the most likely activity to come next in a

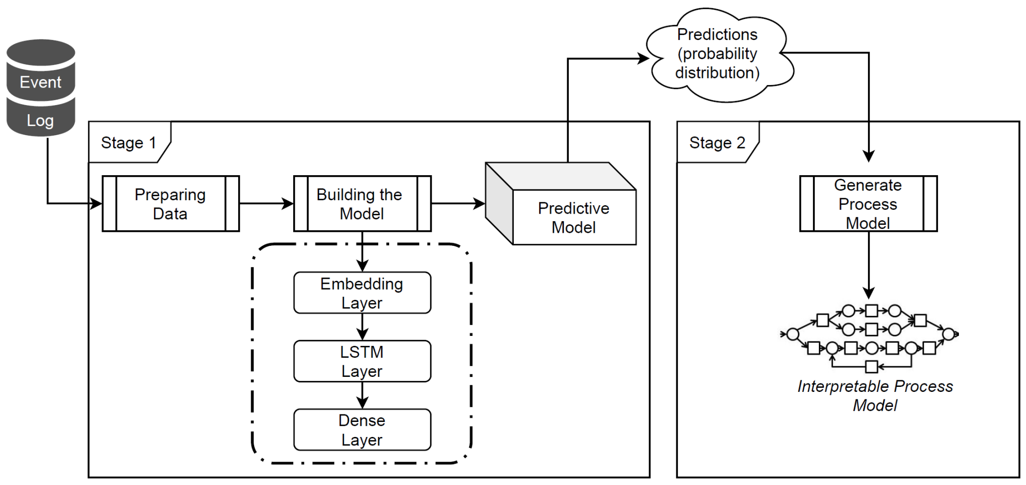


FIGURE 3.1: Flow diagram summarising the proposed approach to process mining

given sequence of events over time. This problem also resembles a time-series problem due to sequence dependence among input variables. In NLP, when predicting the next word to appear in a sentence, the context of the whole sentence is considered to avoid ambiguity (Brownlee, 2017). The same concept was used to frame the input sequences for the LSTM model from event logs. Similar to free text, event logs are also prone to ambiguity because of the varied length of sequences and the existence of repeated activities. To improve the model's training process, the context was broadened by phrasing the problem so that multiple previous time steps were considered when predicting the next time-step. In particular, the event logs were pre-processed according to the following protocol and definitions.

The *NULL* label marks the beginning of each case (or process instance). It becomes the first input activity x_0 at the current time t_0, and the target activity y_0 becomes the activity at time t_1 (this is the first activity occurring in each case of the event log). The next sequence of input activity x_1 becomes the activity at the previous times $\{t_0, t_1\}$, and the target y_1 becomes the next activity at time t_2. The next input activity sequence x_2 becomes the activity at the prior times $\{t_0, t_1, t_2\}$, and the target y_2 becomes the next activity at time t_3; and so on. The inputs X build up for each next input sequence as the prior activities are joined with the current activity. Targets Y are always activities in the next time-step t_{n+1} until the last input sequence contains all activities in the previous time steps, including the current activity. At this point, the label *END* is added to mark the end of a case. This procedure is repeated until all cases in the event log are pre-processed.

Definition 2 (Predicting next activity). Given a trace of activities $t = a_1, a_2, ..., a_t$, the output of the predictive model is the next activity $\{a_{t+1}\}$.

Definition 3 (Predicting complete traces). Given the prefix of a trace $t = a_1, a_2, ..., a_t$ and the value **END** to mark the end of each case, the output of the predictive model is the sequence of activities $\{a_{t+1}, a_{t+2}, ..., \textbf{END}/a_t\}$.

Encoding and Padding

The input sequences are encoded using the *Tokenizer class* from the *Keras* library (Gulli and Pal, 2017). The tokenizer maps each activity in an event log to a unique integer, creating a sequence of integers. Next, the prepared sequences are padded using the *pad sequences()* function of *Keras* (Gulli and Pal, 2017). The function first finds the longest sequence, and then uses the length to pad other sequences to have a uniform length. The targets were dummy-encoded using the *pd.get_dummies()* function of the Pandas library. The function converts categorical variables into dummy variables by placing a value of one when a categorical event occurs and zero when it does not occur. For example, if $A = [a, b, c]$, the respective tokens are $A \rightarrow [1, 2, 3]$, where $a = 1$, $b = 2$ and $c = 3$. Each activity $a_i \in \mathcal{A}$ is encoded as a vector (A_i) of length $|A| + 3$ such that the first $|A|$ features are all set to zero, except the one occurring at the index of the current activity a_i, which is set to one. Table 3.1 shows an example of prepared input sequences, where $[1, 2, 3, 4, 5, 6]$ are the resulting tokens or integers of tokenised activities $[NULL, a_1, a_2, a_3, a_4, a_5]$. The target column contains the encoded dummies (i.e., the converted categorical labels) $[a_1, a_2, a_3, a_4, a_5, END]$, where every activity is set to zero except the target activity, which is set to one. Whenever the target is *END* encoded as [1000000] in the target column, the next process instance is visited. This procedure is repeated for all process instances in the event log.

TABLE 3.1: Example of prepared data for training the proposed LSTM models

Time-step (t)	Input	Target
t_0	[0, 0, 0, 0, 0, 0]	[0100000]
t_1	[0, 0, 0, 0, 0, 1]	[0010000]
t_2	[0, 0, 0, 0, 1, 2]	[0001000]
t_3	[0, 0, 0, 1, 2, 3]	[0000010]
t_4	[0, 0, 1, 2, 3, 5]	[0000100]
t_5	[0, 1, 2, 3, 5, 4]	[0000001]
t_6	[1, 2, 3, 5, 4, 6]	[1000000]
t_0	[0, 0, 0, 0, 0, 0]	[0100000]
t_1	[0, 0, 0, 0, 0, 1]	[0000010]
t_2	[0, 0, 0, 0, 1, 5]	[0001000]
...	...	...

3.1.2 Proposed Models for Predicting Next Activities in Business Processes

Model Architectures

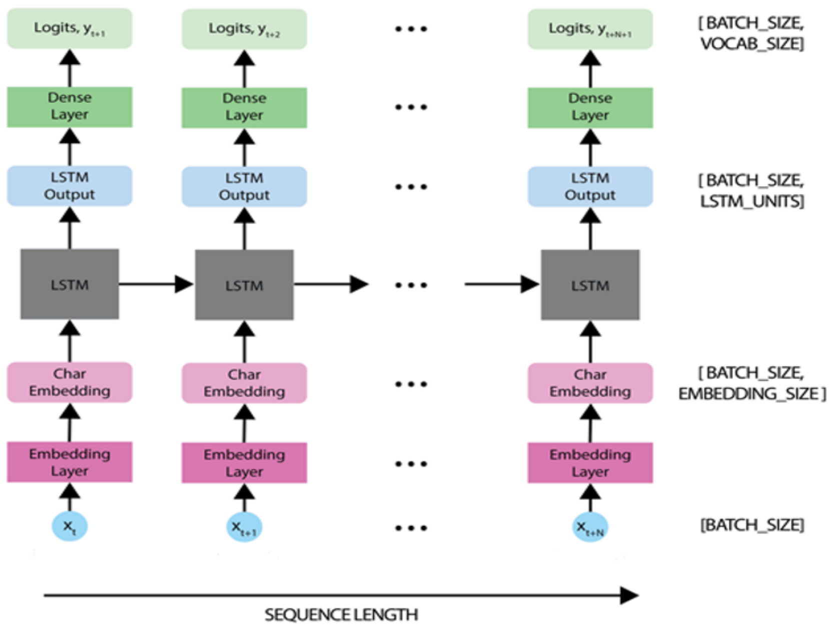
In addition to the unique way of pre-processing event logs described above, another distinctive feature of the proposed models compared to other LSTM models for PM presented in the literature is their architecture. In particular, an additional embedding layer is employed as an interface between the input and LSTM layers of the network. The embedding layer takes three arguments: the vocabulary size (i.e. the set of unique activities in a log), embedding vector space size, and length of input sequences. These parameters are dependent on event log characteristics. The output of the embedding layer serves as input to the LSTM layer. A single LSTM layer is followed by an additional fully connected dense layer as the output layer. The output layer uses the softmax activation function to ensure the output takes the form of probability distributions. While some of the existing LSTM models for PM add a dense layer on top of the LSTM layer, they either do not use embedding at all or implement it differently (e.g. see (Tax et al., 2017; Camargo, Dumas, and González-Rojas, 2019)). Furthermore, the existing LSTM models for PM employ two LSTM layers, whereas the proposed models have only one LSTM layer. The first model (Model 1) employs a unidirectional LSTM, while the second (Model 2) employs a BLSTM. The latter trains two LSTMs instead of one on the input sequence: the first – on the input sequence as it is and the second – on a reversed version of the input sequence. This gives additional context to the network, resulting in faster and improved learning.

Figures 3.2a and 3.2b illustrate the network architectures of the proposed DL models, where the output of each layer is the input to the next layer. In both architectures, the first *Input Layer* contains an activity sequence as constructed during the log event pre-processing procedure (see the previous section for details). The size of (ActNum) is determined by the number of unique activities in the event log. The *Embedding Layer* encodes the input sequence into a sequence of dense vectors of dimension (DimSize). While the dimension size can be tuned for better model performance, the choice should be guided by the number of unique activities in the log. The *LSTM Layer* in Figure 3.2a and *BLSTM Layer* in Figure 3.2b are characterised by the number of neurons (NeuronNum). Here, the vector sequence is transformed into a single vector of size (NeuronNum), containing information about the entire sequence. There are no set criteria on how many hidden neurons an LSTM layer should have; it is problem-dependant. For the problem considered here, the number of neurons is chosen considering the event log size and the number of unique activities in the log. A *Dropout* probability can be added to prevent overfitting and improve learning. The *Dense Layer*, which is the last layer, outputs the probability for each unique activity in a log to appear next in the sequence, including the extra *END* activity added to mark the end of the process. Hence, the output takes the form of a vector of size TargetNum (i.e. the number of expected targets), which is (ActNum + 1).

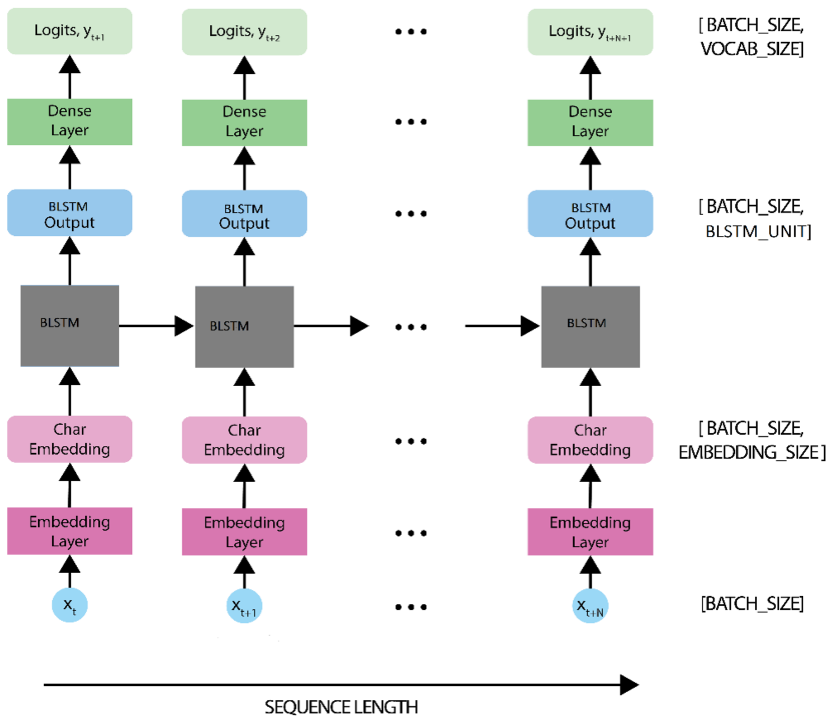
Training LSTM Models

Providing the multi-class classification problem (i.e. predicting an activity from a list of activities), the categorical cross-entropy loss function was used during the training process of each model. The Adam optimisation algorithm (an implementation of the gradient descent algorithm) was used to track the loss and accuracy at the end of each epoch. The goal of the training process was to minimise the log loss by adjusting the trainable parameters (i.e. weights of the network).

Each model was fitted using training data, a variable number of training epochs, and batch sizes. Each epoch trained the model on the entire training event log while maintaining the weights and biases learned from the previous epochs. At the end of each epoch, test data were run through the model, and the average accuracy across all possible next event cases in the test set was returned.



(A) Model 1.



(B) Model 2.

FIGURE 3.2: Deep learning network architectures of the two proposed LSTM models: (a) Model 1 - unidirectional LSTM; (b) Model 2 - bidirectional LSTM

Next Activity Prediction

After the training phase, the LSTM models are ready to be used to predict next activities and, eventually, complete traces. Algorithm 1 lists the pseudo-code of the prediction algorithm. The algorithm takes the prefix $\mathcal{T}_p$ and the trained LSTM model

μ as inputs and returns the complete trace $\mathcal{T}$. First, $\mathcal{T}_p$ is encoded using *tokenizer*. Then, the tokenised prefix is padded to the left with zeros to form a sequence of the same length and fed into the embedding layer. The resulting vector-matrix from the embedding layer is fed into the model μ, which then generates the probability distribution over different possible activities that can occur in the next position of the trace. The activity with the highest probability is returned as the next activity β. A new trace is obtained by concatenating the current prefix with the new predicted activity. The procedure is repeated until the model predicts the end activity denoted as *END* and until it does the same for all traces in the event log.

Algorithm 1 Next activity prediction

```
Require: Prefix: $\mathcal{T}_p$, LSTM model: $\mu$
 1: $h \leftarrow 0$
 2: $\mathcal{T} \leftarrow \mathcal{T}_p$
 3: while $(\beta <> END)$ and $(h < Log\_End)$ do
 4:     $\mathcal{T}_p \leftarrow tokenize(\mathcal{T}_p)$
 5:     $\mathcal{T}_p \leftarrow pad(\mathcal{T}_p)$
 6:     $\mathcal{T}_p \leftarrow embed(\mathcal{T}_p)$
 7:     $P \leftarrow Predict(\mu, \mathcal{T}_p)$
 8:     if $\beta \leftarrow highest\_probability$ then
 9:         $\mathcal{T} \leftarrow \mathcal{T}_p \cdot \beta$    {concatenate the current prefix with the new next activity}
10:         $h \leftarrow h + 1$
11:         Return  Complete trace: $\mathcal{T}$
12:     end if
13: end while
```

3.1.3 Graphs Explaining the Prediction Process of LSTM Models

In this thesis, a DFG is used to explain the decision-making process of each LSTM model when predicting the next activities in a business process. The DFG is generated according to Algorithm 2 based on the probabilities obtained at each iteration of Algorithm 1. Each arc (a directed edge) of the DFG is annotated with a prediction probability based on the following definitions.

Definition 4 (Directly-follows Probability). Given a sequence of activities $\{a_1, a_2, ..., a_n \in \mathcal{E}\}$, the directly-follows probability between a_1 and a_2 is the LSTM model's next activity probability assignment for a_2.

Definition 5 (Directly-follows Graph). Given a directly follows probability matrix, its DFG is a directed graph $\mathcal{G} = (i, o, N, E)$, where i is the start event, o is the end event, N is a non-empty set of nodes and $E \subseteq (x, y)|x, y \in N$ is the set of edges.

A process is a directed graph if there exists a graph node for each activity of the process $(N = A)$. The edges of the graph represent the possible transitions between the activities. For any two parallel activities $a, b \in A$, there will be directed arcs (a, b) and (b, a) between them.

The *START* and *END* nodes are introduced to produce sound process graphs. This is necessary to prevent deadlocks or lack of synchronisation (Aalst, Weijters, and Maruster, 2004). The prediction probability matrix is generated first and then used to build a graph from a set of traces. The matrix describes the activities that succeed other activities in the set of traces. The rows in the matrix represent the trace prefixes, while the columns represent the activity that directly follows. The numbers in the matrix cells represent the next activity prediction probabilities.

Generating the probability matrix involves feeding the first activity of each trace a_1 to the LSTM model and getting the predictions of each possible activity being the next in the sequence. The activity with the highest probability is selected as the most likely next event a_2. Next, the combination of the first activity and the next predicted activity $\{a_1, a_2\}$ is fed into the LSTM model to predict the next activity a_3; and so on. For each trace, this continues until the LSTM model predicts that the next activity will be END, which automatically marks the end of each process instance.

The process graph is generated by traversing each row in the matrix, choosing the column with the highest probability as the most likely next activity, and creating a transition between the preceding and succeeding activity (drawing an edge E between the nodes N). This is repeated until all rows in the matrix are visited. Algorithm 2 highlights the procedure. To begin with, a set of all start activities (η_0) (i.e., activities that appear first in each trace), a set (N) of all established nodes(to avoid having duplicate activities in the graph), and a set of edges (E) are initialised. From the *START* node, a transition is created to the start activity of the first trace. The start activity is used to determine the next and subsequent activities, creating a transition between each preceding and succeeding activity β (i.e. from the matrix, the activity with the highest probability in each row). This is repeated for each case until the end of the matrix is reached. The activity that appears as the last activity o in each trace is joined to the *END* node. Parallel transitions are observed if two or more activities have the same prediction probabilities, which indicates that the process may be branching (i.e. two or more activities may follow the previous activity in the process).

The transition probabilities P are appended to its edges to make the graph more informative. Furthermore, a probability threshold ϕ is introduced as a parameter to allow tuning the complexity of the graph. Although the main purpose of the graph is to explain the decision-making process of the LSTM model when predicting the next events in a business process, it can also be used to perform various PM tasks. For example, by setting a high probability threshold, one can visualise the most common way of the business process execution. In contrast, by setting the threshold to a low value, one can capture less common instances of the business process execution, potentially indicating cases of non-conformance.

Another benefit of the proposed approach is its ability to generate traces for building a *single process graph* for each case by simply specifying the *case id*. This can greatly benefit when the interest is in investigating particular cases.

Algorithm 2 Generating a process graph based on LSTM predictions

```
Require: Predicted Activities: ${\{a_{1,1}, a_{1,2}, ...a_{1,x}\}...\{a_{n,1}, ...a_{n,x}\}}$, Threshold: $\phi$
 1: Create START node
 2: Create END node
 3: $\eta_0 \leftarrow \emptyset$   {nonempty list of all start events}
 4: $N \leftarrow \emptyset$   {nonempty set of established nodes}
 5: for $a \in A$ do
 6:   $\eta_0\{i\} \leftarrow a_{i,1}$
 7:   $E : START \rightarrow \eta_0\{i\}$   {Create a transition}
 8:   $counter \leftarrow 0$
 9:   $event\_list \leftarrow \emptyset$   {$event\_list \subseteq a$}
10:   $o \leftarrow \eta_0\{i\}$
11:   if $\beta \leftarrow highest\_probability$ and $\beta \notin N$ then
12:     $N \leftarrow (N, \beta)$
13:   else if $P \leq \phi$ then
14:     $E : o \rightarrow \beta$
15:     $E : Append(P)$   {add probability to the edge}
16:     $o \leftarrow \beta$
17:     $E : o \rightarrow END$
18:     Return  Process graph: $\mathcal{G}$
19:   end if
20: end for
```

3.2 Experiments

Algorithms 1 and 2 were implemented as a set of Python scripts using Python 3.6. LSTM models were built using the Keras (Gulli and Pal, 2017) and Tensorflow (Abadi et al., 2016) libraries. Process graphs were generated using the Graphviz library (Ellson et al., 2001), while trace graphs for each process instance were generated using the NetworkX library (Hagberg, Swart, and S Chult, 2008). The experiments were carried out using the Google Colab free Tesla K80 GPU.

3.2.1 Evaluation

To evaluate the predictive performance of the proposed LSTM models, they were applied to the following five real-life logs: Helpdesk (Polato, 2017), BPIC 2012 (Dongen, 2012), BPIC 2013I (Steeman, 2013b), BPIC 2013C (Steeman, 2013a) and Road traffic fine management process (Leoni and Mannhardt, 2015). The details of these logs are provided in the next section and Table 3.2. The accuracy of the next event predictions of the two proposed models was compared with that of three other LSTM models proposed by Tax et al. (Tax et al., 2017), Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) and Lin et al. (Lin, Wen, and Wang, 2019). These studies used only the Helpdesk and BPIC 2012 logs from the five real-life logs listed above.

Tax et al. (Tax et al., 2017) used one-hot encoding to encode events as input to an LSTM model that had two layers and 100 neurons in each layer. Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) extracted n-grams of fixed sizes for each event log trace, which they used as input to their LSTM model consisting of two stacked LSTM layers and a dense output layer. Also, Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) used pre-trained embedded dimensions in all their experiments. Lin et al. (Lin, Wen, and Wang, 2019) used a two-layer LSTM-based encoder-decoder architecture with 32 neurons per layer for each encoder and decoder cell. Random embedding was established for each event. Only the S-Pred model proposed by Lin et al. (Lin, Wen, and Wang, 2019) was considered in this study as it is concerned with predicting the next events based solely on sequences of events (i.e., without considering additional information such as event attributes).

Being the most popular, the Helpdesk log was used to generate process graphs to visually demonstrate the ability of the approach presented in this paper to explain the LSTM decision-making process when predicting event sequences of business processes. We are the first to propose such an approach to the best of our knowledge.

3.2.2 Real-life Event Logs

The proposed approach was evaluated on the following five real-life event logs. The original log files were presented in the XES format but, for the purpose of this study, were converted to CSV files using the ProM tool. The sizes of these event logs vary. Table 3.2 shows the statistics.

Helpdesk (Polato, 2017): This event log contains records of real-life events from a ticketing management process of the help desk of an Italian software company.

BPIC 2012 (Dongen, 2012): This event log represents a loan application process of a German financial institution. It contains three intertwined sub-processes. The first alphabet of each task name identifies which sub-process this task originated.

BPIC 2013I and BPIC 2013C (Steeman, 2013b) and (Steeman, 2013a): These are the real-life event logs of the Volvo IT incident and problem management system called VINST. The first includes *Incident* cases of the incident management process, while the second includes *Closed* cases of closed problems in the problem management process, both of which are used in the experiment.

Road Traffic Fine Management Process (Leoni and Mannhardt, 2015): This is a real-life event log of an information system that manages traffic fines.

TABLE 3.2: Event log description

Event Log	Number of Traces	Number of Events	Number of Activities
Helpdesk	4,580	21,348	14
BPIC 2012	13,087	262,200	25
BPIC 2013I	7,554	65,533	13
BPIC 2013C	1,487	6,660	7
Road Traffic Process	150,370	561,470	11

3.2.3 Experimental Setup

The following experimental protocol was adopted to closely follow the experiments conducted by Tax et al. (Tax et al., 2017), Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) and Lin et al. (Lin, Wen, and Wang, 2019) for a fair comparison with their models:

1. Similar to Tax et al. (Tax et al., 2017) and Camargo et al. (Camargo, Dumas, and González-Rojas, 2019), each event log was divided into two sets: a *training set* composed of 70% of traces used as input for training the LSTM models and a *test set* composed of the remaining 30% used to evaluate the predictions and assess the models' ability to generalise (Lin et al. (Lin, Wen, and Wang, 2019)

also used 70% of traces for training but only 20% of traces to test their models' accuracy in predicting next events). The splitting was performed using the algorithm proposed by (Evermann, Rehse, and Fettke, 2017), which is consistent with the method used by Tax et al. (Tax et al., 2017).

2. For the embedding layer, the vocabulary size was set to the number of unique activities in the log. The number of embedding dimensions was guided by the vocabulary size; hence it varied across the event logs.

3. The networks had one hidden LSTM layer with a dropout probability of 0.2. The dropout technique was used to avoid over-fitting and improve learning by temporarily removing a cell from the network randomly.

4. The number of training epochs was set to 50 for all logs.

The parameters of the proposed models, namely, the number of embedding dimensions and LSTM neurons, were tuned for each event log separately for better performance. In particular, 32 embedding dimensions were used for the BPIC13 log, and 50 dimensions were used for the three other logs; 100 neurons were used in the LSTM layer for all logs. For a fair comparison with the other three LSTM models proposed by Tax et al. (Tax et al., 2017), Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) and Lin et al. (Lin, Wen, and Wang, 2019), each model's performance was evaluated based on the average accuracy score achieved across all possible next event cases in the test set.

3.3 Results and Discussion

3.3.1 Model Predictive Performance

Table 3.3 summarises the performance of the proposed LSTM models on the considered event logs based on the accuracy of predicting the next activity over the test set. The results are compared to those achieved by the other three models proposed by Tax et al. (Tax et al., 2017), Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) and Lin et al. (Lin, Wen, and Wang, 2019). For the comparison, only the Helpdesk and BPIC2012 event logs were used since they are the only logs out of the five considered here, for which results are reported in the other three studies.

TABLE 3.3: Prediction results over the test sets

Implementation	Event Log	Next Activity Prediction Acurracy
Model 1: Unidirectional LSTM	Helpdesk	**0.912**
	BPIC 2012	**0.854**
	BPIC 2013I	0.694
	BPIC 2013C	0.640
	RTFMP	0.814
Model 2: Bidirectional LSTM	Helpdesk	0.907
	BPIC 2012	0.853
	BPIC 2013I	**0.698**
	BPIC 2013C	**0.644**
	RTFMP	**0.815**
Tax et al. (Tax et al., 2017)	Helpdesk	0.712
	BPIC 2012	0.760
	BPIC 2013I	-
	BPIC 2013C	-
	RTFMP	-
Camargo et al. (Camargo, Dumas, and González-Rojas, 2019)	Helpdesk	0.789
	BPIC 2012	0.786
	BPIC 2013I	-
	BPIC 2013C	-
	RTFMP	-
Lin et al. (Lin, Wen, and Wang, 2019)	Helpdesk	0.808
	BPIC 2012	0.814
	BPIC 2013I	-
	BPIC 2013C	-
	RTFMP	-

It can be noticed from Table 3.3 that the proposed Model 1 and Model 2 outperform the other three models on the Helpdesk event log, achieving accuracy scores of 91.2% and 90.7% compared to 71.2%, 78.9% and 80.8% achieved by Tax's et al. (Tax et al., 2017), Camargo's et al. (Camargo, Dumas, and González-Rojas, 2019) and Lin's et al. (Lin, Wen, and Wang, 2019) models, respectively. On the BPIC2012 event log, the proposed Model 1 and Model 2 achieve 85.4% and 85.3%, respectively, compared to 76.0%, 78.6% and 81.4% achieved by Tax's et al. (Tax et al., 2017), Camargo's et al. (Camargo, Dumas, and González-Rojas, 2019) and Lin's et al. (Lin, Wen, and Wang, 2019) models, respectively.

The better performance of the proposed models can be explained by the difference in the event log pre-processing approach and the models' architecture. In particular, Tax et al. (Tax et al., 2017) encoded activities in event logs using one-hot encoding. This type of encoding might be suitable for logs with a small number of activities but becomes inefficient for logs with many activities. Furthermore, while Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) employed the word embedding technique, they trained an independent network to coordinate the embedded dimension, which they used throughout their experiments as non-trainable parameters (which is different from our approach of employing an embedding layer as part of one network architecture). They probably did this to shorten the training time while improving the quality of the predictive model. Still, this approach cannot guarantee the compatible prediction accuracy. Finally, the proposed models employ only one LSTM layer, compared to the two LSTM layers used by Tax et al. (Tax et al., 2017), Camargo et al. (Camargo, Dumas, and González-Rojas, 2019) and Lin et al. (Lin, Wen, and Wang, 2019). We believe that this simplification contributes to the better performance of our models in the case of the Helpdesk log, which has a simple sequence flow.

Another observation that can be made from Tables 3.2 and 3.3 is that the proposed models perform well on the larger RTFMP log (Model 1: 81.4%; Model 2: 81.5%) but not so well on the smaller BPIC2013I and BPIC2013C logs (Model 1: 69.4% and 64.0%; Model 2: 69.8% and 64.4%). The latter result can be explained by the relatively small number of traces (e.g. compared to the BPIC 2012 log) and the relatively large number of events (e.g. BPIC2013C compared to the Helpdesk log) included in the BPIC2013 logs (i.e., the logs included many repetitive traces). This means that the models are expected to pick up the sequences of activities that can happen in reality among many possible combinations of these activities when provided only with a small number of examples of these combinations that happened in reality. Model 2 performs slightly better than Model 1 on the BPIC2013I, BPIC2013C, and RTFMP logs in terms of both training and test accuracy. Thus, forward and backward learning in Model 2 is beneficial in the cases of large logs and complex logs with repetitive activities. At the same time, it took longer to train Model 2 than Model 1. Hence, there is a trade-off between the accuracy and training time of the models. Since Model 2 improves accuracy slightly compared to Model 1, the latter is preferred because it is both simple and accurate.

3.3.2 Graphical Representation of Model Predictions

In addition to presenting a better performing LSTM model for PM, another contribution of this study is proposing the idea of generating graphs of different complexity to visually explain the decision-making process of an LSTM model when predicting the next events in business processes. These graphs can be used to explore the performance of the LSTM model and identify difficult cases with which the model must deal so that measures could be taken to improve the model performance in such cases. Furthermore, the graphs can be used to perform various PM tasks such as model discovery, conformance checking, and investigating cases of non-compliance.

Figures 3.3, 3.4, 3.5 and 3.6 illustrate the graphs generated using the outputs of Model 1 for the Helpdesk log. In particular, Figure 3.3 shows the graph generated based on the predictions of Model 1 on the training set, while Figure 3.4 shows the same for the test set. The probability threshold was set to 0 for both graphs so that all transitions present in the log could be represented in the graphs. The numbers on the edges indicate the probability scores of the respective activities that will be next in the process as predicted by the LSTM model.

When comparing the two graphs, one can notice their consistency, which indicates a good ability of both the model to generalise and the graph built over the training set to interpret the model's behaviour. They were converted to adjacency matrices to formally verify the similarity between the training and test graphs. Then the similarity score between the matrices was calculated by taking the sum of the differences between the values in the corresponding cells of the two matrices and dividing it by the number of non-zero values in the test matrix. Since we are interested in evaluating the interpreting ability of the training graph, the training matrix was reduced to the test matrix size, and the transitions that were captured during the training process but did not appear during testing were excluded from the calculation. A score of 1 obtained this way would mean the two graphs are identical, whereas a score of 0 would mean they are completely different. Two cases were considered for the calculation. In *Case 1*, binary matrices were constructed, where the value of 1 in a cell corresponded to the transition between the corresponding two activities, and the value of 0 represented the fact that there was no transition between the two activities. In *Case 2*, the matrices were constructed in the same way as in *Case 1*, except that instead of the value 1, the probability of transition was recorded as predicted by the LSTM model.

Tables 3.4 and 3.5 represent the adjacency matrices built for binary-based *Case 1* using the process graphs illustrated in Figures 3.3 and 3.4, respectively. Tables 3.6 and 3.7 represent the same for probability-based *Case 2*. After dividing the sum of the difference between the training and test matrices by the number of non-zero values in the test matrix (41), scores of 0.80488 and 0.81708 were obtained for *Case 1* and *Case 2*, respectively. Providing that the scores are close to one, it can be concluded that the two graphs are similar, indicating the model's ability to generalise and the training graph to interpret the model's behaviour.

Figure 3.5 shows a less complex version of the graph presented in Figure 3.3, which was achieved by setting the probability threshold at 0.8. This means that all predicted transitions with a probability less than the set threshold are pruned and not included in the graph, which results in a simpler visualisation. For example, the probability threshold can be adjusted to any value required by a task at hand to understand the performance of the LSTM model or perform PM tasks. Finally, Figure 3.6 shows the graph of a single trace from the Helpdesk log. This type of graph can be useful for investigating specific instances of LSTM model predictions or business process execution.

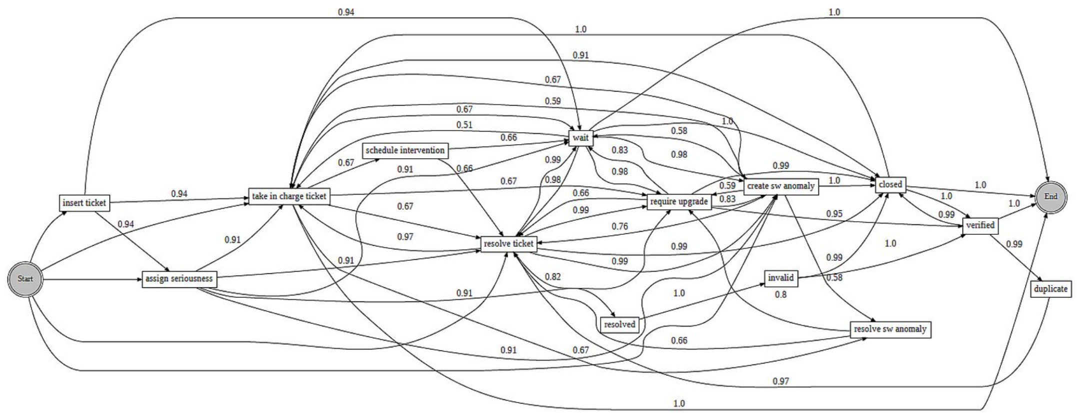


FIGURE 3.3: Graph demonstrating the decision-making process of the unidirectional LSTM model (Model 1) when generating likely event sequences representing a business process based on the training set of the Helpdesk log with the probability threshold set to 0. The graph nodes represent activities, while the edges represent transitions between the activities. The numbers on the edges represent the probabilities of the transitions as predicted by the LSTM model.

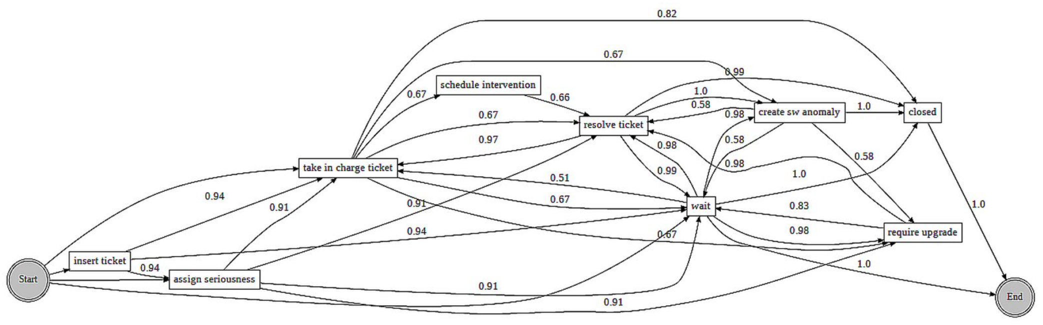


FIGURE 3.4: Graph demonstrating the decision-making process of the unidirectional LSTM model (Model 1) when generating likely event sequences representing a business process based on the Helpdesk log test set with the probability threshold set to 0. The features of the graph are the same as in Figure 3.3.

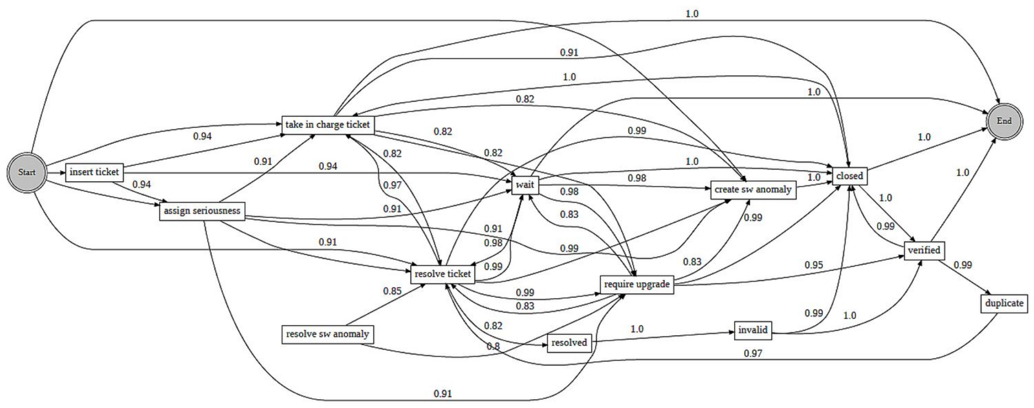


FIGURE 3.5: Graph demonstrating the decision-making process of the unidirectional LSTM model (Model 1) when generating likely event sequences representing a business process based on the test set of the Helpdesk log with the probability threshold set to 0.8. The features of the graph are the same as in Figure 3.3.

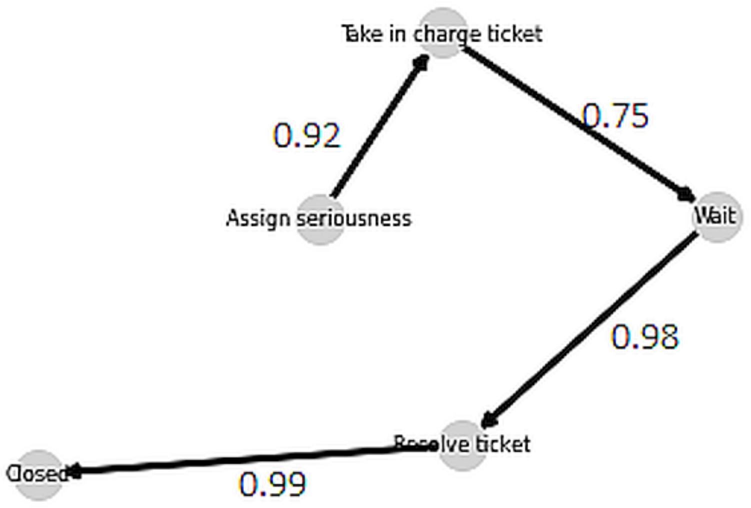


FIGURE 3.6: Single trace process graph generated based on the decisions of the unidirectional LSTM model (Model 1) for the test set of the Helpdesk log. The features of the graph are the same as in Figure 3.3

TABLE 3.4: **Case 1:** Training matrix created from the graph generated using the *train set* of the Helpdesk log. The rows and columns represent activities. The values on the intersections of columns and rows represent the transitions between the corresponding activities, with 1 indicating the presence of the transition and 0 indicating its absence.

0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	1	0	1	1
0	1	0	0	1	0	1	0	1	0	1	1
0	0	0	1	0	0	1	0	1	0	1	1
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1
0	0	0	1	1	0	1	0	0	1	1	0

TABLE 3.5: **Case 1:** Test matrix created from the graph generated using the *test set* of the Helpdesk log. The rows and columns represent activities. The values on the intersections of columns and rows represent the transitions between the corresponding activities, with 1 indicating the presence of the transition and 0 indicating its absence.

0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	1	0	1	1
1	1	0	0	1	0	1	0	1	0	1	1
0	0	0	1	0	0	1	0	1	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	1
0	0	0	1	1	0	1	0	1	1	0	0

TABLE 3.6: **Case 2:** Training matrix created from the graph generated using the *train set* of the Helpdesk log. The rows and columns represent activities. The values on the intersections of columns and rows represent the probabilities of the transitions between the corresponding activities.

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.79	0.86
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.69
0.0	0.0	0.0	0.92	0.92	0.0	0.0	0.0	0.92	0.0	0.92	0.92
0.0	0.73	0.0	0.0	0.73	0.0	0.81	0.0	0.73	0.0	0.73	0.73
0.0	0.0	0.0	0.54	0.0	0.0	1.0	0.0	0.99	0.0	0.99	0.99
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.97	0.97	0.97	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.72	0.0	0.0	0.0	0.72	0.0	0.0	0.0	0.0	0.0	0.72	0.72
0.0	0.0	0.0	0.0	0.0	0.82	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.85	0.0	0.0	0.0	0.0	0.0	0.0	0.87
0.0	0.0	0.0	0.97	0.97	0.0	0.97	0.0	0.0	0.99	0.0	0.0

TABLE 3.7: **Case 2:** Test matrix created from the graph generated using the *test set* of the Helpdesk log. The rows and columns represent activities. The values on the intersections of columns and rows represent the probabilities of the transitions between the corresponding activities.

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.79	0.89
0.0	0.0	0.0	0.0	0.69	0.0	0.0	0.0	0.0	0.0	0.0	0.69
0.0	0.0	0.0	0.92	0.92	0.0	0.0	0.0	0.92	0.0	0.92	0.92
0.73	0.73	0.0	0.0	0.73	0.0	0.84	0.0	0.73	0.0	0.73	0.73
0.0	0.0	0.0	0.54	0.0	0.0	1.0	0.0	0.99	0.0	0.99	0.99
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.97	0.97	0.97	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.48	0.0	0.0	0.48	0.72	0.0	1.0	0.0	0.0	0.0	0.72	0.54
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.85	0.0	0.99	0.0	0.85	0.0	0.0	0.97
0.0	0.0	0.0	0.97	0.97	0.0	0.97	0.0	1.0	0.99	0.0	0.0

3.4 Summary

This chapter presented a new approach to PM by combining the benefits of widely used graph-based methods for process discovery and DL methods for predicting event sequences. The proposed approach consists of two stages: building an accurate LSTM model to predict business process event sequences based on event logs and generating a DFG explaining the decision-making process of the LSTM model when predicting business process event sequences. Two model architectures were proposed, one using a unidirectional LSTM and another using a bidirectional LSTM. Both outperformed the state-of-the-art LSTM models for PM based on two real-life event logs. Additionally, three real-life logs were used to demonstrate the advantages and limitations of the proposed LSTM models. The bidirectional LSTM model achieved slightly better results than the unidirectional LSTM on more complex or larger logs, but at the cost of training time. The capability of generated graphs to explain the LSTM models was demonstrated visually. An approach to determine the similarity between the graphs was proposed to further validate the generalising ability of the model, which gave a satisfactory result.

The contribution of this study is two-fold. First, two LSTM models were proposed that achieve higher accuracy when predicting the next activities in business processes than existing LSTM models. The better performance of the proposed models can be attributed to a different way of pre-processing event logs to generate inputs for the models and the model network architectures that employ an embedding and a dense layer, in addition to an LSTM layer (unidirectional and bidirectional for the first and second models, respectively).

Second, a way of generating graphs representing each model's decision-making process was proposed. These graphs can be used to explore the performance of the LSTM model and identify complex cases for the model to deal with so that measures can be taken to improve the model performance in such cases. Furthermore, the graphs can be used for various PM purposes such as business process discovery and conformance checking as an alternative to the existing graph-based methods that learn process models directly from data rather than through a machine learning model. The advantage of the proposed approach, in this case, is the ability of the generated graph to explain the decision process of the accurate LSTM to a degree of generality set by the user through a probability threshold. While we demonstrate the predictive performance of the proposed LSTM models through experiments employing real-life logs, the ability of the resulting graphs to explain the LSTM models is shown visually. Another advantage of the proposed approach is its model-agnostic nature. This means that the graph generation part is independent of the predictive modelling part; any other machine learning model (or DL architecture) can be used instead of LSTM.

In the next chapter (Chapter 4), we build upon the methods presented in this chapter to propose two new methods, called PGraphDD-QM and PGraphDD-SS (where 'P' stands for *process*, 'DD' for *drift detection*, 'QM' for *quality metrics* and 'SS' for *similarity score*) for detecting sudden concept drifts in business processes from a control-flow perspective. Since the Bi-directional LSTM model (Model 2) improves accuracy slightly compared to the uni-directional LSTM model (Model 1), the latter is used in implementing our future methods because it is both simple, accurate and has a faster training time.

Chapter 4

Concept Drift Detection

Chapter 2 highlighted some benefits of detecting drifts in business processes and how early detection of drifts can enable organisations to take proactive measures and avoid adverse effects resulting from random changes in the business process behaviour. The chapter also mentioned some limitations of the state-of-the-art drift detection methods, such as not detecting drift in online settings and detecting drifts with a long delay because some approaches require waiting for the trace to complete. Furthermore, the majority of existing methods rely on statistical tests over trace distributions, which means that the methods may have insufficient data samples when there is high variability in a log. Such a statistical approach also affects the performance of the methods when logs exhibit many distinct traces over the total number of traces, which can be frequently observed, for example, in healthcare processes.

Chapter 3 presented a new approach to PM by combining the benefits of widely used graph-based methods for process discovery and DL methods for predicting event sequences. This chapter builds on the methods presented in Chapter 3 to propose two new methods, called PGraphDD-QM and PGraphDD-SS (where 'P' stands for *process*, 'DD' for *drift detection*, 'QM' for *quality metrics* and 'SS' for *similarity score*) for detecting sudden concept drifts in business processes from a control-flow perspective. Unlike existing methods that perform statistical analysis on features extracted from event logs, the proposed methods represent business processes as graphs. The methods are used to detect drift in both the offline scenario, where historical data are available to analyse drifts that have already happened (i.e. considering event logs), and the online scenario, where data are coming in real time (i.e. considering event streams).

This chapter is structured as follows. Section 4.1 introduces the proposed concept drift detection approach. Section 4.2 presents the proposed methods for offline and online drift detection. Section 4.3 describes the experiments performed to evaluate the proposed methods for offline and online drift detection, while section 4.4 presents and discusses the results obtained. Finally, Section 4.5 summarises the chapter. Part of the work described in this chapter has been published in (Hanga, Kovalchuk, and Gaber, 2022).

4.1 Approach to Concept Drift Detection

This chapter presents two new methods, PGraphDD-QM and PGraphDD-SS, for addressing concept drift detection, and the purpose is to decide whether the new or recently observed process behaviour shows significant changes compared to the previously observed process behaviour. The proposed drift detection methods are based on the approach introduced in Chapter 3. According to this approach, a unidirectional LSTM model is first trained on an event log, as detailed in Section

4.1.1. The trained model is then used to find the probabilities that each event present in the log will appear in the business process next. Finally, these probabilities are used to generate a DFG, as detailed in Section 4.1.2, representing the likely business process model as believed by the LSTM model.

The solutions for both drift detection methods depend on properly representing the process behaviour in consecutive periods. For offline drift detection, the incoming event traces are represented as *process graphs* generated based on two event logs covering different time periods. To detect drifts, a uni-directional LSTM model trained on an event log covering one period of time is applied to another event log covering a different time period. Process graphs representing the behaviour of the two different time periods are generated using the decisions of the LSTM model about the most probable business process flow.

For online drift detection, the incoming event traces received continuously over time are represented as *graph streams*. To detect drifts, an LSTM model trained on a stream of logged events that covered a previous period of time is applied to a newly generated stream of events as they occur. Graph streams representing the process behaviour of different time periods (i.e., the previous and new time periods) are generated using the decisions of the LSTM model about the most probable business process flow.

According to PGraphDD-QM, the model performance, in both offline and online scenarios, is then separately estimated over the previous and new process/stream graphs using the F-score metric, and the two sets of measures are compared. The change in values of the two sets of measures is assumed to indicate a concept drift. According to PGraphDD-SS, the DFGs generated based on the LSTM model decisions for two different time periods are used to verify the drift, both visually by detecting structural changes and by measuring the similarity score between the adjacency matrices of the two different graphs to estimate the amount of changes observed in the business process after the drift has occurred.

4.1.1 Long Short-term Memory for Predicting Next Activities

According to the approach proposed in Chapter 3, an LSTM model is trained to establish the most likely activity to come next in a given sequence of events over time. The model's training process was improved by broadening the context and phrasing the problem so that multiple previous time steps are considered when predicting the next time-step. Specifically, event logs are pre-processed according to the following protocol and definitions.

The label *NULL* is used to mark the start of each case (or process instance). It becomes the first input activity x_0 at the current time t_0, and the target activity y_0 becomes the activity at time t_1 (this is the first activity occurring in each case of the event log). The next sequence of input activity x_1 becomes the activity at the previous times $\{t_0, t_1\}$, and the target y_1 becomes the next activity at time t_2. The next input activity sequence x_2 becomes the activity at the prior times $\{t_0, t_1, t_2\}$, and the target y_2 becomes the next activity at time t_3; and so on. The inputs X build up for each next input sequence as the previous activities join the current activity. Targets Y are always activities in the next time step t_{n+1} until the last input sequence contains all activities in the previous time steps, including the current activity. At this point, the *END* label is added to mark the end of a case. This procedure is repeated until all cases in the event log are pre-processed.

The input sequences are encoded using *Tokenizer class* from the Keras library (Gulli and Pal, 2017). The tokenizer maps each activity in an event log to a unique integer creating a sequence of integers.The prepared sequences are padded to the left using the *pad sequences()* function from Keras. This function finds the longest sequence and uses its length as a standard to pad other sequences for the same length. The targets are dummy-encoded using the pd.get_dummies() function from the Pandas library. The function converts categorical values into dummy numerical values.

Next, a unidirectional LSTM model is defined, compiled, and fitted using the preprocessed event log. The model is composed of an embedding layer (which serves as an interface between the input and LSTM layers of the network), a single LSTM layer, and a fully connected dense layer as the output layer (which uses the softmax activation function to ensure the output takes the form of probability distributions). The trained LSTM model is then used to predict each next activity in the business process. A prediction probability matrix is constructed for the succeeding activity predictions. These probabilities are used to generate a visually explainable process model graph in the form of a DFG representing the decision-making process of the LSTM model on the likely business process.

4.1.2 Directly-follows Graph for Representing LSTM Decisions

A DFG is a directed graph whose nodes represent activities and edges represent directly-follows relations between these activities. Each edge in the DFG is annotated with a directly-follows probability, denoting the LSTM model's next activity prediction.

A DFG is generated on the basis of the probabilities output by the LSTM model. In Chapter 3, a DFG is used to explain the decision-making process of the LSTM model when predicting the subsequent events in a business process. The DFG is constructed by traversing each row in the prediction probability matrix, picking the column with the highest probability, which becomes the most likely next activity, and then creating a transition between each preceding and succeeding activity by drawing an edge between the nodes. The procedure is repeated until all rows in the matrix are visited. The outcome is a process graph, which can be used to analyse the performance of the LSTM model and to identify difficult cases with which the model must deal so that measures can be taken to improve the model performance in such cases. A probability threshold is introduced as a parameter to allow tuning the complexity of the graph, making it possible for the level of detail in the graph to be adjusted. The graphs generated this way can also be used to perform various PM tasks such as model discovery, conformance checking, and investigating cases of non-compliance. This chapter extends the work in Chapter 3 by demonstrating how the graphs constructed based on LSTM predictions can also be used to detect concept drifts in business processes.

4.2 Proposed Methods: PGraphDD-QM and PGraphDD-SS

This section presents some preliminaries and discusses the steps involved in offline and online drift detection methods. The proposed methods are called PGraphDD-QM and PGraphDD-SS (where 'P' stands for *process*, 'DD' for *drift detection*, 'QM' for *quality metrics* and 'SS' for *similarity score*).

4.2.1 Offline scenario: Detecting Drift in *Event Logs*

An event log is a set of traces, each capturing the sequence of events observed for a given case or process instance, ordered by timestamp. For example, the event log

$L = [\sigma_1, \sigma_2, \sigma_1, \sigma_1, \sigma_2, \sigma_1]$, where $\sigma_1 = \langle a, b, c, d \rangle$ and $\sigma_2 = \langle a, c, b, d \rangle$, defines a log containing six traces (four occurrences of σ_1 and two of σ_2) and a total of 24 events. Formally:

Definition 1 (Trace, Event log). Let A be a set of activities. $A+$ is the set of all finite non-empty sequences of activities from A. $\sigma \in A+$ is called a trace when σ represents a firing activity sequence of a process model. An event log L is a multi-set of traces from $A+$.

The problem of business concept drift detection can be formulated as locating a point in time when there is a difference between the observed behaviour before and after the point, if any. The basic idea in handling concept drift is that the characteristics of the traces before the change point differ from the characteristics of the traces after the change point.

Definition 2 (Concept drift). Let G be a process graph, and let $G_0, G_1, ..., G_n$ be $n+1$ different process models and $T_0 < T_1 < ... < T_n$ be $n+1$ time periods. $G(T_i) = G_i$ represents the graph used in T_i. $G(T_0) = G_0$ is the initial graph. When the time period $T_i (0 < i \leq n)$ arrives, the current graph will change into G_i instantly, and the traces are still recorded in the same event log. Such a phenomenon is referred to as concept drift, with $T_1, ..., T_n$ being change points.

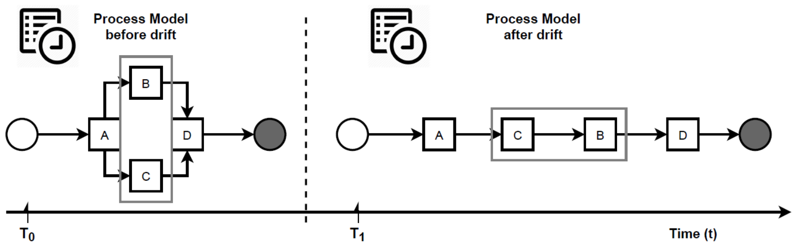
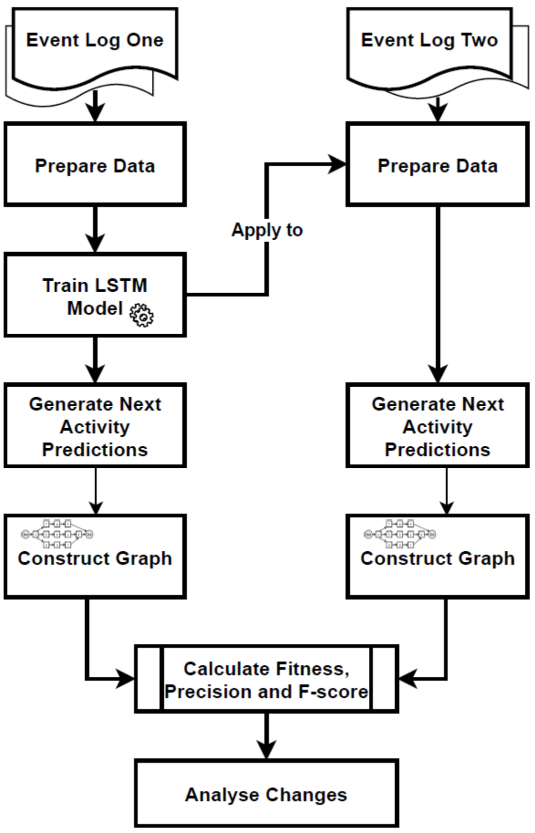
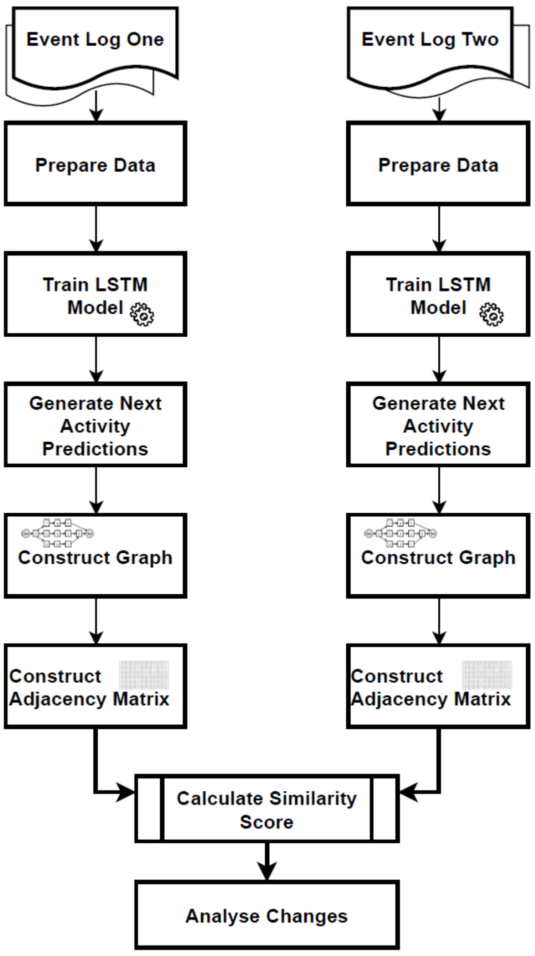


FIGURE 4.1: Concept drift phenomenon in process mining

Given an event log $L = (\sigma_1, \sigma_2, .., \sigma_n)$, the aim is therefore to find out the moments when the changes occurred in the log. From Definition 4.2.1, it can be observed that the model behaviour before a change is not the same as after a change. Consider the example in Figure 4.1. If the traces for T_0 and T_1 are collected, it will be observed that they differ. Therefore, to detect a drift, a natural idea is to compare a number of traces before and after a candidate change point. Bose et al. (Bose et al., 2013), Maaradj et al. (Maaradji et al., 2017), Martjushev et al. (Martjushev, Bose, and Aalst, 2015) all adopted this solution, which is, however, susceptible to two challenges: (i) how to measure the differences between two sets of traces and (ii) how many traces to collect for testing (i.e. deciding the window size). These authors used feature extraction and statistical hypothesis testing as a solution for the first challenge, while for the second challenge, they introduced fixed and adaptive window size strategies. However, the actual performance heavily depends on the choice of the window size, which is not quite satisfactory: a wrong window size can lead to false negatives and false positives, as well as difficulty in locating the exact point of the drift. We propose an entirely different idea to avoid such disadvantages.



(A)


Event Log One
Event Log Two
Prepare Data
Prepare Data
Train LSTM Model
Train LSTM Model
Generate Next Activity Predictions
Generate Next Activity Predictions
Construct Graph
Construct Graph
Construct Adjacency Matrix
Construct Adjacency Matrix
Calculate Similarity Score
Analyse Changes

(B)

FIGURE 4.2: Proposed approach to detecting concept drifts in event logs of business processes: (A) PGraphDD-QM (B) PGraphDD-SS

Figure 4.2 summarises the proposed methods PGraphDD-QM and PGraphDD-SS for detecting concept drifts in business processes in the offline scenario. Algorithm 3 lists the pseudo-code of PGraphDD-QM (Figure 4.3(a)) for offline concept drift detection, which includes the following steps:

1. Divide the event log into two sub-logs representing two time periods (lines $3-4$): reference sub-log L_i and detection sub-log L_{i+1}.

2. Pre-process the sub-logs (line 6). This involves encoding and padding the traces from both the detection and reference sub-logs (L_1 and L_2) as described in Section 4.1.1.

3. Define, compile and train an LSTM model μ on the pre-processed traces from the reference sub-log L_1 (line 7) as described in Section 4.1.1 and in Chapter 3 to obtain the fitted model $\mu\prime$.

4. Apply the LSTM model $\mu\prime$ trained on the traces of reference sub-log L_1 to both reference sub-log L_1 and detection sub-log L_2 (lines 9) to make predictions of the complete trace of each process instance.

5. Construct prediction probability matrices $P_{ij}^{l_1}$ and $P_{ij}^{l_2}$ for reference and detection windows respectively (lines $13-14$), as described in Section 4.1.1 and in Chapter 3 .

6. Generate two DFG process models G^{l_1} and G^{l_2} using $P_{ij}^{l_1}$ and $P_{ij}^{l_2}$ respectively (lines $15-16$), as described in Section 4.1.2 and in Chapter 3.

7. Calculate two sets of performance metrics (fitness, precision and F-score) based on the predictions (lines $18-21$).

8. Compare metrics of the detection window with the threshold ϕ (line 22). The threshold is set based on the F-score values obtained for the reference window. An F-score below a threshold signals a drift presence.

9. Analyse the change by inspecting the detection window (lines $23-34$).

Algorithm 3 PGraphDD-QM: Offline Concept Drift Detection using Quality Measurements

```
Require: Event log: $L$, LSTM model: $\mu$, Threshold: $\phi$
 1: $fScoreLog \leftarrow []$
 2:    {split event logs into sub-logs $L_i$ and $L_{i+1}$}
 3: $L_i, L_{i+1} \leftarrow splitLog(L)$
 4: $L_1 \leftarrow L_i$, $L_2 \leftarrow L_{i+1}$
 5: while $L_1, L_2 \neq 0$ do
 6:    $prepare(L_1, L_2)$
 7:       {compile and train an lstm model on the $L_1$ sub-log}
 8:    $\mu'_1 \leftarrow (\mu, L_1)$
 9:       {Apply the trained model to both $L_1$ and $L_2$}
10:    $l_{1pred} \leftarrow predict(\mu', L_1)$
11:    $l_{2pred} \leftarrow predict(\mu', L_2)$
12:       {build next event prediction probability matrices, construct graphs}
13:    $P_{ij}^{l_1} \leftarrow generate(Pred\_Prob\_Mat, l_{1pred})$
14:    $P_{ij}^{l_2} \leftarrow generate(Pred\_Prob\_Mat, l_{2pred})$
15:    $G^{l_1} \leftarrow drawGraph(i, o, N, E, P_{ij}^{l_1}, l_{1pred})$
16:    $G^{l_2} \leftarrow drawGraph(i, o, N, E, P_{ij}^{l_2}, l_{2pred})$
17:       {calculate the quality metrics}
18:    $p \leftarrow performance()$
19:    $fitness \leftarrow p.fitness()$
20:    $precision \leftarrow p.precision()$
21:    $fScore \leftarrow p.fScore(fitness, precision)$
22:    if $fScore \leq \phi$ then
23:       $Drift\ detectected\ in\ L_2$
24:          {change point detected and reported}
25:    end if
26: end while
```

Algorithm 4 PGraphDD-SS: Offline Concept Drift Detection using Similarity Score

```
Require: Event log: $L$, LSTM model: $\mu$, Threshold: $\phi$
 1: $simScore \leftarrow []$
 2:     {split event logs into sub-logs $L_i$ and $L_{i+1}$}
 3: $L_i, L_{i+1} \leftarrow splitLog(L)$
 4: $L_1 \leftarrow L_i$, $L_2 \leftarrow L_{i+1}$
 5: while $L_1, L_2 \neq 0$ do
 6:     $prepare(L_1, L_2)$
 7:         {compile and train two lstm model on the two sub-logs $L_1$ and $L_2$}
 8:     $\mu'_1 \leftarrow (\mu, L_1)$, $\mu'_2 \leftarrow (\mu, L_2)$
 9:         {Make predictions for both}
10:     $l_{1pred} \leftarrow predict(\mu'_1, L_1)$
11:     $l_{2pred} \leftarrow predict(\mu'_2, L_2)$
12:         {build next event prediction probability matrices, construct graphs}
13:     $P^{l_1}_{ij} \leftarrow generate(Pred\_Prob\_Mat, l_{1pred})$
14:     $P^{l_2}_{ij} \leftarrow generate(Pred\_Prob\_Mat, l_{2pred})$
15:     $G^{l_1} \leftarrow drawGraph(i, o, N, E, P^{l_1}_{ij}, l_{1pred})$
16:     $G^{l_2} \leftarrow drawGraph(i, o, N, E, P^{l_2}_{ij}, l_{2pred})$
17:         {generate adjacency matrices, measure similarity score}
18:     $A^{l_1}_{ij} \leftarrow generate(adjacency\_matrix, G^{l_1})$
19:     $A^{l_2}_{ij} \leftarrow generate(adjacency\_matrix, G^{l_2})$
20:     $p \leftarrow performance()$
21:     $simScore \leftarrow p.getScore()$
22:     $M \leftarrow 0$ {set a counter of non-zero values}
23:     $M \leftarrow getM(A^{l_1}_{ij}, A^{l_2}_{ij})$
24:     $absoluteVal \leftarrow (A^{l_1}_{ij}, A^{l_2}_{ij})$
25:     $sumVal \leftarrow absoluteVal.abs().sum().sum()$
26:     $simScore \leftarrow 1 - sumVal/M$
27:     if $simScore \leq \phi$ then
28:         $Drift\ \ detectected\ \ in\ \ L_2$
29:             {change point detected and reported}
30:     end if
31: end while
```

Quality Metrics

The quality dimensions considered are *Fitness, Precision and F-score*:

Fitness is a model's ability to reproduce the behaviour contained in the log. This study uses the fitness measure proposed in (Augusto et al., 2019). It indicates the degree to which each trace in the log can be aligned with a corresponding trace produced by the process model (DFG in our case). A fitness score of 1 means that the model (DFG) can reproduce all traces in the log.

Definition 3: Fitness. A Fitness or recall measure $fitness \in L \times M \rightarrow [0,1]$ aims to quantify the fraction of observed behaviour that the model allows. Let $l \in L$ and $m \in M$ be an event log and a process model, and T be a set of traces. Then,

$$Fitness = \frac{|T(l) \cap T(m)|}{|T(l)|}. \tag{4.1}$$

Precision is a model's ability to generate only the behaviour found in the log. This study uses the precision measure proposed in Augusto et al., 2019. A precision score of 1 indicates that any trace produced by the process model (DFG in our case) is contained in the log.

Definition 4: Precision. A precision measure $precision \in L \times M \rightarrow [0,1]$ quantifies the fraction of behaviour allowed by the model that was actually observed. Let $l \in L$ and $m \in M$ be an event log and a process model, and T be a set of traces. Then:

$$Precision = \frac{|T(l) \cap T(m)|}{|T(m)|}. \tag{4.2}$$

F-score is a single measure of accuracy; it is the harmonic mean of fitness and precision, calculated as

$$\text{F-score} = \frac{2 \times \text{Fitness} \times \text{Precision}}{\text{Fitness} + \text{Precision}}.$$

To complement PGraphDD-QM, PGraphDD-SS (Figure 4.2(b)) compares the structure of the DFGs generated based on the prediction output of two LSTM models, one trained on traces from the reference window and the other trained on traces from the detection window covering two different periods. Algorithm 4 lists the pseudo-code of PGraphDD-SS, which includes the following steps:

1. First, divide the event log into two sub-logs representing two time periods (lines $3-4$): reference sub-log L_i and detection sub-log L_{i+1}.

2. Pre-process the sub-logs (line 6). This involves encoding and padding the traces from both the detection and reference sub-logs (L_1 and L_2) as described in Section 4.1.1.

3. Define and compile an LSTM model μ and train it first on preprocessed traces of reference sublog L_1 (line 8) as described in Section 4.1.1 and in Chapter 3 to obtain the fitted model $\mu_1\prime$ and then on the pre-processed traces of detection sub-log L_2 to obtain the fitted model $\mu_2\prime$.

4. Use each of the trained LSTM models to predict the complete trace of each process instance of the sub-log it has been trained on (lines $10-11$).

5. Construct prediction probability matrices $P_{ij}^{l_1}$ and $P_{ij}^{l_2}$ for the reference and detection windows, respectively (lines $13-14$), as described in Section 4.1.1 and in Chapter 3.

6. Generate two DFG process models G^{l_1} and G^{l_2} using the $P_{ij}^{l_1}$ and $P_{ij}^{l_2}$, respectively (lines $15-16$), as described in Section 4.1.2 and in Chapter 3.

7. Build two adjacency matrices $A_{ij}^{l_1}$ and $A_{ij}^{l_2}$ for each of the constructed DFG process models, respectively (lines $18-19$).

   Definition 5: Adjacency matrix. Let $\mathcal{G}$ be a DFG with a vertex set $V = v_1, ..., v_n$. G can be transformed into an adjacency matrix A, where A is a square $n \times n$ matrix, such that its element $A_{ij} = 1$ when there is an edge from vertex v_i to vertex v_j, and $A_{ij} = 0$ when there is no edge.

8. Calculate the similarity score $simScore$ using the adjacency matrices of the two DFG process models generated (lines $20-26$).

9. Compare the similarity score with the threshold ϕ. The threshold is set based on the highest similarity score values obtained. A similarity score below the set threshold signals a drift presence (line 27).

10. Analyse the change by inspecting the detection window (lines $28 - 29$).

Measuring the Similarity Score

In addition to visually comparing the similarity between the two graphs representing two snapshots of a business process taken over different windows, the similarity is also verified formally in PGraphDD-SS. The two graphs are first converted into two adjacency matrices to achieve this. Then, the similarity score between the matrices is calculated by taking the sum of the differences between the values in corresponding cells of the two matrices and dividing it by the number of non-zero values in the matrix with the least number of activities. If the two graphs, namely, the first graph representing the reference window and the second graph representing the detection window, happen to differ in size (i.e. one has more transitions than the other), the transitions present in one window but absent in the other are introduced when constructing the adjacency matrix of the latter to enable the calculation of the similarity score. These additional transitions are initialised to zeros since they have not occurred in reality. A similarity score of 1 indicates that the two graphs are identical (i.e. there is no drift). In contrast, a similarity score of 0 indicates that they are entirely different (i.e., represent different business processes). A similarity score between 0 and 1 indicates the presence of drift. For the calculation of the similarity score, binary matrices are constructed. A value of 1 in a cell indicates that there is a transition between the corresponding two activities. On the contrary, a value of 0 represents the fact that there is no transition between the two activities.

4.2.2 Online scenario: Detecting Drift in *Event Streams*.

An event stream is an indefinite sequence of events, where each event represents an occurrence of interest at a certain period of time.

Definition 6: Trace, event stream. Let A be a set of activities and $A+$ be a set of all non-empty finite sequences of activities from A. $\sigma \in \mathcal{A}+$ is called a trace when σ represents a firing activity sequence of a process model. An event stream $\mathcal{S}$ is a multi-set of infinite event traces from $\mathcal{A}+$.

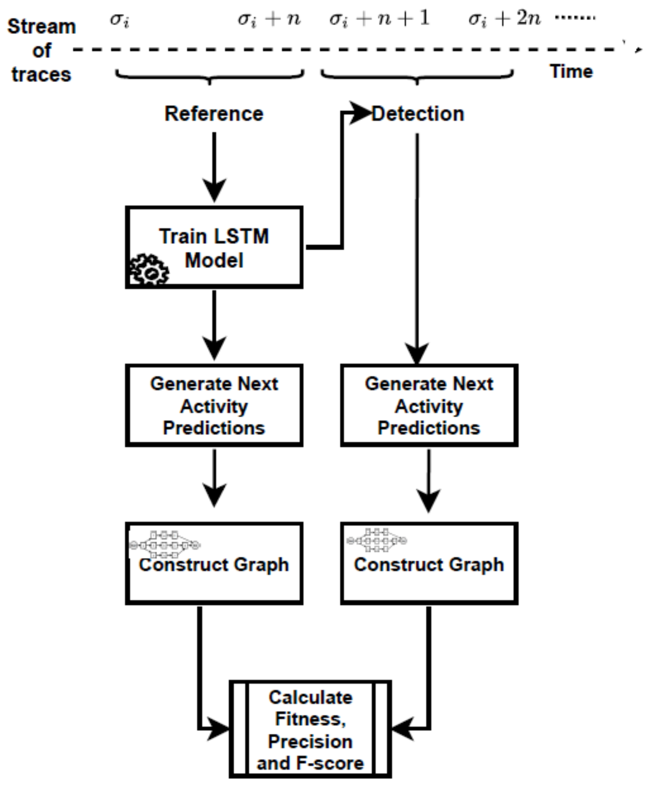
In PM, a business process can be represented as a graph built using an event stream, with nodes representing activities and edges representing the transitions between the activities. The dynamism of a business process (i.e. changes in the activities and transitions between them) can be represented as a graph stream.

Definition 7: Graph Stream. A graph stream G_s is a sequence of elements $e = (x, y; t)$, where x and y are node labels, and edge (x, y) occurs in a time period t. A stream $G_s = \langle e_1, e_2, ..., e_m \rangle$ typically defines a graph $G = (V, E)$, where V denotes a set of nodes (or vertices) and E denotes a set of edges.

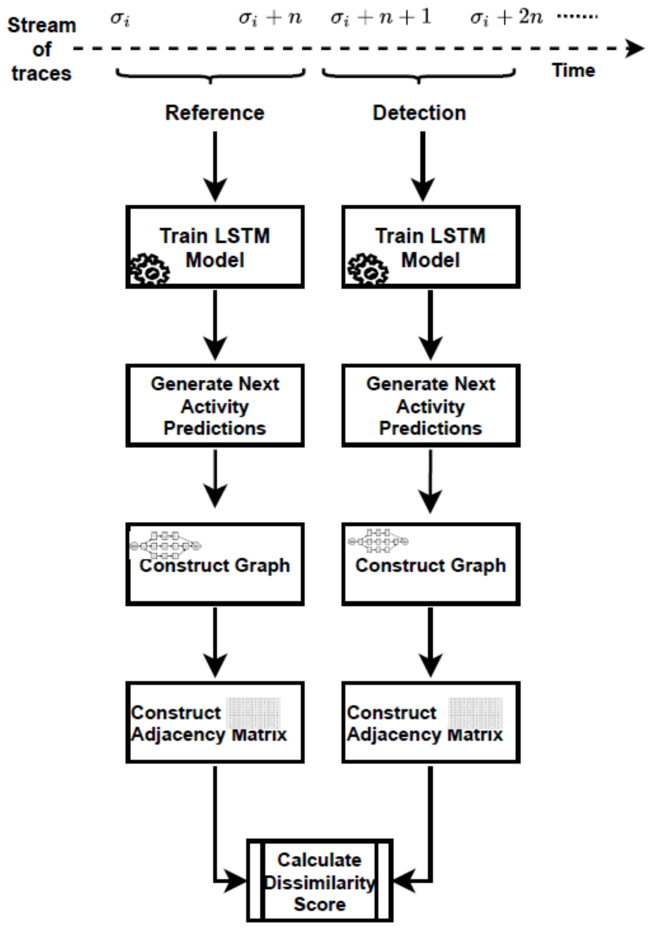
The problem of detecting a concept drift in a graph stream can be formulated as locating a point when there is a difference between the observed behaviour before and after the point. The basic idea behind detecting concept drifts is that the characteristics of the graph stream before the change point differ from those of the graph stream after the change point.

Definition 8: Concept drift in Graph Stream. Let $G_s = (\sigma_1, \sigma_2, .., \sigma_n)$ be a graph stream, $S_0, S_1, ...\infty$ be an indefinite number of different graph streams and $T_0 < T_1 < ... < \infty$ be an indefinite number of time periods. $G_s(T_i) = G_{s_i}$ represents the graph used in T_i. $S(T_0) = S_0$ is the initial graph. When a time period $T_i(0 < i \leq \infty)$ arrives, the current graph will change into G_{s_i} instantly, and the traces will continue to be updated in the same event stream. This phenomenon is called *concept drift*. $T_1, ..., T_\infty$ are called change points. Figure 4.1 illustrates a concept drift that occurred at the change point T_1.

Given a graph stream $G_s = (\sigma_1, \sigma_2, .., \sigma_n)$, the aim is thus to detect the moments when changes occur in the stream.



(A)



(B)

FIGURE 4.3: Proposed approach to detecting concept drifts in event streams of business processes: (A) PGraphDD-QM (B) PGraphDD-SS

Figure 4.3 summarises the proposed methods to detect concept drifts in graph streams that represent changing business processes. According to the first method (PGraphDD-QM) illustrated in Figure 4.3(a), an LSTM model is first trained using reference traces (i.e., traces from the previous time period or stream). This LSTM model is then applied to a newly generated stream of traces. The model's performance over the old and new streams is compared in terms of the F-score metric, which is the harmonic mean of fitness and precision. Fitness or recall is the ability of the model to reproduce the behaviour contained in the stream of traces. Precision is the ability of the model to generate only the behaviour found in the stream of traces. Intuitively, if a change were introduced to a process, an LSTM model trained on traces representing the old process (i.e., process before the change) would perform poorly on traces representing the new process (i.e., process after the change). To quantify this performance deterioration, a threshold is introduced. The value of the F-score performance metric on the newly generated stream of events below this threshold indicates the presence of drift. In contrast, those above the threshold indicate that there is no drift. Algorithm 5 lists the pseudo-code of PGraphDD-QM, which includes the following steps:

1. Split the recently observed stream of traces into two windows (lines $6-13$): the detection window $\mathcal{D}$ (most recent traces) and the reference window $\mathcal{R}$ (older traces).

2. Preprocess the traces (i.e., encode and pad) the traces from both the detection and reference windows as described in Section 4.1.1 (line 13).

3. Define and compile an LSTM model μ and train it on the preprocessed traces of the reference window $\mathcal{R}$ (line 14) as described in Section 4.1.1 and in Chapter 3 to obtain a fitted model $\mu\prime$.

4. Apply the LSTM model $\mu\prime$ trained on the traces from the reference window $\mathcal{R}$ to both the reference window $\mathcal{R}$ and the detection window $\mathcal{D}$ to make predictions of the complete trace of each process instance (lines $16-17$).

5. Construct prediction probability matrices P_{ij}^{R} and P_{ij}^{D} for reference and detection windows respectively (lines $19-20$), as described in Section 4.1.1 and in Chapter 3.

6. Generate two DFG process models $\mathcal{G}^{R}$ and $\mathcal{G}^{D}$ using P_{ij}^{R} and P_{ij}^{D} respectively (lines $21-22$), as described in Section 4.1.2 and in Chapter 3.

7. Calculate two sets of performance metrics (fitness, precision and F-score) based on the predictions (lines $24-27$).

8. Compare the F-score value of the detection window with the threshold ϕ. The threshold is set based on the F-score values obtained for the reference window. An F-score below a threshold signals a drift presence (line 28).

9. Analyse and report concept drift by inspecting the detection window (lines $29-30$).

10. Repeat the process for each new run read from the stream by sliding both the reference and detection windows to the right until the end of the stream is reached.

Algorithm 5 PGraphDD-QM: Online Concept Drift Detection using Quality Metrics

```
Require: Event stream: $\mathcal{S}$, LSTM model: $\mu$, Threshold: $\phi$
 1: $fScoreLog \leftarrow []$ , $DriftLog \leftarrow []$
 2:   {read event stream, split windows, prepare data, train LSTM model and make
      predictions}
 3: $e \leftarrow fetch(eventStream)$
 4: $w \leftarrow WinSize$ {the window size}
 5: $parts \leftarrow splitLog(e, w)$
 6: for all $i$ in range(0, parts) do
 7:   $j \leftarrow i + 1$
 8:   if $i = parts - 1$ then
 9:     $j \leftarrow 0$
10:     $\mathcal{R} \leftarrow part(i)$ , $\mathcal{D} \leftarrow part(j)$
11:   end if
12:   while $w \neq 0$ do
13:     $prepare(\mathcal{R}, \mathcal{D})$
14:     $\mu' \leftarrow (\mu, \mathcal{R})$     {train an LSTM model on the reference window}
15:       {Apply the trained model to both reference and detection windows}
16:     $R_{pred} \leftarrow predict(\mu', \mathcal{R})$
17:     $D_{pred} \leftarrow predict(\mu', \mathcal{D})$
18:       {build next event prediction probability matrices, construct graphs}
19:     $P^R_{ij} \leftarrow generate(Pred\_Prob\_Mat, R_{pred})$
20:     $P^D_{ij} \leftarrow generate(Pred\_Prob\_Mat, D_{pred})$
21:     $G^R \leftarrow drawGraph(i, o, N, E, P^R_{ij}, R_{pred})$
22:     $G^D \leftarrow drawGraph(i, o, N, E, P^D_{ij}, D_{pred})$
23:       {calculate the quality metrics}
24:     $p \leftarrow performance()$
25:     $fitness \leftarrow p.fitness()$
26:     $precision \leftarrow p.precision()$
27:     $fScore \leftarrow p.fScore(fitness, precision)$
28:     if $fScore \leq \phi$ then
29:       $print("drift\ found\ in\ window" + str(j))$
30:       $Report \leftarrow (i, j, fScore)$
31:     end if
32:   end while
33: end for
```

Algorithm 6 PGraphDD-SS: online concept drift detection using similarity score

```
Require: Event stream: $\mathcal{S}$, LSTM model: $\mu$, Threshold: $\phi$
 1: $simScore \leftarrow [], DriftLog \leftarrow []$
 2: $e \leftarrow fetch(eventStream)$
 3: $w \leftarrow winSize$ {the window size}
 4: $parts \leftarrow splitLog(e, w)$
 5: for all $i$ in range(0, parts) do
 6:   $j \leftarrow i + 1$
 7:   if $i = parts - 1$ then
 8:     $j \leftarrow 0$
 9:     $\mathcal{R} \leftarrow part(i), \mathcal{D} \leftarrow part(j)$
10:   end if
11:   while $w \neq 0$ do
12:     $prepare(\mathcal{R}, \mathcal{D})$
13:       {compile and train two lstm model on the reference and detection window}
14:     $\mu'_1 \leftarrow (\mu, \mathcal{R}), \mu'_2 \leftarrow (\mu, \mathcal{D})$
15:       {Make predictions for both}
16:     $R_{pred} \leftarrow predict(\mu'_1, \mathcal{R})$
17:     $D_{pred} \leftarrow predict(\mu'_2, \mathcal{D})$
18:       {build next event prediction probability matrices, construct graphs}
19:     $P^R_{ij} \leftarrow generate(Pred\_Prob\_Mat, R_{pred})$
20:     $P^D_{ij} \leftarrow generate(Pred\_Prob\_Mat, D_{pred})$
21:     $G^R \leftarrow drawGraph(i, o, N, E, P^R_{ij}, R_{pred})$
22:     $G^D \leftarrow drawGraph(i, o, N, E, P^D_{ij}, D_{pred})$
23:       {generate adjacency matrices, measure similarity score}
24:     $A^R_{ij} \leftarrow generate(adjacency\_matrix, G^R)$
25:     $A^D_{ij} \leftarrow generate(adjacency\_matrix, G^D)$
26:     $p \leftarrow performance()$
27:     $simScore \leftarrow p.getScore()$
28:     $M \leftarrow 0$ {set a counter of non-zero values}
29:     $M \leftarrow getM(A^R_{ij}, A^D_{ij})$
30:     $absoluteVal \leftarrow (A^R_{ij} - A^D_{ij})$
31:     $sumVal \leftarrow absoluteVal.abs().sum().sum()$
32:     $simScore \leftarrow 1 - sumVal/M$
33:     if $simScore \leq \phi$ then
34:       $print(''drift\ found\ in\ window'' + str(j))$
35:         {change point detected and reported}
36:     end if
37:     $Report \leftarrow (i, j, simScore)$
38:   end while
39: end for
```

To complement PGraphDD-QM, PGraphDD-SS (Figure 4.3(b)) compares the structure of the DFGs generated based on the prediction output of two LSTM models, one trained on traces from the reference window and the other trained on traces from the detection window covering two different periods (i.e., reference and detection). Algorithm 6 lists the pseudo-code of PGraphDD-SS, which includes the following steps:

1. Split the recently observed stream of traces into two windows (line $5 - 10$): the detection window $\mathcal{D}$ (most recent traces) and the reference window $\mathcal{R}$ (older traces).

2. Pre-process (i.e., encode and pad) the traces from both the detection and reference windows as described in Section 4.1.1 (line 12).

3. Define and compile an LSTM model μ and train it first on the pre-processed traces from the reference window $\mathcal{R}$ (line 14) as described in Section 4.1.1 and in Chapter 3 to obtain the fitted model $\mu_1\prime$ (line 15) and then on the pre-processed traces from the detection window $\mathcal{D}$ to obtain the fitted model $\mu_2\prime$.

4. Use each of the trained LSTM models to predict the complete trace of each process instance of the window it has been trained on (lines $16 - 17$).

5. Construct prediction probability matrices for each of the two windows, respectively (lines $19 - 20$), as described in Section 4.1.1 and in Chapter 3.

6. Generate two DFG process models (lines $21 - 22$) as described in Section 4.1.2 and in Chapter 3 using the respective prediction probability matrices of each window.

7. Build adjacency matrices for each of the DFG process models constructed, respectively (lines $24 - 25$).

   Definition 9: Adjacency matrix. Let $\mathcal{G}$ be a DFG with a vertex set $V = v_1, ..., v_n$. G can be transformed into an adjacency matrix A, where A is a square $n \times n$ matrix, such that its element $A_{ij} = 1$ when there is an edge from the vertex v_i to the vertex v_j and $A_{ij} = 0$ when there is no edge.

8. Calculate the similarity score of the two DFG process models using the adjacency matrices generated (lines $26 - 32$).

9. Compare the similarity score with the threshold ϕ. The threshold is set based on the highest similarity score values obtained. A similarity score below the set threshold signals the presence of drift (line 33).

10. Analyse and report concept drift by inspecting the detection window (lines $34 - 35$).

11. Repeat the process for each new run read from the stream by sliding both reference and detection windows to the right until the end of the stream is reached.

4.3 Experiments

This section presents the datasets used for the experiments and the experimental setup to evaluate the two methods in both offline and online scenarios. The experiments were carried out using the Google Colab free Tesla K80 GPU. All stages depicted in Figures 4.2 and 4.3 were implemented as a set of Python scripts using Python 3.6. LSTM models were built using the Keras (Gulli and Pal, 2017) and Tensorflow (Abadi et al., 2016) libraries. Process graphs were generated using the Graphviz library (Ellson et al., 2001). The original log files are in XES format. For the purposes of this study, they were converted to CSV files using the ProM tool (Van Dongen et al., 2005).

4.3.1 Offline Drift Detection

Two publicly available real-life event logs were used to evaluate both PGraphDD-SS and PGraphDD-QM: BPIC 2015 (Dongen, 2015) and Helpdesk (Polato, 2020). The BPIC 2015 dataset already includes five variations of the same process. However, for the Helpdesk log, four different modifications (labelled $'Helpdesk\_N'$, where N is the modification number) were artificially created to explore the impact of different process changes on the performance of the proposed methods. All the logs are detailed in the next subsections and summarised in Table 4.1.

TABLE 4.1: Summary of the event logs used in the experiments

Event Log	Numder of Events	Number of Traces	Number of Activities
Helpdesk	21,348	4,580	14
Helpdesk_1	23,938	4,580	15
Helpdesk_2	19,442	4,580	13
Helpdesk_3	20,533	4,582	14
Helpdesk_4	25,027	4,582	14
BPIC2015_1	52,217	1,199	398
BPIC2015_2	44,354	832	410
BPIC2015_3	59,681	1,409	383
BPIC2015_4	47,293	1,053	356
BPIC2015_5	59,083	1,156	389

Helpdesk Logs

The original Helpdesk log (Polato, 2020) contains records of real-life events from a ticketing management process of the help desk of an Italian software company. To simulate the presence of concept drifts, the log is systematically altered by applying simple change patterns to the log in turn (Table 4.2). As a result, the following four additional logs were obtained, each capturing different change operations commonly identified in business process models, namely addition, removal, and re-positioning of events in a process:

1. *Helpdesk_1*: an activity is added. For the first variant of the Helpdesk log, a new activity, 'Emergency', was added to the log.
2. *Helpdesk_2*: an activity is removed. For the second variant of the Helpdesk log, the activity 'Duplicate' was removed from the log. This should result in a graph with one less node and thus zero probabilities in the second adjacency matrix, leading to a drop in the similarity score between the original and modified process.
3. *Helpdesk_3*: change in flow. For the third variant of the Helpdesk log, some changes were applied at the location of a branching node in the process graph. In particular, all traces following *'Assign seriousness → Take in charge ticket → Wait → Resolve ticket → Closed' and 'Assign seriousness → Take in charge ticket → Require upgrade → Resolve ticket → Closed'* were replaced with *'Assign seriousness → Take in charge ticket → Wait → Require upgrade → Resolve ticket → Closed'* (i.e. both 'Wait' and 'Require upgrade' take place before moving to 'Resolve ticket' and eventually 'Closed' in the new variation).

4. *Helpdesk_4*: incorporating all changes. For the fourth variant of the Helpdesk log, all three change patterns (addition, removal, change in the flow) were included in the event log. This case allows testing the sensitivity of the proposed methods to the number of changes in the control flow present in the new log compared to the old one.

TABLE 4.2: Change patterns applied to the original Helpdesk log.

Event Log	**Change Pattern**
Helpdesk_1	Add activity
Helpdesk_2	Remove activity
Helpdesk_3	Alter a parallel branch
Helpdesk_4	Incorporate all three change patterns

BPIC 2015 logs

BPIC 2015 is a real-life dataset that contains cases of building permit applications provided by a Dutch municipality. This log collection was originally used in the Business Process Intelligence Contest (BPIC 2015) (Dongen, 2015). Five log files are available in the collection, each provided by one of the five Dutch municipalities. The logs contain many different activities, each labelled with a code and a Dutch and English label. Although the processes in the five municipalities should be identical, they differ in reality. A large behaviour in each municipality is not observed in the other municipalities. There are distinctions in sub-processes between the municipalities regarding the frequency of occurrence and behaviour within the sub-processes. This may have resulted from the changes to procedures, rules, or regulations. As there are about 1,170; 828; 1,349; 1,049 and 1,153 different execution paths for the BPIC2015_1; BPIC2015_2; BPIC2015_3; BPIC2015_4 and BPIC2015_5 logs, respectively, almost all cases are unique from the control-flow perspective.

Experimental Setup for Offline Drift Detection

The following experimental protocol was adopted to evaluate the proposed methods for detecting concept drift in business processes in offline scenarios:

1. For PGraphDD-SS, the similarity score between each pair of process graphs covering two different time periods (1st and 2nd graph, 1st and 3rd, 2nd and 4th, 4th and 5th, etc.) was measured.

   The following two cases were considered for calculating the similarity score:

   - *Case 1*: constructing binary matrices, where a value of 1 in a cell corresponds to the fact that the transition between the two corresponding activities exists, whereas a value of 0 represents the fact that there is no transition between the two activities.
   - *Case 2*: constructing matrices in the same way as in Case 1, except taking the probability of transition was recorded as predicted by the LSTM model instead of the value of 1.

   Taking into account the size and complex structure of the process graphs constructed from the BPIC 2015 logs, unlike those constructed from the Helpdesk log, different probability thresholds, namely, $0.0, 0.2, 0.6, 0.8, 1.0$, were used in turn during the experiment. The higher the threshold set, the less complex the process graph constructed, and vice versa. The idea is to allow for ease of analysis and to observe the effect of different thresholds on similarity scores.

2. For PGraphDD-QM, two event logs covering two different time periods were used: one representing an old event log (for Helpdesk, the same base log) and the other a new event log (for Helpdesk, each of the modified versions). An LSTM model was trained on the old log and then applied to both the old and new logs to make the next activity predictions. Two sets of performance metrics (fitness, precision, and F-score) were calculated based on the predictions and compared between the two event logs.

3. For the embedding layer of the LSTM model, the vocabulary size was set to the number of unique activities in the considered log. The number of embedding dimensions was guided by the vocabulary size; hence, it varied across the event logs.

4. The network architecture of the LSTM model included one hidden LSTM layer with a dropout probability of 0.2 set based on preliminary experiments. The dropout technique was used to avoid over-fitting and improve learning by temporarily removing a cell from the network randomly.

5. The number of training epochs was set to 50 for all logs based on preliminary experiments.

6. The parameters of the LSTM model, namely, the number of embedding dimensions, batch size, and LSTM neurons, were tuned for each event log separately for better performance. In particular, 32 embedding dimensions were used for Helpdesk logs, and 415 embedding dimensions were used for BPIC 2015 logs; a batch size of 20 was used for the Helpdesk log and a batch size of 1500 was used for the BPIC 2015 logs; 100 neurons were used in the LSTM layer for all logs.

4.3.2 Online Drift Detection

Two publicly available event logs were used to evaluate the performance of the online concept drift detection methods. Namely, the loan application process (Maaradji et al., 2015) and Dutch municipality (BPIC 2015) (Dongen, 2015) logs. The details of the two datasets are presented in turn below. Accuracy (calculated as a harmonic mean of precision and recall) and mean delay (calculated as the average number across all windows of log traces between the point when a drift occurred and when it was detected) were used as performance metrics (Ho, 2005). The binary case (i.e. *Case 1*) was considered to calculate the similarity score in online drift detection. For both methods, the threshold was tuned for each dataset according to the values obtained over the first reference windows (which contain no drift by default). In particular, for the Loan Application Process dataset, the thresholds were set to 0.9 and 1.0 for PGraphDD-QM and PGraphDD-SS, respectively, while for the municipality dataset of BPIC 2015, the thresholds were set to 0.9 and 0.3 for PGraphDD-QM and PGraphDD-SS, respectively. The results obtained for the loan application process logs (Maaradji et al., 2015) were compared to those reported in (Maaradji et al., 2017; Seeliger, Nolle, and Mühlhäuser, 2017; Sousa et al., 2021). Although there exists a study using the BPIC 2015 logs for drift detection (Hassani, 2019), the results reported cannot be directly compared with ours.

Loan Application Process Dataset

The loan application process dataset comprises 72 synthetic event logs generated from a base model comprising 15 activities, one start, and three end events. The logs exhibit different control-flow structures, including loops and parallel and alternative branches (Figure 4.4). To generate the logs, the base model was systematically modified applying, in turn, one of the twelve simple change patterns described in (Weber, Reichert, and Rinderle-Ma, 2008) (Table 4.3). These modifications reveal different change patterns, which are categorised into insertion ("I"), resequentialisation ("R") and optionalisation ("O"). More complex drifts were created by combining the simple change patterns; This involved randomly applying a pattern from each category in a nested way, thus resulting in additional event logs: "IOR", "IRO", "OIR", "ORI", "RIO", and "ROI".

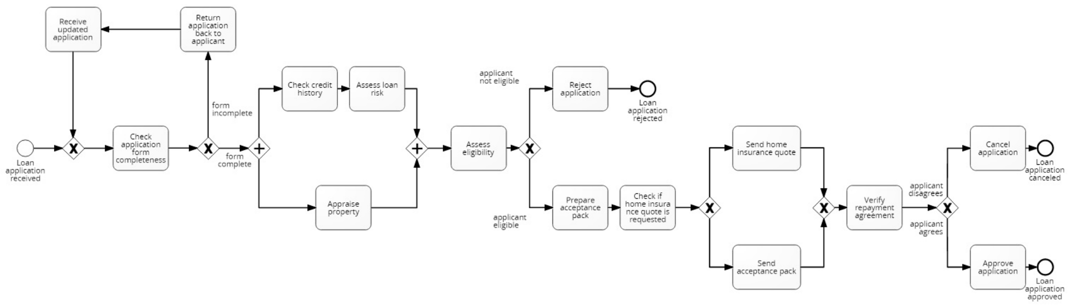

Receive updated application
Return application back to applicant
Loan application received
Check application form completeness
form incomplete
form complete
Check credit history
Assess loan risk
Appraise property
Assess eligibility
applicant not eligible
applicant eligible
Reject application
Loan application rejected
Prepare acceptance pack
Check if home insurance quote is requested
Send home insurance quote
Send acceptance pack
Verify repayment agreement
applicant disagrees
applicant agrees
Cancel application
Loan application canceled
Approve application
Loan application approved

FIGURE 4.4: Base BPMN model of the loan application process

To vary the distance between the drifts, four event logs of sizes 2500, 5000, 7500 and 10000 were generated for each of the change patterns by combining a fixed number of alternating instances from the base model, then a fixed number of instances from the modified model, leading to a total of 72 logs. Each event log generated in this way contains precisely nine process drifts.

TABLE 4.3: Control-flow change patterns for synthetic event logs adopted from (Maaradji et al., 2015). "I": insertion; "R": resequentialisation; "O": optionalisation.

Code	Simple change pattern	Category
re	Add/remove fragment	I
cf	Make two fragments conditional/sequential	R
lp	Make fragment loopable/non-loopable	O
pl	Make two fragments parallel/sequential	R
cb	Make fragment skippable/non-skippable	O
cm	Move fragment into/out of conditional branch	I
cd	Synchronise two fragments	R
cp	Duplicate fragment	I
pm	Move fragment into/out of parallel branch	I
rp	Substitute fragment	I
sw	Swap two fragments	I

BPIC 2015 Dataset

The same BPIC 2015 municipality dataset introduced in 4.3.1 was used. However, here, similar to Hassani, 2019, the five logs were merged for the experiments to get one log with four reliable concept drifts.

Experimental Setup for Online Drift Detection

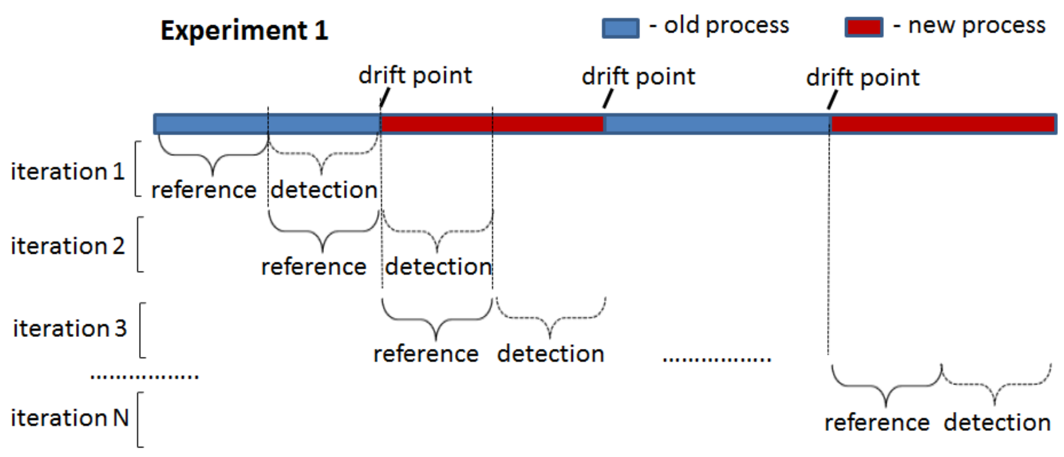


FIGURE 4.5: Experiment 1: drift introduced at the start of the detection window

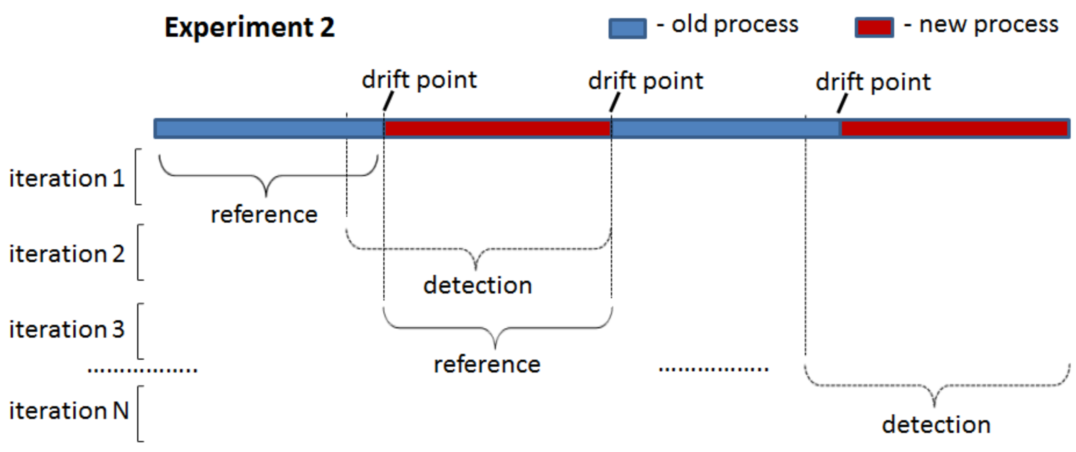


FIGURE 4.6: Experiment 2: drift introduced in the middle of the detection window

The two proposed drift detection methods were tested in two experiments: (1) drift introduced at the beginning of the detection window (Figure 4.5) and (2) drift introduced in the middle of the detection window (Figure 4.6). In the first experiment, all the traces in the detection window were generated by a process model different from that used to generate traces for the reference window. In the second experiment, the detection window contained traces generated by two different process models: an old one used for the reference window and a new one. The point of the switch from the old to the new process in the detection window (i.e. the ratio between the number of traces generated by the old and new process models) was varied in the second experiment to assess the sensitivity of the proposed drift detection methods to the number of traces generated by the old process model still present in the analysed window containing a drift.

To demonstrate the ability of the proposed methods to detect drifts in the first experiment, the LSTM model was initially trained on the first half of the reference window and applied to its second half. In this case, the model is expected to achieve an F-score close to 1 since the same process model generated traces in both portions. This F-score value was used to set the threshold for PGraphDD-QM. Next, the model was trained on the second half of the reference window and applied to the first half of the detection window. Since the model was trained on traces generated by one process model but applied to traces generated by another, a drop in the F-score value below the threshold is expected, indicating a drift. The procedure was repeated by shifting windows forward until the end of the considered log was reached. For all the iterations, it was noted whether the F-score value was always above the threshold when the LSTM model was trained and applied to traces generated by the same process model (i.e., not triggering false alarms) and below the threshold when the LSTM model was trained on traces generated by one process model but applied to traces generated by another process model (i.e., detecting drifts when they actually happened). For PGraphDD-SS, the similarity score was compared across the windows in the same manner, assuming that a similarity score above the threshold indicated no drift, and a similarity score below the threshold meant a drift.

The second experiment was designed to explore the behaviour of the proposed drift detection methods on the stitches of the traces generated by different process models and obtain the delay metric. In this case, each detection window was constructed to include traces before and after the change point (i.e. drift) at different percentages. Initially, the detection window was set to have 90% of traces generated by the old process model and 10% of traces generated by the new process model. The F-score value was checked and a drift was assumed to be detected if the F-score value was below the set threshold. If drift was not detected, the detection window was modified to include old and new traces in the ratios of 80%:20%, 70%:30% and so on until drift was detected. The delay was set to the number of traces generated by the new process model at the point of the detected drift (i.e., if the drift was detected in a ratio of 30%:70%, the delay was set to the number of traces in the 30% block). A short delay between a change and its detection is highly desirable.

The following experimental protocol was adopted to evaluate the proposed methods for detecting concept drifts in business processes in online scenarios:

1. For both methods, the threshold was tuned for each dataset according to the values obtained over the first reference windows (which contain no drift by default).

2. For the embedding layer of the LSTM model, the vocabulary size was set to the number of unique activities in the log considered. The size of the vocabulary guided the number of embedding dimensions; hence, it varied across the event logs.

3. The network architecture of the LSTM model included one hidden LSTM layer with a dropout probability of 0.2. The dropout technique was used to avoid over-fitting and improve learning by randomly removing a cell from the network.

4. The number of training epochs was set to 50 for all synthetic logs and 10 for the BPIC 2015 logs based on preliminary experiments.

5. The parameters of the LSTM model, namely, the number of embedding dimensions, batch size, and LSTM neurons, were tuned for the different event

log sizes for better performance. In particular, 250; 500; 750; 1,000 embedding dimensions and batch sizes were used for each of 2,500; 5,000; 7,500; 10,000 loan application process logs, respectively; 1,000 embedding dimensions and a batch size of 1000 were used for the BPIC 2015 logs; 100 neurons were used in the LSTM layer for all logs.

4.4 Results and Discussion

This section presents and discusses the results of the experiments carried out for the two methods in both offline and online scenarios.

4.4.1 Offline Drift Detection: PGraphDD-QM

Tables 4.4–4.7 show the results of drift detection using PGraphDD-QM on the considered event logs based on the quality metrics (fitness, precision, and F-score) obtained for two different time periods.

Table 4.4 lists the fitness, precision, and F-score values calculated based on the LSTM model trained using the original (base) Helpdesk log (considered as the old event log) and applied to each of its modified versions (considered as new event logs, Section 4.3.1). The values of all the three metrics for the $Helpdesk\_1$ and $Helpdesk\_2$ logs are 0.99 and 0.97, respectively, implying that the two time periods (old and new logs) are respectively 99% and 97% similar, i.e., insignificant drifts are detected in the new log, with only one activity being added or removed, compared to the old log (it is worth noting that the metric values are all 1.0 when comparing the original event log to itself, see Table 4.4). Changes to the flow introduced in the $Helpdesk\_3$ event log resulted in a stronger indication of drift, with precision dropping to 0.85 and thus F-score dropping to 0.90, with fitness remaining at 0.97. The most significant indication of drift can be observed for the $Helpdesk\_4$ log that incorporates all three types of changes, with fitness dropping to 0.94, precision to 0.84, and F-score to 0.88. From these results, it can be concluded that the more significant drop in the values of the quality metrics indicates a more significant drift (i.e., a higher number of changes can be observed in the new logged events compared to the old log).

Tables 4.5 and 4.6 show the results obtained by splitting each of the modified versions of the Helpdesk log into 5 and 8 portions (or folds), respectively, and then comparing each fold in turn to the Helpdesk log, thus simulating a more dynamic offline scenario, where only a small portion of the new event log is available for analysis. Similar to considering a larger event log, a more significant drift is detected in the case of $Helpdesk\_4$ folds, as indicated by the lowest fitness scores.

Table 4.7 shows the results of comparing the pairs of the five BPI20C15 logs against each other, with one log in the pair representing the old log (on which the LSTM model was trained) and the other log representing the new log (in which drifts are detected). It can be noticed from the table that the values of the quality measures are significantly lower across all pairs compared to those obtained for the Helpdesk log variations, indicating more severe drifts. Indeed, the business processes captured in the BPIC2015 logs vary much more than those captured in the Helpdesk variations. To confirm this, the ProM tool was used to view activities, traces, and the process flows captured in the event logs of the five municipalities. This analysis revealed that almost all cases follow a unique execution path and there is a unique variant for almost every case. For example, BPIC2015_1 has 943 cases and only 888 variants. This explains the low fitness, precision, and F-score values observed across all pairs of event logs, except in the cases where the event logs were compared with themselves. For example, the values of 0.0017 for fitness, 0.0024 for precision and 0.0020 for F-score obtained when comparing BPIC2015_1 and BPIC2015_2 logs indicate a high degree of drift, i.e., a lot of changes have occurred in the way the processes were executed between the two time periods. In contrast, the value of 1 across all the metrics obtained when comparing each BPIC2015 log to itself (e.g. BPIC2015_1 against BPIC2015_1) indicates that the proposed method captured no drift, which is what one would expect when comparing two identical business processes.

The experiments with folds conducted for the Helpdesk log were unnecessary for the BPIC2015 logs, given their complexity. This is because the performance metric values are already very low, and splitting the logs into folds would result in many unique traces being captured in each fold, thus returning 0 scores across all the metrics due to no overlaps between traces across the folds.

TABLE 4.4: Results of offline process drift detection using PGraphDD-QM for Helpdesk log and its variations

Event Logs	**Fitness/Recall**	**Precision**	**F-Score**
Helpdesk/Helpdesk_1	0.9925	0.9913	0.9919
Helpdesk/Helpdesk_2	0.9706	0.9706	0.9706
Helpdesk/Helpdesk_3	0.9705	0.8462	0.9041
Helpdesk/Helpdesk_4	0.9412	0.8421	0.8889
Helpdesk/Helpdesk	1.0000	1.0000	1.0000

TABLE 4.5: Results of offline process drift detection using PGraphDD-QM for Helpdesk log and its variations using five folds

Event Log	**Folds**	**Fitness/Recall**	**Precision**	**F-Score**
Helpdesk/Helpdesk_1	**Fold 1**	0.88	0.88	0.88
	Fold 2	0.90	0.90	0.90
	Fold 3	0.92	0.92	0.92
	Fold 4	0.90	0.90	0.90
	Fold 5	0.90	0.90	0.90
Helpdesk/Helpdesk_2	**Fold 1**	0.88	0.88	0.88
	Fold 2	0.88	0.88	0.88
	Fold 3	0.90	0.90	0.90
	Fold 4	0.89	0.89	0.89
	Fold 5	0.89	0.89	0.89

Helpdesk/Helpdesk_3	**Fold 1**	0.73	0.73	0.73
	Fold 2	0.72	0.72	0.72
	Fold 3	0.71	0.71	0.71
	Fold 4	0.74	0.74	0.74
	Fold 5	0.74	0.74	0.74
Helpdesk/Helpdesk_4	**Fold 1**	0.62	0.62	0.62
	Fold 2	0.63	0.63	0.63
	Fold 3	0.62	0.62	0.62
	Fold 4	0.64	0.64	0.64
	Fold 5	0.64	0.64	0.64

TABLE 4.6: Results of offline process drift detection method for Helpdesk log and its variations using eight folds

Event Log	**Folds**	**Fitness/Recall**	**Precision**	**F-Score**
Helpdesk/Helpdesk_1	**Fold 1**	0.91	0.91	0.91
	Fold 2	0.90	0.90	0.90
	Fold 3	0.90	0.90	0.90
	Fold 4	0.89	0.89	0.89
	Fold 5	0.89	0.89	0.89
	Fold 6	0.92	0.92	0.92
	Fold 7	0.90	0.90	0.90
	Fold 8	0.89	0.89	0.89

Helpdesk/Helpdesk_2	**Fold 1**	0.90	0.90	0.90
	Fold 2	0.88	0.88	0.88
	Fold 3	0.89	0.89	0.89
	Fold 4	0.87	0.87	0.87
	Fold 5	0.88	0.88	0.88
	Fold 6	0.87	0.87	0.87
	Fold 7	0.87	0.87	0.87
	Fold 8	0.89	0.89	0.89
Helpdesk/Helpdesk_3	**Fold 1**	0.69	0.69	0.69
	Fold 2	0.69	0.69	0.69
	Fold 3	0.69	0.69	0.69
	Fold 4	0.69	0.69	0.69
	Fold 5	0.70	0.70	0.70
	Fold 6	0.72	0.72	0.72
	Fold 7	0.68	0.68	0.68
	Fold 8	0.93	0.93	0.93
Helpdesk/Helpdesk_4	**Fold 1**	0.63	0.63	0.63
	Fold 2	0.62	0.62	0.62
	Fold 3	0.64	0.64	0.64
	Fold 4	0.67	0.67	0.67
	Fold 5	0.65	0.65	0.65
	Fold 6	0.66	0.66	0.66
	Fold 7	0.63	0.63	0.63
	Fold 8	0.66	0.66	0.66

TABLE 4.7: Results of offline process drift detection using PGraphDD-QM for BPIC2015 logs

Event Logs	Fitness/Recall	Precision	F-Score
BPIC2015_1/BPIC2015_2	0.0017	0.0024	0.0020
BPIC2015_1/BPIC2015_3	0.0026	0.0022	0.0024
BPIC2015_1/BPIC2015_4	0.0009	0.0010	0.0009
BPIC2015_1/BPIC2015_5	0.0017	0.0017	0.0017
BPIC2015_1/BPIC2015_1	1.0000	1.0000	1.0000
BPIC2015_2/BPIC2015_3	0.0012	0.0007	0.0009
BPIC2015_2/BPIC2015_4	0.0012	0.0010	0.0011
BPIC2015_2/BPIC2015_5	0.0024	0.0017	0.0020
BPIC2015_2/BPIC2015_2	1.0000	1.0000	1.0000
BPIC2015_3/BPIC2015_4	0.0029	0.0038	0.0033
BPIC2015_3/BPIC2015_5	0.0007	0.0009	0.0008
BPIC2015_3/BPIC2015_3	1.0000	1.0000	1.0000
BPIC2015_4/BPIC2015_5	0.0000	0.0000	0.0000
BPIC2015_5/BPIC2015_5	1.0000	1.0000	1.0000

4.4.2 Offline Drift Detection: PGraphDD-SS

Tables 4.8 and 4.9 show the results of PGraphDD-SS drift detection on the considered event logs based on the similarity score between each pair of graphs representing business processes from two different time periods.

In Table 4.8, the process graph generated from the original Helpdesk log is compared against each process graph generated from its modified versions as described in Section 4.3.1. Two sets of results are presented in Table 4.8: for the binary case (*Case 1*) and the probability case (*Case 2*). First, the process graph of the original Helpdesk log is compared against the process graph of the same original Helpdesk log, A similarity score of 1.0 was obtained for Case 1 and 1.0 for Case 2, suggesting the two graphs are identical (i.e., no change detected). Then, the Helpdesk_1 process graph was compared with the Helpdesk process graph of the original Helpdesk log. A similarity score of 0.97 was obtained for Case 1 and 0.98 for Case 2, indicating a minor change. A similarity score of 0.90 for Case 1 and 0.92 for Case 2 was obtained when the process graph of Helpdesk_2 log was compared against the process graph of Helpdesk log; the scores still suggest minor changes detected. A decrease in similarity scores was observed when the Helpdesk log process graph was compared to the Helpdesk_3 and Helpdesk_4, especially the latter, which showed the lowest similarity score of 0.71 for Case 1, and 0.74 for Case 2. This is expected as Helpdesk_4 incorporates all three change patterns (Section 4.3.1), hence a greater change is anticipated. Overall, the similarity score in all the combinations (i.e., the base Helpdesk log compared to the four modified versions) was high. This indicates that the detected drifts are insignificant, which is expected given the minor modifications made to the alternative Helpdesk logs. When comparing the similarity scores of the two cases obtained across the logs (i.e., Case 1 and Case 2), it can be said that Case 1 (the binary case) better captures drifts in all modified versions of the original log. Furthermore, it is observed that the more changes introduced in the original process, the better the performance of PGraphDD-SS in offline drift detection.

Table 4.9 shows the pairwise comparison of all five BPIC2015 event logs. Given the complex nature of the logs, each contains a large number of event cases and lengthy execution paths (Table 4.1). A probability threshold was used to generate

TABLE 4.8: Results of offline process drift detection using PGraphDD-SS for Helpdesk logs: *Case*1: binary; *Case*2: probabilities

Event Log	**Similarity Score (%)**	
	Case 1	**Case 2**
Helpdesk/Helpdesk	1.00	1.00
Helpdesk/Helpdesk_1	0.97	0.98
Helpdesk/Helpdesk_2	0.90	0.92
Helpdesk/Helpdesk_3	0.76	0.85
Helpdesk/Helpdesk_4	0.71	0.74

the process graphs based on the predictions of the LSTM model to filter out less probable transitions between activities as predicted by the LSTM model. This implies that the generated graph would appear less complex if the probability threshold is set to a higher value, allowing only highly probable process paths to appear in the graph, while setting the threshold to 0 would result in a more complex graph since all transitions with a probability higher than 0 would feature in the graph. Different threshold values were tested to determine the sensitivity of the similarity score. It can be seen from the results listed in Table 4.9 that higher threshold values yield higher similarity scores. For example, when comparing BPIC2015_1 against BPIC2015_2, using a probability threshold of 0 (i.e., when including all transitions as predicted by the LSTM model), a similarity score of 0.69 was obtained for Case 1 and 0.45 for Case 2. The scores suggest an average change in behaviour in the process graph for the next time period. However, on the same logs, using a probability threshold of 1 (i.e., when including only the transitions that the LSTM model is 100% confident about as being the only way of transiting from one activity to another), a similarity score of 0.00 was obtained for Case 1 and 0.00 for Case 2, suggesting a total change in behaviour in the process graph for the next time period. The same pattern was observed in most pairs when playing with the different thresholds.

A factor that may have contributed to the big difference in the observed similarity scores is that, on average, when comparing the BPIC2015 logs, there are about ten sub-processes covering 30% of the event logs found in all of them. This means that on average another 70% of the behaviour in each log is not observed in each of the other logs (Martin et al., 2015). Hence, when all transitions are included (i.e. when a low probability threshold is used), the possibility of capturing a similar behaviour in both time periods is high. However, when a higher probability threshold is set, only the highly likely transitions are included in the graphs, thus making the possibility of capturing a similar behaviour less likely in the time periods.

In the probability case (*Case*2), an average similarity score of 0.44 is observed across all pairs when the threshold is 0.0, 0.40 when the threshold is 0.2, 0.25 when the threshold is 0.4, 0.15 when the threshold is 0.6, 0.10 when the threshold is 0.8 and 0.0 when the threshold is 1.0.

Just like for the Helpdesk log, a difference is observed between the similarity scores calculated for the binary case (*Case*1) and the probability case (*Case*2). For the Helpdesk log, the similarity scores for Case 1 are lower than those for Case 2. However, for the BPIC2015 logs, the similarity scores for Case 1 are higher than those for Case 2 across all probability thresholds. The difference in results obtained for the Helpdesk and BPIC2015 logs can be explained by the more complex nature of the BPIC2015 logs.

TABLE 4.9: Results of offline process drift detection method using PGraphDD-SS for BPIC2015 logs: *Case*1: binary; *Case*2: probabilities

Event Log	**Threshold**	**Similarity Score (%)**	
		Case 1	Case 2
BPIC2015_1/BPIC2015_2	0.0	0.69	0.45
	0.2	0.65	0.40
	0.4	0.49	0.26
	0.6	0.33	0.20
	0.8	0.12	0.09
	1.0	1.00	0.00
BPIC2015_1/BPIC2015_3	0.0	0.69	0.47
	0.2	0.66	0.44
	0.4	0.50	0.28
	0.6	0.33	0.18
	0.8	0.20	0.11
	1.0	0.88	0.07
BPIC2015_1/BPIC2015_4	0.0	0.68	0.46
	0.2	0.66	0.44
	0.4	0.50	0.28
	0.6	0.34	0.18
	0.8	0.20	0.11
	1.0	0.00	0.00
BPIC2015_1/BPIC2015_5	0.0	0.69	0.44
	0.2	0.66	0.41
	0.4	0.50	0.28
	0.6	0.32	0.15
	0.8	0.18	0.10
	1.0	0.00	0.00

BPIC2015_2/BPIC2015_3	0.0	0.69	0.44
	0.2	0.65	0.39
	0.4	0.49	0.26
	0.6	0.34	0.82
	0.8	0.18	0.91
	1.0	1.00	1.00
BPIC2015_2/BPIC2015_4	0.0	0.69	0.43
	0.2	0.65	0.41
	0.4	0.51	0.28
	0.6	0.35	0.18
	0.8	0.18	0.07
	1.0	1.00	0.00
BPIC2015_2/BPIC2015_5	0.0	0.71	0.48
	0.2	0.66	0.41
	0.4	0.49	0.25
	0.6	0.32	0.16
	0.8	0.17	0.08
	1.0	0.08	0.02
BPIC2015_3/BPIC2015_4	0.0	0.67	0.43
	0.2	0.64	0.39
	0.4	0.50	0.28
	0.6	0.34	0.19
	0.8	0.19	0.11
	1.0	0.00	0.00

BPIC2015_3/BPIC2015_5	0.0	0.69	0.44
	0.2	0.65	0.41
	0.4	0.50	0.28
	0.6	0.34	0.18
	0.8	0.19	0.10
	1.0	0.00	0.00
BPIC2015_4/BPIC2015_5	0.0	0.66	0.38
	0.2	0.62	0.35
	0.4	0.47	0.23
	0.6	0.29	0.11
	0.8	0.13	0.04
	1.0	0.00	0.00

The results presented in Tables 4.8 and 4.9 indicate that the proposed method can be used to effectively detect concept drifts in business processes in the offline scenario. Comparing the results obtained for the two logs (Helpdesk and BPIC2015), it can be concluded that the method is sensitive to the number of changes (or drifts) present in logs representing different time periods: the higher the similarity score, the more insignificant the difference between the compared business processes. This means that the method can not only indicate the presence of drifts but also their severity.

4.4.3 Online Drift Detection: PGraphDD-QM

Loan Application Process

In the first experiment for the loan application process dataset, PGraphDD-QM achieved an average accuracy score of 99% across all log sizes over 11 out of 17 change patterns. For the remaining change patterns ('cb', 'cm', 'ior', 'pl', 'rio' and 'lp'), the method achieved an average accuracy of 66%. These results suggest that the effectiveness of PGraphDD-QM depends on the nature of the event log. For example, in the case of the 'cb' change pattern, the fragment "'Prepare acceptance pack' – 'Check application form completeness'" always present in the reference window may be omitted in the detection window (i.e. the process branches, meaning that the two activities are executed in some cases but not always, resulting in a portion of traces containing the two activities and in a portion of traces not containing them). Since the LSTM model trained on the reference window, where the activities are always present, would still get points for correctly predicting the case of these activities being present in the detection window, PGraphDD-QM may fail to detect the drift if the proportion of the traces containing the activities is greater than the proportion of the traces not containing these activities.

TABLE 4.10: Results across all log sizes for the "cd" change pattern of the Loan Application Process dataset using PGraphDD-QM

Windows	**cd-2500**	**cd-5000**	**cd-7500**	**cd-10000**	**Drift Detected?**	**Actual Drift?**	**Correct Detection?**
0,1	0.98	0.99	0.99	0.99	No	No	Yes
1,2	0.53	0.47	0.55	0.50	Yes	Yes	Yes
2,3	0.50	0.94	0.99	0.99	No	No	Yes
3,4	0.62	0.49	0.54	0.47	Yes	Yes	Yes
4,5	0.96	0.99	0.99	0.99	No	No	Yes
5,6	0.42	0.45	0.50	0.49	Yes	Yes	Yes
6,7	0.69	0.99	0.99	0.99	No	No	Yes
7,8	0.56	0.48	0.48	0.63	Yes	Yes	Yes
8,9	0.94	0.98	0.99	0.99	No	No	Yes
9,10	0.89	0.51	0.49	0.49	Yes	Yes	Yes
10,11	0.98	0.97	0.99	0.99	No	No	Yes
11,12	0.25	0.50	0.48	0.48	Yes	Yes	Yes
12,13	0.95	1.00	0.99	0.99	No	No	Yes
13,14	0.37	0.49	0.44	0.49	Yes	Yes	Yes
14,15	0.65	0.99	0.98	0.99	No	No	Yes
15,16	0.50	0.45	0.52	0.48	Yes	Yes	Yes
16,17	0.96	0.99	0.99	0.99	No	No	Yes
17,18	0.82	0.45	0.48	0.51	Yes	Yes	Yes
18,19	0.92	0.97	1.00	0.99	No	No	Yes
19,0	0.33	0.46	0.67	0.49	Yes	Yes	Yes

Table 4.10 lists F-scores achieved by PGraphDD-QM, over each pair of windows slid across the 'cd' log of sizes 2,500, 5,000, 7,500 and 10,000 as an example to explain how the decision on whether a drift has occurred or not is derived. F-score values above the set thresholds indicate absence of drift. In contrast, a drop in the values below the thresholds indicates the presence of drift. Considering our knowledge of the event log, we can confirm whether drift happened or not in reality. The Yes/No answers in the *'Drift detected?'* column in the table indicate whether the methods detected a drift or not, respectively. The Yes/No answers in the *'Actual drift?'* column indicates whether the drift has actually occurred or not. The Yes/No answers in the *'Correct detection?'* column indicate whether PGraphDD-QM detected the drift correctly or not, respectively. It can be seen that the decisions reported in the *'Detected drift?'* column of both tables are confirmed as accurate detections in the *'Correct detection?'* column. This means that PGraphDD-QM could detect all nine drifts for the 'cd' logs of all sizes. Indeed, it can be noticed from the tables that high F-score values were returned for each window pair containing traces generated by the same process model and low values for each window pair containing traces generated by different process models.

BPIC 2015

Table 4.11 lists F-scores obtained for BPIC 2015 using PGraphDD-QM. It can be seen from the table that PGraphDD-QM was able to find all four concept drifts. However, it reported two false positives, thus achieving an accuracy score of 80%. The false positives can be attributed to the complex nature of the five business processes and the high variability of the traces within each of the five logs.

TABLE 4.11: Results obtained for BPIC 2015 using PGraphDD-QM

Windows	F-score	Drift Detected?	Actual Drift?	Correct Detection?
0,1	0.92	No	No	Yes
1,2	0.87	Yes	Yes	Yes
2,3	0.88	Yes	No	No
3,5	0.89	Yes	Yes	Yes
4,5	0.90	No	No	Yes
5,6	0.21	Yes	Yes	Yes
6,7	0.84	Yes	No	No
7,8	0.67	Yes	Yes	Yes

4.4.4 Online Drift Detection: PGraphDD-SS

Loan Application Process

In the first experiment for the Loan Application Process dataset, PGraphDD-SS achieved an average accuracy score of 98% over 16 out of 17 change patterns in the first experiment. However, over the remaining 'ior' change pattern, PGraphDD-SS achieved a low average accuracy score of 58%, which can be explained by the composite nature of this change pattern, where simple change patterns are nested within each other. Such a build means that there is a point in time when a new behaviour resembles an old behaviour from a previous time, which is challenging for the method to handle, as indicated by the four false negatives made by the LSTM model for each of the log sizes.

Table 4.12 lists similarity scores achieved by PGraphDD-SS over each pair of windows slid across the 'cd' log of sizes 2,500, 5,000, 7,500 and 10,000 as an example to explain how the decision on whether a drift has occurred or not is derived. Similarity values above the set thresholds indicate drift absence. On the contrary, a drop in values below the thresholds indicates the presence of drift. Considering our knowledge of the event log, we can confirm whether the drift occurred or not in reality. The Yes/No answers in the *'Drift detected?'* column in the table indicate whether the methods detected a drift or not, respectively. The Yes/No answers in the *'Actual drift?'* column indicates whether the drift has actually occurred or not, respectively. The Yes/No answers in the *'Correct detection?'* column indicate whether PGraphDD-SS detected the drift correctly or not. It can be observed that the decisions reported in the *'Detected drift?'* column of the table are confirmed as accurate detections in the *'Correct detection?'* column. This means that PGraphDD-SS could detect all nine drifts for the 'cd' logs of all sizes. It can be observed from the table that high similarity values were returned for each window pair containing traces generated by the same process model and low values for each window pair containing traces generated by different process models.

BPIC 2015

Table 4.13 reports the results achieved by PGraphDD-SS over the BPIC 2015 dataset. It can be noticed from the table that significantly lower similarity scores were achieved across all window pairs of the BPIC 2015 dataset compared to the loan application process dataset, which can again be explained by the much more complex nature of the BPIC 2015 logs and the higher variability across traces within each log compared

TABLE 4.12: Results across all log sizes for the "cd" change pattern of the Loan Application Process dataset using PGraphDD-SS

Windows	**cd-2500**	**cd-5000**	**cd-7500**	**cd-10000**	**Drift Detected?**	**Actual Drift?**	**Correct Detection?**
0,1	1.00	1.00	0.93	1.00	No	No	Yes
1,2	0.86	0.86	0.86	0.86	Yes	Yes	Yes
2,3	1.00	1.00	1.00	1.00	No	No	Yes
3,4	0.86	0.86	0.86	0.86	Yes	Yes	Yes
4,5	1.00	1.00	1.00	1.00	No	No	Yes
5,6	0.86	0.86	0.86	0.86	Yes	Yes	Yes
6,7	1.00	1.00	1.00	1.00	No	No	Yes
7,8	0.86	0.86	0.86	0.93	Yes	Yes	Yes
8,9	1.00	0.93	1.00	0.93	No	No	Yes
9,10	0.86	0.93	0.86	0.86	Yes	Yes	Yes
10,11	1.00	1.00	1.00	1.00	No	No	Yes
11,12	0.86	0.86	0.86	0.86	Yes	Yes	Yes
12,13	1.00	1.00	1.00	1.00	No	No	Yes
13,14	0.86	0.86	0.86	0.86	Yes	Yes	Yes
14,15	1.00	1.00	1.00	1.00	No	No	Yes
15,16	0.86	0.86	0.86	0.86	Yes	Yes	Yes
16,17	1.00	1.00	1.00	1.00	No	No	Yes
17,18	0.86	0.86	0.86	0.86	Yes	Yes	Yes
18,19	1.00	1.00	1.00	1.00	No	No	Yes
19,0	0.86	0.92	0.86	0.86	Yes	Yes	Yes

TABLE 4.13: Results obtained for BPIC 2015 using PGraphDD-SS

Windows	**Similarity score**	**Drift Detected?**	**Actual Drift?**	**Correct Detection?**
0,1	0.31	No	No	Yes
1,2	0.27	Yes	Yes	Yes
2,3	0.36	No	No	Yes
3,5	0.15	Yes	Yes	Yes
4,5	0.32	No	No	Yes
5,6	0.30	No	Yes	No
6,7	0.39	No	No	Yes
7,8	0.24	Yes	Yes	Yes

to the Loan Application Process logs. Three drifts were detected correctly, while one was missed, resulting in an accuracy score of 75%.

Further analysis of the results revealed that PGraphDD-SS is more stable and accurate on average than PGraphDD-QM (Figure 4.7). At the same time, it should be noted that PGraphDD-SS requires training two LSTM models, whereas PGraphDD-QM requires training only one. Hence, PGraphDD-QM can be recommended for resource-constrained environments, while PGraphDD-SS should be used when achieving accurate and stable results is more critical than efficiency.

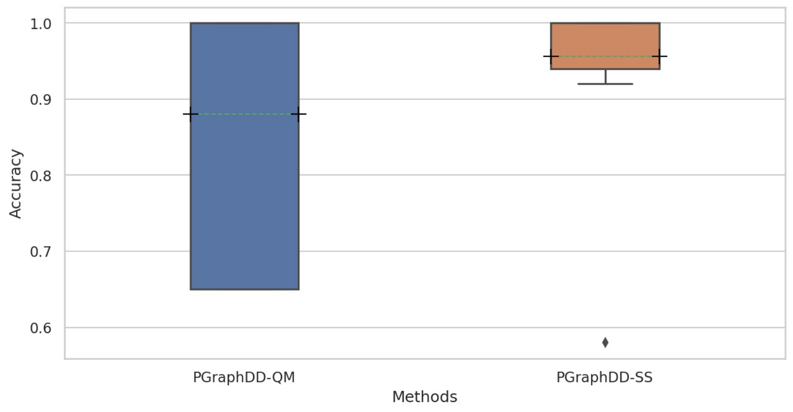


FIGURE 4.7: Distribution of accuracy scores achieved by PGraphDD-SS and PGraphDD-QM across all change patterns and log sizes of the Loan Application Process dataset

The results obtained for BPIC 2015 indicate that both PGraphDD-QM and PGraphDD-SS can perform well regardless of the complexity and peculiarities of logs, but require setting F-score and similarity threshold values on a case-by-case basis. It is important to note that the thresholds can automatically be set for each case based on the results obtained over two known covering traces generated by the same process model.

Comparison with State-of-the-art Methods

Table 4.14 lists accuracy scores for each change pattern averaged over the four event log sizes achieved by the proposed PGraphDD-QM and PGraphDD-SS methods, and the existing approaches reported in (Maaradji et al., 2017; Seeliger, Nolle, and Mühlhäuser, 2017; Sousa et al., 2021). The highest accuracy values for each change pattern across all compared methods are highlighted in bold.

PGraphDD-QM detected drifts with high accuracy between 90% and 100% in 12 out of 17 change patterns, while PGraphDD-SS detected them in 16 out of 17 change patterns (Table 4.14). PGraphDD-QM outperformed state-of-the-art methods in some change patterns such as ‘cd’, ‘cf’, and ‘rio’. However, it did not perform on par with the other methods in change pattern ‘ior’. PGraphDD-SS outperformed PGraphDD-QM (Table 4.14) in change patterns ‘cb’, ‘cm’, ‘lp’, and ‘rio’. For example, PGraphDD-QM could not find all the drifts in “cb” because it may have been tricked by the appearance of both skippable and non-skippable fragments in the detection window. However, according to PGraphDD-SS, these changes were captured in the adjacency matrices generated from the pair of graphs and thus could not be missed in the similarity score calculation. Overall, the proposed methods match

TABLE 4.14: Average accuracy achieved over the Loan Application Process logs by the proposed and state-of-the-art methods

#	PGraphDD-QM	PGraphDD-SS	Maaradji et al.	Seeliger et al.	Gaspar et al.
cb	0.65	0.92	0.92	**0.97**	0.76
cd	**1.00**	0.97	0.88	0.95	0.90
cf	0.98	**1.00**	0.98	0.98	0.98
cm	0.65	0.92	**1.00**	0.97	0.91
cp	**1.00**	**1.00**	**1.00**	0.98	**1.00**
ior	0.65	0.58	**1.00**	0.96	-
iro	**1.00**	**1.00**	**1.00**	0.94	-
lp	0.73	**1.00**	**1.00**	0.76	0.76
oir	**1.00**	**1.00**	0.98	0.73	-
ori	**1.00**	0.94	**1.00**	0.98	-
pl	0.65	**1.00**	**1.00**	0.95	0.90
pm	**1.00**	**1.00**	**1.00**	0.98	1.00
re	**1.00**	**1.00**	**1.00**	0.90	0.72
rio	0.95	**1.00**	0.98	0.97	-
roi	**1.00**	0.92	**1.00**	**1.00**	-
rp	**1.00**	**1.00**	0.96	0.97	0.98
sw	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

the performance of state-of-the-art methods, while not requiring generating features manually.

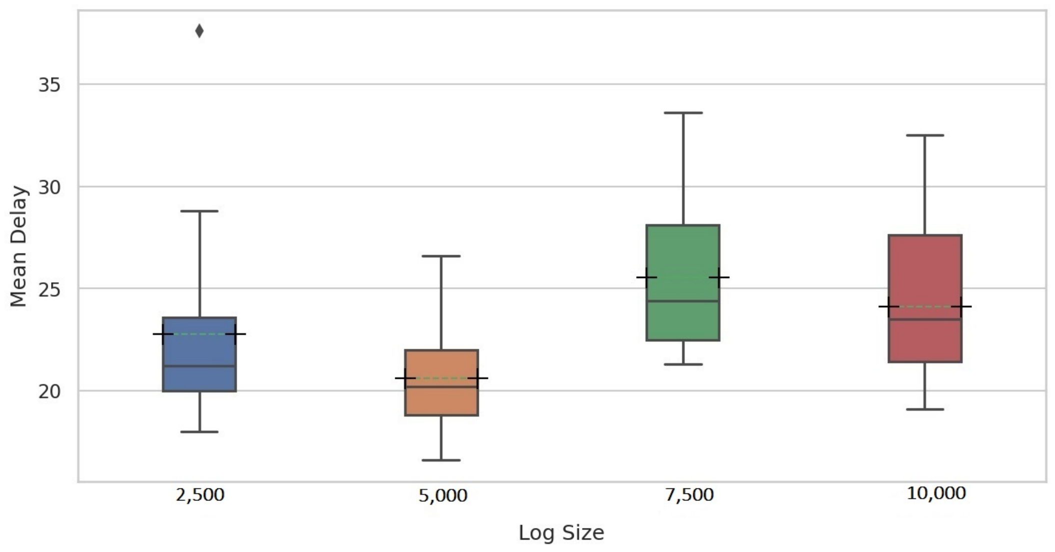


FIGURE 4.8: Distribution of mean delay scores achieved by PGraphDD-QM per change pattern for each of the four log sizes of the Loan Application Process dataset

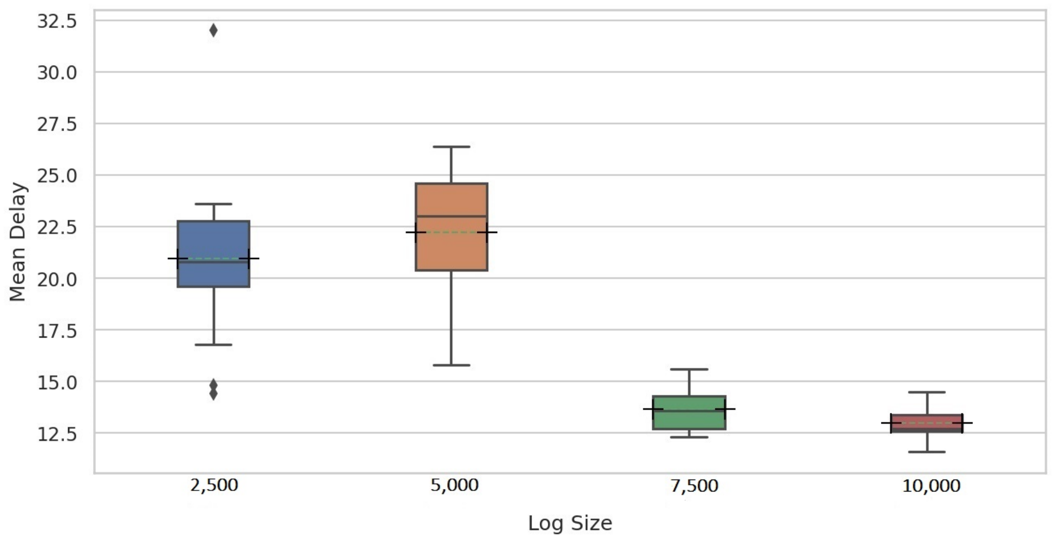


FIGURE 4.9: Distribution of mean delay scores achieved by PGraphDD-SS per change pattern for each of the four log sizes of the Loan Application Process dataset

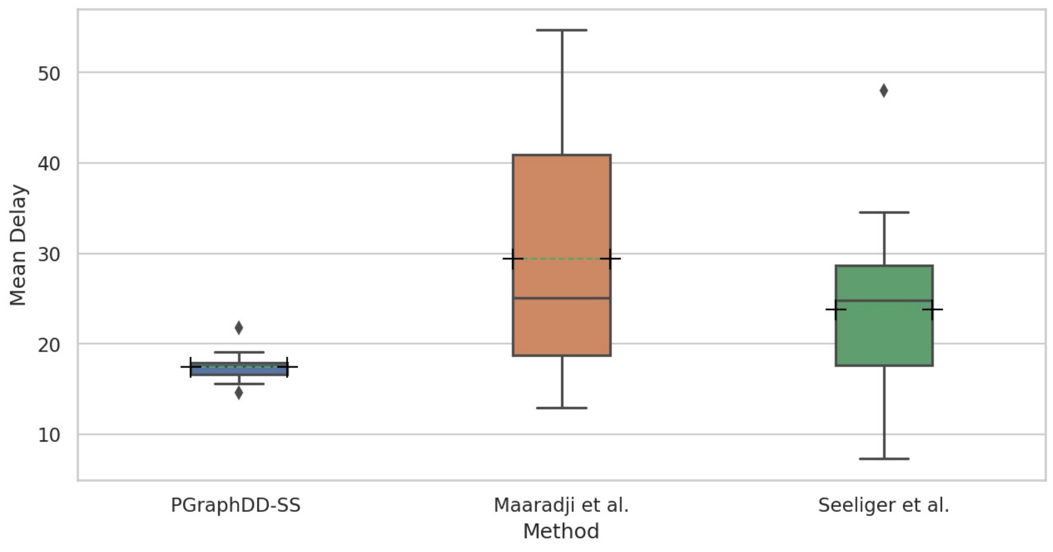


FIGURE 4.10: Distribution of mean delay scores per change pattern across all log sizes of the Loan Application Process dataset: PGraphDD-QM and PGraphDD-SS versus Maaradji et al.(Maaradji et al., 2017) and Seeliger et al. (Seeliger, Nolle, and Mühlhäuser, 2017)

For a fair comparison with Maaradji et al.(Maaradji et al., 2017) and Seeliger et al. (Seeliger, Nolle, and Mühlhäuser, 2017), a fixed window size of 100 traces was used for all log sizes in the second experiment that evaluated the delay of the proposed methods in detecting drifts. Figures 4.8 and 4.9 show the mean delay per change pattern for each of the four log sizes for PGraphDD-QM and PGraphDD-SS, respectively. It can be noticed from the figures that PGraphDD-QM performs better on reasonably small log sizes (5,000), whereas PGraphDD-SS performs better on larger log sizes (7,500 and 10,000). The performance of both methods on very small log sizes (2,500) is not stable.

Figure 4.10 compares the mean delays per change pattern across the four log sizes achieved by PGraphDD-QM and PGraphDD-SS with those achieved by the methods reported by Maaradji et al. (Maaradji et al., 2017) and Seeliger et al. (Seeliger, Nolle, and Mühlhäuser, 2017). It can be observed from the figure that both PGraphDD-QM and PGraphDD-SS outperform the state-of-the-art methods, with PGraphDD-SS achieving the most stable results and the shortest drift detection delays compared to the other three methods.

TABLE 4.15: Mean delay per change pattern of the Loan Application Process dataset compared to Maaradji et al. (Maaradji et al., 2017) and Seeliger et al. (Seeliger, Nolle, and Mühlhäuser, 2017)

#	**Mean delay**, *d*			
	PGraphDD-QM	PGraphDD-SS	Maaradji et al. (ADWIN)	Seeliger et al.
cb	25.5	**18.1**	54.7	18.9
cd	27.7	**17.8**	32.4	28.7
cf	25.4	**16.7**	19.1	34.6
cm	21.0	**17.9**	40.9	19.2
cp	24.4	**16.6**	18.7	17.6
ior	21.0	17.5	16.7	**13.0**
iro	19.7	**16.5**	43.8	27.2
lp	19.3	**18.8**	46.3	48.0
oir	19.5	**17.1**	43.7	28.01
ori	20.5	17.8	**12.9**	14.3
pl	32.5	**19.1**	36.5	26.3
pm	20.9	17.1	**12.9**	24.8
re	25.0	**14.6**	38.1	33.0
rio	22.2	**17.9**	25.1	20.8
roi	21.9	21.8	19.8	**7.3**
rp	25.5	15.6	16.9	**12.7**
sw	25.7	**15.9**	20.8	29.6

Table 4.15 lists mean delays for each change pattern averaged over the four different event log sizes for PGraphDD-QM and PGraphDD-SS, as well as the state-of-the-art methods reported in (Maaradji et al., 2017; Seeliger, Nolle, and Mühlhäuser, 2017; Sousa et al., 2021). The lowest mean delay values for each change pattern are highlighted in bold. It can be seen from the table that PGraphDD-SS performs better than the other methods in 12 out of 17 logs (cb, cd, cf, cm, cp, iro, lp, oir, pl, re, rio, sw).

One study was found to have carried out experiments using the BPIC 2015 logs (Hassani, 2019). However, the metrics employed in that study are not comparable to those reported in this and other studies on drift detection in PM. In particular, the study presented in (Hassani, 2019) used measures such as dependency, edge, and routing distances to identify drifts. The authors reported that of the four drifts resulting from concatenation of the five BPIC 2015 logs, the first three drifts were detected, but not the fourth when using the distance measure. The edge distance helped detected all four drifts but with low confidence for the first and last drifts. Finally, the routing distance also allowed to detect all drifts but with low confidence for the first drift. As discussed in Section 4.4.3, PGraphDD-QM detected three of the four drifts, while PGraphDD-SS detected all four drifts. However, the overall performance of the two methods was affected by some false positives reported over pairs of windows taken from the same logs (something not explored in (Hassani, 2019)). At the same time, it is not possible to provide a full evaluation of false positives due to the absence of the ground truth about the individual municipality logs (i.e., there could be changes in the business processes of each municipality over time constituting drifts that are not formally recognised).

4.5 Summary

Modern-day business processes are prone to changes over time as a result of changing circumstances and conditions. To proactively respond to these changes, also known as concept drifts, businesses require mechanisms to detect and analyse them. This chapter introduced two novel methods (PGraphDD-QM and PGraphDD-SS) for detecting sudden concept drifts in dynamic business processes in offline and online settings. The results of the experiments presented in the chapter demonstrated that the methods perform well in both settings. Furthermore, in the online setting, the methods perform on par with state-of-the-art methods, achieving similar accuracy in detecting drifts, with shorter delays than the other methods, while offering the following advantages over existing solutions.

First, unlike existing methods based on the statistical analysis of graphs representing business processes, the proposed methods employ DL, which does not require the user to construct features. Second, the proposed methods employ graphs that explain the decision-making process of the DL models predicting the next activities in the sequence of business processes, thus allowing the user to verify what has changed when a drift is detected. Finally, the relative insensitivity of the LSTM model to the length of the interval may have contributed to the detection of drifts with minimal delay.

The next chapter (Chapter 5) presents a new method called PGRaphDL (where 'P' stands for *process*, 'DL' for *deep learning*) for localising sudden concept drifts in dynamic business processes. PGRaphDL is built based on the methods presented in Chapters 3 and 4 to demonstrate an additional benefit of using the decision-making ability of the LSTM model when predicting the next activities in business processes to confirm localised drifts.

Chapter 5

Drift Localisation

Previous chapter (Chapter 4) demonstrated the successful ability of the two proposed methods, PGraphDD-QM and PGraphDD-SS, to detect drifts in business processes. Another objective in handling concept drifts is drift localisation. The drift localisation task is concerned with detailing the drift subject, i.e. identifying the parts of control-flow changes that relate to a concept drift. Early detection and localisation of drifts can minimise the gap between intended processes and their actual execution. It can make monitoring of business processes easy and keeps the results of process analyses up-to-date. Unlike the goal of the drift detection task, which is clear and universal over related work, no common definition is established regarding the output of a drift localisation approach. State-of-the-art drift localisation methods generally compare the process behaviour before and after the drift to report differences.

This chapter introduces a new method called PGraphDL (where 'P' stands for *process*, 'DL' for *deep learning*) for localising sudden concept drifts in dynamic business processes. Unlike existing methods that find statistically significant changes in process behaviour and output natural language statements to capture their differences, the proposed method is based on DL and extends the methods introduced in Chapters 3 and 4 to demonstrate an additional benefit of using the decision-making ability of the LSTM model when predicting the next events in a business process to confirm localised drifts.

The chapter is structured as follows. Section 5.1 details the proposed approach. Section 5.3 describes the experiments and discusses the results. Finally, Section 5.4 summarises the chapter. Part of the work described in this chapter has been published in (Hanga, Kovalchuk, and Gaber, 2022)).

5.1 Change Localisation

Detecting a drift without localising it does not provide a complete picture of the change that occurred in a process. Change localisation aims to identify the entities involved in the drift and unravel what has changed in the behaviour of a process. While drift detection alerts organisations that a process has changed, drift localisation sheds more light on where the process has changed.

Definition 1: Concept drift localisation. Let G be a process graph and let $G_0, G_1, .., G_n$ be $n+1$ different process models and $T_0 < T_1 <, .., < T_n$ be $n+1$ time periods. $G(T_i) = G_i$ represents the graph used in T_i. $G(T_0) = G_0$ is the initial graph. When the time period $T_i (0 < i \leq n)$ arrives, the current graph will change into G_i instantly, and the traces are still recorded in the same event log. Such a phenomenon is referred to as concept drift, with $T_1, ..., T_n$ being called change points. **Drift localisation** identifies the entities that have changed when a drift has occurred. Figure 4.1 illustrates a concept drift that occurred at the change point T_1. In this case, the location of the drift is around activities 'B' and 'C': these activities are executed in parallel before the drift and sequentially after the drift.

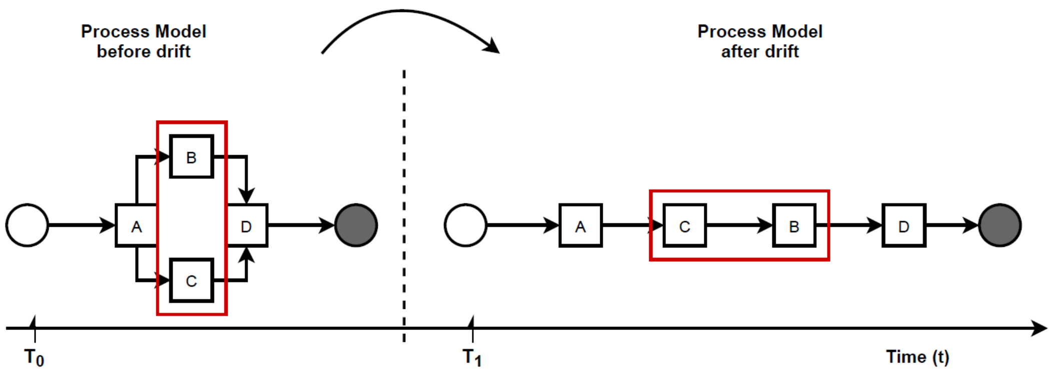


FIGURE 5.1: Concept of drift localisation

Localisation approaches commonly require an additional data structure, thus increasing complexity. As such, the majority of state-of-the-art approaches focus solely on drift detection. This chapter proposes to use the directly-follows process graphs constructed based on two different (reference and detection) windows as described in Section 4.1.2 to identify the locations of process drifts. The proposed change localisation method, PGraphDL, allows one to gather details about the structural change and detected modifications when comparing two process graphs.

5.2 Proposed method: PGraphDL

This section presents a new method called PGraphDL (where 'P' stands for process, 'DL' for deep learning) for localising sudden concept drifts in dynamic business processes. The method is based on training a uni-directional LSTM model to predict next activities in business processes and to generate DFGs based on the predictions of the model as described in Sections 4.1.1 and 4.1.2, respectively.

Algorithm 7 lists the pseudo-code of PGraphDL, which includes the following steps:

1. Take two process graphs $\mathcal{G}^R$ and $\mathcal{G}^D$ constructed based on two different windows (i.e. reference and detection) as input. The graphs can be generated using Algorithm 4 proposed in Chapter 4.
2. Select any path of interest from the base process model (i.e. $\mathcal{G}^R$) by specifying an index (line 9).
3. Compute the positional score for each candidate path in $\mathcal{G}^D$ by searching the best matching path in $\mathcal{G}^D$ for each possible path in $\mathcal{G}^R$ (line 10). The positional score (*positional_score*) is calculated as the number of activities in $\mathcal{G}^R$ located in the same position in $\mathcal{G}^D$ divided by the length of the selected path in $\mathcal{G}^R$.
4. A positional score of 1 indicates that the paths in two graphs are identical, i.e., there is no drift (lines $11 - 12$).

5. A positional drift is declared when each activity in $\mathcal{G}^D$ is not found in the same position as the activity in question in $\mathcal{G}^R$ (lines $13-14$).
6. Construct a process graph of each path and highlight the drift positions with dotted circles to visualise the drifts (line 16).
7. Display the graph of the selected path also to clarify the explanation (line 15).
8. Repeat the process for each new path of interest selected from the base process model.

Algorithm 7 PGraphDL: concept drift localisation

```
Require: Two process graphs: $\mathcal{G}^R$ and $\mathcal{G}^D$   {i.e two graphs constructed based on
    reference and detection window}, Threshold: $\phi$
 1: $\phi \leftarrow 0$ {all transitions inclusive}
 2: $positional\_score \leftarrow 0$
 3: let $\mathcal{G}^R \leftarrow \{p_1, p_2, ..., p_n\}$   {paths in $G_1$}
 4: let $\mathcal{G}^D \leftarrow \{p_1, p_2, ..., p_m\}$   {paths in $\mathcal{G}^R$}
 5: let $A_n$ be activities in a path in $\mathcal{G}^R$
 6: let $A_m$ be activities in a path in $\mathcal{G}^D$
 7: let $P_{\mathcal{G}^R}$ denote a path graph in $\mathcal{G}^R$
 8: let $P_{\mathcal{G}^D}$ denote a path graph in $\mathcal{G}^D$
 9: select path $i$ from $G_1$   {compute the positional score for each candidate path in
    $\mathcal{G}^D$}
10: $positional\_score = no\_of\_activities\_in(\mathcal{G}^R)/the\_length\_of\_the\_selected\_path\_in(\mathcal{G}^R)$
    {the candidate path in $\mathcal{G}^D$ with the max positional score is selected as the best
    matching path}
11: if $positional\_score = 1$ then
12:     no drift detected!
13: else if $A_n \in \mathcal{G}^R$ is not in the same position as $A_m \in \mathcal{G}^D$ then
14:     declare positional drift!
15:     return: $P_{\mathcal{G}^R}$
16:     return: $P_{\mathcal{G}^D}$   {with drift_position highlighted in dotted circles}
17: end if
```

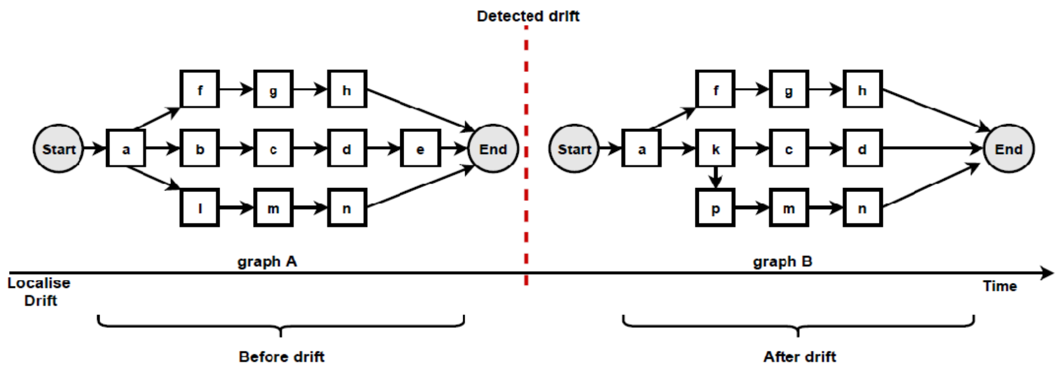


FIGURE 5.2: Two process graphs constructed based on two different windows: reference and detection

According to Algorithm 5, two process graphs (*graph A* and *graph B*) constructed based on two different windows (reference and detection), respectively, are taken as input (Figure 5.2). A user selects any path of interest from the base process model by specifying an index. The best matching path is searched in graph B for each possible path in graph A by computing the positional score for each candidate path in graph B. The positional score is calculated as the number of activities in graph A located in the same position in graph B divided by the length of the selected path in graph A. For example, if there are five activities in the selected path of graph A and four matching activities in graph B, then the positional score is 4/5. The candidate path in graph B with the maximum positional score is selected as the best matching path in graph A. A maximum positional score of 1 indicates that the paths in two graphs are identical (i.e. there is no drift). At the same time, each activity in graph B that is not found in the same position as the activity in question in graph A is declared as a positional drift. For example, if 'b' is an activity on the selected path of graph A with a positional index of '1', but there is an activity 'k' in the positional index '1' of graph B, then 'k' is declared as positional drift. In the experiments outlined in the next section, the probability threshold was set to 0 when identifying candidate paths (i.e. all transitions in the graph inclusive). The process graph of each path is constructed with drift positions highlighted in dotted circles to visualise drifts. The graph of the selected path is also displayed to clarify the explanation (Figure 5.3).

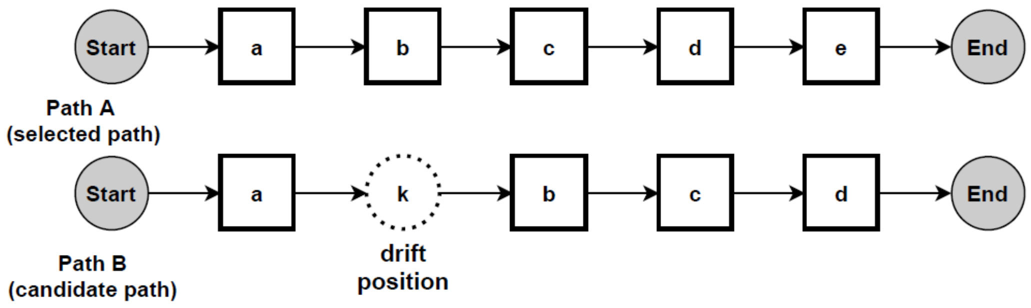


FIGURE 5.3: Change localisation according to PGraphDL: Path A is the selected path from graph A, while Path B is the candidate path from graph B with drift position highlighted

5.3 Experiments

All stages of PGraphDL were implemented as a set of Python scripts using Python 3.6. LSTM models were built using the Keras (Gulli and Pal, 2017) and Tensorflow (Abadi et al., 2016) libraries. The process graphs were generated using the Graphviz library (Ellson et al., 2001). The experiments were carried out using the Google Colab free Tesla K80 GPU.

The same event logs used in the experiments presented in Chapters 4 and 5 were used to evaluate the performance of PGraphDL: Loan Application Process (Maaradji et al., 2015), BPIC 2015 (Dutch municipality) (Dongen, 2015) and Helpdesk (Polato, 2020). To localise the drifts, the graphs generated over two different windows were compared visually and analytically. The accuracy of identified changes for the Loan

Application Process and Helpdesk datasets can be validated using knowledge about the changes injected into the event logs.

5.3.1 Localising Drifts in Loan Application Process Logs

Figure 5.4 shows two DFGs constructed for a reference window before a drift and a detection window after the drift, respectively, for the Loan Application Process dataset. In particular, the reference window graph (Figure 5.4(a)) represents the base BPMN model of the loan application process (Section 4.3.2, Figure 4.4), whereas the detection window graph (Figure 5.4(b)) represents a modified version of the base model after introducing the 'sw' change pattern. It can be observed that before the drift (Figure 5.4(a)), activities 'Prepare acceptance pack' and 'Check if home insurance quote is requested' occurred before activity 'Verify repayment agreement'. However, after the drift (Figure 5.4(b)), activities 'Prepare acceptance pack' and 'Check if home insurance quote is requested' swapped places with activity 'Verify repayment agreement'.

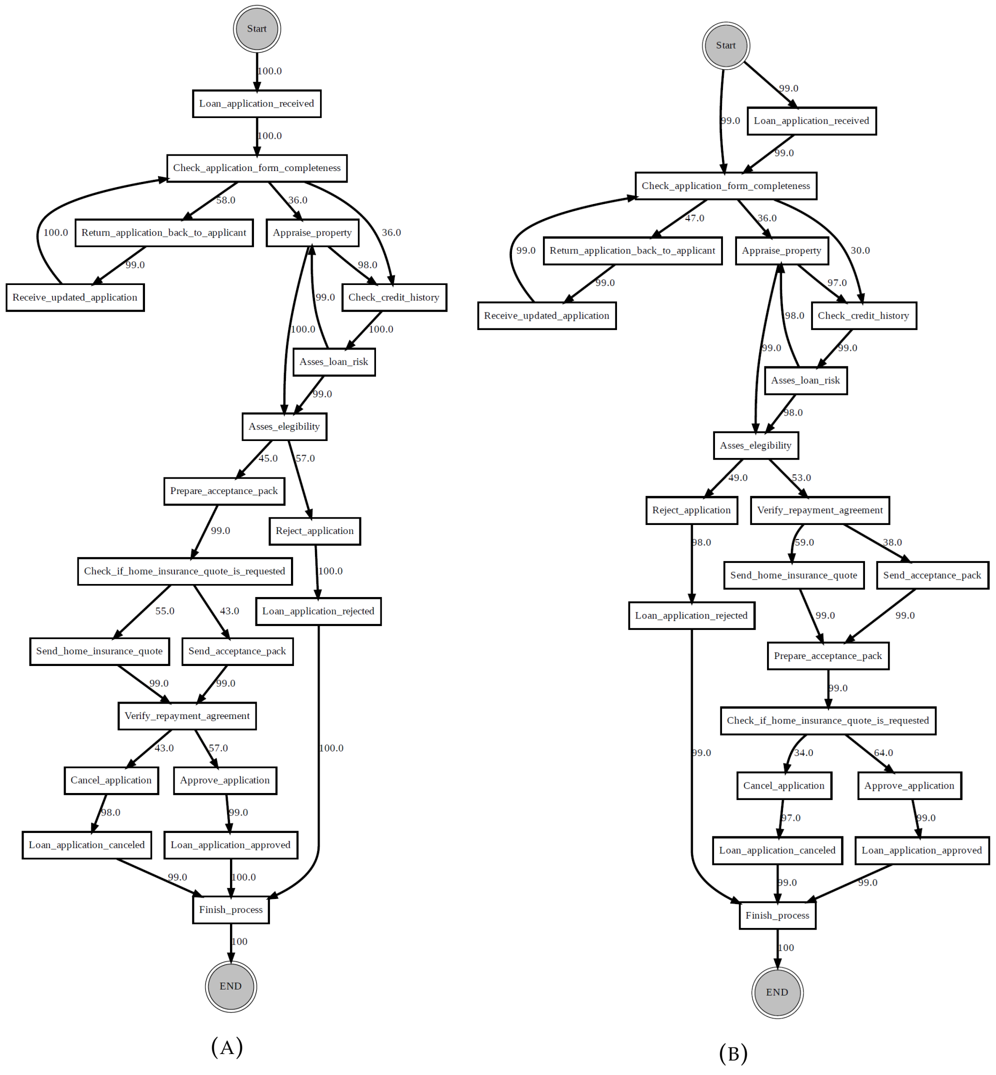

Start
Loan_application_received
Check_application_form_completeness
Return_application_back_to_applicant
Receive_updated_application
Appraise_property
Check_credit_history
Asses_loan_risk
Asses_elegibility
Prepare_acceptance_pack
Reject_application
Check_if_home_insurance_quote_is_requested
Loan_application_rejected
Send_home_insurance_quote
Send_acceptance_pack
Verify_repayment_agreement
Cancel_application
Approve_application
Loan_application_canceled
Loan_application_approved
Finish_process
END
(A)
(B)

FIGURE 5.4: Directly-follows process graphs constructed using the 'sw' change pattern log from the Loan Application Process dataset: (a) reference window; (b) detection window

Table 5.1 lists the reasons extracted from the pair of graphs generated for some of the change patterns using PGraphDL. A reason is marked as correct if the modification to the event log corresponds to the activities identified as the reasons for the process drift. The correct reason for the process drift from the graphs is extracted in all cases.

TABLE 5.1: Reasons for the drifts extracted from the graphs generated using PGraphDL for each change pattern of the loan application process dataset

Change pattern	**Reason for drift extracted from graphs**	**Reason correct?**
cb	'Prepare acceptance pack' and 'Check application form completeness' are non-skippable before the drift but skippable after drift.	Correct
cd	'Check credit history' and 'Assess loan risk' are sequential before the drift but happen at the same time after drift.	Correct
cf	'Send home insurance quote' and 'send acceptance pack' are conditional before the drift but sequential after drift.	Correct
ori	'Added activity' is added in-between Send home insurance quote' and 'send acceptance pack' which are made sequential and loopable after the drift.	Correct
rp	'Verify repayment agreement' is always in place before the drift but replaced with 'Replaced activity' after the drift.	Correct
sw	'Prepare acceptance pack' and 'Check if home insurance quote is requested' swapped place with 'Verify repayment agreement' after the drift.	Correct

To evaluate PGraphDL, a path selected from the graph representing the base loan application process was compared with the paths from the graphs representing the modified versions of the base process. After selecting the candidate path for each graph based on the positional scores, drift positions were visualised as described in Section 5.1.

Figure 5.5 shows an example of localising drifts in the 'sw' log, where two fragments are swapped in some specific parts of the log. It can be seen from the figure that before the drift (path A), activity 'Check if home insurance quote is requested' is executed after activity 'Prepare acceptance pack', whereas after the drift (path B), activities 'Prepare acceptance pack' and 'Check if home insurance quote is requested' swapped places with activities 'Verify repayment agreement' and 'Send home insurance quote'. Figure 5.6 considers the 're' log, where a fragment is added or removed in some specific parts of the log. It can be noticed from the figure that before the drift (path A), activity 'Appraise property' is followed by activity 'Assess eligibility', then by activity 'Reject application', whereas after the drift (path B), activity 'Assess eligibility' is removed, while activity 'Reject application' happens after activity 'Appraise property'. Finally, Figure 5.7 considers the 'rp' log with a substituted activity. It can be noticed from the figure that before the drift (path A), activity 'Verify repayment agreement' is executed after activity 'Send home insurance quote', whereas after the drift (path B), the activity 'Verify repayment agreement' is substituted with activity 'Replaced activity'.

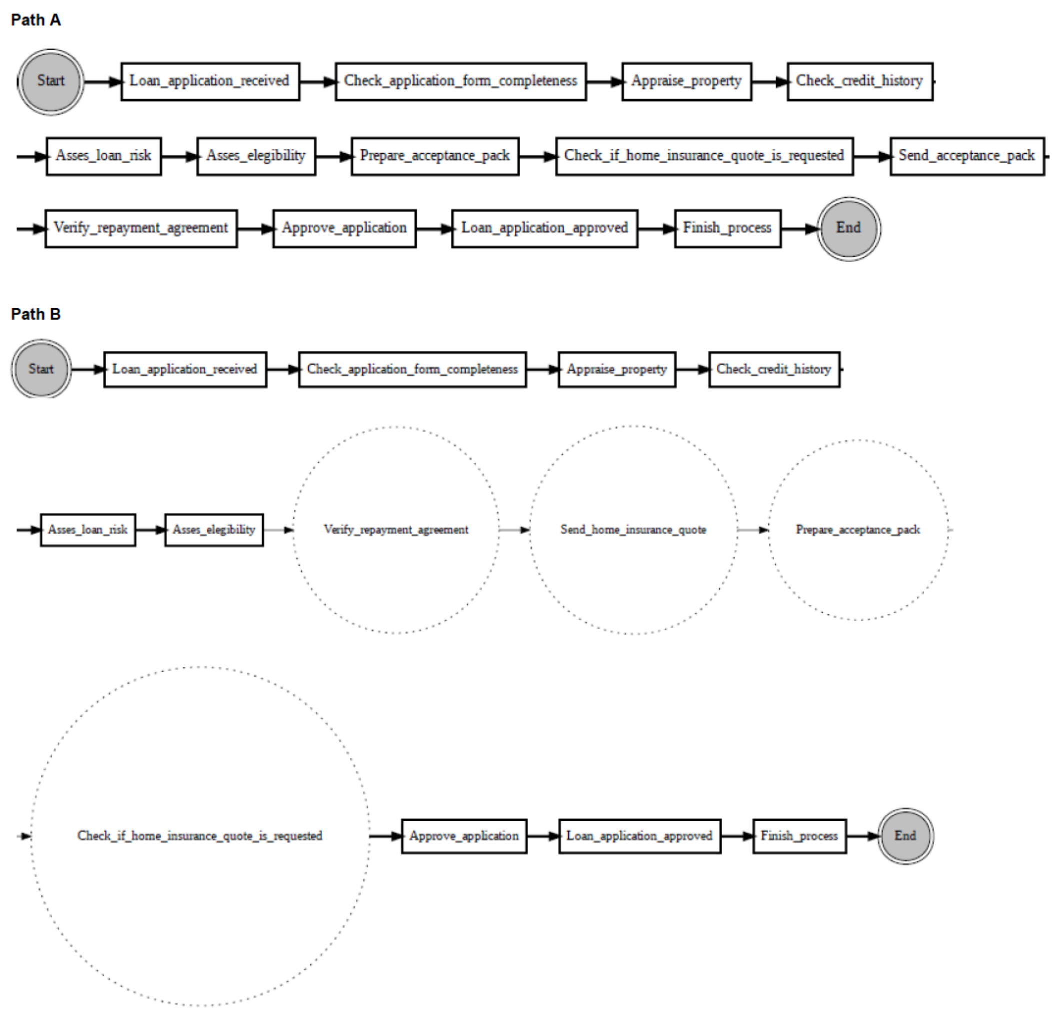

Path A
Start
Loan_application_received
Check_application_form_completeness
Appraise_property
Check_credit_history
Asses_loan_risk
Asses_elegibility
Prepare_acceptance_pack
Check_if_home_insurance_quote_is_requested
Send_acceptance_pack
Verify_repayment_agreement
Approve_application
Loan_application_approved
Finish_process
End
Path B
Start
Loan_application_received
Check_application_form_completeness
Appraise_property
Check_credit_history
Asses_loan_risk
Asses_elegibility
Verify_repayment_agreement
Send_home_insurance_quote
Prepare_acceptance_pack
Check_if_home_insurance_quote_is_requested
Approve_application
Loan_application_approved
Finish_process
End

FIGURE 5.5: Drift localisation in the Loan Application Process logs: Path A – reference path from reference window of 'sw' process graph; Path B – candidate path from detection window of 'sw' process graph

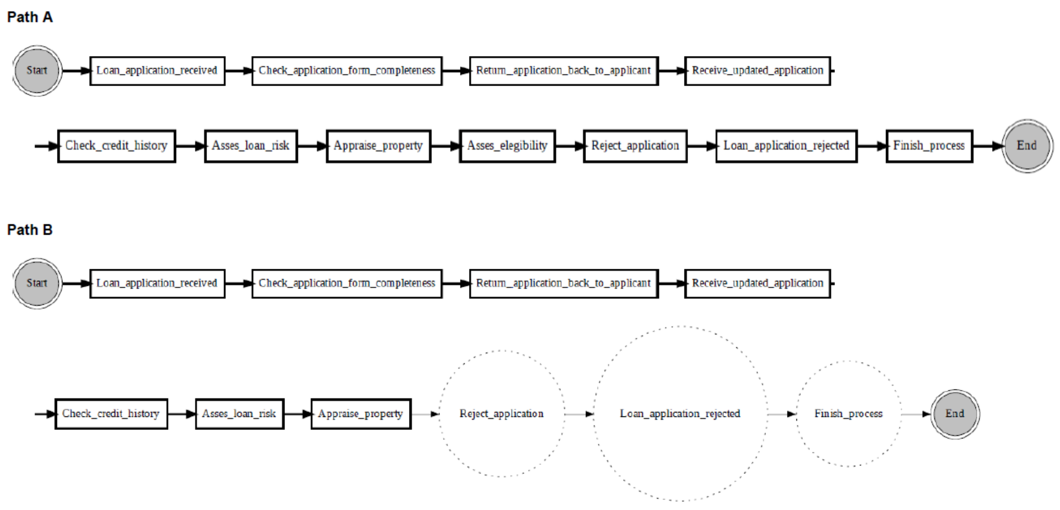

Path A
Start
Loan_application_received
Check_application_form_completeness
Return_application_back_to_applicant
Receive_updated_application
Check_credit_history
Asses_loan_risk
Appraise_property
Asses_elegibility
Reject_application
Loan_application_rejected
Finish_process
End
Path B
Start
Loan_application_received
Check_application_form_completeness
Return_application_back_to_applicant
Receive_updated_application
Check_credit_history
Asses_loan_risk
Appraise_property
Reject_application
Loan_application_rejected
Finish_process
End

FIGURE 5.6: Drift localisation in the Loan Application Process logs: Path A – reference path from reference window of 're' process graph; Path B – candidate path from detection window of 're' process graph

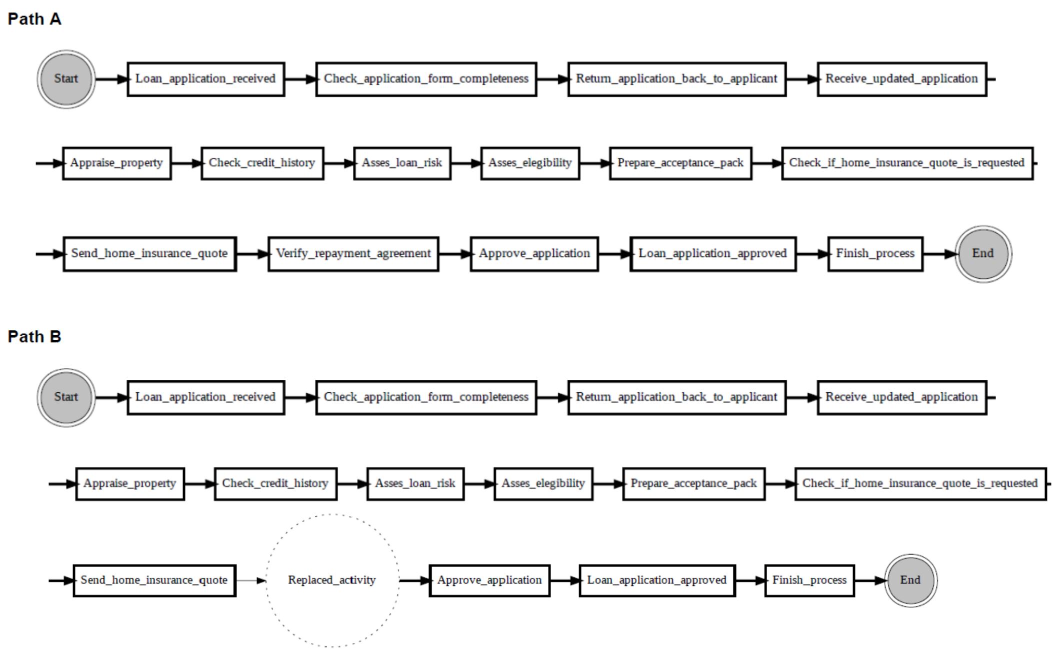

Path A
Start
Loan_application_received
Check_application_form_completeness
Return_application_back_to_applicant
Receive_updated_application
Appraise_property
Check_credit_history
Asses_loan_risk
Asses_elegibility
Prepare_acceptance_pack
Check_if_home_insurance_quote_is_requested
Send_home_insurance_quote
Verify_repayment_agreement
Approve_application
Loan_application_approved
Finish_process
End
Path B
Start
Loan_application_received
Check_application_form_completeness
Return_application_back_to_applicant
Receive_updated_application
Appraise_property
Check_credit_history
Asses_loan_risk
Asses_elegibility
Prepare_acceptance_pack
Check_if_home_insurance_quote_is_requested
Send_home_insurance_quote
Replaced_activity
Approve_application
Loan_application_approved
Finish_process
End

FIGURE 5.7: Drift localisation in the Loan Application Process logs: Path A – reference path from reference window of 'rp' process graph; Path B – candidate path from detection window of 'rp' process graph

5.3.2 Localising Drifts in Helpdesk Log

A selected path in the DFG generated using the original Helpdesk log was compared with the paths in the DFGs generated using the four modified versions of the Helpdesk log. After selecting the candidate path for each graph based on the positional scores, the drift positions were visualised as described in Section 5.1. Figure 5.8 illustrates an example of localising drifts in the *Helpdesk*_1 log, where activity was added compared to the original Helpdesk log. It can be noticed from the figure that before the drift (path A), activities 'Assign seriousness' and 'Take in charge ticket' were sequential. After the drift (path B), the activity 'Emergency' is executed between the two activities. The introduction of this new activity to the sequence has pushed forward the occurrence of 'Take in charge ticket', 'Resolve ticket' and 'Closed', which is why these activities are also circled in dotted lines in the figure. It can thus be concluded that the reason for the drift observed in the new time period (represented by the modified log) compared to the old one (represented by the original log) is the addition of an activity to the process.

Figure 5.9 shows an example of the localisation of drifts in the *Helpdesk*_2 log, where activity was removed compared to the original Helpdesk log. It can be noticed from the figure that before the drift (path A), activity 'Duplicate' is executed after activity 'Verified', whereas after the drift (path B), activity 'Duplicate' does not appear anymore; activity 'Resolve ticket' is now executed after the activity 'Verified'. In Figure 5.10, which considers the *Helpdesk*_3 log with the changed flow, before the drift (path A), activity 'Wait' is executed before activity 'Resolve ticket', whereas after the drift (path B), activity 'Require upgrade' happens after activity 'Wait' and before activity 'Resolve ticket'. In Figure 5.11, which considers the *Helpdesk*_4 log incorporating different changes, before drift (path A), activities 'Assign seriousness', 'Create SW anomaly', 'Take in charge ticket', 'Require upgrade', 'Verified', 'Duplicate', 'Resolve ticket' and 'Closed' can be seen to happen sequentially. A significant change can be noticed after the drift (path B), where a new activity 'Emergency' is introduced between activities 'Assign seriousness' and 'Create SW anomaly', while activity 'Duplicate' does not appear anymore, and activity 'Resolve ticket' is instead executed after activity 'Verified'.

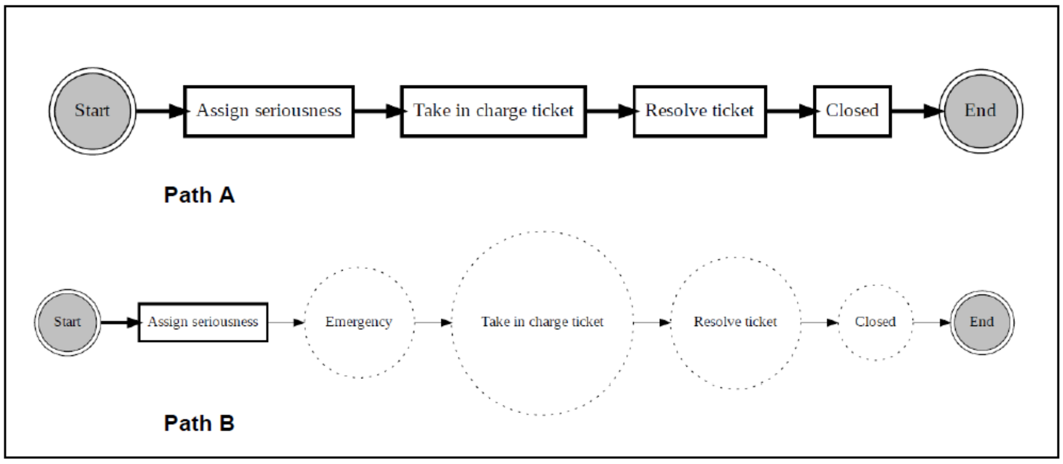


FIGURE 5.8: Path graphs depicting drift localisation: Path A – reference path from original Helpdesk log process graph; Path B – candidate path from *Helpdesk_1* process graph

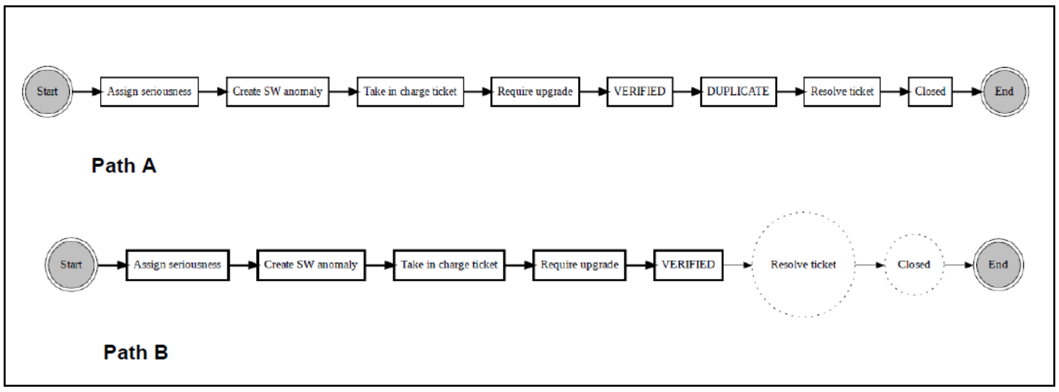


FIGURE 5.9: Path graphs depicting drift localisation: Path A – reference path from original Helpdesk log process graph; Path B – candidate path from *Helpdesk_2* process graph

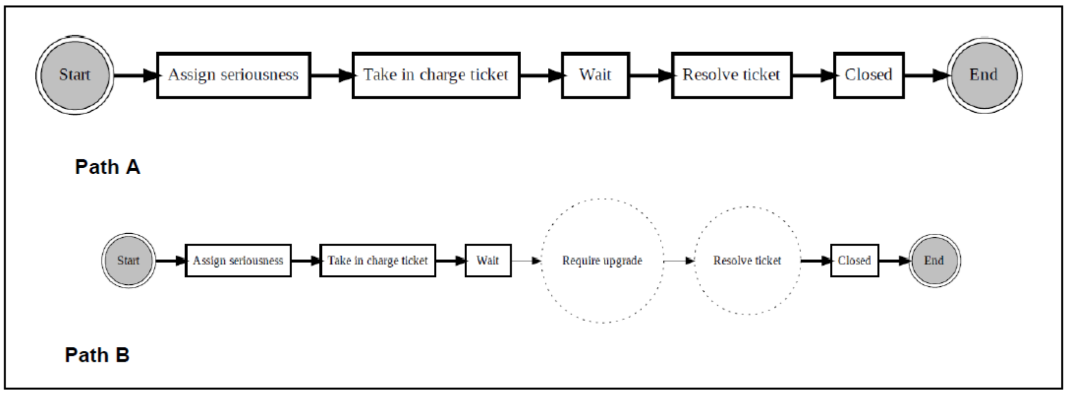


FIGURE 5.10: Path graphs depicting drift localisation: Path A – reference path from original Helpdesk log process graph; Path B – candidate path from *Helpdesk*_3 process graph

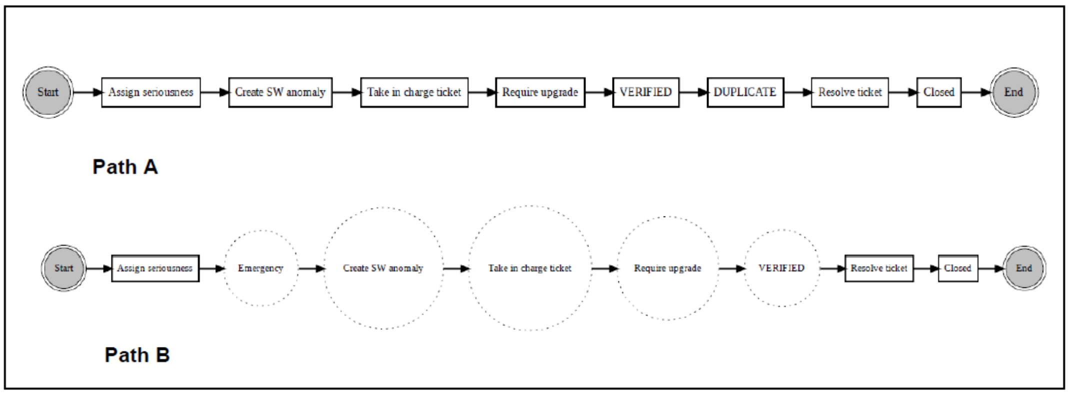


FIGURE 5.11: Path graphs depicting drift localisation: Path A – reference path from original Helpdesk log process graph; Path B – candidate path from *Helpdesk*_4 process graph

5.3.3 Localising Drifts in BPIC 2015 Logs

Two examples of drift localisation are presented for BPIC 2015. The drifts can be easily localised in the same manner for the other pairs of the BPIC 2015 logs.

In Figure 5.12, one path from the graph generated using the BPIC2015_1 reference window is compared to the closest path detected by the proposed drift localisation method in the graph generated using the BPIC2015_2 reference window. It can be seen from the figure that before the drift (path A), 46 activities were executed sequentially. After the drift (path B), some of those activities were replaced, while other new activities were introduced. For example, after the drift (path B), activities ’01_HOOFD_015’ and ’01_HOOFD_030_2’ in the reference graph were replaced with activities ’05_EIND_010’ and ’03_GBH_005’ in the detection graph. It can also be noticed that after the drift (path B), some activities such as ’01_HOOFD_061’ and ’01_HOOFD_510_2’ executed in path A happen sporadically.

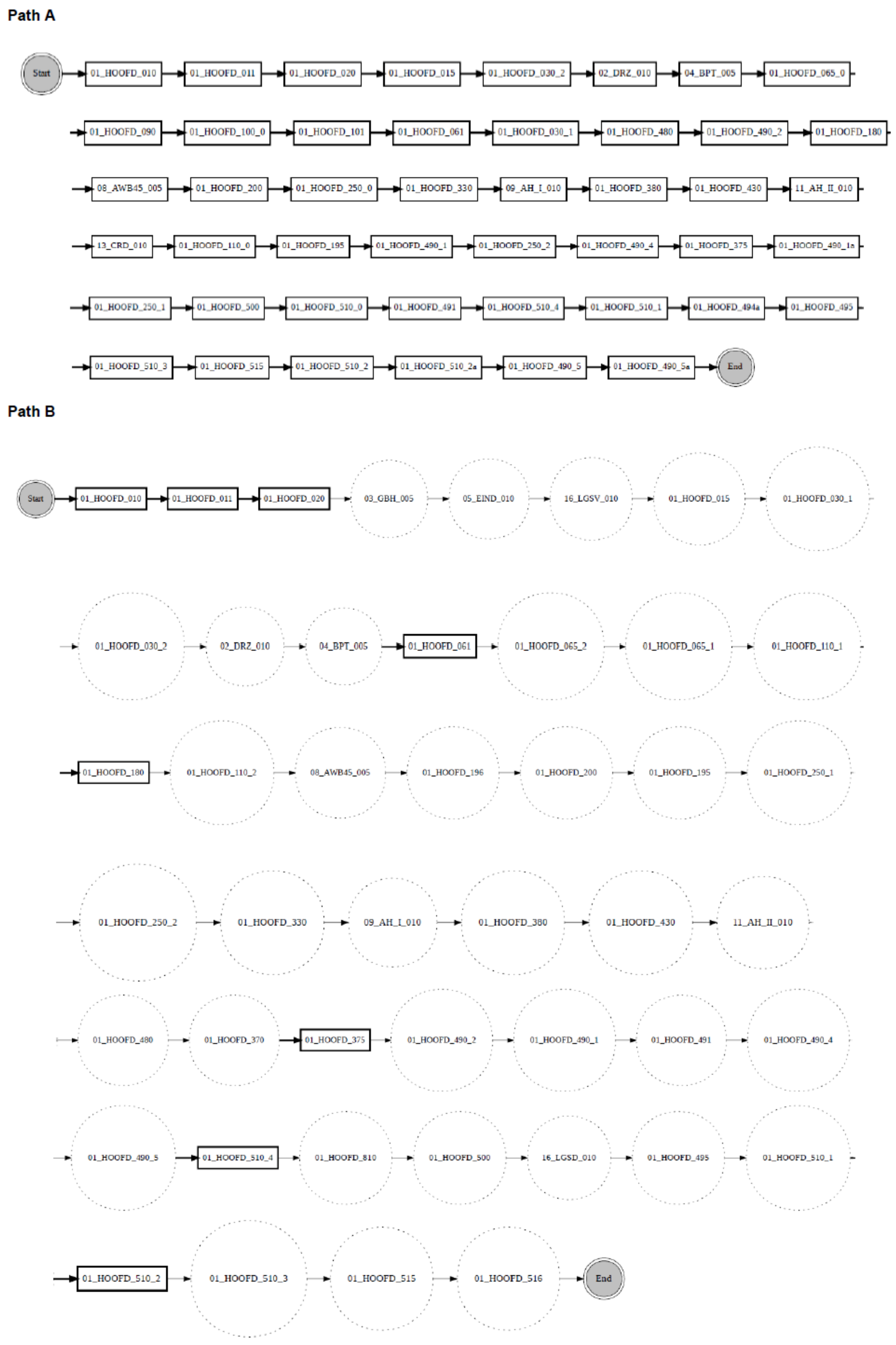


FIGURE 5.12: Drift localisation: Path A – reference path from BPIC 2015 reference process graph; Path B – candidate path from BPIC 2015 detection process graph

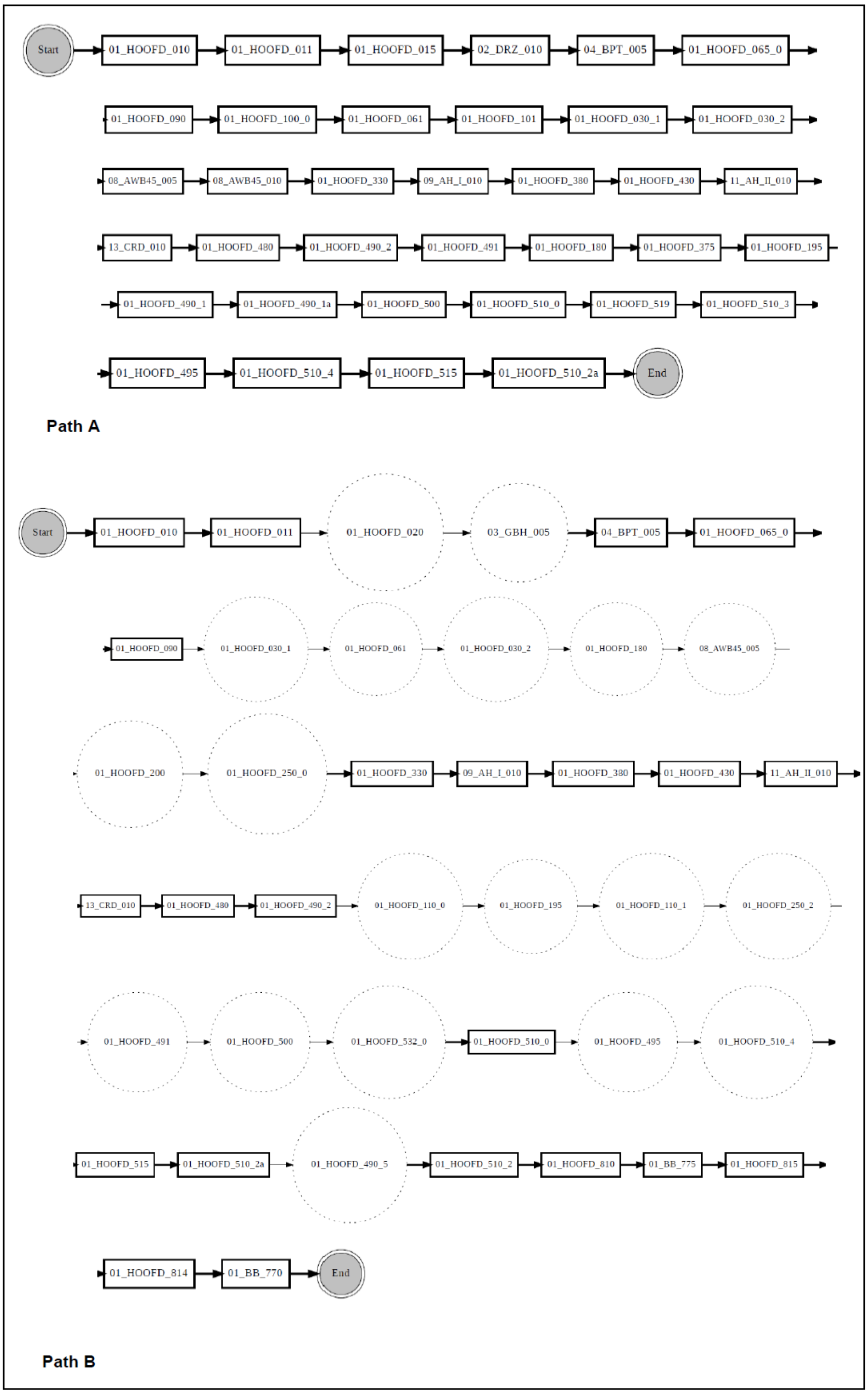


FIGURE 5.13: Path graphs depicting drift localisation: Path A – reference path from BPIC2015_1 process graph; Path B – candidate path from BPIC2015_2 process graph

In Figure 5.13, one path from the process graph of BPIC15_1 is compared to the closest path from the process graph of BPIC15_2 as detected by the proposed drift localisation method. It can be seen from the figure that before the drift (path A), 36 activities were executed sequentially, whereas after the drift (path B), some of those activities were replaced, while other new activities were introduced. For example, after the drift (path B), activities '01_HOOFD_010' and '02_DRZ_010' were replaced by activities '01_HOOFD_020' and '03_GBH_005'. Furthermore, before the drift (path A), the process execution ends after activity '01_HOOFD_510_2a', whereas seven new activities were executed after drift (path B) '01_HOOFD_510_2a'. The drifts can be easily localised similarly to the other pairs of the BPIC2015 logs.

5.4 Summary

This chapter introduced a new method, called PGraphDL, for localising process drifts from both event streams and event logs. According to the method, two process graphs constructed based on two different windows (reference and detection) are taken as input. A user selects any path of interest from the reference process model by specifying an index. The best matching path is searched in the detection graph for each possible path in the reference graph by computing the positional score for each candidate path in the detection graph. The method can help visually and analytically localise the parts of a process that have changed from graphs representing two different periods of that process.

Chapter 6

Conclusion and Future Directions

This chapter summarises the contributions of the thesis and suggests future directions for the current work. The thesis studies the impact of combining DL and graph-based methods on solving some PM tasks. The objectives identified in the thesis are mainly concerned with (i) building a high-performing LSTM model for predicting business process activity sequences from event logs; (ii) explaining the decision-making process of the LSTM model when predicting the next activities in business processes via DFGs; (iii) detecting and localising sudden concept drift in business processes.

6.1 Summary

PM emerged from BPM to help organisations discover, assess, and improve workflows using processes recorded in event logs. It is used to automatically extract process models from event logs to analyse how processes are being implemented in reality, thereby providing the avenue for monitoring and improvement. Several approaches to PM have been proposed in the literature. In particular, probabilistic-based approaches, frequency-based heuristics, genetic-based heuristics, theory of regions, and hybrid methods have been used for process discovery (Augusto et al., 2019). The applicability and effectiveness of these approaches depend on event log features and the structure of the processes. When applied to real-life event logs, the majority of existing process discovery methods demonstrate limitations such as producing broad and spaghetti-like models or models with poor fitness and precision. In other words, these methods cannot discover process models that would express the observed behaviour in the best possible way. It has not proven easy to achieve a compromise between the four process discovery quality metrics; fitness, precision, generalisation and complexity (Augusto et al., 2019).

DL approaches have recently gained popularity in the PM field. In particular, several RNN architectures such as LSTM have been applied to predict next activities in business processes, time of occurrence and completion, and resources that trigger the activities. LSTM can retain information over a long period of time; as such, it can learn long-term dependencies in a sequence, preventing prior information from being lost (Hochreiter and Schmidhuber, 1997). This quality enables an LSTM model to consider all process instances when making a prediction. Compared to the earlier PM techniques, LSTM-based methods have shown better performance in terms of prediction accuracy and generalising ability.

Noise, duplicate tasks, and undetected or missing tasks often found in real-life event logs are challenging issues for PM algorithms to address. Thus, algorithm performance is highly dependent on event log quality and associated processes (Pérez-Alfonso et al., 2015). Because of that, PM methods applied to unstructured logs discover process models that are often purposely unsound representations of the actual processes, thus compromising the quality criteria. While process models are visually explainable, their implausibility makes them less reliable for making predictions (Evermann, Rehse, and Fettke, 2017). As a result, there are debates in the PM community on which methods to use. Recent studies such as (Evermann, Rehse, and Fettke, 2017) and (Tax, Teinemaa, and Zelst, 2020) consider the DL's ability to generalise and yield high prediction accuracy more imperative than generating visually explainable process models represented using graphical notation such as Petri-nets, the BPMN language, Event-driven Process Chains or UML activity diagrams (Tello-Leal et al., 2018). Unlike these studies, we argue that both predictive accuracy and explainability are crucial in the PM field. In existing models generated using DL-based methods, the process structures are implicitly rejected, while no visually explainable process graphs are produced, which limits the value of such models.

Business processes are also prone to constant changes, motivated by numerous internal and external factors. The changes can be due to seasonal trends, constantly evolving market conditions, and customer preferences, and the introduction of new rules and regulations. Changes push organisations to adapt and update their business processes constantly. To proactively respond to these changes, also known as concept drifts, businesses require mechanisms to detect and analyse them. In this regard, several techniques have been proposed in the literature to detect process drifts, i.e. statistically significant changes in the process behaviour. However, the techniques that are being used currently have some limitations. First, they do not work with event streams and, as such, cannot detect inter-trace drifts or detect them with a long delay. Although detecting drifts in an offline setting from historical event logs is equally important for making investigations, organisations can fully utilise the benefits of drift detection in an online setting (i.e. over streams of events). This can enable process stakeholders to take timely corrective measures and avoid or reduce the impact of unintended consequences. Furthermore, existing techniques do not perform well with highly variable business processes, e.g. hospital processes, whose logs feature high trace variability. Finally, they only focus on detecting drifts in event logs without providing any visual solution for localising process changes underpinning the drifts.

This thesis fulfils the following four objectives:

1. Build an accurate LSTM model for predicting business process activity sequences from event logs;

2. Construct a DFG explaining the decision-making process of the LSTM model when predicting next activities in business processes;

3. Detect and localise concept drift in business processes;

4. Validate the research using real-life event logs.

To address the first and second objectives, Chapter 3 presented a new approach to PM that combines the benefits of widely used graph-based methods for process discovery and DL methods for predicting sequences. The proposed approach consists of two stages: building an accurate LSTM model for predicting business process activity sequences based on event logs and generating a DFG explaining the decision-making process of the LSTM model when predicting the next activities in business processes. According to the approach, an LSTM model is first trained on an event log. This model is then employed to find the probabilities for each activity present in the log to appear in the business process next. Finally, these probabilities are used to generate a visually explainable process model graph that represents the decision-making process of the LSTM model. A probability threshold is introduced as a parameter to manage the graph complexity and thus enable faster and more directed business process analysis, including discovering the most common event sequences and unusual or suspicious behaviours.

In this thesis, two model architectures were tested: one using a unidirectional LSTM and another using a bidirectional LSTM. Both architectures were demonstrated to outperform the state-of-the-art LSTM models for PM based on real-life event logs. The better performance of the proposed models can be attributed to a different way of pre-processing event logs when generating inputs for the models and the model network architectures that employ an embedding and a dense layer, in addition to an LSTM layer (unidirectional and bidirectional for the first and second models, respectively). At the same time, the bidirectional LSTM model achieved slightly better results than the unidirectional LSTM on more complex or larger logs, but at the cost of training time.

The ability of generated graphs to explain the decision-making process of the LSTM models was visually demonstrated. An approach was proposed to determine the similarity between the graphs to further validate the generalising ability of the models, which gave a satisfactory result. In particular, graphs can be used to understand the performance of LSTM models and perform a variety of PM tasks such as process discovery, conformance check, and investigation of non-compliance cases.

To tackle the third and fourth objectives, Chapter 4, presented two novel methods, PGraphDD-QM and PGraphDD-SS, for detecting sudden concept drifts in dynamic business processes.

For these tasks, we use the uni-directional over the bi-directional LSTM model because of its simplicity and faster training time. The first drift detection method involves training an LSTM model on a previous event log (off-line scenario) or stream of events (on-line scenario) and applying it to a newly generated log/stream. The model performance on the old and new logs/streams is then compared based on the F-score metric. Similar F-score values over the two logs/streams indicates that there is no drift, whereas a drop of the F-score value over the new log/stream below a set threshold indicates the presence of drift. The more significant the drop in the F-score value, the more changes are expected in the new log/stream compared to the old one.

The second drift detection method involves training two LSTM models on event logs (offline scenario) or streams of events (online scenario) covering two different periods. These models are then used to generate two graphs representing business processes for the two analysed periods as believed by the LSTM models. Next, two adjacency matrices are generated based on the two graphs to measure the similarity between the two business processes. A drop in the similarity score below a set threshold indicates a drift. The more significant drop in the similarity score, the more changes are expected to be present in the new log/stream compared to the old one.

An evaluation of the proposed methods for detecting drifts in business processes using synthetic and real-life logs demonstrated that the methods performed on par with state-of-the-art methods, achieving similar accuracy in detecting drifts, with shorter delays compared to the other methods, while also offering the following advantages over existing solutions:

- The majority of the existing drift detection approaches concentrate on detecting drifts through their effects on the resulting process instances, i.e. cases in event logs, instead of detecting drifts by comparing process models depicting different process instances before and after the drifts have occurred. The latter approach is adopted in the methods proposed in this thesis. In particular, the proposed methods use and compare process graph models constructed based on two windows (or two different time periods), reference, and detection.

- Unlike the methods based on the statistical analysis of graphs representing business processes, the proposed methods employ DL that does not require the user to construct features.
- The proposed methods employed graphs that explain the decision-making process of the DL models trained to predict the next activities in business processes, thus allowing the user to verify what has changed in the process when a drift is detected.
- The relative insensitivity of the DL models with proposed LSTM architectures to interval length may have contributed to detecting drifts with minimal delay.

Finally, Chapter 5 presented a new method, called PGraphDL, for localising detected process drifts. According to the method, two process graphs constructed based on two different windows (or time periods), reference and detection, are taken as input. The user is asked to select any path of interest from the reference graph by specifying an index. The best matching path is then searched in the graph generated for the detection window for each possible path in the reference graph by computing the positional score for each candidate path in the detection graph. The method can visually and analytically localise the parts of the process that have changed from graphs representing two different periods of a business process.

6.2 Potential Applications

The proposed methods can find use in many different application domains. For example, adopting methods in healthcare may provide efficient solutions for saving lives. In particular, the proposed methods can be used to discover disease trajectories, which can reveal disease correlations and temporal disease progression, thus equipping clinicians with tools for predicting and preventing future complications in individual patients (Kusuma et al., 2020). Previous studies have based their solutions for discovering disease trajectories on statistical analysis (Jensen et al., 2014) and knowledge graphs (Vlietstra et al., 2020). Both approaches have limitations: While the former approach is prone to statistical bias, the latter is not scalable and requires significant expert input. Furthermore, both approaches are not designed to track changes in disease trajectories over time, e.g., to study the implications of the COVID-19 pandemic on population health and determine the subset of comorbidities contributing to the death outcome following coronavirus disease. In contrast, the proposed methods can be easily applied to temporal sequences of diagnoses extracted from electronic healthcare records at scale and in a continuous manner, allowing clinicians to track changes in disease progression patterns, perform temporal analysis of comorbidities using the generated graphs, and predict patient outcomes based on their history.

The oil and gas industry is another complex domain where the proposed methods can be useful for preventing theft, improving the efficiency of supply chain management, and adapting to world events promptly. The industry involves many interconnected tasks and parties (Hanga and Kovalchuk, 2019). The actions of every party can be logged, and cases of non-compliance can be identified easily by visualising the intended and actual processes as graphs. Any changes and weaknesses in supply chain management can be equally efficiently spotted through detecting and localising drifts in the discovered process graphs evolving over time.

Purchasing and supply chain activities increasingly demand higher transparency, traceability (Kshetri, 2018) and cyber security (Ivanov, Dolgui, and Sokolov, 2019). This has led to the adoption of advanced technologies such as RFID, IoT sensors (Bienhaus and Haddud, 2018) and Blockchain (Centobelli et al., 2021) to cope with these demands. These technologies are likely to increase the volume and granularity of event data across various supply chains. This is another avenue where the proposed methods can be helpful.

It should be noted that the proposed methods are designed to detect changes in business processes rather than predict process outcomes. For example, the methods can detect changes in the process required from bank staff to follow to approve a loan, while they are not suitable for predicting whether a specific loan application will be approved or not.

6.3 Future Work

There are many potential directions for future work to extend the work presented in this thesis. Some examples of possible extensions of the proposed methods are listed below.

6.3.1 Predicting Next Activities in Business Processes

In this thesis, the LSTM model is used to predict the next activities in business processes. Some identified future research directions are as follows:

- Experiment with other DL architectures, such as encoder-decoder networks. One advantage of the proposed approach in this thesis is its model-agnostic nature. This means that the graph generation part is independent of the predictive modelling part. As such, any model can be used in place of the LSTM.
- Use the proposed approach as a conformance checking method. To achieve this, the graphs generated according to the proposed methods can be evaluated using the metrics widely used in the PM community for conformance checking. In particular, conformance comprises several orthogonal dimensions such as fitness, precision, generalisation, and structure (Rozinat and Aalst, 2008; Weerdt et al., 2010).
- Evaluate the proposed methods on other real-life event logs.

6.3.2 Graphs Explaining Decisions of LSTM Models

In this thesis, DFGs were used to explain the decision-making process of LSTM models trained to predict the next activities of business process. These DFGs appear to be simpler than the models discovered using Petri-nets and BPMN. The potential extensions of the presented work in this regard include the following:

- Improve the generation process of DFGs, and, in particular, distinguish concurrency and causality relations by possibly adding symbols or using different colours for arcs.

- Apply the proposed methods to operational processes, such as analysing hospital treatment processes and improving customer service.

6.3.3 Concept Drift Detection and Localisation

The proposed methods for detecting and localising drifts in business processes can be extended in the following ways:

- As described in Chapter 2, there are four classes of drifts: sudden, gradual, recurring, and incremental. The drift detection methods presented in Chapter 4 focus on detecting sudden drifts only. As such, when applied to event streams containing other classes of drifts, the proposed methods may detect these drifts as sudden or fail to detect them. Therefore, another avenue for future work is to extend the methods to detect other classes of drifts. Several methods have been proposed in the literature for detecting other classes of drifts from event streams. For example, the methods proposed in this thesis can be easily extended by applying the same strategy used in (Maaradji et al., 2017) to detect gradual drifts in trace streams. That is, apply PGraphDD-QM and PGraphDD-SS to two consecutive sudden drifts to determine whether they represent separate changes or define the start and end of a single gradual drift.

- Extend the drift localisation method by enumerating all detection paths in descending order of proximity to the reference path.

- Evaluate the proposed methods using other real-life event logs and explore the ways of applying the methods to analyse chains of events affecting the stock market, for example.

- Package the proposed methods as a library for existing popular PM software and provide users with an interactive visualisation tool for extensive exploration of changes in case they are more significant than those considered in this study.

Bibliography

Aalst, MP Wil van der (2011a). “Process Mining. Discovery, Conformance and Enhancement of Business Processes”. In:

Aalst, Wil van der (2011b). “Using process mining to bridge the gap between BI and BPM”. In: *Computer* 44.12, pp. 77–80.

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

Aalst, Wil MP van der (2010). “Process discovery: Capturing the invisible”. In: *IEEE Computational Intelligence Magazine* 5.1, pp. 28–41.

— (2012). “Distributed process discovery and conformance checking”. In: *International Conference on Fundamental Approaches to Software Engineering*. Springer, pp. 1–25.

Aalst, Wil MP Van der (2016). *Process mining: data science in action*. Springer.

Abadi, Martín et al. (2016). “Tensorflow: A system for large-scale machine learning”. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283.

Alvarez, Camilo et al. (2018). “Discovering role interaction models in the Emergency Room using Process Mining”. In: *Journal of biomedical informatics* 78, pp. 60–77.

Armas-Cervantes, Abel et al. (2014). “Behavioral comparison of process models based on canonically reduced event structures”. In: *International Conference on Business Process Management*. Springer, pp. 267–282.

Augusto, Adriano et al. (2018). “Automated discovery of process models from event logs: Review and benchmark”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.4, pp. 686–705.

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

Beest, Nick RTP van et al. (2016). “Log delta analysis: Interpretable differencing of business process event logs”. In: *International Conference on Business Process Management*. Springer, pp. 386–405.

Bengio, Yoshua (2008). “Neural net language models”. In: *Scholarpedia* 3.1, p. 3881.

Bienhaus, Florian and Abubaker Haddud (2018). “Procurement 4.0: factors influencing the digitisation of procurement and supply chains”. In: *Business Process Management Journal*.

Bifet, Albert and Ricard Gavalda (2007). “Learning from time-changing data with adaptive windowing”. In: *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, pp. 443–448.

Bolt, Alfredo, Wil MP van der Aalst, and Massimiliano De Leoni (2017). “Finding process variants in event logs”. In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, pp. 45–52.

Bolt, Alfredo, Massimiliano de Leoni, and Wil MP van der Aalst (2018). “Process variant comparison: using event logs to detect differences in behavior and business rules”. In: *Information Systems* 74, pp. 53–66.

Bose, RP Jagadeesh Chandra et al. (2011). “Handling concept drift in process mining”. In: *International Conference on Advanced Information Systems Engineering*. Springer, pp. 391–405.

Bose, RP Jagadeesh Chandra et al. (2013). “Dealing with concept drifts in process mining”. In: *IEEE transactions on neural networks and learning systems* 25.1, pp. 154–171.

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

Broucke, Seppe KLM vanden and Jochen De Weerdt (2017). “Fodina: a robust and flexible heuristic process discovery technique”. In: *decision support systems* 100, pp. 109–118.

Brownlee, Jason (2016). *Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras*. Machine Learning Mastery.

— (2017). *Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems*. Machine Learning Mastery.

Buijs, Joos CAM, Boudewijn F van Dongen, and Wil MP van Der Aalst (2012). “On the role of fitness, precision, generalization and simplicity in process discovery”. In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, pp. 305–322.

Buijs, Joos CAM and Hajo A Reijers (2014). “Comparing business process variants using models and event logs”. In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, pp. 154–168.

Camargo, Manuel, Marlon Dumas, and Oscar González-Rojas (2019). “Learning accurate LSTM models of business processes”. In: *International Conference on Business Process Management*. Springer, pp. 286–302.

Carmona, Josep and Ricard Gavalda (2012). “Online techniques for dealing with concept drift in process mining”. In: *International Symposium on Intelligent Data Analysis*. Springer, pp. 90–102.

Centobelli, Piera et al. (2021). “Surfing blockchain wave, or drowning? Shaping the future of distributed ledgers and decentralized technologies”. In: *Technological Forecasting and Social Change* 165, p. 120463.

Conforti, Raffaele et al. (2016). “BPMN Miner: Automated discovery of BPMN process models with hierarchical structure”. In: *Information Systems* 56, pp. 284–303.

Dakic, Dusanka et al. (2018). “BUSINESS PROCESS MINING APPLICATION: A LITERATURE REVIEW.” In: *Annals of DAAAM & Proceedings* 29.

De Weerdt, Jochen et al. (2012). “A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs”. In: *Information Systems* 37.7, pp. 654–676.

Di Francescomarino, Chiara et al. (2017). “An eye into the future: leveraging a-priori knowledge in predictive business process monitoring”. In: *International Conference on Business Process Management*. Springer, pp. 252–268.

Di Francescomarino, Chiara et al. (2018). “Predictive process monitoring methods: Which one suits me best?” In: *International conference on business process management*. Springer, pp. 462–479.

Dongen, B.F. (Boudewijn) van (2015). *Business Process Intelligence (BPI) Challenge 2015*. URL: `https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1`.

Dongen, Boudewijn van (Apr. 2012). “BPI Challenge 2012”. In: DOI: `10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`. URL: `https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204`.

Dumas, Marlon et al. (2013). *Fundamentals of business process management*. Vol. 1. Springer.

Elkhawaga, Ghada et al. (2020). “CONDA-PM—A Systematic Review and Framework for Concept Drift Analysis in Process Mining”. In: *Algorithms* 13.7, p. 161.

Ellson, John et al. (2001). “Graphviz—open source graph drawing tools”. In: *International Symposium on Graph Drawing*. Springer, pp. 483–484.

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

Ferreira, Diogo R (2017). *A primer on process mining*. Springer.

Ferreira, Diogo R and Cláudia Alves (2011). “Discovering user communities in large event logs”. In: *International Conference on Business Process Management*. Springer, pp. 123–134.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Graves, Alex and Jürgen Schmidhuber (2005). “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. In: *Neural networks* 18.5-6, pp. 602–610.

Gulli, Antonio and Sujit Pal (2017). *Deep learning with Keras*. Packt Publishing Ltd.

Günther, Christian W and Wil MP Van Der Aalst (2007). “Fuzzy mining–adaptive process simplification based on multi-perspective metrics”. In: *International conference on business process management*. Springer, pp. 328–343.

Gutiérrez, Luis and Brian Keith (2018). “A systematic literature review on word embeddings”. In: *International Conference on Software Process Improvement*. Springer, pp. 132–141.

Hagberg, Aric, Pieter Swart, and Daniel S Chult (2008). *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Hanga, Khadijah Muzzammil, Yevgeniya Kovalchuk, and Mohamed Medhat Gaber (2020). “A Graph-Based Approach to Interpreting Recurrent Neural Networks in Process Mining”. In: *IEEE Access* 8, pp. 172923–172938.

— (2022). “PGraphD*: Methods for Drift Detection and Localisation Using Deep Learning Modelling of Business Processes”. In: *Entropy* 24.7, p. 910.

Hanga, K.M and Y. Kovalchuk (2019). “Machine learning and multi-agent systems in oil and gas industry applications: A survey”. In: *Computer Science Review* 34, p. 100191. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2019.08.002.

Hao, Xing, Guigang Zhang, and Shang Ma (2016). “Deep learning”. In: *International Journal of Semantic Computing* 10.03, pp. 417–439.

Harremoës, Peter and Gábor Tusnády (2012). “Information divergence is more χ 2-distributed than the χ 2-statistics”. In: *2012 IEEE International Symposium on Information Theory Proceedings*. IEEE, pp. 533–537.

Harris, Zellig S (1954). “Distributional structure”. In: *Word* 10.2-3, pp. 146–162.

Hassani, Marwan (2019). “Concept Drift Detection Of Event Streams Using An Adaptive Window.” In: *ECMS*, pp. 230–239.

Hinkka, Markku, Teemu Lehto, and Keijo Heljanko (2018). “Exploiting event log event attributes in RNN based prediction”. In: *Data-Driven Process Discovery and Analysis*. Springer, pp. 67–85.

Ho, Shen-Shyang (2005). “A martingale framework for concept change detection in time-varying data streams”. In: *Proceedings of the 22nd international conference on Machine learning*, pp. 321–327.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.

Hompes, BFA et al. (2015a). "Detecting changes in process behavior using comparative case clustering". In: *International Symposium on Data-Driven Process Discovery and Analysis*. Springer, pp. 54–75.

Hompes, BFA et al. (2015b). "Discovering deviating cases and process variants using trace clustering". In: *Proceedings of the 27th Benelux Conference on Artificial Intelligence (BNAIC), November*, pp. 5–6.

Ivanov, Dmitry, Alexandre Dolgui, and Boris Sokolov (2019). "The impact of digital technology and Industry 4.0 on the ripple effect and supply chain risk analytics". In: *International Journal of Production Research* 57.3, pp. 829–846.

Jensen, Anders Boeck et al. (2014). "Temporal disease trajectories condensed from population-wide registry data covering 6.2 million patients". In: *Nature Communications* 5.4022.

Jokonowo, Bambang et al. (2018). "Process Mining in Supply Chains: A Systematic". In: *International Journal of Electrical and Computer Engineering (IJECE)* 8.6, pp. 4626–4636.

Ko, Ryan KL, Stephen SG Lee, and Eng Wah Lee (2009). "Business process management (BPM) standards: a survey". In: *Business Process Management Journal*.

Kratsch, Wolfgang et al. (2021). "Machine learning in business process monitoring: a comparison of deep learning and classical approaches used for outcome prediction". In: *Business & Information Systems Engineering* 63.3, pp. 261–276.

Kshetri, Nir (2018). "1 Blockchain's roles in meeting key supply chain management objectives". In: *International Journal of information management* 39, pp. 80–89.

Kurniati, Angelina Prima, Guntur Kusuma, and Gede Wisudiawan (2016). "Implementing heuristic miner for different types of event logs". In: *vol* 11, pp. 5523–5529.

Kusuma, G. et al. (2020). "Process Mining of Disease Trajectories: A Feasibility Study". In: *Proceedings of the 13th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2020)*, pp. 705–712.

Lakshmanan, Geetika T et al. (2015). "A markov prediction model for data-driven semi-structured business processes". In: *Knowledge and Information Systems* 42.1, pp. 97–126.

Lau, Henry CW et al. (2009). "Development of a process mining system for supporting knowledge discovery in a supply chain network". In: *International Journal of Production Economics* 122.1, pp. 176–187.

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

Leemans, Sander J. J. (2015). "Process Discovery and Exploration". In: *Business Process Management Workshops*. Ed. by Fabiana Fournier and Jan Mendling. Cham: Springer International Publishing, pp. 582–585. ISBN: 978-3-319-15895-2.

Leemans, Sander JJ, Dirk Fahland, and Wil MP Van Der Aalst (2013). "Discovering block-structured process models from event logs-a constructive approach". In: *International conference on applications and theory of Petri nets and concurrency*. Springer, pp. 311–329.

Leoni, M. (Massimiliano) de and Felix Mannhardt (Feb. 2015). "Road Traffic Fine Management Process". In: DOI: `10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5`. URL: `https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249`.

Lewis, Nigel Da Costa (2016). *Deep Time Series Forecasting with Python: An Intuitive Introduction to Deep Learning for Applied Time Series Modeling*. ND Lewis.

Li, Tianyang et al. (2017). “Unraveling process evolution by handling concept drifts in process mining”. In: *2017 IEEE International Conference on Services Computing (SCC)*. IEEE, pp. 442–449.

Li, Yang and Tao Yang (2018). “Word embedding for understanding natural language: a survey”. In: *Guide to big data applications*. Springer, pp. 83–104.

Lin, Li, Lijie Wen, and Jianmin Wang (2019). “MM-Pred: a deep predictive model for multi-attribute event sequence”. In: *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, pp. 118–126.

Liu, Na, Jiwei Huang, and Lizhen Cui (2018). “A framework for online process concept drift detection from event streams”. In: *2018 IEEE International Conference on Services Computing (SCC)*. IEEE, pp. 105–112.

Lu, Xixi et al. (2016). “Detecting deviating behaviors without models”. In: *International Conference on Business Process Management*. Springer, pp. 126–139.

Maaradji, Abderrahmane et al. (2015). *Business Process Drift*. DOI: 10.4121/uuid:aa01a720-4616-43e9-af67-370942019f48. URL: https://data.4tu.nl/articles/dataset/Business_Process_Drift/12712436/1.

Maaradji, Abderrahmane et al. (2016). “Fast and accurate business process drift detection”. In: *International Conference on Business Process Management*. Springer, pp. 406–422.

Maaradji, Abderrahmane et al. (2017). “Detecting sudden and gradual drifts in business processes from execution traces”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.10, pp. 2140–2154.

Manoj Kumar, MV, Likewin Thomas, and B Annappa (2015). “Capturing the sudden concept drift in process mining”. In: *Algorithms & Theories for the Analysis of Event Data (ATAED’15, Brussels, Belgium, June 22-23, 2015)*, p. 132.

Martin, Niels et al. (2015). “An Exploration and Analysis of The Building Permit Application Process in Five Dutch Municipalities”. In: *Report for the BPI Challenge* 72, p. 73.

Martjushev, J, RP Jagadeesh Chandra Bose, and Wil MP van der Aalst (2015). “Change point detection and dealing with gradual and multi-order dynamics in process mining”. In: *International Conference on Business Informatics Research*. Springer, pp. 161–178.

Mauro, Nicola Di, Annalisa Appice, and Teresa Basile (2019). “Activity prediction of business process instances with inception CNN models”. In: *International conference of the italian association for artificial intelligence*. Springer, pp. 348–361.

Medeiros, Ana Karla A de, Anton JMM Weijters, and Wil MP van der Aalst (2007). “Genetic process mining: an experimental evaluation”. In: *Data Mining and Knowledge Discovery* 14.2, pp. 245–304.

Mueller, Ronald and Imaan Ali (2021). *Process mining vs. Business Process Management*. URL: https://www.macrosoftinc.com/process-mining-vs-business-process-management/.

Neu, Dominic A, Johannes Lahann, and Peter Fettke (2021). “A systematic literature review on state-of-the-art deep learning methods for process prediction”. In: *Artificial Intelligence Review*, pp. 1–27.

Nguyen, Hoang et al. (2018). “Multi-perspective comparison of business process variants based on event logs”. In: *International Conference on Conceptual Modeling*. Springer, pp. 449–459.

Ostovar, Alireza, Sander JJ Leemans, and Marcello La Rosa (2020). “Robust drift characterization from event streams of business processes”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 14.3, pp. 1–57.

Ostovar, Alireza et al. (2016). “Detecting drift from event streams of unpredictable business processes”. In: *International Conference on Conceptual Modeling*. Springer, pp. 330–346.

Pasquadibisceglie, Vincenzo et al. (2019). “Using convolutional neural networks for predictive process analytics”. In: *2019 international conference on process mining (ICPM)*. IEEE, pp. 129–136.

Paszkiewicz, Zbigniew (2013). “Process mining techniques in conformance testing of inventory processes: an industrial application”. In: *International Conference on Business Information Systems*. Springer, pp. 302–313.

Pérez-Alfonso, Damián et al. (2015). “Recommendation of Process Discovery Algorithms Through Event Log Classification”. In: *Mexican Conference on Pattern Recognition*. Springer, pp. 3–12.

Polato, Mirko (July 2017). “Dataset belonging to the help desk log of an Italian Company”. In: DOI: 10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb. URL: https://data.4tu.nl/articles/dataset/Dataset_belonging_to_the_help_desk_log_of_an_Italian_Company/12675977.

— (2020). *Dataset belonging to the help desk log of an Italian Company*. URL: https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb.

Rama-Maneiro, Efrén, Juan Vidal, and Manuel Lama (2021). “Deep learning for predictive business process monitoring: Review and benchmark”. In: *IEEE Transactions on Services Computing*.

Repta, Dragos et al. (2018). “Automated process recognition architecture for cyber-physical systems”. In: *Enterprise Information Systems* 12.8-9, pp. 1129–1148.

Rojas, Eric et al. (2016). “Process mining in healthcare: A literature review”. In: *Journal of biomedical informatics* 61, pp. 224–236.

Roldán, Juan Jesús et al. (2018). “Analyzing and improving multi-robot missions by using process mining”. In: *Autonomous Robots* 42.6, pp. 1187–1205.

Rozinat, Anne and Wil MP Van der Aalst (2008). “Conformance checking of processes based on monitoring real behaviour”. In: *Information Systems* 33.1, pp. 64–95.

Schonenberg, Helen et al. (2008). “Process flexibility: A survey of contemporary approaches”. In: *Advances in enterprise engineering I*. Springer, pp. 16–30.

Schuster, Mike and Kuldip K Paliwal (1997). “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11, pp. 2673–2681.

Seeliger, Alexander, Timo Nolle, and Max Mühlhäuser (2017). “Detecting concept drift in processes using graph metrics on process graphs”. In: *Proceedings of the 9th Conference on Subject-Oriented Business Process Management*, pp. 1–10.

Sousa, Rafael Gaspar de et al. (2021). “Concept drift detection and localization in process mining: an integrated and efficient approach enabled by trace clustering”. In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pp. 364–373.

Srinivasan, S (2017). *Guide to big data applications*. Vol. 26. Springer.

Steeman, Ward (Apr. 2013a). “BPI Challenge 2013, closed problems”. In: DOI: 10.4121/uuid:c2c3b154-ab26-4b31-a0e8-8f2350ddac11. URL: https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_closed_problems/12714476.

— (Apr. 2013b). “BPI Challenge 2013, incidents”. In: DOI: 10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee. URL: https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_incidents/12693914.

Stefanini, Alessandro et al. (2018). “Performance analysis in emergency departments: a data-driven approach”. In: *Measuring Business Excellence*.

Storn, Rainer and Kenneth Price (1997). "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11.4, pp. 341–359.

Suriadi, Suriadi et al. (2013). "Understanding process behaviours in a large insurance company in Australia: A case study". In: *International Conference on Advanced Information Systems Engineering*. Springer, pp. 449–464.

Syamsiyah, Alifah et al. (2017). "Business process comparison: A methodology and case study". In: *International Conference on Business Information Systems*. Springer, pp. 253–267.

Syring, Anja F, Niek Tax, and Wil MP van der Aalst (2019). "Evaluating conformance measures in process mining using conformance propositions". In: *Transactions on Petri Nets and Other Models of Concurrency XIV*. Springer, pp. 192–221.

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

Tax, Niek et al. (2017). "Predictive business process monitoring with LSTM neural networks". In: *International Conference on Advanced Information Systems Engineering*. Springer, pp. 477–492.

Teinemaa, Irene et al. (2019). "Outcome-oriented predictive process monitoring: review and benchmark". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13.2, pp. 1–57.

Tello-Leal, Edgar et al. (2018). "Predicting Activities in Business Processes with LSTM Recurrent Neural Networks". In: *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*. IEEE, pp. 1–7.

Unuvar, Merve, Geetika T Lakshmanan, and Yurdaer N Doganata (2016). "Leveraging path information to generate predictions for parallel business processes". In: *Knowledge and Information Systems* 47.2, pp. 433–461.

Van Der Aalst, Wil (2012). "Process mining: Overview and opportunities". In: *ACM Transactions on Management Information Systems (TMIS)* 3.2, pp. 1–17.

Van Der Aalst, Wil MP et al. (2007). "Business process mining: An industrial application". In: *Information Systems* 32.5, pp. 713–732.

Van Dongen, Boudewijn F et al. (2005). "The ProM framework: A new era in process mining tool support". In: *International conference on application and theory of petri nets*. Springer, pp. 444–454.

Vlietstra, Wytze J. et al. (2020). "Identifying disease trajectories with predicate information from a knowledge graph". In: *Journal of Biomedical Semantics* 11.9.

Wani, M Arif et al. (2020). *Advances in deep learning*. Springer.

Weber, Barbara, Manfred Reichert, and Stefanie Rinderle-Ma (2008). "Change patterns and change support features–enhancing flexibility in process-aware information systems". In: *Data & knowledge engineering* 66.3, pp. 438–466.

Weerdt, Jochen De et al. (2010). "A critical evaluation study of model-log metrics in process discovery". In: *International Conference on Business Process Management*. Springer, pp. 158–169.

Weijters, AJMM, Wil MP van der Aalst, and A Ana Karla (2006). *de Medeiros."Process Mining with the Heuristics Miner-algorithm"*. Tech. rep. BETA Working Paper Series. Vol. WP 166. 2006 (cit. on pp. 41, 182).

Weijters, AJMM and Joel Tiago S Ribeiro (2011). "Flexible heuristics miner (FHM)". In: *2011 IEEE symposium on computational intelligence and data mining (CIDM)*. IEEE, pp. 310–317.

Xu, Wei and Alexander Rudnicky (2000). "Can artificial neural networks learn language models?" In:

Yeshchenko, Anton et al. (2019). “Comprehensive process drift detection with visual analytics”. In: *International Conference on Conceptual Modeling*. Springer, pp. 119–135.

Zerbino, Pierluigi, Alessandro Stefanini, and Davide Aloini (2021). “Process science in action: A literature review on process mining in business management”. In: *Technological Forecasting and Social Change* 172, p. 121021.

Zheng, Canbin, Lijie Wen, and Jianmin Wang (2017). “Detecting process concept drifts from event logs”. In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, pp. 524–542.