Università Politecnica delle Marche

Doctoral School on Information Engineering
Curriculum "Computer, Management and Automation Engineering"

Encoding & Characterization of process models for Deep Predictive Process Monitoring.

Author:
Andrea Chiorrini

Supervisors:
Prof.ssa Claudia Diamantini
Prof. Domenico Potena

“The greatest challenge to any thinker
is stating the problem in a way that will allow a solution”

Bertrand Russell

Abstract

Ever-increasing digitalization of all aspects of life modifies the operative executions of most human tasks and produces a huge wealth of information, in the form of data logs, that could be leveraged to further improve the general quality of such executions. One way of leveraging such information is to predict how the execution of such tasks will unfold until their completion so as to be capable of supporting the managers in determining, for example, whether to intervene to prevent undesired process outcomes or how to best allocate resources. In the present thesis, it is proposed an approach that uses the information about the parallelism among activities for the Predictive Process Monitoring tasks, by representing process executions with their corresponding Instance Graph and processing them using deep graph convolutional neural networks. Also, to define the scope to best apply such an approach is devised a novel metric that manages to effectively measure the parallelism in a business process model. Lastly, the definition of a set of metrics that describe the execution context of an activity inside a process to represent the activity itself is presented. This is used both to define a querying mechanism for activities in processes and to introduce the notion of "location" as a further relevant prediction target for Predictive Process Monitoring techniques. The proposed techniques have been experimentally evaluated using several real-world datasets and the results are promising.

Abstract

La sempre crescente digitalizzazione di molti aspetti della vita, sta modificando l'esecuzione operativa di molte attività umane, producendo anche una grande quantità di informazione sotto forma di log di dati. Questi possono essere sfruttati per migliorare la qualità di queste esecuzioni. Un modo per sfruttare queste informazioni è usarle per predire come l'esecuzione di un'attività umana possa evolvere fino al suo completamento, così da supportare i manager nel determinare, per esempio, se intervenire per prevenire delle situazioni indesiderate o per meglio allocare le risorse a disposizione. Nella presente tesi, si propone un approccio che usa l'informazione relativa al parallelismo presente tra le attività per eseguire i task tipici del Predictive Process Monitoring. Questo viene fatto rappresentando le esecuzioni di processo con il corrispondente Instance Graph e processandole utilizzando delle graph convolutional neural networks. Inoltre, per definire gli ambiti in cui tale approccio funziona al meglio nel presente elaborato si illustra una nuova metrica ideata per misurare il parallelismo all'interno dei processi di business. Infine, è presentato un insieme di metriche che descrivono il contesto di esecuzione di una attività all'interno di un processo per rappresentare l'attività stessa. Questo è utilizzato sia per definire un meccanismo di "querying" per le attività all'interno dei processi sia per introdurre la nozione di "location" come un ulteriore obiettivo di predizione per le tecniche di Predictive Process Monitoring. Gli approcci proposti sono stati valutati utilizzando vari dataset reali e i risultati ottenuti sono promettenti.

Contents

Chapter 1

Introduction

1.1 Overview

In the last years, we are witnessing a profound and ever-increasing digitalization of most aspects of life. Where it is applied, this digitalization not only modifies the operative execution of tasks but also produces a wealth of data, in the form of logs, that could be leveraged to further improve the general quality of such executions. As a consequence, today's organizations store lots of data from their business processes. These business processes may be the running of a hospital or a factory, granting a loan, or booking a vacancy. In general, we may say that any flow of related activities executed by people and/or machines to achieve a specific goal is a business process. The increasing availability of process-related data enabled the development of advanced techniques to support process analysis and improvements, such as Process Mining ones which can extract knowledge from event logs commonly available in today's information systems. *These techniques provide means to discover, monitor and improve processes in a variety of application domains. There are two main drivers for the growing interest in process mining. On the one hand, more and more events are being recorded, thus, providing detailed information about the history of processes. On the other hand, there is a need to improve and support business processes in competitive and rapidly changing environments.* [1] Lately, with the rise and spread of machine learning a new research field is emerging under the name of *Predictive Process Monitoring* in the process mining scientific community. *Predictive Process Monitoring* can be described as a set of tasks that aim to predict how a running execution of a process will unfold up until its completion. Such tasks may be the outcome of a process execution, its completion time, as well as the sequence of its future activities, or the full execution time of each of them[2]. Such capacity of " looking ahead" during a process execution can support the managers in determining, for example, whether to intervene to prevent undesired process outcomes or how to best allocate resources [3].

Notably, process executions are characterized by (complex) control-flow constructs, like concurrency, choices, and loops [4]. However, these structures are flattened in the event log, since traces only record the sequence of executed activities, possibly with additional data properties. Consequently, a single construct flow can correspond in the event log to several different sequences of events. For instance, a parallel construct involving two or more activities can correspond to a number of sequences equal to all the possible order permutations of the activities. Despite this fact, little work has been done in the literature on leveraging such control-flow information to the predictive process monitoring tasks. Hence, the present thesis.

1.2 Contribution of this work

The present work aims at proposing novel approaches, and methodologies for the Deep Predictive Process Monitoring tasks. The main focus of the thesis and all its proposed contributions regards leveraging the control-flow information, especially the one regarding parallelism, for the next activity prediction task.

Indeed, its main contributions are the following.

- the introduction of an approach that uses the information about the parallelism among activities for the PPM tasks. This is done by representing each trace with its corresponding Instance Graph and processing them using deep graph convolutional neural networks designed to natively manage graph structures.

- the definition of a novel metric that manages to effectively measure the parallelism in a business process model, this is done to investigate a way of characterizing processes so as to determine a way to possibly identify where the previous Instance Graph-based approach can lead to good results.

- the definition of a set of metrics that describe the "execution context" of an activity inside a process to represent the activity itself. This is done to extend the work of previous contributions so as to take into account all the information that a process model provides about a particular activity. This set of metrics is used both to define a querying mechanism for activities in processes and to introduce the notion of "location" as a further relevant prediction target for PPM techniques. We remark that such "execution context" is derived from the process model of the considered activity and it is explicitly devised to take into account the control flow information about the activity itself.

The rest of the thesis is organized as follows: in Chapter 2 we present some related work both on the Predictive Process Monitoring tasks and on the measuring and encoding of processes and processes activities, as well as the benchmark datasets used throughout the whole thesis. In Chapter 3, some core definitions used throughout the thesis are introduced. In Chapter 4, it is presented the design and implementation of the Instance Graph-based approach to the next activity prediction task. Chapter 5 addresses the problem of evaluating the level of parallelism in a business process, proposing a novel metric to do it and its scaled variant. Chapter 6 is devoted to the presentation of an encoding of activities in processes and the usage of such encoding to compare processes and define a novel prediction task in the predictive process monitoring field. Finally, Chapter 7 draws conclusions and discusses future work. In Appendix A.1 it is shown a preliminary study on the application of Reinforcement Learning to Predictive Process Monitoring.

Chapter 2

Related Work

This Section outlines some of the most relevant contributions related to the present work that can be found in recent literature. At the end of this chapter, all the datasets available in the literature and used in this thesis are described.

2.1 Predictive Process Monitoring

Predictive process monitoring made its appearance as a process mining task in the first decade of the 2000s [5]–[7], receiving increasing attention in the latest years, as witnessed by the number of recent surveys and the volume of research products [8]–[10]. Three kinds of predictions can be considered [8], [9]: prediction of (typically continuous) measures of interest like the remaining execution time, overall duration, or cost of an ongoing case [7], [11], [12]; prediction of categorical values like the final outcome or class of risk of a case [9], [13]; predictions related to the sequence of next activities that will be performed [14]. In [15], the authors propose how to deal with more than one of these tasks. As another dimension, approaches can be distinguished into model-aware and model-agnostic. In model-aware approaches, predictions rely on a formal process model, either designed by domain experts or discovered by process discovery techniques, whereas model-agnostic approaches only consider traces contained in the event log [13]. Leveraging a process model allows to exploit some form of control-flow information. On the other hand, when the model describes the prescribed or most common behaviors, overlooking exceptions, or sensible deviations from it, predictions for real executions may suffer from this abstraction. Furthermore, other perspectives can be taken into account besides control-flow, like the time or duration of events and resources performing activities.

For what concerns the next-activity(ies) prediction task, few proposals rely on some kind of process model in combination with traditional machine learning techniques. [14] adopts both a process mining algorithm to discover a general process model and decision trees to calculate transition probabilities from a given activity to neighboring activities from instance specific data, so as to define an instance-specific probabilistic process model for each process execution. Assuming a Markov property for processes, the approach does not consider the path information of previously executed activities to train decision trees, but only data that is progressively produced during the execution of the process starting from the first task. Path information is instead explicitly represented in the approach proposed by [16], which proposes different encoding models to represent the parallel branch each activity belongs to, derived from the overall process model using token-replay principles. [15] relies on annotated transition systems, where (Naïve Bayes) classifiers and ($\in$-SVR) regressors are used for annotations to predict remaining time and the sequence of next activities. Authors introduce a notion of similarity among the states of the transition system to deal also with non-fitting traces, taking into account also the issue of non-stationary processes. A different approach to deal with event logs involving exceptional behaviors has been proposed by [17], which employs sequential pattern mining techniques to derive partial process models that are then used to train classification or regression models. [13] introduces a framework to mine probabilistic finite automata from data by grammatical inference.

Recently, Deep Learning techniques have gained increasing interest in predictive process monitoring as it is well described in a recent survey [18]. The approaches rely upon the power of deep architectures to build complex features and on the success of recurrent architectures in processing sequential data, like log traces are. Hence the majority of approaches are model-agnostic. For what concerns the next-activity prediction, Long Short-Term Memory (LSTM) is one of the first and most adopted architectures [19]–[21]. LSTM trained with a Generative Adversarial Nets learning scheme has also been proposed

[22], tackling the lack of sufficient training data that often impacts performances. An alternative approach is that of [23] where it is proposed to transform traces into image-like data, thus unleashing the full potential of Convolutional Neural Networks (CNN). Although more traditional Deep Learning architectures like Multi Layer Perceptrons have been largely overlooked, in [24] experiments demonstrate that they can achieve good performance on some datasets. Other approaches like reinforcement learning or transformers have also been experimented [25], [26]. In all these proposals, different learning architectures, different input data encodings and attributes characterize the approaches. However, a common feature is the inherently sequential structure of inputs and the consequent inability to fully capture the structure of process executions. Few previous studies have proposed to encode structural information from the process model for Recurrent Neural Network models. For example, [27] proposes an approach which first detects loops in log traces and then uses this information to improve the results of a LSTM-based next-activity classifier. Their approach also allows to incorporate domain knowledge related to execution constraints.

In [28], [29] also proposed a different strategy to take into account process structure for the next activity classification task, by using graphs to represent the processes.

A proposal to directly process graphs to predict the next activity has been done by [24]. This approach has some similarities with the proposal presented in this thesis. First of all, it adopts a process discovery approach (inductive mining with Directly-Follows Graphs) for building a model of the process. Second, it adopts a Graph Convolutional Neural Network (GCNN) to learn the prediction. With respect to [24], our approach adopts a different, instance-specific, graph model in the form of Instance Graph, managing also non-fitting traces. Furthermore, in [24] the network architecture is composed of a single graph convolutional layer followed by two fully connected layers, while here a variation of the Deep Graph Convolutional Neural Network (DGCNN) of [30] is exploited. As another difference, if many events in a trace correspond to the same activity, only the features of the most recent event are retained in [24], whereas the Instance Graphs adopted in this work can present the same activity more than once.

2.2 Measuring & Encoding

2.2.1 Measuring Parallelism in Processes

Several metrics have been devised to measure relevant aspects of business processes [31]–[33]. Among them, one of particular interest is how much parallel a process is. It is acknowledged that increasing the number of activities performed in parallel is a way to improve the performance of a business process [34], although this improvement comes at the cost of a more complex process design, development, and maintenance. Having a synthetic metric to quantify the parallelism of a process may thus provide an assessment of the process and guide certain design choices.

In literature, a widely adopted metric is the Degree of Parallelism (DoP), defined as the maximum number of parallel activities that can be executed in that process [31], [35]. In particular, [35] proposes different algorithms to compute the DoP for three classes of BPMN processes. In [31] it is observed that the DoP, theoretically, can be computed by determining the bound of a Petri net, which is the maximum number of tokens in a marking of the net. The computation of such bound requires the construction of the reachability graph, whose derivation is known to be an NP-complete problem or ever harder for some classes of Petri nets [36], [37]. A more efficient general procedure is then proposed for a wide class of BPMN processes, which exploits the notion of Labeled Transition System and model checking. An extension [38] considers timed business processes modeled in BPMN, where execution times are associated with BPMN constructs such as activities and flows. The DoP metric is concerned with the "worst" case scenario, i.e. the portion of process presenting the highest number of parallel activities, neglecting other parts of the process.

Other metrics aimed at assessing the overall complexity of a business process are discussed in [32]. In particular, it is argued that complexity of the parallelism of a process can be measured by the Average Degree of Transitions

(ADT) which is the average number of incoming and outgoing arcs of transitions in a Petri net.

2.2.2 Encoding Process Structure in Activities

When event logs tracking process executions are available, process mining techniques can be used to derive alternative representations of process executions. This can be done either by combining information from the event log and the process model, as done by, e.g., alignments or trace replay techniques, or inferring directly follows mappings directly from the event log [39], [40]. Over time, the need to browse and query processes has induced the scientific community to investigate further possible representations. In [41] the authors present the usage and comparison of various possible techniques, in particular graph matching techniques, alignment techniques and causal footprint, i.e. an abstract representations of the behavior captured by a process model to define similarity metrics at a process model level of granularity.

In contrast to these techniques, the present work focuses on a finer granularity, i.e. at the level of single activities. Lately, due to the attention that has been devoted to predictive tasks in the process mining field, how to encode the information regarding a business process is becoming crucial. Several works [40], [42], [43] have tackled the problem of encoding traces, i.e. the sequence of activities or events that represent a particular process execution, but only few works exist regarding the representation of a singular activity in a process. Other works have also tried to encode the information regarding only a partial trace, in order to leverage such information to perform predictions [24], [44], [45]. Another recent proposal [46] consists in the usage of neural networks to learn a representation of activities, logs, and process models in order to derive an informative and low-dimensional embedding. In [47] word2vec has been used in conjunction with an LSTM network to label nodes in business process models. It is worth noting that approaches based on neural network representations don't have a clear semantic. Differently from those approaches we purposely design each feature to measure a well-defined structural characteristic and to be more suited to human comprehension.

TABLE 2.1: Overview of benchmark dataset. $|\sigma|$ represents the trace length.

Dataset	N.traces	Tot.events	N.act.types	Min $|\sigma|$	Max $|\sigma|$	Avg $|\sigma|$
Helpdesk	3804	13710	9	1	14	3
BPI12W	9658	72413	6	1	74	20
BPI12	13087	262200	23	3	175	38
RfP	6886	50568	21	1	20	7
TP	7065	86581	51	3	90	12
ID	6449	72151	34	3	27	11
PC	2099	18246	29	1	21	9

2.3 Dataset

For our experiments in this thesis, we selected some of the benchmark datasets commonly used in literature, whose characteristics are reported in Table 2.1.

The *Helpdesk* dataset [48] contains traces from a ticketing management process of the help desk of an Italian software company.

The *BPI12* dataset [49] tracks personal loan applications within a global financing organization. The event log is a merge of three parallel sub-processes. We considered both the full *BPI12* and the *BPI12W* sub-process, related to the work items belonging to the application. We retained only the completed events in the two logs, as done in previous work.

The *BPI20* dataset[50] is taken from the reimbursement process at TU/e. The data is split into travel permits and several request types from which we selected four datasets. *Requests for Payment* (RfP) sub-log contains cost declaration referred to expenses that should not be related to trips. *Travel Permit* (TP) includes all related events of travel permits declarations and travel declarations. *International Declarations* (ID), contains events pertaining to international travel expense claims. *Prepaid Travel Cost* (PC) contains events pertaining to travel expense claims for prepayment. In all four latter datasets, the resource performing the activity is included in the activity itself, thus producing a lot of different activity types.

Chapter 3

Preliminaries

In this Chapter, we introduce some core definitions used throughout the thesis.

Definition 1 (Labeled Petri Net) *A* labeled Petri net *is a tuple* (P, T, F, A, ℓ) *where* P *is a set of places,* T *is a set of transitions,* $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation connecting places and transitions,* A *is a set of labels for transitions, and* $\ell : T \nrightarrow A$ *is a partial function that associates a label with a subset of the transitions in* T*. Transitions not associated with any label are called invisible transitions.*

Figure 3.1 shows the Petri net obtained from a real-life process concerning the ticketing management process of the help desk of an Italian software company [48].

Transitions represent process activities, namely well-defined tasks that have to be performed within the process, and places are used to represent states. Invisible transitions do not correspond to process activities and are used for routing purposes. We indicate the set of invisible transitions as $T_H \subseteq T$.

Specific executions of a process, so-called *process instances*, are typically recorded in logs. More precisely, the execution of an activity generates an *event*, which is a complex entity characterized by a set of *properties*.

Definition 2 (Event, Trace, Log) *Let* A_L *be the set of all activity names,* C *be the set of all case (aka, process instance) identifiers,* H *be the set of all timestamps,* U *a set of variable values,* V *a set of variable names. An event* $e = (a, D, c, i, t) \in A_L \times (V \nrightarrow U) \times C \times \mathbb{J} \times H$ *is a tuple consisting of an executed activity* $a \in A_L$*, a function* D *which assigns a value to some process variables (possibly all of them), a case identifier* $c \in C$ *and a number* $i \in \mathbb{J} \subseteq \mathbb{N}^+$*. A case corresponds to a single process execution;*

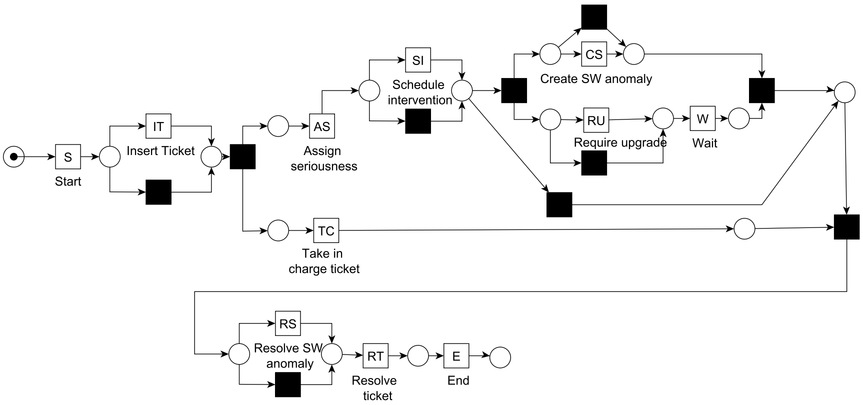


FIGURE 3.1: Petri net mined with the Inductive Miner from the Helpdesk event log. Transition labels are displayed below each transition, while inside each square is the corresponding acronym.

Case ID	**Activity**	**Timestamp**
Case 2	Start	03/04/2012 16:55
Case 2	Assign seriousness	03/04/2012 16:55
Case 2	Take in charge ticket	03/04/2012 16:55
Case 2	Resolve ticket	05/04/2012 17:15
Case 2	End	05/04/2012 17:15
Case 3	Start	29/10/2010 18:14
...	...	...

TABLE 3.1: Excerpt from the HelpDesk event log

the number i identifies the position of the event within the sequence of events that occurred within a case, finally $t \in \tau$ is the timestamp. The set of events is denoted by $\mathcal{E}$. An event trace $\sigma_L \in \mathcal{E}^*$ is a sequence of events with the same case id. An ***event log*** *is a multi-set of event traces L.*

Table 3.1 shows an excerpt of the event log for the Helpdesk process mentioned above. Through this thesis, we will use the notation $act(e)$, $case(e)$, $pos(e)$, $time(e)$, and $var\_name(e)$ to refer to, respectively, the activity, the case id, the position in the sequence, the timestamp and the attribute named var_name of an event e. For instance, let e_2 be the second event in Table 3.1; $act(e2)$ is "Assign seriousness", while $time(e_2)$ is "03/04/2012 16:55". Here we introduce also the *projection* operator $\pi_{Att}(x)$, which is used to build the projection of a tuple x on a subset of its attributes Att. For instance, given an event e_i we can define the projection $\pi_{A_L,C,\mathbb{J}}(e_i) = (act(e_i), case(e_i), pos(e_i))$.

With a slight abuse of notation, we extend this operator to traces as follows: $\pi_{A_L,C,\mathbb{J}}(\sigma_i) = \langle \pi_{A_L,C,\mathbb{J}}(e_1), \ldots, \pi_{A_L,C,\mathbb{J}}(e_n) \rangle$.

Definition 3 (Prefix trace) *A prefix of length k of a trace $\sigma = \langle e_1, e_2, \ldots e_n \rangle \in \mathcal{E}^*$, is a trace $p_k(\sigma) = \langle e_1, e_2, \ldots e_k \rangle \ \in \mathcal{E}^*$ where $k \leq n$.*

For example, let us indicate with σ_1 the trace involving the events with case id *Case 2* in Table 3.1. The prefix of length 3 of σ_1 is
$p_3 = \langle$ (Start, $\{\}$, Case 2, 1, 03/04/2012 16:55), (Assign seriousness, $\{\}$, Case 2, 2, 03/04/2012 16:55), (Take in charge ticket, $\{\}$, Case 2, 3, 03/04/2012 16:55)$\rangle$. Note that, in this example, the function D corresponds to an empty set, since we don't have any additional data attributes in the log.

A well-known issue of log traces is that events are logged in a trace according to the timestamp of the corresponding activities, thus hiding possible concurrency among activities. To address this issue, log traces can be converted in so-called *Instance Graph* [51]. These are directed, acyclic graphs which represent the real execution flow of process activities.

Definition 4 (Causal Relation) *A* Causal Relation *(CR) is a relation on the set of activities, $CR \subseteq A \times A$. $a_1 \rightarrow_{CR} a_2$ denotes that $(a_1, a_2) \in CR$.*

Elements of CR represent the order of execution of a pair of activities of a process. In this work, we consider causal relations extracted from existing process models. To this end, given a labeled Petri net (P, T, F, A, ℓ) representing a process model, we introduce the notion of *direct path* between transitions t_1, $t_2 \in T$ as follows: $dp(t_1, t_2)$ if and only if $\exists p \in P$ s.t. $(t_1, p) \in F \wedge (p, t_2) \in F$. It should be noted that t_1, t_2 can be either transitions with or without labels (i.e., hidden transitions). In this setting, $a_1 \rightarrow_{CR} a_2$ if and only if $l(t_1) = a_1 \ \wedge \ l(t_2) = a_2 \wedge (\exists dp(t_1, t_2) \vee (\exists s = \langle t_{h_1}, \ldots, t_{h_n} \rangle$ s.t. each $t_{h_i} \in T_H \wedge \exists dp(t_1, t_{h_1}) \wedge \exists dp(t_{h_n}, t_2) \ \wedge \ \forall i, j \in \{1, \ldots, n\}, i < j \ \exists \ dp(t_{h_i}, t_{h_j})))$. Informally, this definition considers a causal relation only a direct connection between two labelled transitions, where the first one generates one of the input tokens for the second one, disregarding possible (hidden) transitions occurring in between.

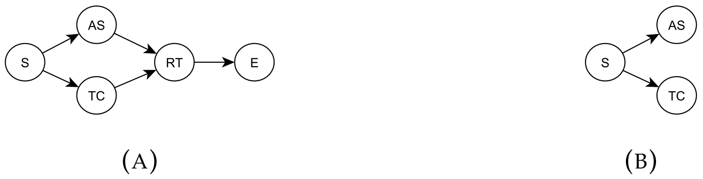


FIGURE 3.2: IG for σ_1 (a) and its prefix of length 3 (b)

Definition 5 (Instance Graph) *Let $\sigma = \langle e_1, \ldots, e_n \rangle \in L$ be a trace and let $\sigma' = \pi_{A_L,\mathbb{J}}(\sigma)$ be its projection on the activity and position sets. An* Instance Graph *(or* IG*) γ_σ of σ is a directed acyclic graph (E, W) where:*

- *$E = \{e \in \sigma'\}$ is the set of nodes, corresponding to the events occurring in σ';*
- *$W = \{(e_h, e_k) \in E \times E \mid h < k \wedge act(e_h) \rightarrow_{CR} act(e_k) \wedge (\forall e_q \in E(h < q < k \Rightarrow act(e_h) \not\rightarrow_{CR} act(e_q)) \vee \forall e_w \in E\ (h < w < k \Rightarrow act(e_w) \not\rightarrow_{CR} act(e_k)))\}$ is the set of edges, defining a partial order over E;*

Similarly to what we have done for trace and trace prefixes, starting from the definition of IGs we can introduce the notion of *Graph prefix*.

Definition 6 (Prefix Instance Graph (prefix-IG)) *Let (E, W) be the instance graph of some trace σ. Let $\widetilde{E}_k$ be the set of events in the prefix trace $p_k(\sigma)$ of size k. We define the prefix instance graph of size k of σ as the graph $p_k((E, W)) = (\widetilde{E}_k, W \cap (\widetilde{E}_k \times \widetilde{E}_k))$. Informally, a prefix graph $p_k(g_j)$ is a subgraph of g_j involving only k nodes of g_j, i.e., nodes included in the corresponding prefix trace.*

Consider σ_1 and the set CR derived from the Petri net in Figure 3.1. In Figure 3.2, Figure 3.2a shows the IG corresponding to the trace, while Figure 3.2b shows its prefix of length 3. For the sake of simplicity, we only use activity acronyms to label the graph nodes rather than showing the index and the complete names. We will adopt the same simplification when drawing IGs throughout the rest of the thesis.

Chapter 4

Instance Graphs for Predictive Process Monitoring

This chapter introduces a novel approach to tackle the next-activity prediction challenge. The approach uses information about the parallelism among activities for the task. This is done by representing each trace with its corresponding Instance Graph and processing them using deep graph convolutional neural networks designed to natively manage such graph structures. This work has also been published in [52].

4.1 Methodology

Here we introduce a novel approach to tackle the next-activity prediction challenge. Formally, this problem corresponds to learning a classifier able to label a prefix trace with the activity to be executed next. Figure 4.1 shows the proposed approach. Given an event log and its process model expressed as a Petri net, the approach i) represents each trace with its corresponding Instance Graph (IG), ii) enriches the built IG with additional perspectives regarding the sequential execution and, when available, additional event attributes, and iii)

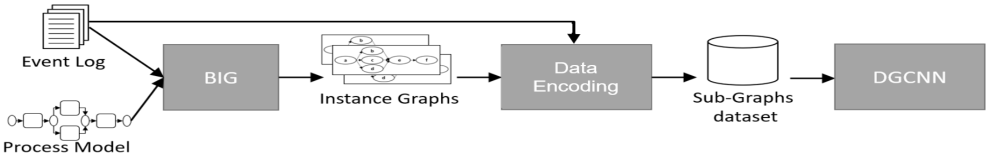


FIGURE 4.1: The BIG-DGCNN methodology pipeline

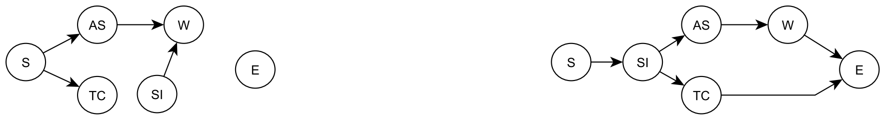
processes such IGs through graph neural networks, designed to work with graph data structures, to train a classifier to perform the next-activity prediction task. The approach used to build the IGs is robust against the possible presence of outliers or anomalous behaviors. In other words, even in the presence of anomalous behaviors the approach returns instance graphs without structural anomalies and that provide a high-quality model for the corresponding process behaviors. The set of instances graphs is then used to train the graph neural network. For the classifier, among the various architectures proposed in the literature, we chose to adopt the Deep Graph Convolutional Neural Network (DGNN) [30]. In the following, we will refer to our methodology as BIG DGCNN and to its multi-perspective enriched variants as Multi-BIG-DGCNN. The following subsections delve into each step of the approach.

4.1.1 Building Instance Graphs

This step takes as input an event log and a process model and converts each sequential trace in the event log into an Instance Graph. It should be noted that we assume to have a single starting and a single ending activity for each process execution. This is necessary to ensure that possible parallelism at the beginning or at the end at the execution can be properly modeled. This constraint, however, does not pose a significant limitation to the applicability of the approach. In fact, it is always possible, if necessary, to apply a simple data preprocessing procedure to the log and to the model to introduce artificial start and end activities. As regards the process model, this can be either provided by a domain expert or extracted by a process discovery algorithm.

To generate the IGs, in this text we refer to the Building Instance Graph (BIG) algorithm proposed in [51], which is able to handle traces that do not conform to the model. BIG is a two-steps algorithm. First, an IG is built for each trace as in Definition 5, by using the set of causal relations extracted by the process model provided in input. In the presence of non compliant events, however, this procedure generates anomalous, low-quality IGs. As an example, let us consider the following trace:[1] $\sigma_2 = \langle(1, S), (2, SI), (3, AS), (4, TC),$

[1] For the sake of simplicity, we directly show the projected trace obtained by another trace from the Helpdesk log. Furthermore, for the sake of readability, we only use activity acronyms.



(A) IG built for trace σ_2

(B) Repaired IG built for trace σ_2

FIGURE 4.2: IG repair examples

$(5, W)$, $(6, E)\rangle$. This trace is not compliant with respect to the model in Figure 3.1 since the activity SI is executed before AS and the activity RT is missing. Figure 4.2a shows the IG built for this trace according to Definition 5. The anomalies mentioned above led to generate a disconnected graph, since W and TC should both be linked to RT. Furthermore, connections among nodes do not reflect the temporal order of occurrence of the events, in particular for SI. In terms of semantics, these models over-generalize the process behavior. For example, the only execution constraint for SI is to be executed before W. Even worse, activities of each part of the disconnected graph can be executed in any order with respect to the activities of the other part. To deal with the issues mentioned above, in [51] an *IG repairing* procedure is applied to IGs corresponding to anomalous traces, which transforms them into graphs capable of also representing the anomalous traces without over-generalizing. First, anomalous traces (and, hence, IGs) are recognized in the event log by means of a conformance checking technique [53]. Then, tailored rules are applied for repairing IGs with deleted and inserted events. For deleted events, the repairing consists in identifying the nodes which should have been connected to the deleted activity properly connecting them. For the insertion repairing, we have to change the edges connecting the nodes corresponding to the event(s) before and the event(s) after the inserted event, to connect such nodes with the node corresponding to the inserted event in the graph, taking into account the causal relations among its predecessors and successors in the trace.

Figure 4.2b shows the outcome of the repairing procedure for the IG corresponding to σ_2. The repairing of the deletion of RT has been realized by connecting its predecessors W and TC with its successor E while the inserted event SI has been connected to the events occurring before/after the anomalous event in the trace. It should be noted that there are two main forces driving the repairing procedures. On the one hand, we want to obtain a representation as precise as possible of the occurred anomaly, limiting the number of behaviors represented by the repaired IG. On the other, we want to preserve concurrency relations described by the model. For this reason, the insertion of the event SI after S is repaired by connecting SI to both the causal successors of S[2]. It is worth noting that the repaired graphs do not fulfil Definition 5 with respect to the original causal relation set CR; however, they still fulfil the definition according to the new set of causal relation CR', obtained by extending CR to include all the pair of activities linked by edges added/modified during the repairing procedure.

We would like to point out that the repairing procedure is an essential component of the BIG algorithm. Without the repairing, the presence of even few non-compliant events in a trace can lead to disconnected graphs and/or graphs with a high degree of parallelism, which can hide temporal relations among process activities. These graphs are likely to hamper the performance of the classifier; therefore, we advocate that not-repaired IGs should not be used to train the classifier.

4.1.2 Data Encoding

This step aims at building a labeled prefix-IG dataset, enriched with additional data perspectives derived from the event log and can be further split into two phases. First, we extract all the prefix-IGs from the set of IGs derived at the previous step. Then, we enrich the prefixes according to the set of perspectives that we want to consider for the analysis. Both steps are detailed below.

Prefix-IG generation

Given the set of n Instance Graphs IG, the goal of this step is to build the dataset $S = \{(p_i(g_j), a_l)\}$ where $p_i(g_j) = (E_p, W_p)$ is a prefix-IG of length i of a

[2] Note that for deviations occurring within parallel constructs other repair configurations are available, e.g., by adding an additional parallel branch involving the inserted activities. Refer to [51] for additional details.

	E	**W**	**Label**
$p_2(g)$	$\{(1, S), (2, SI)\}$	$\{((1, S), (2, SI))\}$	AS
$p_3(g)$	$\{(1, S), (2, SI), (3, AS)\}$	$\{((1, S), (2, SI)), ((2, SI), (3, AS))\}$	TC

TABLE 4.1: prefix-IG of lenght 2 and 3 extracted from the IG in Figure 4.2b

.

graph $g_j = (E_j, W_j)$, where $i \in [1, \|E_j\| - 1], g_j \in IG$ and a_l corresponds to the next activity of the partial execution described by $p_i(g_j)$. It is straightforward to see that from each IG, we produce $N - 1$ pairs of S.

The building of the prefix-IG set is accomplished by using the total order of the events in the trace. Indeed, recall that each node in an Instance Graph corresponds to an event in the corresponding trace (see Definition 5). Therefore, it is possible to link each node in an IG to a progressive index representing the position of the corresponding event in the trace. This index determines the order of the nodes, which we use to progressively build the prefix-IG set.

In particular, given an IG g, the graph prefix $p_2(g)$ is obtained by selecting the first two nodes and the edge(s) between them. This prefix is labelled with the activity of the event in position 3. The next prefix is then derived by extending $p_2(g)$ with the node of index 3 and the edges connecting it to $p_2(g)$. The associated label is the activity of the event in position 4. The procedure is repeated until the activity corresponding to the last node of the graph is selected as the label. As an example, let us consider the trace σ_1 introduced in Chapter 3, whose IG g is reported in Figure 4.2b. Table 4.1 shows two prefix-IGs extracted by g, of length 2 and length 3, respectively.

Multi-Perspective Prefix-IG Enrichment

The prefix-IGs built at the previous step model the activity name and the corresponding causal relations for each event of a process execution. This step aims at *enriching* the prefix-IGs in order to incorporate additional data perspectives. In practice, this is realized by linking each node of each prefix-IG with the set of features which the analyst wants to take into account for the prediction. Formally, let M be the set of perspectives (aka, features) chosen by the analyst for the prediction, G be the set of feature values and $Val\_M : M \rightarrow 2^G$ the function defining the values admissible for each feature. Given the set of IG-prefixes S built at the previous step, the goal is to build the dataset $S^{'} = \{(p_i^{\prime}(g_j), a_l)\}$, where $p_i^{\prime}(g_j) = (E_p, W_p, Val\_M)$ is a *Multi-Perspective Enriched* prefix-IG of lenght i of a graph $g_j = (E_j, W_j)$, $i \in [1, \|E_j\| - 1]$.

We consider two sets of features; *direct* features, corresponding to data attributes stored in the event log, and *indirect* features, derived from the information available in the trace. Note that the set of direct features corresponds to the set D introduced in Definition 2; therefore, $D \subset M$. For the indirect features, we are especially interested to *time-related* features, which are used to encode information about the sequential execution order of the traces from which the IGs have been extracted.

The use of this kind of information has been previously used in literature [20], [23]. However, thanks to the use of instance graphs in place of log traces, in our framework the temporal intervals are computed for each activity with respect to its causal predecessor rather than with respect to the preceding activity in the sequence. We argue that such computation provides a more accurate representation of what actually happened within the process execution, thus providing more robust information to be used for the prediction in place of the sequence-based features. These features are defined as follows.

Let CR be the set of causal relations defined among the activities of the event log L. Let us consider the prefix $p_i(g_j) \in S$ and let us indicate with n_i the node corresponding to the event at the i-th position in the trace corresponding to g_j. With a slight abuse of notation, in the following we use $act(n_i)$, $time(n_i)$ to indicate the activity and the timestamp of the event e_i. This is justified by the fact that for each node of each prefix-IG there exists a unique mapping to the position of the event of the corresponding trace, from which the corresponding information can be accessed.

The first temporal feature we define is $\Delta_{t_{n_i}}$, which represents the time between the current event and its predecessor in the graph. For all nodes n_i, let $pred_{n_i} = \{n_j \mid (act(n_j), act(n_i)) \in CR\}$ denote the set of all nodes that are causal predecessors of n_i. We define

$$\Delta_{t_{n_i}} = \begin{cases} 0 & \text{if } pred_{n_i} = \emptyset \\ \min_{n_j \in pred_{n_i}} \frac{time(n_i) - time(n_j)}{\Delta_{\max_e}} & \text{otherwise} \end{cases}$$

where $time(n_i)$ is the timestamp of the event at index i, and $\Delta_{\max_e}$ is the maximum interval between two consecutive adjacent nodes. In addition to $\Delta_{t_{n_i}}$, we use two other temporal features. The first one represents for each event the time it occurred with respect to the start of the process. The other feature allows to take into account at which point an activity has occurred with respect to the corresponding working week (i.e, since midnight on previous Sunday). This can provide valuable information for the classifier, since activities of a business process are likely to be carried out within office hours. Formally:

$$t_{d_{n_i}} = \frac{time(n_i) - t_0}{\Delta_{\max_t}}; \quad t_{w_{n_i}} = \frac{time(n_i) - t_{w_0}}{\Delta t_w}$$

where t_{w_0} is the timestamp of the last passed Sunday midnight, and t_0 is the start timestamp of the process. Δt_w is the amount of time in a week, while $\Delta_{\max_t}$ is the maximum trace duration. Note that Δt_w, $\Delta_{\max_e}$ and $\Delta_{\max_t}$ are normalization factors computed on the entire event log to make features varying in the range $[0, 1]$, as it improves the performance of the network.

Once the direct features have been selected from the event log and the indirect ones have been computed, we compute the mapping function Val_M for each node of each prefix, thus generating the dataset S'.

As an example, Table 4.2 shows the prefixes discussed above enriched with the temporal features. Note that since the first three events of σ_1 all have the same timestamps, the temporal features are all the same for these prefixes.

The final processing step consists in transforming the feature set in the format requested by the classifier. In particular, the Deep Convolutional Neural Network we select for our architecture takes in input a vector $FV = [FV_e, FV_W, Label]$ where:

- $FV_e = \{fv_1, \ldots, fv_n\}$ where fv_i corresponds to a feature vector describing one node of the graph, i.e., $fv_i \in A_L \times Val\_M$. Note that we exploit

	E	**W**	*Val_M*	**Label**
$p'_2(g)$	$\{(1,S),$ $(2,SI)\}$	$\{((1,S),$ $(2,SI))\}$	$\{(1,\{\Delta_{t_{n_i}} = 0, t_{d_{n_i}} = 0, t_{w_{n_i}} = 0.27\}),$ $(2, \Delta_{t_{n_i}} = 0, t_{d_{n_i}} = 0, t_{w_{n_i}} = 0.27\})\}$	AS
$p'_3(g)$	$\{(1,S),$ $(2,SI),$ $(3,AS)\}$	$\{((1,S),$ $(2,SI)),$ $((2,SI),$ $(3,AS))\}$	$\{(1,\{\Delta_{t_{n_i}} = 0, t_{d_{n_i}} = 0, t_{w_{n_i}} = 0.27\}),$ $(2,\{\Delta_{t_{n_i}} = 0, t_{d_{n_i}} = 0, t_{w_{n_i}} = 0.27\}),$ $(3,\{\Delta_{t_{n_i}} = 0, t_{d_{n_i}} = 0, t_{w_{n_i}} = 0.27\})\}$	TC

TABLE 4.2: Enriched prefix-IG of lenght 2 and 3 extracted from g.

the *one-hot* encoding to encode both the name of the activity and the possible categorical features in M.

- $FV_W = \{(i,j) \mid 1 \leq i,j \leq |FV_e|\}$ is a set of tuples corresponding to the set of the edges of the graph;
- *Label* corresponds to the classification label associated with the graph.

4.1.3 Deep Graph Convolutional Neural Network

As model architecture to perform the next-activity prediction we use a Deep Graph Convolutional Neural Network (DGCNN) proposed in [30].

The DGCNN is composed of three sequential stages as shown in Figure 4.3.First it has several graph convolutional layers which extract the features from the nodes local substructure whose features values can be used to define a consistent vertex ordering. Second it has a SortPoolingLayer which sorts the vertex features according to the order defined in the previous stage, selecting the top nodes. In this way the dimension of the input is unified. At last, a 1-D convolutional layer and a dense layer take the obtained representation to perform predictions.

The graph convolutional layer adopted by DGCNN is represented by the following formula:

$$Z = f(\tilde{D}^{-1}\tilde{A}XW) \tag{4.1}$$

where $\tilde{A} = A + I$ is the adjacency matrix (A) of the graph with added self-loops(I), $\tilde{D}$ is its diagonal degree matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, $X \in \mathbb{R}^{n \times c}$ is the graph nodes information matrix (in our case the one-hot encoding of the activity labels associated with the nodes), $W \in \mathbb{R}^{c \times c'}$ is the matrix of trainable weight parameters, f is a nonlinear activation function, and $Z \in \mathbb{R}^{n \times c'}$ is the output activation matrix. In the formulas, n is the number of nodes of the input graph (in our case, the graph prefix), c is the number of features associated with a node, and c' is the number of features in the next layer tensor representation of the node.

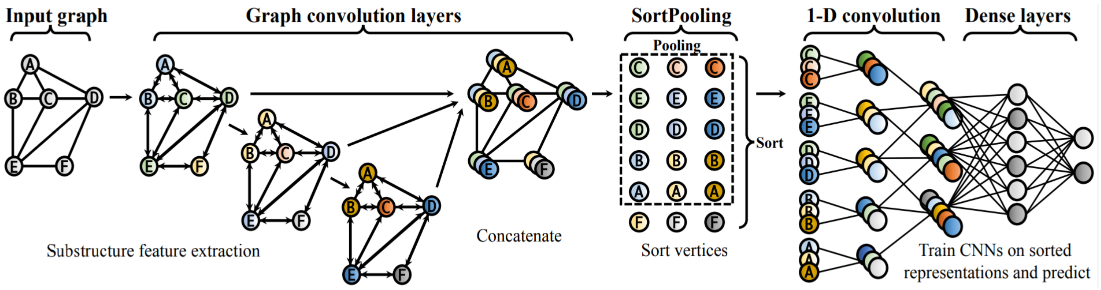
In a graph, the convolutional operation aggregates node information in local neighborhoods so to extract local structural information. To extract multi-scale structural features, multiple graph convolutional layers (eq. 4.1) are stacked as follows:

$$Z^{k+1} = f(\tilde{D}^{-1}\tilde{A}Z^kW^k) \tag{4.2}$$

where $Z^0 = X$, $Z^k \in \mathbb{R}^{n \times c_k}$ is the output of the k^{th} convolutional layer, c_k is the number of features of layer k, and $W^k \in \mathbb{R}^{c_k \times c_{k+1}}$ maps c_k features to c_{k+1} features.

The graph convolutional outputs $Z^k, k = 1, ..., h$ are then concatenated in a tensor $Z^{1:h} := [Z^1, ..., Z^h] \in \mathbb{R}^{n \times \sum_1^h c_k}$ which is then passed to the SortPoolingLayer. It first sorts the input $Z^{1:h}$ row-wise according to Z^h, and then returns as output the top m nodes representations, where m is a user-defined parameter. This way, it is possible to train the next layers on the resulting fixed-in-size graph representation.

In the original proposal the DGCNN includes a 1-D convolutional layer, followed by several MaxPooling layers, one further 1-D convolutional layer followed by a dense layer and a softmax layer. Here we simplify the architecture leaving only one 1-D convolution layer with dropout [54] followed by a dense and a softmax layer. This is because the process mining domain tend to present smaller graphs in comparison with those of typical application domains of graph neural networks [55]. For further information we refer the interested reader to [30].



credit to: Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification (2018)

FIGURE 4.3: DGCNN architecture [30].

This section describes the experiments we carried out on multiple real-world datasets to assess the performance of our approach w.r.t. state-of-the-art competitors. We first provide a description of the experimental set-up, the selected datasets and the competitors. Then, we discuss the obtained results.

4.2 Experiments

4.2.1 Experimental Setup

We compared our approach against a set of representative competitors from the literature. In particular, we chose one representative of neural network architecture per type used in next-activity prediction in previous work, namely, LSTM, CNN, MLP, and GCNN. The criteria used to select the competitors were:

- the availability of the source code to reproduce the experiments,
- a claim of good performance on one or more benchmark datasets commonly used in literature,
- the absence of a particular encoding mechanism apart from those necessary to apply their architecture.

When in doubt, we selected the most acknowledged papers on the basis of citations and place of publication. On the basis of these criteria we selected:

- [24] for MLP,

- [24] for GCNN (specifically the Laplacian binary),
- [23] for the CNN,
- [20] for the RNN (LSTM to be specific).

We highlight that for [20] we had to reimplement the code since it was too outdated w.r.t. the used python modules. Also, for the GCNN proposed in [24], we had to add to the adjacency matrix self-loops in order to guarantee its invertibility with every dataset, as done in other cases in the literature [30]. Finally, we remark that for each competitor we used the hyper-parameters search methods provided in their code or, when not available, their claimed best hyper-parameters. If neither was available, we used the parameters provided by their code. To provide an as fair as possible comparison we just used the timestamp-based features so that no extra information is used with respect to any of the competitors (though some of them may have used extra ones). All the datasets used are those presented in sec. 2.3.

4.2.2 Parameter settings

The presented methodology involves two algorithms requiring the setting of parameters: the infrequent Inductive Miner (iIM) [56], used to extract the process model from a given event log, and the DGCNN. The iIM builds the model after filtering out infrequent behaviours according to a noise threshold. We changed the noise threshold in steps of 10% from 0% to 100% and selected the smallest noise threshold that granted at least a 90% fitness (i.e., how much the discovered model can accurately reproduce the cases recorded in the log[3]). Using this criterion, the obtained models are capable of representing the vast majority of traces while still maintaining a good degree of generalization, thus providing a favorable setting for the classification task.

Regarding the parameters of the DGCNN, we set the number of 1-D convolutional layers to one, followed by a dense layer, both with 64 neurons.We used ADAM [57] as optimization algorithm and trained the network for 100 epochs with an early stopping. We used as loss function the categorical cross

[3] Here we refer to the state-of-the-art notion of fitness proposed by Adriansyah et al ([53])

entropy, a fixed batch size of 64 and a fixed dropout percentage of 0.1. For all datasets, we tested all the methods using the same 67%-33% train-test split (of chronologically ordered traces) and varied the following parameters:

- the number of nodes selected by the SortPooling layer (*m*), in {3,5,7,30}

- the number of stacked graph convolutional layer (*h*), in {2,3,5,7}

- the initial learning rate (*lr*), in $\{10^{-2}, 10^{-3}, 10^{-4}\}$

The configurations that provide the best performance reported as (m, h, lr) are (7,3,10^{-4}), (7,2,10^{-4}), (7,3,10^{-2}), (7,3,10^{-2}), (7,3,10^{-2}), (7,3,10^{-3}), (7,3,10^{-3}), respectively for Helpdesk, BPI12W, BPI12, BPI20 RfP, BPI20 TP, BPI20 ID, BPI20 PC.

For all datasets, the best number of selected nodes is always 7. The most reasonable cause for this behaviour is that for all dataset the number of samples with a prefix shorter than 8 is the vast majority. We also notice that this explanation also holds for the small number of stacked graph convolutional layers. All the experiments have been performed using either pytorch geometric [58] with torch version 1.10.0 or tensorflow 2.5 [59], on an NVIDIA GeForce GTX 1080 GPU, a Intel(R) Core(TM) i7-8700K CPU@3.70GHz, and a 32 GB RAM.

4.2.3 Evaluation metrics

To compare the results obtained by the tested classifiers, we exploit two metrics widely used for classification tasks, namely *accuracy* and *F1 score*.

The accuracy measures the proportion of the correctly classified samples out of all samples, i.e., $Accuracy = \frac{T}{N}$, where N is the number of samples and T is the sum of all the samples correctly classified. The overall F1 score is computed as the weighted average of the F1 scores computed for each class, weighted w.r.t. the corresponding number of samples. The F1 score for each class $F1_i$ is computed as the harmonic mean of *precision* and *recall* for class i, i.e. $F1_i = \frac{P_i \cdot R_i}{P_i + R_i}$. The *precision* P_i is computed as $P_i = \frac{TP_i}{TP_i + FP_i}$ and the *recall* R_i is computed as $R_i = \frac{TP_i}{TP_i + FN_i}$.$TP_i$ is the number of i-class samples correctly classified, FP_i corresponds to the number of samples wrongly classified as class

TABLE 4.3: Comparison results; measured accuracy and F score.

	Dataset							
Approach		Helpdesk	BPI12W	BPI12	RfP	TP	ID	PC
Multi-BIG DGCNN	Acc	**86.15%**	**71.32%**	76.09%	**90.64%**	78.50%	88.44%	85.80%
	F1	**83.19%**	**69.89%**	71.12%	**87.51%**	76.13%	**86.28%**	**83.90%**
BIG DGCNN	Acc	85.18%	70.85%	72.90%	90.03%	78.29%	73.72%	85.61%
	F1	82.93%	69.06%	68.31%	87.16%	76.08%	70.63%	83.14%
GCNN	Acc	80.42%	64.75%	60.92%	88.16%	61.33%	81.74%	79.52%
	F1	76.73%	59.77%	58.95%	86.05%	60.12%	78.05%	76.44%
MLP	Acc	82.16%	66.17%	71.80%	89.81%	76.83%	86.82%	85.38%
	F1	77.45%	62.11%	66.07%	87.38%	74.61%	84.31%	84.56%
CNN	Acc	85.02%	66.36%	78.45%	89.11%	**82.52%**	**88.89%**	**85.93%**
	F1	82.13%	63.48%	75.92%	85.62%	**80.23%**	86.18%	83.65%
LSTM	Acc	74.49%	66.08%	**79.06%**	90.24%	76.89%	87.96%	82.65%
	F1	72.13%	61.61%	**75.48%**	86.72%	72.60%	84.93%	79.74%

i (aka, *false positives*); while FN_i corresponds to the number of samples of class i wrongly classified as some other class (aka, *false negatives*).

Moreover, we evaluate the Average Ranking (AR), the Success Rate Ratio Ranking (SRR) and the ranking (R) [60]. The former is simply the average of ranks achieved by a given approach on all datasets. The SRR shows the success rate ratio of approach i, and it is measured by first calculating the average of accuracy (F1 score) ratios on all k datasets $SRR_{i,j} = (\sum_1^k SRR_{i,j}^k)/k$, where $SRR_{i,j}^k = Acc_i^k/Acc_j^k$ $(F1_i^k/F1_j^k)$ is the ratio of accuracies (F1) achieved by approaches i, j on dataset d_k. The SRR of the approach i (SRR_i) is then obtained as $SRR_i = \sum_j SRR_{i,j}/(m-1)$, where m is the number of competitor approaches. Finally, R is the ranking computed over the SRR.

4.3 Results and Discussion

Table 4.3 reports the results achieved by each approach over the tested datasets. The best values for each dataset are highlighted in bold. To assess the impact of the enrichment phase on the classification performance, we tested two versions of our approach, i.e., the one exploiting only the control-flow information (BIG-DGCNN) and the one exploiting the enriched IGs (Multi-BIG-DGCNN).

The first interesting insight is that considering multiple perspectives is overall beneficial for classification performance. In fact, Multi-BIG-DGCNN is consistently better than BIG-DGCNN over all tested datasets. The strongest differences can be observed in BPI12, which shows improvements in accuracy and the F1 score respectively of 3.19% and 2.81%, and in the ID dataset, where the accuracy and the F1 score improved of, respectively, 14.72% and 15.65%. These results suggest that the set of features used for the enrichment have strong predictive capabilities for these two datasets. On the other hand, focusing on the pure workflow perspective, we can state that BIG-DGCNN is a better approach than GCNN.

Moving to the comparison with the competitors, Multi-BIG-DGCNN achieves best results in terms of F1 score on five datasets out of seven. In Helpdesk, BPI12W and RfP Multi-BIG-DGCNN also achieves the best accuracy performance. CNN turns out to be the best on TP and achieves the best accuracy values on ID and PC, whereas LSTM is the best on BPI12. Overall, considering the F1 score, it seems that Multi-BIG-DGCNN shows a better consistency over all datasets. To demonstrate this, we report in Table 4.4 the overall comparison expressed in terms of AR, SRR and R for both accuracy and F1 score figures of merit. We observe that, for what concerns AR, Multi-BIG-DGCNN is the best approach, followed by CNN and then BIG-DGCNN and LSTM. It also turns out to be the best approach according to the SRR metrics, though values show that it is basically comparable with CNN. Considering that the CNN encodes a richer set of aggregated temporal features than Multi-BIG-DGCNN, results are encouraging and demonstrate the viability of instance graphs processed by DGCNN, since this kind of information may also be added when deemed useful for prediction purposes.

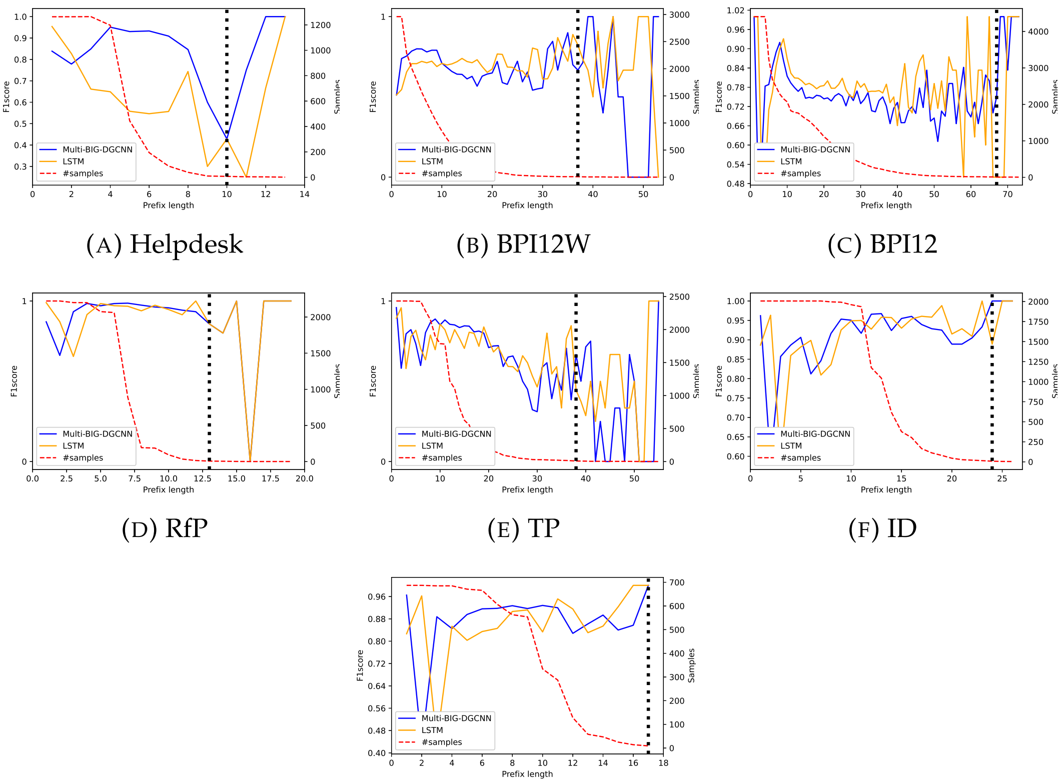
It is also worth noting that the BPI12W dataset, where both our approaches obtained the biggest improvement with respect to the second best approach, is also the dataset with the highest percentage of activities in a short loop,

TABLE 4.4: Literature comparison, rankings

	Accuracy			F1		
Approach	AR	SRR	R	AR	SRR	R
Multi-BIG-DGCNN	1.50	1.248	1	1.43	1.255	1
BIG-DGCNN	2.88	1.206	5	2.88	1.210	3
GCNN	5.00	1.113	6	4.75	1.110	6
MLP	3.75	1.207	3	2.88	1.203	4
CNN	2.00	1.246	2	2.38	1.253	2
LSTM	3.25	1.206	4	3.88	1.198	5

which is known to be a difficult situation for next-activity prediction. A reasonable explanation for this result is that the graph convolution mechanism is naturally robust to such repetitions since it can aggregate the information of nearby nodes, which is exactly the scenario we have when a specific activity is repeated several times.

In addition to analyze the overall behavior of the approach, we are also interested in understanding how it varies among the different prefix sizes. Figure 4.4 shows the trend of the F1 score with respect to the different prefix lengths across all the datasets. We compare the performance of Multi-BIG-DGCNN (blue line) against those of LSTM (orange line). We chose to compare these two approaches because the LSTM approach proposed by Tax et al. is the one with the set of features more similar to ours. The main differences are that we consider for each event the time w.r.t. the start of the process, rather than within the day (i.e., w.r.t. midnight) and that we consider causal relations in computing temporal intervals between an event and its successor(s), rather than considering subsequent events in the trace (see Section 4.1.2). Therefore, we can reasonably assume that differences in performance are likely to be due either to the different architectures employed, i.e., sequential vs graph-based, or to the explicit use of information on the process structure in the feature set. In addition to the F1 score performance, in the figures a red, dotted line shows how the sample size varies with the increase of the prefix length. To provide some additional insights on the size of the sample set for the different prefix lengths, a vertical, dotted black line is placed to separate results obtained on



(A) Helpdesk

(B) BPI12W

(C) BPI12

(D) RfP

(E) TP

(F) ID

(G) PC

FIGURE 4.4: F1 score of Multi-BIG-DGCNN and LSTM on the tested datasets plotted against the prefix lengths, together with the number of samples.

prefix lengths with at least ten samples (on the left of the line), to those obtained on fewer samples (on the right of the line).

In the following, we focus the discussion on the prefixes on the left of the black line, i.e., prefixes involving at least ten samples. This is justified by the fact that for prefix lengths involving a very scarce number of samples, even a difference of a few samples classified correctly or incorrectly can deeply impact the results. Note that most of the F1 score plots in Figure 4.4 show a very unstable result in the neighborhood of the black line for both classifiers, which seems to confirm that a limit of 10 is reasonable for these datasets.

The figure shows that Multi-BIG-DGCNN usually performs close to or higher than LSTM on the shorter prefixes; however, the performance get worse for longer prefixes. Since the shorter prefixes correspond to the higher number of samples, outperforming the competitor in the shorter prefixes allows Multi-BIG-DGCNN to obtain a higher accuracy than LSTM in the corresponding dataset. An exception is represented by the dataset BPI12, where LSTM obtains comparable or better results along all the prefix lengths, which indeed results in a higher overall average accuracy as shown in Table 4.3.

The reason for the worsening in performance of Multi-BIG is unclear. A hypothesis can be related to the fact that the Deep Graph Convolutional Neural Network needs more samples to train. Another justification can be found in the architectural properties of the DGCNN. These networks determine which nodes are the most important for the prediction exploiting the information gained during the convolutional layers adopted during the training. In doing so for the analyzed datasets, it is reasonable that the network gives higher importance to nodes belonging to the shorter prefixes, since they are the most frequent and the most relevant ones for the overall prediction performance. However, these nodes are also the least informative ones for the longer, less frequent prefixes, with the result of a drop in performance.

Chapter 5

Measuring Parallelism in Process Models

This Chapter is devoted to the definition of a novel metric that manages to effectively measure the parallelism in a business process model. This is done to investigate a way of characterizing processes so as to determine where the Instance Graph-based approach, presented in the previous Chapter 4, can lead to good results. This work has also been published in [61].

Several notations exist to describe a process model, like Petri nets [62], or BPMN [63]. In the following, we will discuss the properties of the different metrics on a set of paradigmatic processes described in Petri nets notation.

Note that we focus on operational business processes, that are represented by a sub-class of Petri nets called Workflow nets (WF-net). A WF-net has one start place, one end place, and all transitions and all places are on a path from start to end[64].

We recall that a specific execution of the process generates a so-called process instance, which is the partially ordered set of activities that are performed to achieve the completion of a single execution of a process, that can be modeled by an IG.

5.1 Use Cases Description

In this section, we introduce 7 use cases, in the form of Petri nets. Although simple, these processes have been designed to capture some paradigmatic situations allowing us to enlighten the properties of the different metrics from the

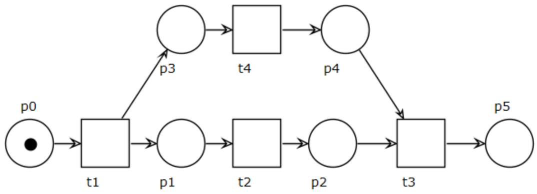


FIGURE 5.1: Use case 1

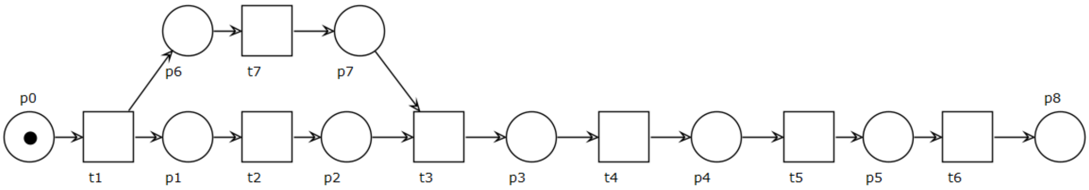


FIGURE 5.2: Use case 2

literature and those proposed here. These examples can also be easily scaled.

We designed use case 1 (Fig. 5.1), as a simple starting base case so to compare the added structural complexity of the other models with it.

Use case 2, (Fig. 5.2) only adds to use case 1 a sequence of transitions after the parallelism. Hence, for this model, it is desirable that a metric lowers its score.

Concerning use case 3 (Fig. 5.3), we designed it to highlight the opposite behaviour: it is basically a sequence of two use case 1, so it would be reasonable to see an increase in the metric score.

Use case 4 (Fig. 5.4), has been designed so to compare it with both use cases 2 and 3. We expect to see an increased score w.r.t. the former, while it is more difficult to compare the quantity of parallelism with the latter. Indeed, use case 4 has a greater number of activities that can be performed simultaneously, while use case 3 has a greater number of parallel block.

Similarly, we designed use case 5 (Fig. 5.5) as a problematic example to compare with use cases 1 and 2, since there are a greater number of activities in the parallel branch.

Use case 6 (Fig. 5.6) is introduced to show the effect of synchronisation on the parallel complexity of the process with respect to use case 5.

Lastly, we consider use case 7 (Fig. 5.7) to show the effects of an or split on the various metrics.

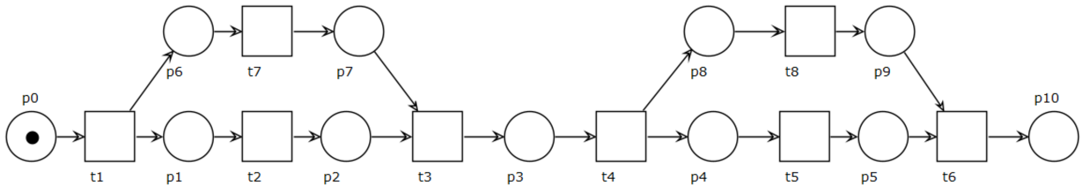


FIGURE 5.3: Use case 3

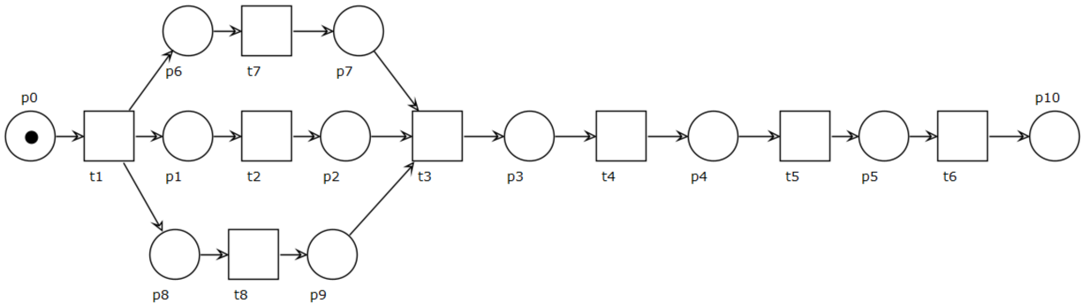


FIGURE 5.4: Use case 4

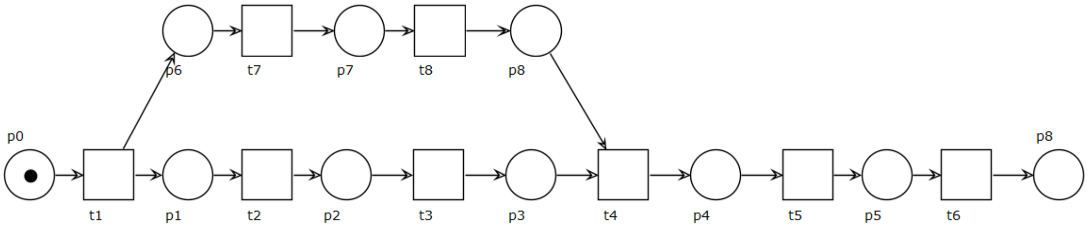


FIGURE 5.5: Use case 5

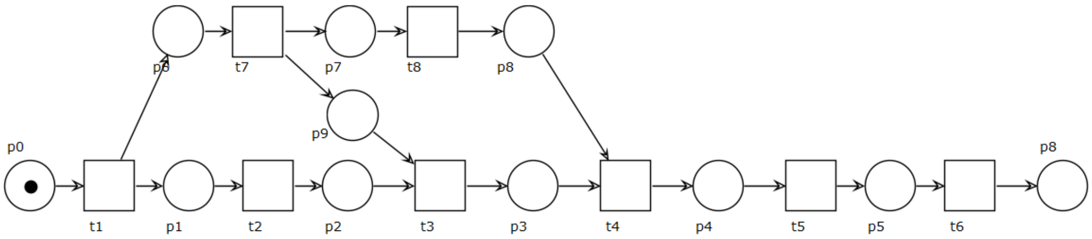


FIGURE 5.6: Use case 6

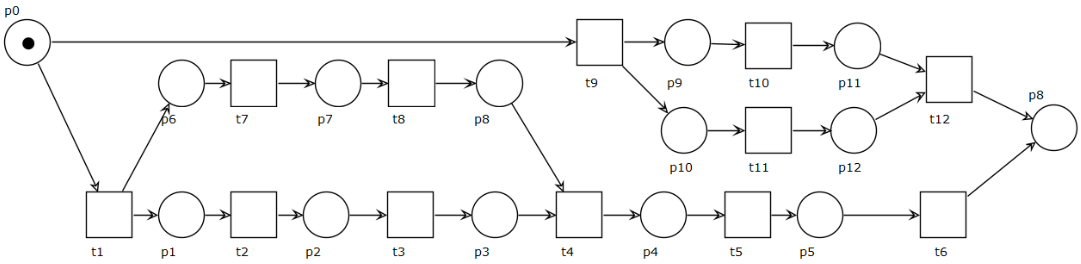


FIGURE 5.7: Use case 7

5.2 Metrics for Process Parallelism Evaluation

5.2.1 Metrics Based on Model perspective

In this subsection we introduce two popular metrics, whose values are calculated directly from the structure of the process.

Average Degree of Transition

The Average Degree of Transition (ADT) has been introduced in the context of Petri net-based business process as a mean to measure control flow parallel complexity [32]. It is defined as: $ADT = \frac{\sum_i^T deg(t_i)}{|T|}$, where T is the set of all transitions in the Petri net, and $deg : T \rightarrow N$ is the function that associates with a transition the number of its incoming and outgoing arcs. It is relevant to notice that such metric assumes values in $[2; +\infty)$, a value of 2 represents a purely sequential process whereas bigger values indicate the presence of more parallelism. As examples, considering use case 1 we can compute the ADT as $10/4 = 2.5$ whereas for the use case 2 it is $16/7 = 2.29$.

Degree of Parallelism

The most commonly used metric to quantify the parallelism of a process is the maximum number of activities that can be executed in parallel, called Degree of Parallelism (DoP). It is computed by evaluating the number of activities enabled by a marking of the Petri net. For example, for the use case 1 the value of DoP is 2 due to the marking $\langle p1, p3 \rangle$ which enables both transitions $t2$ and $t4$; for the process in Figure 5.4 the DoP is 3 since the marking $\langle p1, p6, p8 \rangle$

enables the maximum number of transitions, namely $t2, t7$ and $t8$. Techniques to calculate the DoP have been proposed in [31], [35].

Comparison between DoP and ADT

First of all we notice that as the DoP calculate a maximum parallelism, it is hence suited to assess peak behaviours, whereas ADT shows a more comprehensive average complexity of the process. In order to highlight this fact, let us consider use cases 2 and 3. The value of the DoP metric is 2 for both processes, but the latter clearly shows a more complex behaviour due to the presence of a further parallel block. This is captured by ADT whose value is increased from 2.29 to 2.5. Clearly DoP is able to capture the difference between processes in Figures 5.2 and 5.4 passing from 2 to 3. ADT also detect the increased complexity, although with a slightly minor relative increment passing from 2.29 to 2.5. We remark that the ADT metric returns the same value for use cases 3 and 4, demonstrating that ADT is unable to capture the increased complexity inherent to a parallelism with a higher number of parallel activities. The values of DoP and ADT metrics for all the use cases are shown in the last two columns of Table 5.1. The limits discussed for the two metrics motivated us to focus on a different aspect of a process model to define a metric. Specifically, in the following subsection we describe two metrics which take into account the number of variants represented by an IG.

5.2.2 Metrics Based on Instance Graphs

We can say that each IG represents a specific set of parallel branches that exist in the process and that the parallel complexity of such set is as high as the number of variants, i.e. all the possible unique sequence of activities, represented by the IG. On this basis, we introduce a metric called Parallel Complexity, and a scaled variant, designed to overcome the limits of both ADT and DoP being more sensitive to both overall and maximum parallelism.

Parallel Complexity (PC)

This metric is defined as $PC = \frac{|V|}{|IG|} - 1$, where $|V|$ is the number of distinct variants allowed by the process model and $|IG|$ is the number of distinct IGs that represents those variants. The minus 1 is a scaling parameter used to reduce to 0 the Parallel Complexity in case of strictly sequential processes. Given a process model, the set of IGs can be easily obtained by a two-step procedure. First, the variants of a process are generated (play-out). Then, using causal relations of the model, an IG is generated for each variant. Computing such metric for the provided use cases, we can see that the metric shows an hybrid behaviour with respect to DoP and ADT, i.e., it is sensitive to both overall parallelism and maximum parallelism. On the one hand, as regards the use cases 2 and 3 we can notice that PC scores respectively 1 and 3 managing to reflect the increased overall parallelism as ADT does, but DoP does not. On the other hand, for use cases 3 and 4, PC takes the values 3 and 5 respectively, highlighting the difference between the two examples as DoP does but ADT does not. Agreeing behaviours are displayed considering use cases 2 and 4 as both ADT and DoP detect the increment. However, it should be noted that PC scores a bigger relative increase. Indeed, the percentage increment of ADT, DoP and PC are 9.2%, 50% and 400%, respectively. This is due to the fact that adding a branch to a parallelism does not increase linearly its complexity as the possible actual executions of such parallelism grow combinatorially. A relevant advantage of the PC metric is shown comparing use case 6 with use case 5. The former adds to the latter a synchronisation place $p9$ which locks the execution of $t3$ to $t7$, therefore reducing the existing parallelism. In this situation, ADT, instead of a reduction, even scores an increment due to the arcs used to connect p9. DoP ignore the change altogether. On the contrary we note that PC correctly reflects the change passing from 5 of Figure 5.5 to 4 of Figure 5.6. A similar situation is displayed by use cases 3 and 5, where ADT lowers its value instead of increasing it. Use case in Figure 5.7 shows that PC also works well if choices exist in the process model, giving rise to more than one IG. Since this use case is composed by a choice between 1 and 5, PC correctly returns a score of 3 which is the average of its values for these two use cases. A similar

TABLE 5.1: Metrics for all use cases

Use case	$\|IG\|$	$\|V\|$	$\|T\|$	$\|S\|$	PC	SPC	ADT	DoP
1	1	2	4	2	1	0.5	2.5	2
2	1	2	7	2	1	0.2	2.29	2
3	1	4	8	4	3	0.75	2.5	2
4	1	6	8	2	5	0.83	2.5	3
5	1	6	8	2	5	0.83	2.25	2
6	1	5	8	3	4	0.8	2.5	2
7	2	8	12	4	3	0.375	2.33	2

behaviour is displayed by ADT which scores 2.33 which is in-between the composing use cases ADT values. The main limit of PC is that it doesn't scale well with respect to process length. To explain the point, let us consider use cases 1 and 2. The former shows a higher overall parallelism, since a larger portion of process 2 is strictly sequential. Instead, PC values for both processes is 1, whereas this difference is captured by ADT. This motivates the introduction of a scaled variant of the PC.

Scaled Parallel Complexity (SPC)

In order to overcome the described limit of the proposed metric, we devised a scaling mechanism that takes into account the number of activities. $SPC = \frac{PC}{(|T| - |S|)}$, where $|T|$ is the total number of transitions in the model and $|S|$ is the number of transitions that have more than one outgoing or incoming arcs. They in fact represent the point where a parallel branch starts (AND split) and where a parallel branch synchronises (AND merge) respectively. We can see from Table 5.1 that this new metric maintains all the good behaviours that PC has. It also displays the desired behaviour that PC does not have: when considering use cases 1 and 2, we can note that SPC decreases as required thanks to the scaling factor that takes into account the presence of the sequence in use case 2. We also notice that this scaling leads to smaller relative increment with respect to PC. In contrast to PC, when considering use case 7 we see that SPC fails to produce a coherent value. Indeed, use case 7 is a process with an alternative choice between use case 1 and use case 5. Nevertheless, SPC score drops to 0.375 which is not in the range of scores for use cases 1 and 5. This is due to how the scaling is implemented. Currently the mechanism considers all transitions in the process model even though the number of transitions in the derived IG could be far less. This is because an IG only displays the activities from one of the optional branches of the process. This over-reduces the score by over-estimating the denominator of the SPC formula. This makes the actual SPC formula unfit for many kinds of processes as it doesn't manage properly choice structures. We plan on improving this metric by modifying the scaling mechanism, for instance by applying it previously to the number of variants produced by a specific instance-graph and then computing the average of such values.

Chapter 6

A new representation for Events in Processes

This Chapter presents a set of metrics that describe the "execution context" of an activity inside a process to represent the activity itself. This is done to extend the work of the previous Chapters so as to take into account all the information that a process model provides about a particular activity. The set of metrics is used both to define a querying mechanism for activities in processes and to introduce the notion of "location" as a further relevant prediction target for PPM techniques. We remark that such "execution context" is derived from the process model of the considered activity and it is explicitly devised to take into account the control flow information about the activity itself.

Section 6.1 presents the metrics and their usage in a case study to show how they can be used to derive measures of similarities. This is also presented in publication [65].

Section 6.2 presents how such metrics can be used to introduce a novel PPM task that does not predict directly the next activity but, instead, predicts the "execution context" of the next activity. This work is also published in [66].

6.1 A Process Model-based Encoding

In this section we introduce a set of features that describe the context of an activity inside a process, thus embedding the process structure in the activity representation. We introduce the features, explain the rationale under them, and discuss how they can be used to derive a measure of similarity among activities.

6.1.1 Methods

This section describes the proposed features. They have been designed for Petri nets, which is the best-investigated formalism for process modeling [67]. In particular, in this work we focus our investigation on block-structured Petri net. Each set of such features represents a single process activity describing its execution context, intended as the control-flow constructs in which the activity is involved (e.g., parallelism, choice, loop), so that activities with similar structural properties will be close to each other in the feature space.

Path Length

This feature represents the position of the activity w.r.t. the beginning of the process. Given an activity a_i in the set of all process activities $\mathcal{A}$, we define the longest path $LP(a_i)$ as the maximum number of activities in a path from the beginning of the process to a_i, excluding hidden activities and loops. The Path Length for a_i is then computed as follows:

$$PL(a_i) = \frac{LP(a_i)}{\max\limits_{a_j \in \mathcal{A}} LP(a_j)} \tag{6.1}$$

The Path Length assumes values in $]0,1]$, where small values indicate that the activity is located near the beginning of the process and values close to 1 are of activity near the end of the process. For the first activity in the process the value is $\frac{1}{\max\limits_{a_j \in \mathcal{A}} LP(a_j)}$, while for the farthest one it is 1. It should be noted that the farthest activity is not necessarily the last activity in the process, although the two concepts often coincide. The feature is computed using the longest path so to ensure coherent order among all process activities representation, w.r.t. the process model. To better understand this, consider activities c and e in Figure 6.1, we can notice that if we used the shortest instead of the longest we would obtain a value of $\frac{3}{4}$ for activity c and of $\frac{2}{4}$ for activity e, but activity

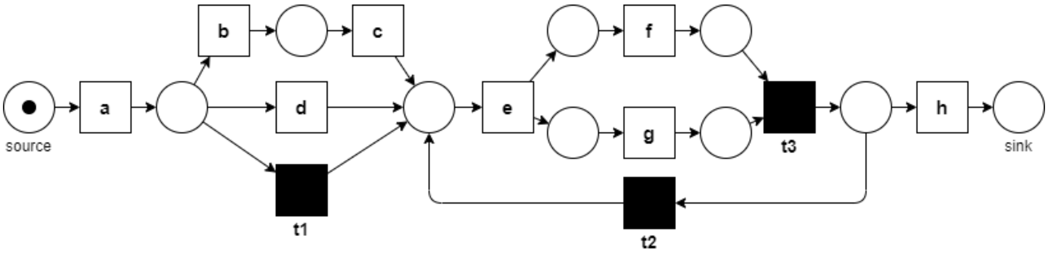


FIGURE 6.1: Example Petri net 1

e is always executed after activity c, according to the model. Hence, wanting to maintain in the feature the relative order of the activities in the process the shortest path would be inadequate.

In Figure 6.1, the maximum length of the longest path is 6, namely $< a, b, c, e, f, h >$ (or $< a, b, c, e, g, h >$). Hence, for the first and last activities we have $PL(a) = \frac{1}{6} = 0.17$ and $PL(h) = 1$, which correspond to the lowest and highest value of the feature respectively. The activities b and d have the same value because they both have to wait for one activity to be executed before they can be activated ($PL(b) = PL(d) = 0.33$), whereas c has to wait for both a and b to be executed ($PL(c) = 0.5$). Regarding the activity e, it should be noted that it can be started only if the parallel block preceding e is terminated, that is both branches $< b, c >$ and $< d >$ are executed. So the delay of e depends on the longest branch and is $PL(e) = \frac{4}{6} = 0.67$.

Optionality

This feature captures the degree of optionality of a specific activity. Optionality is computed as the reciprocal of the number of the maximum alternative branches in which an activity is involved, as follows:

$$Opt(a_i) = 1 - \frac{1}{n_{\#}(a_i)} \tag{6.2}$$

where $n_{\#}(a_i)$ represents the number of alternative branches for the activity a_i.

We note that the "number of the maximum alternative branches" means that - in the case of nested choice structures - we take into account all the possible alternative branches to the considered activity, and count the number of branches that can be alternatively executed at most. Branches with hidden activity are only count once, in case of nested choices.

The feature assumes values in $[0, 1[$, where 0 represents an activity that is not involved in any alternative branches and values close to 1 are related to activities in choice block with many possible branches. The feature is not computed for hidden activities, and backward branches (i.e., defining loops) are not taken into account in the formula.

Let's consider again the example in Figure 6.1. Here we have two choice blocks: after the activity a and before the activity h. For the former, we have $Opt(b) = Opt(c) = Opt(d) = 0.67$ because there are three possible alternatives: execute the activities b and c, perform d or neither (passing through the hidden activity $t1$ in the lowest branch). For the latter, we do not consider the branch with $t2$ because it represents a loop, so $Opt(h) = 0$. The optionality for all other activities in the process model assumes the value 0.

Parallelism

This feature is used to calculate the degree of parallelism of an activity, i.e. with how many activities it is in parallel. It is calculated counting the number of parallel branches to the activity. The Parallelism (Par) for the activity a_i is computed as follows:

$$Par(a_i) = 1 - \frac{1}{n_{\parallel}(a_i)} \tag{6.3}$$

where $n_{\parallel}(a_i)$ represents the number of parallel branches for the activity a_i. In the case of nested parallel blocks, the maximum number of possible branches in parallel is considered.

We note that the "maximum number of possible branches in parallel" means that - in the case of nested parallelism block - we take into account all the possible parallel branches to the considered activity, and count the number of branches that can be parallelly executed at most. The feature assumes values in $[0, 1[$, where 0 represents a sequential activity and the closer the value is to 1, the higher the number of parallel branches. The feature is not computed for hidden activities.

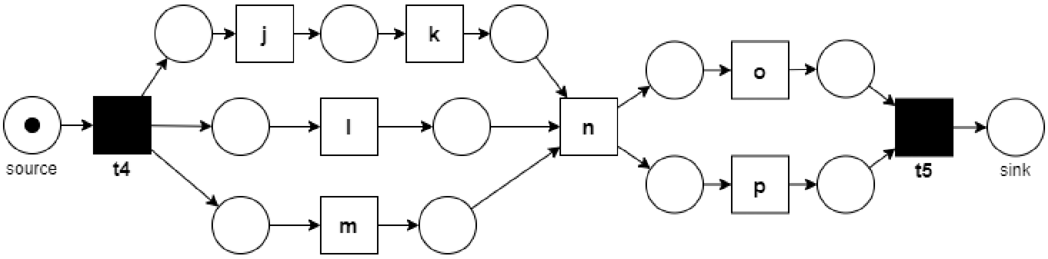


FIGURE 6.2: Example Petri net 2

In Figure 6.2, we have two parallel blocks: one from hidden activity $t4$ to activity n, and the other from n to $t5$. As for the former, we have $Par(j) = Par(k) = Par(l) = Par(m) = 1 - \frac{1}{3} = 0.67$, whereas in the latter block $Par(o) = Par(p) = 0.5$. The feature for the activity n assumes the value 0, since it can not be executed in parallel with any other activity.

Parallelism Path Length

This feature is based on the PL feature; namely, it captures the position of an activity within the parallel block in which the activity is located. The formula is the same, except that the starting point from which to compute the longest path is the beginning of the parallel block. The feature is computed as follows:

$$PPL(a_i) = \frac{LP_{\parallel}(a_i)}{\max\limits_{a_j \in \mathcal{A}} LP_{\parallel}(a_j)} \tag{6.4}$$

Where, $LP_{\parallel}(a_j)$ is the length of the longest path for the activity a_j within the parallel block. The path takes into account activities in any parallel branch plus the last activity of the parallel block (i.e., the synchronization point); so the activity from which the parallelism stems is not considered. For activities that aren't in a parallelism's branch the value of PPL is equal to 0. The same considerations made for PL take to this feature, but limited to the sub-process related to the parallel block.

Let us consider the two blocks in Figure 6.2. For the former (from $t4$ to n), the longest path is formed by 3 activities (i.e., $< j, k, n >$), hence $PPL(j) = PPL(l) = PPL(m) = 0.33$ and $PPL(k) = 0.67$. In the latter parallel block (from

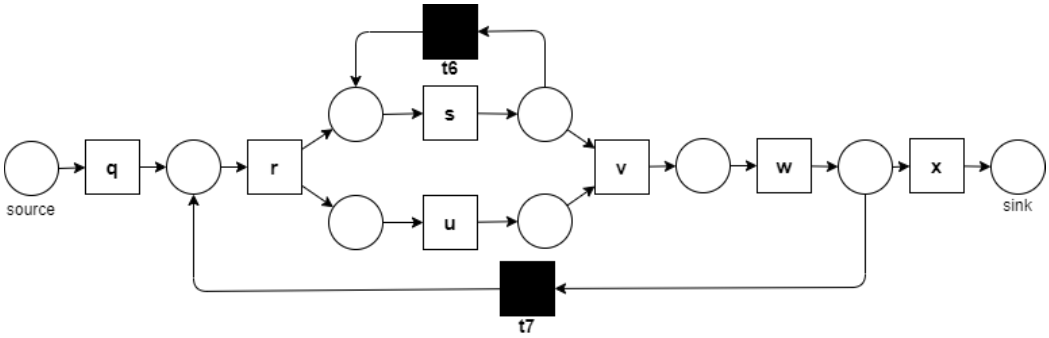


FIGURE 6.3: Example Petri net 3

n to $t5$), the length of the longest path is 1 because the path starts after the first activity of the block and the synchronization point is and hidden activity which is not considered. Hence, $PPL(o) = PPL(p) = 1$.

Self Loopable

This feature evaluates if an activity can be consecutively repeated. In particular, the Self Loopable (SL) feature for the activity a_i is equal to 1 if the activity a_i is in a self-loop, 0 otherwise.

Long Loopable

This feature is concerned with representing if an activity belongs to a sub-process that can be repeated. In this case the Long Loopable (LL) feature assumes the value 1, otherwise 0. The same considerations made for a Straightly Loopable activity also hold here.

Let us consider Figure 6.3, which involves two loops. In the former the activity s can be repeated consecutively, whereas in the latter the sub-process from r to w can be repeated. Since only s is in a self-loop, we have $SL(s) = 1$ and for all others activities SL takes the value 0. The LL, instead, takes the value 1 for all activities in the loop block from r to w (i.e., $LL(r) = LL(s) = LL(u) = LL(v) = LL(w) = 1$). For all others activities, LL is equal to 0.

6.1.2 Case Study

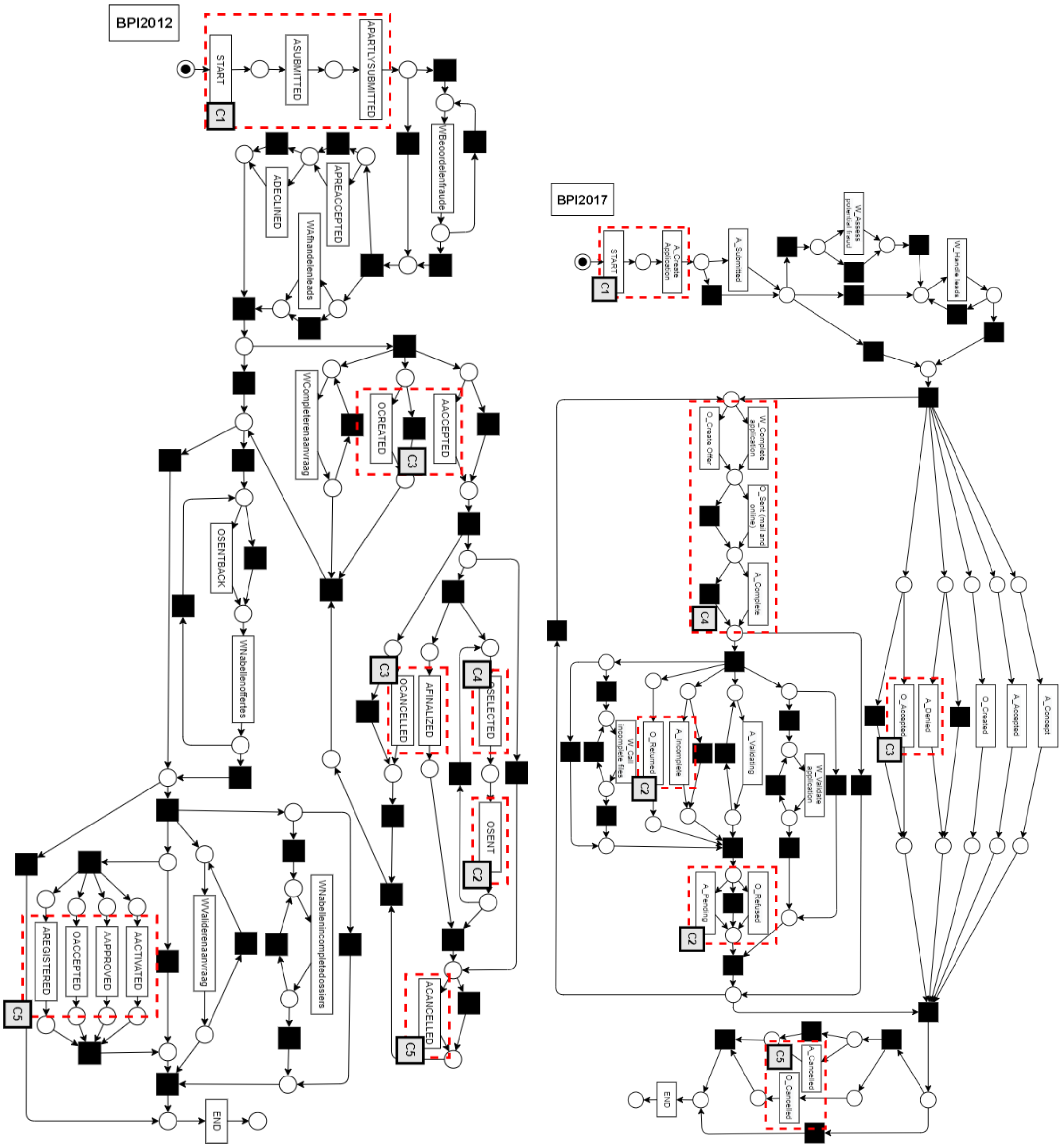
This section discusses a case study which serves as a proof of concept for the features introduced in the present work. We focus on two versions of a loan application process of a Dutch financial institute, the first one is BPI12 [49] and the second is one from 2017 [68] (BPI17). The latter model has some substantial differences from the former, since a new workflow system has been implemented in the company. We argue that this context represents a good case study for our approach. First, the two models under analysis represent the same process, possibly with some variation. Therefore, it is reasonable to assume that the semantics of (part of) the activities overlaps, at least to some degree, which simplifies the analysis of the detected structural similarities and differences. Second, there are several studies available in the literature on these processes, which provide us with useful insights to elaborate upon the observations derived by the structural comparison.

Settings

The embedding strategy has been implemented as a python library[1]. Note that the implementation of the features discussed in Section 6.1 requires to investigate the state space of the process. To this end, here we exploit so-called *process trees* [69], which provide a block-structured representation of a process from which the control-flow relations for each activity can be inferred. We use the pm4py library[2] to derive a process tree from a given Petri net. The construction and analysis of process trees is a more efficient alternative to the exploitation of reachability graphs. As a drawback, it requires a block-structured process model. We argue that this is not a strong limitation, since block-structures nets are widely adopted to support different process analysis tasks, and they are the default representation of well-known state-of-the-art process discovery methods (e.g., Inductive Miner [70]). Figure 6.4a and 6.4b show the Petri Nets for bpi2012 and bpi2017, respectively. The models have been extracted by applying the Inductive Miner algorithm with the default settings on the event logs provided at [49], [68].

[1] `https://github.com/KDMG/Embedding-Structure-in-Activities`

[2] `https://pm4py.fit.fraunhofer.de/`



(A) BPI12

(B) BPI17

FIGURE 6.4: Process model for a loan application in 2012(a) and 2017(b)

The two Petri nets look quite different from each other, thus making it challenging for a human to identify meaningful mappings of similar portions. This highlights the needs for a set of features able to guide the analyst. We first embedded each activity in both processes in a feature vector composed of the features previously introduced. Then, we applied k-means to determine clusters that group activities of both processes. We leveraged a grid search in $[4; 20]$ to determine the best k, that resulted in $k = 14$.

Results

Table 6.1 shows the detected activity clusters, grouping on columns activities belonging to different processes. In this way, we define a mapping from the two sets of activities. Note that if a cluster contains only activities of the same process, such a mapping is not defined. Hence, for the sake of space, we limit our analysis to inter-process clusters, that is clusters containing activities from both processes These clusters are highlighted in Figures 6.4a and 6.4b with a labeled dotted red rectangle. Hereafter we briefly discuss these clusters.

c1: Process initialization The first cluster involves the first sequence of activities in both processes, interrupted with the occurrence of a XOR operator.

c2 and c4:Offer management The second and fourth clusters show two activity mappings related to the management of the offer. In both cases, one activity from bpi2012 activities has been mapped to a set of activities in bpi2017, which may indicate, for instance, that a more fine-grained modelling has occurred for the second model. Furthermore, the order between the activities of the first and of the second mapping reflects the order of the original activities $OSELECTED$ and $OSENT$. The activities $A\_validation$ and $W\_Call\ incomplete\ files$ are not included in the mapping to $OSENT$ because of the presence of a self-loop.

c3: Decision on application These clusters maps together activities related to decisions taken over an application. It is worth noting that the cluster of activities in bpi2012 is not straightforward to derive. Since $AACCEPTED$ and *OCREATED* occur before in the process and are in parallel with a different activity, analyzing the overall process structure, it would seem natural to either separate these activities in different clusters or merge them with the other

TABLE 6.1: Cluster Table

Cluster	2012	2017
c1	START; ASUBMITTED; APARTLYSUBMITTED	START; A_Create Application
c2	OSENT	O_Refused; A_Pending; O_Returned; A_Incomplete
c3	AFINALIZED; OCREATED; OCANCELLED; AACCEPTED	A_Denied; O_Accepted
c4	OSELECTED	W_Complete application; O_Sent (mail and online); O_Create Offer;A_Complete
c5	AACTIVATED; OACCEPTED; AAPROVED; ACANCELLED; AREGISTERED	A_Cancelled; O_Cancelled

occurring nearby. However, by focusing on structural features from the activity perspective, their similarities emerge; indeed, all of these activities are optional, occur in a similar position in the process, have a similar degree of parallelism and do not occur in a self-loop. These characteristics also justify the inter-process mapping with the corresponding activities in bpi2017.

c5: Application finalization Here we have multiple finalization activities from bpi2012 mapped to two finalization activities in bpi2017. This is in line with the fact that in the 2017 model the closure of the process has been significantly simplified w.r.t. the 2012 model. The inclusion of the activity $ACANCELED$ in this mapping set shows consistency between the mappings; indeed, in the original process this activity occurs immediately after the $OSENT$ activity, and in the new process it occurs immediately after the activities mapped to $OSENT$. Such relation would be quite challenging to identify from a visual inspection.

While this case study is mostly intended as a preliminary proof of concept for the introduced features, the clusters discussed above show that our local notion of similarity allows to generate non-contiguous partitions of the process model, thus uncovering relations among activities occurring in different section of the model that would be easily missed by a human analyst or by adopting graph-based, global similarity metric.

6.2 Next-location prediction for process executions

State-of-the-art approaches for the next activity prediction task are focused on predicting which specific process activity will be executed next given the current process execution. This information, however, does not provide the analyst with any insights into the execution context in which the predicted activity is executed. For instance, in the presence of a parallelism it can be argued that a human analyst would like to be informed also on which other activities could occur at that point of the process (and are hence likely to happen shortly after) even though with lower probability. This is particularly true when the chances of execution of the activities are close to each other. In addition, an analyst may want to understand whether the activity to be executed may belong to a cycle, or to know at which point of the process the execution is, e.g. at the beginning or close to the end. If the predictor only returns the activity in output, the only way for the analyst to derive such information consists of a manual inspection of the process model which can be a challenging and error-prone task, since real-life processes can involve complex combination of control-flow constructs with many activities and variants. To tackle this challenge, in this work we introduce an approach that, given an ongoing process instance, aims at predicting the execution context that the process will enter in the next step, hereafter referred to as *location*. Informally, a location is a set of activities with the same structural features, defined on the basis of workflow patterns commonly used in the process modeling domain. We argue that the notion of location provides us with a useful formalism to define activity execution contexts in a rigorous, yet intuitive, way. We provide a precise characterization of the notion of location, and we investigate three possible strategies to predict a location, namely i) an extension of the next-activity prediction problem, where the label of a single activity is predicted and mapped to its location, ii) a direct prediction of the numeric feature vector representing the structural properties of a location, and iii) a direct prediction of the label of the next location. To validate our approach, we carry out a set of experiments on real-world datasets frequently used as benchmarks within the Predictive Process Monitoring (PPM) field. The results obtained show the capability of the approach to support the analyst in the exploration of the process evolution.

The rest of the section is structured as follows. Section 6.2.1 formally introduces the notion of location; Section 6.2.2 provides the problem statement and the methodology to address it; Section 6.2.3 describes the experiments designed to assess the validity of the methodology and discusses the results.

6.2.1 Defining process locations

Given a reference model for the process under analysis, to determine the location of each process activity we exploit seven model-based features to represent the structural characteristics of process activities, as introduced in [65] and here described in 6.1. More precisely, let $\mathcal{A}$ denote the set of process activities, we define an embedding $f_e : \mathcal{A} \rightarrow \mathbb{R}^7$ which maps each activity into a vector of seven structural features, each characterizing a specific property of the activity with respect to the morphology of the overall process.

The idea underlying this representation is that activities with similar structural properties will be close to each other in the feature space. The proposed features have been designed taking inspiration from the standard workflow patterns[71], which capture concepts related to well-known control flow constructs, thus generating an embedding that is understandable for the human analyst. Note that while these features can, in principle, be derived for every process model able to represent the aforementioned control flow constructs, in this work we focus on workflow nets. All the features are the ones described in section 6.1, except for the "Not in Model" which is described right below here.

Not in Model: is a binary feature equal to one when the activity is not present in the given model, 0 otherwise. This feature represents the fact that when an activity is missing from the process model no structural information is available.

To better understand the notion of location let us consider the process model in Petri net notation reported in Figure 6.5. Table 6.2 shows for each activity the values for the structural features introduced above.

Activities with identical feature values determine a location. We use dotted squares to represent locations in Figure 6.5.

Locations L_1 and L_6 correspond to activity A and H, the only activities occurring at the beginning and at the end of the process respectively. From

TABLE 6.2: Structural features computed for the process model in Figure 6.5.

Activity	Path length	Optionality	Parallelism path length	Parallelism	Self loopable	Long loopable	Not in model	Location label
A	0.25	0.0	0	0.0	0	0	0	L_1
B	0.5	0.0	0.5	0.67	0	0	0	L_2
C	0.5	0.0	0.5	0.67	0	1	0	L_3
F	0.75	0.0	1.0	0.67	0	1	0	L_4
D	0.5	0.0	0.5	0.67	0	0	0	L_2
E	0.75	0.0	1.0	0.67	1	0	0	L_5
G	0.75	0.0	1.0	0.67	1	0	0	L_5
H	1.0	0.0	0.0	0.0	0	0	0	L_6

the Table, we can see that the location provides the analyst with additional insights on these activities. First, we can immediately see that none of them is involved in any parallel, choice, or loop construct, since the corresponding features are zero. From the path length, we also get a good idea on where the activity occurs within the process. L_2 involves B and D, since they both occur at the beginning of the parallelism immediately after A; note that C is instead in a different location, L_3, since it is involved in a loop. From the parallelism value, we can derive that each of these activities (except A and H) occur in parallel to other two (indeed, we have a total of 3 branches); furthermore, they occur halfway in the parallelism, since the parallel branch they belong to only has two activities in total. L_4 and L_5 involve activities occurring at the same position in the process, but that differ because of the presence of a self loop for E and G, while F is involved in a long loop.

We argue that the notion of location in the context of predicting process monitoring is an abstraction that can provide the process analyst with a higher-level overview of what can be expected next in the process execution. For instance, after the execution of A, traditional next-activity prediction techniques would return either B, C or D, whilst using the notion of location, either L_2 or L_3 is reported. This higher-level information allows the analyst to grasp important insights on the execution context of the ongoing process instance

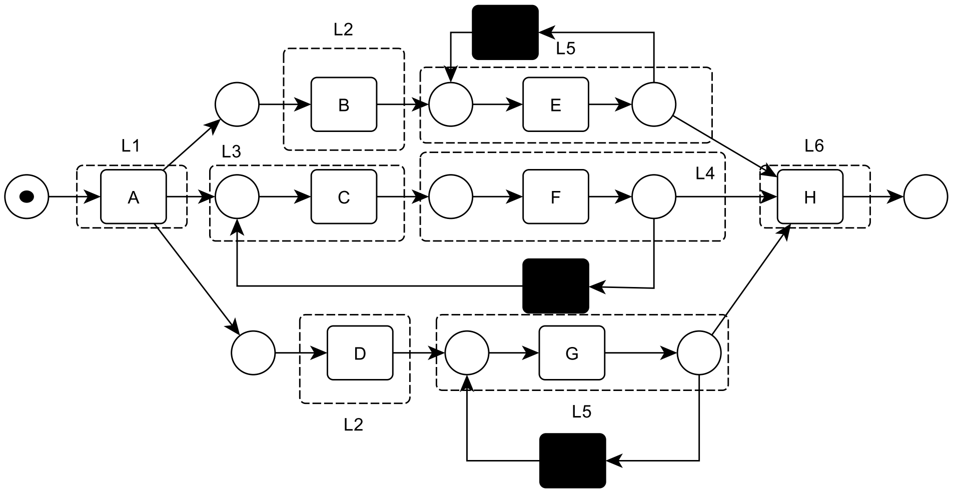


FIGURE 6.5: Example process model with activity locations

which can, in turn, help in getting decisions regarding the next steps. Keeping the notion of location in predicting the next execution steps in the process model in Figure 6.5, after one of the activities in L_2 or L_3 has been executed, the process can either stay in the same location (which suggests that multiple parallel branches are usually initiated before proceeding within a specific branch), or move towards L_4 or L_5, suggesting whether the process moves towards a potential repetition of multiple/single activities respectively.

6.2.2 Methodology

In this work, we introduce an extension of the next-activity prediction problem, hereafter referred to as *next-location* problem. In particular, we propose a robust approach that exploits both information regarding the process structure, extracted from a process model describing the prescribed process behaviors, and knowledge regarding the sequential execution order for the prediction. The proposed approach involves three main steps: i) *extraction* of location features, ii) *data encoding*, which transforms the event log in a set of prefix traces labeled with the corresponding location, and iii) *classifier training* to perform the prediction. The extraction of location features has already been discussed in Section 6.2.1. Hence, in the following subsections we first give a formal definition of the next-location problem statement, then we delve into the data encoding step. We do not discuss the classifier architecture and training, since off-the-shelf techniques can be applied.

Problem statement

In our setting, we assume the availability of an *event log* tracking the executions of the process under analysis and of a process model describing the prescribed process behaviors, either provided by domain experts or mined with process discovery techniques. Note that, within the context of this work, we assume that the provided model is able to properly represent (at least) the core process behaviors. Clearly, a poor quality model will heavily impact the performance of the classification, since the control-flow information used to determine the different locations would not correspond to the real process behaviors. We plan to investigate how to improve the robustness of the method w.r.t. the model in input in future work. An event log consists in a collection of *traces*, i.e., sequences of *events*, each corresponding to the execution of a process activity. Formally, let Π be the set of all prefix traces extracted from L, $\Theta \subset \mathbb{R}^7$ be the set of locations extracted by using the features defined in Section 6.2.1, let S be a set of location labels, and let $f_m : \Theta \ \rightarrow S$ be a function mapping each location to a location label.

The *next-location* prediction problem can be defined as the one of learning a mapping function able to label each prefix trace with the process location of the next state of the process.

Data Encoding

Given the the set Π of m trace prefixes that can be generated from L, the goal of this step consists in determining a function mapping each prefix to its vectorial representation. Formally, let V be a vector of size j, we have to define the function $f_v : \ \Pi \rightarrow V$. In this work, we investigate three strategies to generate the prediction, which require three different kinds of encoding. The first one corresponds to an enrichment of the output of a next-activity classifier, hereafter referred to as *enriched next-activity prediction* (ENAP). The construction of the prefix set for this strategy is the same as in traditional next-activity prediction; namely, we build the dataset $P = \{(f_v(p_i), a_i)\}_{i=2}^{m}$ where p_i is a prefix of length k of a trace $\sigma \in L$ and a_i corresponds to the next activity of the partial execution described by p_i. Once an activity is returned by the classifier, the function f_m is applied to return in output its corresponding location label. The second strategy aims at returning, given the current state of the process execution, directly the process location label, without first predicting the next activity. We refer to this strategy as *next location label prediction* (NLLP). To support this strategy, we build the dataset $P = \{(f_v(p_i), s_i)\}_{i=2}^{m}$ where p_i is a prefix of length k of a trace $\sigma \in L$ and s_i corresponds to the label of the next location of the partial execution described by p_i. Finally, the third strategy corresponds to the prediction of the next location feature vector, instead of its label. In practice this strategy corresponds to a regression problem, hereafter referred to as *direct next-location prediction* (DNLP). For implementing this strategy, we need to build a dataset $P = \{(f_v(p_i)), \theta_i\}_{i=2}^{m}$, where p_i is a prefix of length k of a trace $\sigma \in L$ and $\theta_i \in \Theta$ corresponds to the next location of the partial execution described by p_i.

In all the discussed strategies, the feature vector V can be computed in three different ways; namely, one can generate the one-hot encoding of the process activities, consider their corresponding location, or the combination of the two.

6.2.3 Experiments

This section describes the experiments we carried out on a set of real-world datasets to assess the performance of our approach and to provide some examples on the kind of predictions that are returned.

In particular, we are interested in answering the following questions: *RQ1: How does the direct location prediction performs w.r.t. the location prediction via next-activity prediction? RQ2: What is the impact of the use of the location features on the prediction performance?*

In the following, we first provide a description of the experimental setup and the selected datasets; then, we discuss the results obtained.

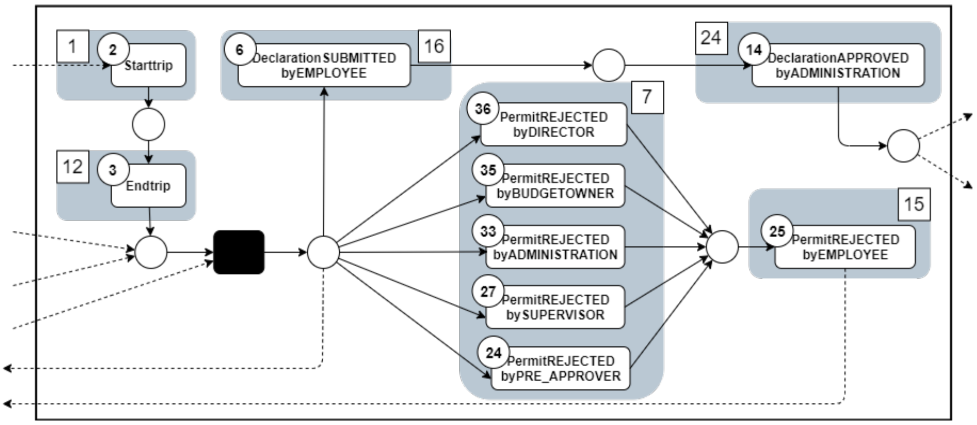

1
2 Starttrip
12
3 Endtrip
6 DeclarationSUBMITTED byEMPLOYEE
16
36 PermitREJECTED byDIRECTOR
35 PermitREJECTED byBUDGETOWNER
33 PermitREJECTED byADMINISTRATION
27 PermitREJECTED bySUPERVISOR
24 PermitREJECTED byPRE_APPROVER
7
24
14 DeclarationAPPROVED byADMINISTRATION
15
25 PermitREJECTED byEMPLOYEE

FIGURE 6.6: Excerpt from the ID process model

Experimental setup

In order to answer RQ1, we recall that the prediction problem may be approached in one of the following way:

- Predict the next activity label, then associate it with the location it belongs (**ENAP**),
- Predict the next location label (**NLLP**),
- Predict the next location (**DNLP**).

To answer RQ2, we consider 3 possible representation for the input data encoding:

- only the one-hot encoding of each activity label (**OHE**);
- only the location of each activity (**ENC**);
- a combination of the two above (**ALL**).

We report the experiments only for the 4 combinations of prediction problem and input representation that, after some first trials, seemed most promising, but we plan to investigate all possible combinations in future works. In particular: $ENAP + OHE$, which implements the classification of the next activity followed by the mapping to the location; $DNLP + ENC$, which determines the

TABLE 6.3: Structural features computed for the activities of process model in Figure 6.6

Activity	**Path length**	**Optionality**	**Paralelism Path Length**	**Paralelism**	**Self Loopable**	**Long Loopable**	**Not in model**	**Location label**
2	0.5556	0.6667	0.9	0.5	0	1	0	**1**
3	0.6111	0.6667	1.0	0.5	0	1	0	**12**
33	0.6667	0.8333	0.0	0.0	0	1	0	**7**
36	0.6667	0.8333	0.0	0.0	0	1	0	**7**
24	0.6667	0.8333	0.0	0.0	0	1	0	**7**
27	0.6667	0.8333	0.0	0.0	0	1	0	**7**
35	0.6667	0.8333	0.0	0.0	0	1	0	**7**
6	0.6667	0.0	0.0	0.0	0	0	0	**16**
14	0.7222	0.0	0.0	0.0	0	0	0	**24**
25	0.7222	0.5	0.0	0.0	0	1	0	**15**

location as a regression problem, returning in output the set of location features; $NLLP + ALL$, which exploits both the location features and the activity label, returning in output the location label; finally, $NLLP + OHE$, which only exploits the activity label and returns in output the location label. All the experiments have been performed using tensorflow 2.5 [59] and python 3.8.10 on a machine with a Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz, 32GB on RAM and two GPUs GeForce GTX 1080 with 6GB of dedicated memory each.

We selected for each approach the same neural network architecture: two LSTM layers each with a dropout and a BatchNormalization layer after them and, finally, a fully connected output layer. We also used the same optimizer, Adam [57] and we used as loss function a mean absolute error for the ENC approach and a categorical cross entropy for all the other approaches. For any unexplained parameter the default value has been used. We also run the tree-structured Parzen estimator (TPE) hyper-optimization algorithm [72] with 30 as maximum number of iterations. In particular, the optimization space was composed of the learning rate in the range $[10^{-6}, 10^{-2}]$ and of the LSTM units and dropout, respectively in the set $\{32, 64, 128, 256, 512\}$ and in the range $[0, 1]$, for both the 1st and 2nd layer (separately), for a total of 5 hyper-optimized parameters. The process models were extracted using the Infrequent Inductive Miner[73] with the lowest noise threshold that granted at least a fitness of 90%. All the above have been performed in a 3-fold fashion division of the dataset where a 20% of the training dataset have been used as validation set.

We selected the following as benchmark datasets (see sec. 2.3):

- *Helpdesk*
- *BPI12*
- *BPI20RfP*
- *BPI20TP*
- *BPI20PC*
- *BPI20ID*

6.2.4 Results and discussion

Example of predicted output

To provide an example of the output returned by our approach, in Figure 6.6 we draw a portion of the model from the ID dataset. Table 6.3 shows the location features and location labels for the activities in the Figure. Locations are also represented in the Figure as grey rectangles.

Let us now discuss the predicted output obtained by the different encodings when the process reaches the *Endtrip* activity. Using *DNLP*, the predictor outputs a location feature vector describing the properties of the next location. The most similar to this one among the all available true feature vectors will be considered the predicted one. In our example, the predicted output was $[0.6567, 0.8463, 0.02, 0.09, 0.01, 0.98, 0.01]$; the most similar location feature vector is $[0.6667, 0.8333, 0.0, 0.0, 0, 1, 0]$, whose label is 7. Using *ENAP*, the predictor first outputs a distribution of probabilities for each process activity; in this case, the predictor returned 36, which is the most likely activity in the set $\{6, 20, 24, 27, 33, 35, 36\}$. That output is then converted in its corresponding location, i.e., 7. This is an example of how it is possible to predict the next location from the next activity. Using *NLLP*, the output will be a distribution of probabilities of the next location classes; in this case, 7 was the most likely location label from set $\{5, 7, 16\}$. We remark that this prediction is performed without the intermediate activity-location mapping used in *ENAP*. As regards the interpretation of the outcome, the user can determine the level of abstraction she prefers, i.e., whether to have in output information only on the location or also on the activity, choosing the appropriate format for the input and the output. In some cases, it might be enough to know that the permit is likely to be rejected by some authority, even without knowing the precise type of rejection. Also, as discussed for the toy example in Section 6.2.1, the analysis of the location feature equips the analyst with a good overview of the current execution context with respect the overall model, which can be quite challenging to derive manually, given the size of the model. For instance, the analyst is immediately aware that activities of location 7 can lead to work repetition, since they are part of a long loop. This makes sense, since it is reasonable to assume that if a permit is rejected, additional actions have to be taken. On the other hand, activities of locations 16 and 5 do not belong to any cycle and they are closer to the end of the process, as shown by the path length. Summarizing, the used notion of location successfully manage to group together all the activities connected to a rejection from a human controller that lead to potentially restart the procedure business process. We would like to highlight once more that the user can derive such information from our output directly, without having to manually inspect the process model.

Analysis of prediction performance

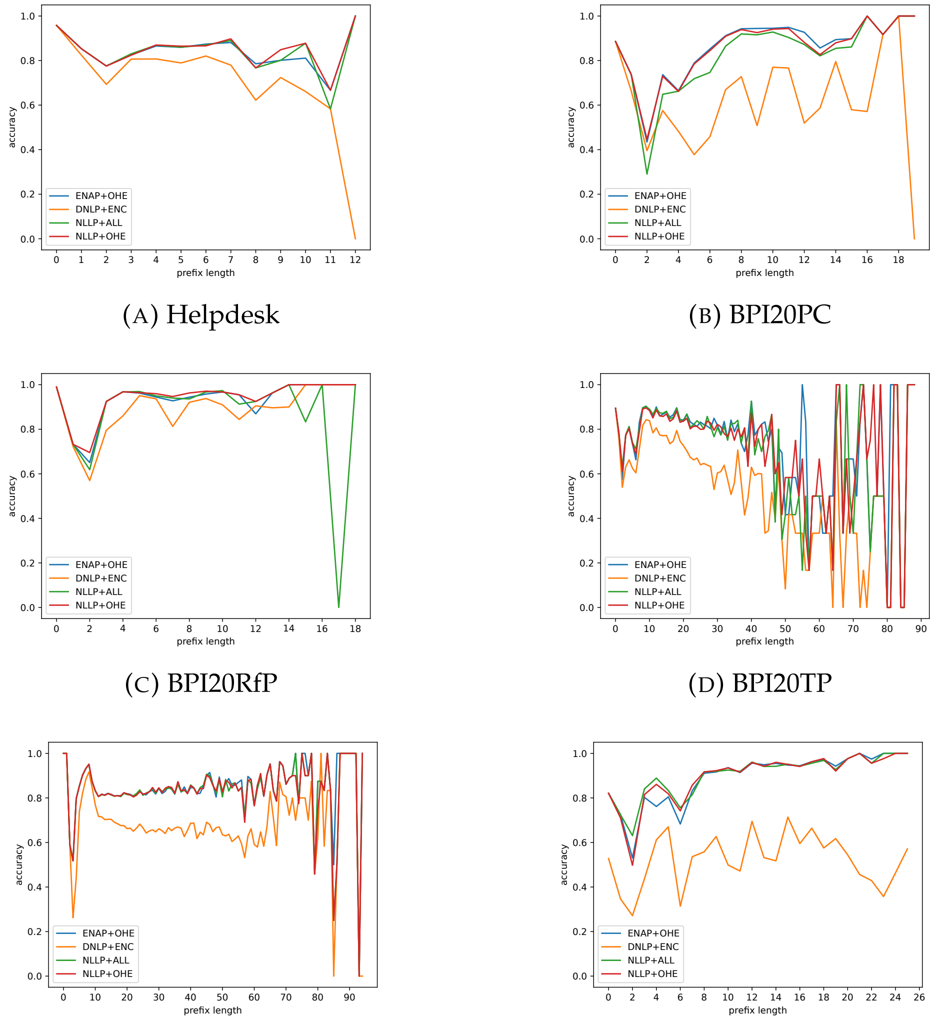
Table 6.4 reports the results obtained over the selected datasets. The $DNLP + ENC$ approach is the one obtaining the worst results. This was expected, since it cannot leverage information related to the specific activities, in contrast with the the other approaches. Instead, there is no clear winner between the two location prediction strategies, since both obtain very similar results. Also regarding the use of the location features no clear trend can be identified. Indeed,

TABLE 6.4: Classification results from the tested encodings. We report the accuracy (**acc**) and the f1 score (**fsc)** for each fold, together with the average results

		ENAP+OHE		DNLP+ENC		NLLP+ALL		NLLP+OHE	
Dataset	**fold**	**acc**	**fsc**	**acc**	**fsc**	**acc**	**fsc**	**acc**	**fsc**
Helpdesk	0	0.85	0.85	0.81	0.80	0.86	0.85	0.86	0.85
	1	0.85	0.85	0.81	0.80	0.86	0.85	0.86	0.85
	2	0.86	0.85	0.84	0.83	0.86	0.85	0.85	0.85
	avg	**0.85**	**0.85**	**0.82**	**0.81**	**0.86**	**0.85**	**0.86**	**0.85**
BPI20PC	0	0.56	0.58	0.42	0.44	0.53	0.55	0.56	0.58
	1	0.88	0.86	0.69	0.69	0.88	0.86	0.88	0.86
	2	0.89	0.88	0.62	0.60	0.77	0.71	0.88	0.86
	avg	**0.78**	**0.78**	**0.58**	**0.57**	**0.73**	**0.70**	**0.77**	**0.77**
BPI20RfP	0	0.90	0.88	0.86	0.83	0.91	0.87	0.90	0.88
	1	0.90	0.86	0.89	0.95	0.89	0.86	0.90	0.86
	2	0.82	0.79	0.71	0.68	0.81	0.79	0.85	0.82
	avg	**0.87**	**0.84**	**0.82**	**0.79**	**0.87**	**0.84**	**0.88**	**0.85**
BPI20TP	0	0.74	0.72	0.60	0.58	0.75	0.73	0.75	0.72
	1	0.84	0.82	0.80	0.77	0.84	0.83	0.84	0.82
	2	0.83	0.82	0.76	0.73	0.84	0.82	0.83	0.80
	avg	**0.80**	**0.79**	**0.72**	**0.70**	**0.81**	**0.79**	**0.80**	**0.78**
BPI12	0	0.82	0.80	0.72	0.70	0.82	0.80	0.82	0.80
	1	0.82	0.80	0.68	0.67	0.82	0.80	0.82	0.80
	2	0.82	0.80	0.73	0.71	0.82	0.80	0.82	0.80
	avg	**0.82**	**0.80**	**0.71**	**0.69**	**0.82**	**0.80**	**0.82**	**0.80**
BPI20ID	0	0.69	0.68	0.41	0.41	0.73	0.72	0.72	0.71
	1	0.87	0.86	0.42	0.39	0.89	0.88	0.87	0.87
	2	0.86	0.83	0.67	0.65	0.88	0.86	0.86	0.86
	avg	**0.80**	**0.79**	**0.50**	**0.48**	**0.83**	**0.82**	**0.82**	**0.81**

$NLLP + ALL$ leads to some improvements w.r.t. $NLLP + OHE$ for the TP and the ID datasets; however, it obtains worse results for the RfP dataset and, in particular, for the PC dataset, suggesting that in those experiments the use of the structural features do not provide useful information for the classifier and, instead, it likely introduced some noise. To shed some light on the reasons underlying these results, we delved into the mined process models to understand how locations were defined. In particular, we computed the ratio between the number of the activities in the process model and the number of identified locations. In this regards, it is worth noting that for the PC dataset, where $NLLP + ALL$ obtained the worst results, we have 24 locations for 30 activity type; this suggests that the introduction of the location features does not bring much abstraction, supporting the hypothesis that they likely introduce more noise than useful information. The other datasets show a lower ratio, which indicates that more activities could be grouped in the same locations, thus presenting a more favorable condition for the application of our approach. Note however that the mapping between the activities and the locations is not the only factor affecting the classification performance. For instance, helpdesk also presents a high activity-location ratio, without resulting in a reduction of performance. This may depend on the fact that helpdesk is an easier problem, having a very small number of activities and quite a linear structure.

Finally, we analyze the performance with respect to prefix lengths, to study the stability of the outcome. Results are reported in Figure 6.7. Overall, the figures confirm the trends we already observed; the various approaches perform very close to each other for most prefix lengths, even though the performance of $DNLP + ENC$ are consistently lower than the others. $NLLP + ALL$ performs quite close to $NLLP + OHE$ in most cases, even though some peaks in performance decrease occur for some prefix length, in particular for the BPI20RfP dataset. BPI20TP and BPI12 show quite unstable results for all the encodings for long prefixes, which is reasonable, since few samples are available for training.



(A) Helpdesk

(B) BPI20PC

(C) BPI20RfP

(D) BPI20TP

(E) BPI12

(F) BPI20ID

FIGURE 6.7: Accuracy across temporal windows.

Chapter 7

Conclusion

The present thesis aims at addressing the field of Predictive Process Monitoring by introducing novel approaches and new perspectives on its problems. The notion of Instance Graph is leveraged to be exploited for the next activity prediction task and to introduce a novel metric to measure parallelism. A new encoding technique for the encoding of process activities is also proposed and it is shown how to use it both to compare processes as well as to define a new kind of task for the Predictive Process Monitoring field. In particular to leverage Instance Graph, BIG-DGCNN has been presented: a model-aware neural approach to address the task of next activity prediction. The model allows to represent process instances in the form of Instance Graphs, thus maintaining information about parallel activities that is missed in the traces recorded in event logs. Graphs are then natively processed by Deep Convolutional Graph Neural Networks to synthesize a classification model able to predict the next activity given a prefix of any length. The adoption of BIG allows to build sound Instance Graphs even for non-fitting traces and makes the approach suitable also for unstructured processes. Furthermore, an extension is proposed which enriches the Instance Graph with additional data perspectives. The comparison with state-of-the-art literature highlights that BIG-DGCNN shows promising performance, especially considering that competitor approaches all take into account some data perspective, whereas BIG-DGCNN only encodes control-flow information. Endowing BIG-DGCNN with temporal information,it demonstrates to favorably compare to other approaches. Further analysis of the performance trend with respect to the prefix length enlightens an interesting difference with respect to the LSTM architecture, that is the decay of performance on longer prefixes.

To characterize the processes in which the BIG-DGCNN approach could better work, it has been addressed the problem of evaluating the level of parallelism in a business process. The limits of the two best-known metrics in the literature have been discussed and a novel metric has been introduced, the Process Complexity and its scaled variant, both based on the notion of Instance Graph. The idea underlying the proposed metric is that the parallel complexity of a process linearly depends on the number of distinct variants allowed by each IG, representing the possible executions of the process. This metric provides a sensible way to evaluate the overall complexity of the process model due to the number and structural relations among parallel activities. None of the metrics, on the other hand, is fully satisfactory showing some incoherent behaviors with respect to the number of activities or the presence of alternative branches. A possible conclusion is that more than one metric should be taken into account when evaluating the parallelism of a process. The results are not yet conclusive as they have been obtained on a set of synthetic use cases. Future works may be devoted to confirming the results both theoretically and empirically on a larger set of process models. Another research direction is toward the introduction of different scaling mechanisms that takes the length of a sequence of activities into due consideration.

Then, in order to exploit the information derived from the process model, a set of features, characterizing the process activities on the basis of structural properties of the control-flow constructs they are involved in, have been defined. We discussed how these features can be used to address the problem of activity identification and comparison between process models. In turn, this enables more sophisticated analyses, ranging from retrieving process models from a process repository starting from a given reference process to the evaluation of the changes that have occurred in a model over time. We applied our embedding strategy on a case study involving two process models describing the same real-world loan management application process at different times. In particular, we used such an embedding to realize an activity clustering between the two process models. The results are promising and show the capabilities of the adopted features to highlight similarities among activities that would be hard to grasp adopting name-based or global similarity features. Nevertheless, these experiments represent a first preliminary study to investigate the potential of the approach. Furthermore, such a set of features has been exploited to propose an approach aimed at predicting the location of a process that is likely to be executed next. The notion of location provides an abstraction mechanism from the level of the single activity, which can be used to supply the process analyst with a higher-level overview of what can be expected next in the process execution. We used seven structural features representing common control-flow constructs to define the notion of location. We provided three different formulations for the next-location prediction problem, with different assumptions on the format of the input and the output. We tested our approach on a dataset involving five real-world event logs. The obtained results have shown little differences among the tested formulations, which obtained in any case good prediction performance. We also discussed an example of the output, to better highlight differences in the output generation obtained by the different formulations and to show how the location features can be used to support the analyst in grasping relevant insights on the execution context.

A relevant research direction includes the refinement of the introduced scaled parallel complexity and a study on its correlation with the performance of both a parallelism-aware approach (like Multi-BIG) and a purely-sequential approach. Future works also include the extension of the Instance Graph-based encoding by the introduction of a set of features that represents the behaviors stored in the event log. This in turn also offers the opportunity to extend the notion of Location and its possible applications and refinements. Experiments on more datasets are for all the above a further research direction.

Appendix A

Further PhD activities

In this Chapter, some of the side activities carried on during the Ph.D. are described. In particular, the work produced following the very first research direction of the Ph.D. course and the consulting activities performed for the Namirial s.p.a. R&D department.

A.1 Reinforcement Learning for Predictive Process Monitoring

Here, it is described the first research direction followed at the beginning of the Ph.D., in the hope that the adoption of the Reinforcement Learning paradigm could result later in an evolution of the predictive process monitoring to prescriptive. This particular research direction was then temporarily suspended in favor of the more promising ones described in the thesis. Reinforcement learning is gaining ever-increasing attention since 2015 when [74] trained an agent that bested many human professional players over various Atari games. This has led the scientific community to further investigate the techniques, leading to various interesting results (e.g., [75], [76]), up until the latest astonishing artificial agent [77] that managed to beat professional human players in Starcraft II, an extremely complex real-time strategy game.

Reinforcement Learning is a particular kind of machine learning paradigm that trains models to directly maximize a reward signal, without assigning any label or necessarily trying to find some hidden structure in the data. An interesting feature of this family of algorithms is that learning is guided by an objective function that takes into account the chain of future decisions and their effects, instead of focusing only on the decision at hand. This could be an interesting feature in the predictive process mining field, where events to be predicted are conditioned by the process workflow. Motivated by past successes of Reinforcement Learning and this observation, we set as our goal to study if it is possible to apply Reinforcement Learning to the field of predictive process monitoring. At the best of our knowledge, this was the first study of this kind. The only other study that applied reinforcement learning in the process mining field is [78], where the problem of efficient resource allocation is considered. Our results enlighten promising features of the approach and interesting research issues towards a prescriptive generative approach to improve processes.

As also confirmed by a successive work [79] where from two different datasets the authors manage to derive process simulations and, using two different KPIs as loss functions, to train actors capable of prescript actions.

A.1.1 Background

In this section, we provide general background knowledge on the Reinforcement Learning paradigm.

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. A reinforcement learning problem is formalized using ideas from dynamical systems theory, specifically, as the optimal control of incompletely-known Markov decision process [80]. The learner (agent) interacts with an environment during a sequence of timesteps composing the learning episode. In the domain of process mining we can think the learning episode as the evolution of the trace, and the occurrence of an event in it as a timestep. At each timestep, an interaction between agent and environment occurs through observations (x) of the environment, actions (a) and rewards (r). The observation is the trace event, the action is the prediction for the next event, and the reward is derived from the comparison between the next event information and the predicted one. The agent's goal is to maximize its cumulative future reward performing its actions, with respect to the state of the environment. The state s_i of the environment is defined as the full sequence of observations and actions performed until timestep i, formally: $s_i = x_1, a_1, x_2, ..., a_{i-1}, x_i$. However, it is complex to use a state composed of a variable number of observations and actions as input. Hence, it is usually preferred to use a constant fixed number of observations and actions. In this chapter we refer to the number of past timesteps considered to define the state as *window size*. Having fixed the window size to a generic k, the state is written as $s_i = x_{i-k}, a_{i-k}, x_{i-k+1}, ..., a_{i-1}, x_i$.

The objective function of the agent at timestep i can be expressed as:

$$R_i = \sum_{t=i}^{T} \gamma^{t-i} r_t, \tag{A.1}$$

where $\gamma < 1$ is the discount factor of future rewards, used to prioritize more recent rewards, and T is the number of timesteps in the whole learning episode. In particular, in this methodology we considers Q learning agents. This type of agents tries to approximate the optimal action-value function $Q^*(s, a)$, learning it from the transitions from a state s_i to a next state s_{i+1} on the basis of the performed action a_i and the received reward r_i. The optimal action-value function may be expressed as:

$$Q^*(s, a) = \max_{\pi} \mathbf{E}\left[R_i | s = s_i, a = a_i, \pi\right] \tag{A.2}$$

which is the maximum expected reward achievable after seeing sequence s and taking action a, by following any behaviour policy π for mapping sequences to actions. This reinforcement learning algorithm is based on the fact that knowing $Q^*(s, a)$ an agent can choose the best sequence of actions at any state, maximizing its reward. Obviously, perfectly knowing $Q^*(s, a)$ it is not always possible, especially in complex environment. Still it is possible to discover $Q(s, a, \theta)$, through a machine learning model, where θ are the parameters of the trained model so that $Q(s, a, \theta) \approx Q^*(s, a)$. In the case of deep Q network (DQN) agents, the model adopted is a deep neural network, and θ are its weights, used to approximate the optimal action-value function. In our study we used as underlying model to approximate the Q-function an LSTM based neural network. The network weights may be adjusted through training using as loss function, that varies at each timestep, the mean squared error defined as follows:

$$L_i(\theta_i) = \mathbf{E}_{(s,a,r,s')} \left[r + \gamma \max_{a'} \tilde{Q}(s', a', \theta_i^-) - Q(s, a, \theta_i) \right]^2 \tag{A.3}$$

in which γ is a discount factor determining a penalty for more future reward, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i used in place of the optimal and unknown $max_{a'}Q^*(s', a')$. Therefore, in contrast to supervised learning where targets are fixed before learning begins, the targets depends on the network weights. Though, since θ_i^- is kept fixed at the ith optimization of $L_i(\theta_i)$, all the optimization problems at each iteration are well defined. In our case we used a so called soft update of θ which updates the parameters at each iteration on the basis of a coefficient β accordingly to the formula: $\theta_{i+1}^- = (1-\beta)\theta_i^- + \beta\theta_i$, where $\beta \in (0, 1)$. It is worth noting that this algorithm is model-free as it solves the Reinforcement Learning task directly, without estimating the system transition dynamics. Also it is an off-policy algorithm, since it learns a greedy policy where $a = argmax_{a'}Q(s, a', \theta)$ but it still ensure, through its behaviour policy, an adequate exploration of the state space through a random action. This allows to discover if there are better actions to perform with respect to the recommended one. In our particular case, for training, we used a Boltzmann Q Policy, which builds a probability law on Q values and returns an action selected randomly according to this law while, for prediction purposes, a GreedyQPolicy is adopted which selects the action with the highest reward.

For further details, the full description of the algorithm can be found in [74], which originally proposed it.

A.1.2 Methodology

This section is devoted to describe the proposed methodology, which uses two agents trained through reinforcement learning to predict activity and execution time of both the one step ahead event as well as the activities suffix and trace time.

In the following, we first describe the pre-processing performed to make event log data suitable for being fed to our system, and then the details on the architecture adopted.

An event e_i in the sequence given in input to the LSTM used here, is logically represented by 4 components, namely the activity a_i, and three temporal features. Each activity is expressed by a binary vector built using the one-hot encoding of the activity type. One-hot encoding has been chosen as it is an effective and popular way of representing categorical data. Its main advantage is that one-hot encoding transformation does not introduce any order or similarity among the representation of categorical data. Regarding the three added temporal features, the first is the time passed between Sunday midnight and the event e_i ($\mathrm{t}_{\mathrm{w}_\mathrm{i}}$ in eq. A.4) which is useful to express the seasonality of the process. The second is the time passed between the completion of an event e_i and the completion of the previous event e_{i-1} ($\mathrm{t}_{\mathrm{e}_\mathrm{i}}$ in eq. A.5), thus substantially corresponding to the event duration (plus possible idle time between the two events). The last temporal feature is the time passed between the start of the trace and the event e_i ($\mathrm{t}_{\mathrm{t}_\mathrm{i}}$ in eq. A.6), which gives information about the progression of the trace. This last one is particularly relevant since there may be a strong correlation between the performed actions and the "age" of the process case.

Formally:

$$t_{\mathrm{w_i}} = \frac{t_{\mathrm{i}} - t_{\mathrm{w}0}}{\Delta t_{\mathrm{w}}} \tag{A.4}$$

$$\begin{cases} t_{\mathrm{e_i}} = 0 & \text{if } i = 1, \\ t_{\mathrm{e_i}} = \frac{t_{\mathrm{i}} - t_{\mathrm{i\text{-}1}}}{\Delta_{\mathrm{max}_e}} & \text{otherwise} \end{cases} \tag{A.5}$$

$$t_{\mathrm{t_i}} = \frac{t_{\mathrm{i}} - t_0}{\Delta_{\mathrm{max}_t}} \tag{A.6}$$

where t_i is the timestamp of the event at index i, t_{w0} is the timestamp of the last passed Sunday midnight, and t_0 is the start timestamp of the process. Δt_w Δ_{max_e} and Δ_{max_t} are normalization factors to make features varying in the range $[0, 1]$, as it improves the performance of the network. Δt_w is the amount of time in a week, while Δ_{max_e} and Δ_{max_t} are, respectively, the maximum event duration and the maximum trace duration. It is also worth noting that given t_{e_i} and both $t_{w_{i\text{-}1}}$ and $t_{t_{i\text{-}1}}$, it is possible to derive the value of both t_{w_i} and t_{t_i}.

Regarding the overall architecture, we show it in figure A.1: the dashed lines enlighten the learning phase, while solid lines refer to the prediction phase. In the system, we have two different agents. Both take as input a sequence of events, in which every event is defined by the three temporal features and the one-hot encoding of the activity as explained before. One agent predicts the one step ahead activity, the next one that will be performed, while the other is devoted to predict its completion time. As said, every DQN agent has an underlying neural network that models the reward function. For each of our two agents, we used an LSTM based neural network to learn and approximate the optimal $Q^*(s, a)$, instead of training them using ground-truth labels, typical of supervised learning. This is done, through the agents' interaction with their respective environment, thanks to which they receive their reward. The LSTM architecture have been chosen because of its widespread adoption in predictive process mining. We hasten to note that DQN agents only work with a discrete action space and therefore they are unable to produce continuous outputs. To address this issue we divided the output time in bins, each representing the range in which the estimated time falls, and we designed the time agent so as to produce bin indexes as outputs.

As explained in section A.1.1, the learning process of our RL agents is based on the notion of transition from a state of the environment to another on the basis of the performed action and its associated reward at each timestep.

We set the reward functions in each environment as binary reward functions: in the time environment, the reward gives a plus one when the predicted bin included the true time, and zero otherwise; similarly in the activity environment the reward gives a plus one when the prediction is correct and a zero otherwise.

For the one step ahead prediction of the next activity and time the two agents work in isolation exploiting their underlying LSTM network model to perform their prediction. For suffix prediction the situation is more complex, as each agent has access only to the information of the first part of the trace. In particular, it reads only the first k events where k is the window size. Hence, each agent needs to rely both on its own prediction and on the other agent's prediction to have all the required inputs for predicting more than one step ahead, as the true information is not available. In a way, the two agents cooperates exchanging messages to inform the other of their prediction, at each timestep. This way the whole sequence may be predicted using the predicted information when the true one is not available. All this is iterated until the end of the trace is predicted.

Formally, at the first iteration we consider the sequence $\sigma_k =< e_1, e_2, \ldots, e_k >$ of events of length k (window size), where e_j be the j-th event of a trace, which is characterized by the tuple $< a_j, t_{w_j}, t_{e_j}, t_{t_j} >$. The time predictor agent α_t and the action predictor agent α_a are defined as follows:

$$\alpha_t : \sigma_k \mapsto t'_{e_{k+1}},$$

$$\alpha_a : \sigma_k \mapsto a'_{k+1},$$

where apex denotes the predicted value. Each agent will inform the other of its prediction and therefore the predicted next event e'_{k+1} will be characterized by the tuple $< a'_{k+1}, t'_{w_{k+1}}, t'_{e_{k+1}}, t'_{t_{k+1}} >$, where $t'_{w_{k+1}}$ and $t'_{t_{k+1}}$ are derived from $t'_{e_{k+1}}$, t_{t_k} and t_{w_k}. Then a new prediction will be performed by each agent using as input $\sigma_{k+1} =< e_2, \ldots, e_k, e'_{k+1} >$. Iterating at the i-th step, the sequence σ_i will be formed by $k - i$ real events and i predicted ones. The algorithm is iterated until the end event of the process is predicted.

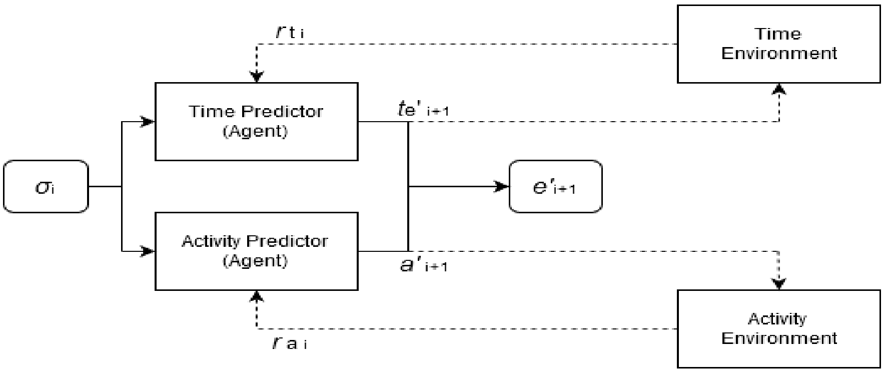


FIGURE A.1: Overall architecture.

A.1.3 Evaluation

In this section we describe the experimental setup used and empirically evaluate the performance of the proposed approach. Results are compared with those of other approaches using LSTM networks for uniformity reasons, so to remark the contribution of RL paradigm.

The experimental dataset used here is $BPI12W$ (see sec. 2.3. As done in [20], to perform our experiments we used chronologically ordered first $2/3$ of the traces as training data, and evaluate the activity and time predictions on the remaining $1/3$ of the traces. The dataset has been pre-processed as explained in section A.1.2. For what concerns the setting of bins defining the output values of the time agent, we analyzed the whole distribution of events duration in the dataset. This allowed to set the various ranges so as to both balance the number of elements in a bin and to maintain a reasonable similarity between elements in the same bin. The resulting bin endpoints are [0, 1, 10, 60, 120, 240, 480, 1440, 2880, 4320, 7200, 10080, 14400, 20160, 30240, 40320, 50400] expressed in minutes. Also note that the chosen endpoints correspond to meaningful time frames such as hours, days or weeks. Figure A.2 shows the distribution of events duration in the dataset. The x-axis is in logarithmic scale for visualization purposes.

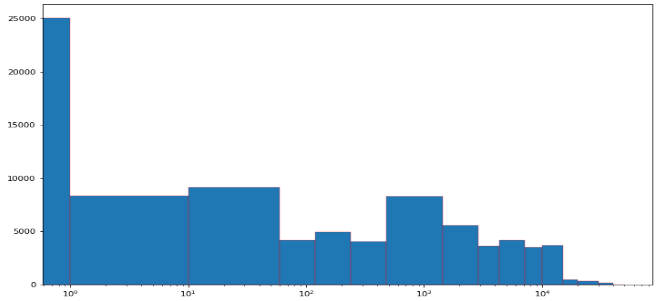


FIGURE A.2: t_e distribution in bins

We performed the experiments using Keras-rl [81], running on a machine with two NVIDIA GeForce GTX 1080, a i7 8700K CPU @3.70 GHz and 32GB RAM. Each agent was trained for 600000 steps and is characterized by the use of a sequential memory of dimension 500000, a BoltzmannQPolicy clipped in range (-15,15) as behaviour train policy, and a GreedyQPolicy as test policy; the target function was updated through soft update using $\beta = 10^{-2}$ as coefficient. The underlying neural network has two hidden LSTM layers with 200 neurons each and ReLU activation; during training we used an Adam optimizer with a learning rate of 10^{-3}. This configuration was kept for all the tested window size, as it had the best performance for approximating the Q function, between those tested.

In order to properly compare our results with previous work, we adopted the same evaluation metrics. For the *One-step Ahead Prediction*, we evaluate our results in terms of accuracy, and in terms of mean absolute error (MAE) in days, for the predicted time. For the purpose of comparison with the baselines we use the MAE but it is important to remember that our time agent predicts ranges of time. Therefore, since we need a continuous value for the time in order to compute the MAE, we choose for this the inferior endpoint of the bin predicted as value. For example, if the predicted bin is the third one, which corresponds to range $[10, 60)$, the time used for computing the MAE will be 10 minutes. For the *Suffix Prediction*, and in particular, for the suffix completion time prediction we consider the absolute trace duration error (TDE)

$$TDE = |t'_{t_f} - t_{t_f}| \tag{A.7}$$

where, with some abuse of notation, f refers to the final event in the true and estimate trace, hence t_{t_f} (t'_{t_f}) represents the total duration of the true (estimated) trace. The TDE is then averaged over all traces. For evaluating the accuracy of the activity *Suffix Prediction* the most well-known and used distance is the Damerau-Levenshtein distance, which is defined as the minimum number of deletion, insertion, substitution and transposition operations needed to transform the first string to the second. In particular, this distance can be normalized dividing its value for the length of the longer string. What we adopted for comparison purposes is the Damerau-Levenshtein similarity expressed as one minus the normalized Damerau-Levenshtein distance.

Regarding the results, in Table A.1 we present the performances achieved for the one step ahead prediction tasks. For next completion time prediction (Table A.1.(a)) we compare our results with the best reported by Tax et al. [20]

TABLE A.1: Comparison of performances for the one step ahead prediction tasks. (a) Next completion time. (b) Next activity.

	MAE (days)	
Window size	Ours	Tax et al.
2	1.34	1.69
10	1.05	1.45
20	0.62	0.98
All	1.17	1.59

(a)

Accuracy		
Ours	Tax et al.	Camargo et al.
71.3%	76%	77.8%

(b)

for different window sizes. Table A.1.(b) reports the accuracy of the next activity prediction of our method, and the ones reported by Tax et al. [20] and Camargo et al. [21]. In [21] the next completion time task is not addressed. In Table A.1.(a) the row "All" reports the average performance over all the tested window sizes. In [20] these correspond to all the values in the range [2,20], whereas in our case we considered the set {2,3,4,5,6,7,10,20}.

It can be seen that our performance in the next completion time prediction are clearly better than the baseline, whilst our accuracy is worse. In particular, the relative improvement in the case of completion time prediction is about 27%, and the relative accuracy degradation is only about 8% with respect to best result provided by [21]. These results may be justified as follows. DQN agents optimize a cumulative reward function that takes into account rewards on future actions, in a sense trying to simulate the future. Completion times show a form of dependency on the total trace duration. For instance, overestimating the duration of early events will lead to an excessively long overall trace duration estimate. This may guide the learner through states with a better generalization ability. On the contrary, a similar relation does not exist for activities in the considered setting, where only the structural perspective of the process (i.e the workflow) is taken into account. Thus enriching the log with other perspectives and in particular with data regarding case-specific and event-specific properties may likely highlight dependencies among activities and thus lead to improved results. We plan to verify such hypothesis in future work.

We also show in Table A.2 the performance achieved in suffix prediction

TABLE A.2: Comparison of performances for the suffix prediction tasks. (a) Completion time. (b) Next activities.

Window	mean TDE (days)		
size	Ours	Tax	Camargo
2	12.66	≈ 14	≈ 11
10	6.17	≈ 9	≈ 9
20	4.63	≈ 6	≈ 9

(a)

DL-similarity		
Ours	Tax	Camargo
0.174	0.3533	0.525

(b)

tasks. The results confirm a better behavior of the proposed RL architecture on the completion time prediction than on the activity prediction task. For the former, the relative improvement is about 21%, which is in line with the one step ahead performance. For the latter, we observe a much worse performance degradation of about the 66% with respect to [21], and about 50% with respect to [20]. This is in part due to an expected error propagation effect, since errors committed at the early suffix prediction stages progressively compromise all the subsequent ones. As another issue reducing our systems performance, we observed that the event agent struggle to predict the end of the trace, leading to excessively long traces. To verify this, we calculated the DL-similarity truncating the predicted traces to the length of the true traces, discovering that performances improves up to a DL-similarity of 0.2974, which is comparable with the accuracy obtained by [20]. For what concerns computational complexity, clearly the time required to train an RL agent is much higher than the LSTM alone. We experimented an increase factor of about 20x of the required training time. This is a well known characteristics of RL training although alternative techniques with better computational performance have been proposed [82]. We plan to investigate them in future work.

A.2 Consulting Activities

The Ph.D. has been co-funded by Namirial s.p.a. a digital transaction manager enterprise and a digital trusted services provider, specialized in guaranteeing digital identities and all connected services, like digital signatures and electronic billing. In the context of collaboration with the enterprise, various

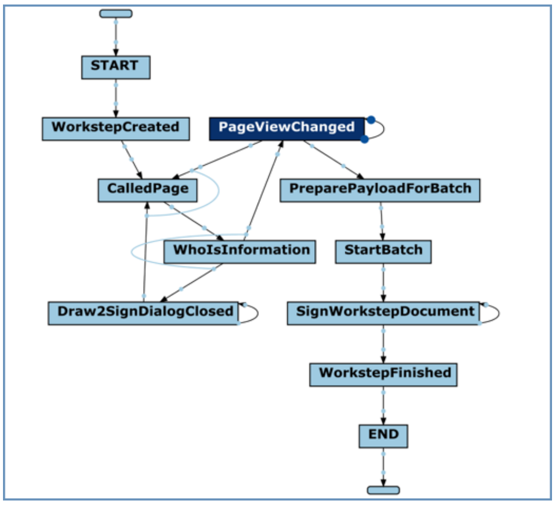


FIGURE A.3: Sign drawing required, process model

consulting activities have been developed. The two most notable are the technical consulting for the detection of counterfeit ID documents and an analysis of their digital signature system performance. It is not possible to precisely describe the former for legal reasons, but the work mostly regarded providing a formalization of the problem as well as technological and technical advice to the R&D head of the enterprise, who then had the proposed solution implemented by his team (also this for legal reasons).

The latter involved a thorough analysis of digital signature logs using data mining and process mining techniques to highlight possible bottlenecks or issues in the flow of work. To do so the CRISP methodology was adopted (CRoss-Industry Standard Process for Data Mining)[83]. The most remarkable result delivered was an analysis showing how the signature process requiring actually drawing the sign discouraged the users, which as a consequence let many days pass before actually signing the document. Figure A.3 shows the process model when the drawing is required, while figure A.4 shows the process model when just a click suffices to sign the document: it can be seen that in the latter, the "Draw2SignDialogClosed" (the signature drawing) event lead to the repetition of the "CalledPage" event which represents the opening of the file and can only be repeated after having firstly closed the document to sign.

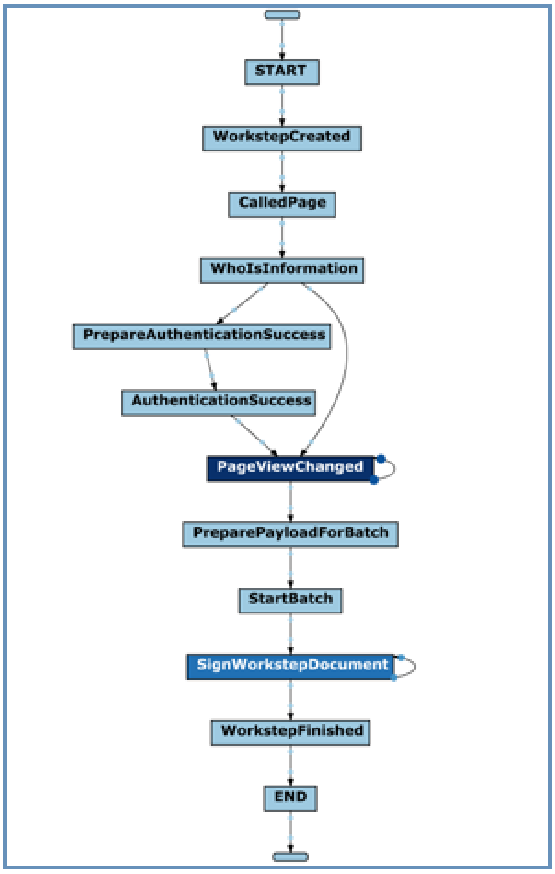


FIGURE A.4: No sign drawing required, process model

Bibliography

[1] W. van der Aalst, A. Adriansyah, A. K. A. de Medeiros, and et al., "Process mining manifesto," in *Business Process Management Workshops*, F. Daniel, K. Barkaoui, and S. Dustdar, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 169–194, ISBN: 978-3-642-28108-2.

[2] F. M. Maggi, C. D. Francescomarino, M. Dumas, and C. Ghidini, "Predictive monitoring of business processes," in *International conference on advanced information systems engineering*, Springer, 2014, pp. 457–472.

[3] A. Appice, N. Di Mauro, and D. Malerba, "Leveraging shallow machine learning to predict business process behavior," in *2019 IEEE International Conference on Services Computing (SCC)*, IEEE, 2019, pp. 184–188.

[4] W. M. van der Aalst, "A practitioner's guide to process mining: Limitations of the directly-follows graph," *Procedia Computer Science*, vol. 164, pp. 321–328, 2019, CENTERIS 2019 - International Conference on ENTERprise Information Systems / ProjMAN 2019 - International Conference on Project MANagement / HCist 2019 - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN/HCist 2019, ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2019.12.189`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1877050919322367`.

[5] D. Grigori, F. Casati, U. Dayal, and M.-C. Shan, "Improving business process quality through exception understanding, prediction, and prevention," 2001, pp. 159–168.

[6] M. Castellanos, N. Salazar, F. Casati, U. Dayal, and M.-C. Shan, "Predictive business operations management," *International Journal of Computational Science and Engineering*, vol. 2, no. 5-6, pp. 292–301, 2006.

[7] W. Van Der Aalst, M. Pesic, and M. Song, "Beyond process mining: From the past to present and future," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6051 LNCS, pp. 38–52, 2010.

[8] C. Di Francescomarino, C. Ghidini, F. M. Maggi, and F. Milani, "Predictive process monitoring methods: Which one suits me best?" In *Business Process Management*, M. Weske, M. Montali, I. Weber, and J. vom Brocke, Eds., Cham: Springer International Publishing, 2018, pp. 462–479.

[9] I. Teinemaa, M. Dumas, M. Rosa, and F. Maggi, "Outcome-oriented predictive process monitoring: Review and benchmark," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 2, 2019.

[10] A. Marquez-Chamorro, M. Resinas, and A. Ruiz-Cortes, "Predictive monitoring of business processes: A survey," *IEEE Transactions on Services Computing*, vol. 11, no. 6, pp. 962–977, 2018.

[11] W. Van Der Aalst, M. Schonenberg, and M. Song, "Time prediction based on process mining," *Information Systems*, vol. 36, no. 2, pp. 450–475, 2011.

[12] A. Senderovich, M. Weidlich, A. Gal, and A. Mandelbaum, "Queue mining for delay prediction in multi-class service processes," *Information Systems*, vol. 53, pp. 278–295, 2015, ISSN: 0306-4379.

[13] J. Becker, D. Breuker, P. Delfmann, and M. Matzner, "Designing and implementing a framework for event-based predictive modelling of business processes," vol. P-234, 2014, pp. 71–84.

[14] G. Lakshmanan, D. Shamsi, Y. Doganata, M. Unuvar, and R. Khalaf, "A markov prediction model for data-driven semi-structured business processes," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 97–126, 2015.

[15] M. Polato, A. Sperduti, A. Burattin, and M. De Leoni, "Time and activity sequence prediction of business process instances," *Computing*, vol. 100, no. 9, pp. 1005–1031, 2018.

[16] M. Unuvar, G. T. Lakshmanan, and Y. N. Doganata, "Leveraging path information to generate predictions for parallel business processes," *Knowledge and Information Systems*, vol. 47, no. 2, pp. 433–461, 2016.

[17] M. Ceci, P. F. Lanotte, F. Fumarola, D. P. Cavallo, and D. Malerba, "Completion time and next activity prediction of processes using sequential pattern mining," in *International Conference on Discovery Science*, Springer, 2014, pp. 49–61.

[18] E. Rama-Maneiro, J. Vidal, and M. Lama, "Deep learning for predictive business process monitoring: Review and benchmark," *IEEE Transactions on Services Computing*, 2021.

[19] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decision Support Systems*, vol. 100, pp. 129 –140, 2017, Smart Business Process Management, ISSN: 0167-9236.

[20] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with lstm neural networks," in *Advanced Information Systems Engineering. CAiSE 2017. Lecture Notes in Computer Science, vol 10253*, 2017, pp. 477–492.

[21] M. Camargo, M. Dumas, and O. González-Rojas, "Learning accurate LSTM models of business processes," in *Proceedings of the 17th International Conference on Business Process Management (BPM'19), Lecture Notes in Computer Science, 11675*, 2019, pp. 286–302.

[22] F. Taymouri, M. L. Rosa, S. Erfani, Z. D. Bozorgi, and I. Verenich, "Predictive business process monitoring via generative adversarial nets: The case of next event prediction," in *Business Process Management*, D. Fahland, C. Ghidini, J. Becker, and M. Dumas, Eds., Cham: Springer International Publishing, 2020, pp. 237–256, ISBN: 978-3-030-58666-9.

[23] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "Predictive process mining meets computer vision," in *"Business Process Management Forum (BPM'20), Lecture Notes in Business Information Processing*, vol. 392, 2020, pp. 176–192.

[24] I. Venugopal, J. Tollich, M. Fairbank, and A. Scherp, “A comparison of deep learning methods for analysing and predicting business processes,” in *Proceedings of International Joint Conference on Neural Networks, IJCNN*, 2021.

[25] A. Chiorrini, C. Diamantini, A. Mircoli, and D. Potena, “A preliminary study on the application of reinforcement learning for predictive process monitoring,” in *Proceedings of 2nd International Conference on Process Mining (ICPM20), Lecture Notes in Business Information Processing*, 2020.

[26] P. Philipp, R. Jacob, S. Robert, and J. Beyerer, “Predictive analysis of business processes using neural networks with attention mechanism,” 2020, pp. 225–230.

[27] C. Di Francescomarino, C. Ghidini, F. M. Maggi, G. Petrucci, and A. Yeshchenko, “An eye into the future: Leveraging a-priori knowledge in predictive business process monitoring,” in *International conference on business process management*, Springer, 2017, pp. 252–268.

[28] B. F. van Dongen and W. M. P. van der Aalst, “Multi-phase process mining: Building instance graphs,” in *Conceptual Modeling – ER 2004*, P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.-W. Ling, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 362–376, ISBN: 978-3-540-30464-7.

[29] W. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters, “Workflow mining: A survey of issues and approaches,” *Data & Knowledge Engineering*, vol. 47, no. 2, pp. 237–267, 2003, ISSN: 0169-023X.

[30] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[31] L. Y. Radu Mateescu Gwen Salaün, “Quantifying the parallelism in BPMN processes using model checking.,” in *The 17th International ACM Sigsoft Symposium on Component-Based Software Engineering (CBSE 2014)*, 2014.

[32] C. Mao, “Control flow complexity metrics for Petri Net-based web service composition,” *Journal of Software*, vol. 5, pp. 1292–1299, Nov. 2010.

[33] K. B. Lassen and W. M. van der Aalst, "Complexity metrics for workflow nets," *Information and Software Technology*, vol. 51, no. 3, pp. 610–626, 2009, ISSN: 0950-5849.

[34] T. H. Davenport, *Process innovation: reengineering work through information technology*. Harvard Business Press, 1993.

[35] Y. Sun and J. Su., "Computing degree of parallelism for BPMN processes.," in *Proceedings of ICSOC'11*, Springer, Ed., 2011, 1–15.

[36] J. Esparza, "Reachability in live and safe free-choice Petri Nets is NP-complete," *Theoretical Computer Science*, 1998.

[37] E. Mayr., "An algorithm for the general Petri Net reachability problem.," *SIAM Journal on Computing*, 1984.

[38] F. Durán, C. Rocha, and G. Salaün, "Computing the parallelism degree of timed bpmn processes," in *Software Technologies: Applications and Foundations*, Cham: Springer International Publishing, 2018, pp. 320–335, ISBN: 978-3-030-04771-9.

[39] W. van der Aalst and et al., "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[40] S. Barbon Junior and et al., "Evaluating trace encoding methods in process mining," in *From Data to Models and Back: 9th International Symposium, DataMod 2020, Virtual Event, October 20, 2020, Revised Selected Papers*, Berlin, Heidelberg: Springer-Verlag, 2020, 174–189, ISBN: 978-3-030-70649-4.

[41] R. Dijkman and et al., "Similarity of business process models: Metrics and evaluation," *Information Systems*, vol. 36, no. 2, pp. 498–516, 2011, Special Issue: Semantic Integration of Data, Multimedia, and Services, ISSN: 0306-4379.

[42] R. P. J. C. Bose and et al., "Context aware trace clustering: Towards improving process mining results," in *Proceedings of the 2009 SIAM International Conference on Data Mining (SDM)*, pp. 401–412.

[43] A. Leontjeva and et al., "Complex symbolic sequence encodings for predictive monitoring of business processes," in *Business Process Management*, Cham: Springer International Publishing, 2015, pp. 297–313, ISBN: 978-3-319-23063-4.

[44] A. Chiorrini and et al., "Exploiting instance graphs and graph neural networks for next activity prediction," in *Process Mining Workshops*, J. Munoz-Gama and et al., Eds., Cham: Springer International Publishing, 2022, pp. 115–126, ISBN: 978-3-030-98581-3.

[45] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "A multi-view deep learning approach for predictive business process monitoring," *IEEE Transactions on Services Computing*, 2021.

[46] P. De Koninck and et al., "Act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes," in *Business Process Management*, Cham: Springer International Publishing, 2018, pp. 305–321, ISBN: 978-3-319-98648-7.

[47] P. Hake and et al., "Supporting business process modeling using rnns for label classification," in *22nd International Conference on Applications of Natural Language to Information Systems, Proceedings*, F. Frasincar and et al., Eds., ser. Lecture Notes in Computer Science, vol. 10260, Springer, 2017, pp. 283–286.

[48] I. Verenich, *Helpdesk*, 2016. DOI: `10.17632/39bp3vv62t.1`.

[49] B. van Dongen, *BPI Challenge 2012*, Apr. 2012. DOI: `10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`. [Online]. Available: `https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204`.

[50] B. van Dongen, *Bpi challenge 2020*, 2020. DOI: `10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51`.

[51] C. Diamantini, L. Genga, D. Potena, and W. van der Aalst, "Building instance graphs for highly variable processes," *Expert Systems with Applications*, vol. 59, pp. 101–118, 2016, ISSN: 0957-4174.

[52] A. Chiorrini, C. Diamantini, A. Mircoli, and D. Potena, "Exploiting instance graphs and graph neural networks for next activity prediction," in *Process Mining Workshops, Lecture Notes in Business Information Processing*, vol. 433, 2021.

[53] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst, "Conformance checking using cost-based fitness analysis," in *2011 ieee 15th international enterprise distributed object computing conference*, IEEE, 2011, pp. 55–64.

[54] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, 1929–1958, Jan. 2014, ISSN: 1532-4435.

[55] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[56] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from incomplete event logs," in *Application and Theory of Petri Nets and Concurrency*, G. Ciardo and E. Kindler, Eds., Cham: Springer International Publishing, 2014, pp. 91–110.

[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, 2015.

[58] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[59] M. Abadi, A. Agarwal, I. Sutskever, and e. a. Oriol Vinyals, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[60] P. B. Brazdil and C. Soares, "A comparison of ranking methods for classification algorithm selection," in *Machine Learning: ECML 2000*, R. López de Mántaras and E. Plaza, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 63–75, ISBN: 978-3-540-45164-8.

[61] A. Chiorrini, C. Diamantini, A. Mircoli, and D. Potena, "Metrics of parallel complexity of operational business processes," in *Proceedings of the 24th International Conference on Enterprise Information Systems - Volume 1: ICEIS,*, INSTICC, SciTePress, 2022, pp. 561–566, ISBN: 978-989-758-569-2. DOI: `10.5220/0011084200003179`.

[62] W. van der Aalst and K. van Hee, "Business process redesign: A petri-net-based approach," *Computers in Industry*, vol. 29, no. 1, pp. 15–26, 1996, ISSN: 0166-3615.

[63] W. van der Aalst, "Business process modeling notation," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds., New York, NY: Springer New York, 2018, pp. 382–383, ISBN: 978-1-4614-8265-9.

[64] W. Aalst, K. Hee, A. Ter, *et al.*, "Soundness of workflow nets: Classification, decidability, and analysis," *Formal Asp. Comput.*, vol. 23, pp. 333–363, May 2011. DOI: `10.1007/s00165-010-0161-4`.

[65] A. Chiorrini, C. Diamantini, L. Genga, M. Pioli, and D. Potena, "Embedding process structure in activities for process mapping and comparison (under publication)," in *New Trends in Database and Information Systems - ADBIS 2022 Short Papers*, ser. Communications in Computer and Information Science, 2022.

[66] A. Chiorrini, C. Diamantini, L. Genga, M. Pioli, and D. Potena, "Towards next-location prediction for process executions," in *2022 4th International Conference on Process Mining (ICPM)*, 2022, pp. 40–47. DOI: `10.1109/ICPM57379.2022.9980785`.

[67] W. Van Der Aalst, *Process mining: data science in action*. Springer, 2016, vol. 2.

[68] B. van Dongen, "BPI Challenge 2017," Feb. 2017. [Online]. Available: `https://data.4tu.nl/articles/dataset/BPI\_Challenge\_2017/12696884`.

[69] A. Polyvyanyy and et al., "Simplified computation and generalization of the refined process structure tree," in *International Workshop on Web Services and Formal Methods*, Springer, 2010, pp. 25–41.

[70] S. J. J. Leemans and et al., "Discovering block-structured process models from event logs - a constructive approach," in *Application and Theory of Petri Nets and Concurrency*, J.-M. Colom and et al., Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 311–329, ISBN: 978-3-642-38697-8.

[71] W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and parallel databases*, vol. 14, no. 1, pp. 5–51, 2003.

[72] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[73] S. J. Leemans, D. Fahland, and W. M. Van Der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *International conference on business process management*, Springer, 2013, pp. 66–78.

[74] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, ISSN: 1476-4687.

[75] D. Silver, A. Huang, and et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, ISSN: 1476-4687.

[76] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[77] O. Vinyals, I. Babuschkin, and et al., "Grandmaster level in starcraft "ii" using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019, ISSN: 1476-4687.

[78] Z. Huang, W. van der Aalst, X. Lu, and H. Duan, "Reinforcement learning based resource allocation in business process management," *Data & Knowledge Engineering*, vol. 70, no. 1, pp. 127 –145, 2011, ISSN: 0169-023X.

[79] S. Branchi, C. Di Francescomarino, C. Ghidini, D. Massimo, F. Ricci, and M. Ronzani, "Learning to act: A reinforcement learning approach to recommend the best next activities," in *Business Process Management Forum*, C. Di Ciccio, R. Dijkman, A. del Río Ortega, and S. Rinderle-Ma, Eds., Cham: Springer International Publishing, 2022, pp. 137–154, ISBN: 978-3-031-16171-1.

[80] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018.

[81] M. Plappert, *Keras-rl*, `https://github.com/keras-rl/keras-rl`, 2016.

[82] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement Learning. Adaptation, Learning, and Optimization*, vol. 12, Springer, Berlin, Heidelberg, 2012.

[83] P. Chapman, J. Clinton, R. Kerber, *et al.*, "Crisp-dm 1.0 step-by-step data mining guide," in Jan. 1999.