Multi-perspective enriched instance graphs for next activity prediction through graph neural network

Journal of Intelligent Information Systems (2023) 61:5–25
https://doi.org/10.1007/s10844-023-00777-1

Multi-perspective enriched instance graphs for next activity prediction through graph neural network

Andrea Chiorrini[1] · Claudia Diamantini[1] · Laura Genga[2] · Domenico Potena[1]

Received: 8 April 2022 / Revised: 27 September 2022 / Accepted: 19 January 2023 /
Published online: 1 May 2023

Abstract
Today's organizations store lots of data tracking the execution of their business processes. These data often contain valuable information that can be used to predict the evolution of running process executions. The present paper investigates the combined use of Instance Graphs and Deep Graph Convolutional Neural Networks to predict which activity will be performed next given a partial process execution. In addition to the exploitation of graph structures to encode the control-flow information, we investigate how to couple it with additional data perspectives. Experiments show the feasibility of the proposed approach, whose outcomes are consistently placed in the top ranking then compared to those obtained by well-known state-of-the-art approaches.

Keywords Process mining · Predictive process monitoring · Next activity prediction · Graph neural network · Instance graph

1 Introduction

Predictive process monitoring (PPM) is an emerging field of process mining whose aim is to predict how a running execution of a process will unfold up until its completion (Maggi et al., 2014). PPM approaches can be used to predict different information on the process,

✉ Andrea Chiorrini
a.chiorrini@pm.univpm.it

Claudia Diamantini
c.diamantini@univpm.it

Laura Genga
l.genga@tue.nl

Domenico Potena
d.potena@unvipm.it

[1] Department of Information Engineering, Polytechnic University of Marche, Via Brecce Bianche 12, Ancona, 60131, Marche, Italy

[2] Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, Groene Loper 3, Eindhoven, 5612 AE, The Netherlands

such as the remaining completion time or the probability of violating a set of constraints. In this work, we focus on the *next-activity* prediction task. Given the current state of execution of a process, the goal consists in predicting which activity will be executed next. Being able to "look ahead" during a process execution can support the managers in determining, for example, the best allocation of resources, or whether to intervene to prevent undesired process outcomes (Appice et al., 2019). A recent trend emerging from the literature consists in the use of deep learning architectures, which outperformed traditional machine-learning and model-based approaches in several studies. Most of the previous work investigated the use of architectures originally developed within the natural language processing field (e.g., LSTM), thus exploiting the sequential nature of traces in the event log. Some approaches also explored the use of architectures commonly used for image classification (e.g., CNN), proposing to use different trace properties to build a multi-dimensional representation of a log resembling the data structure of images. To the best of our knowledge, however, little attention has been paid to exploiting *structural* properties of a process execution when generating a prediction. Process executions are characterized by (complex) control-flow constructs, like concurrency, choices, and loops. 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. This can make it challenging for a sequential-based classifier to learn possible relations between high-level control-flow constructs and the classification target, thus affecting its performance (Evermann et al., 2017a; Metzger & Neubauer, 2018).

Another relevant trend emerging from the literature consists in combining the control-flow information with additional data stored in the event log, thus adopting a so-called *multi-perspective* view on process executions, which has shown to boost classification performance significantly in previous studies (Pasquadibisceglie et al., 2021; Camargo et al., 2019). Indeed, in real-life processes what to do "next" may depend not only on which activities have been executed before but also on, for instance, which resources have been involved, or on the kind of customers, and so on.

Elaborating upon these considerations, in this work we propose an approach able to leverage the structure of the process executions for the prediction, at the same time coupling it with additional data perspectives. Our approach exploits knowledge from a process model, from which instance specific models are derived in the form of instance graphs, coupled with Deep Graph Convolutional Neural Networks (DGCNN) to deal natively with the prediction of the next activity from graph prefixes. The instance graphs are then *enriched* with additional data derived from the event log. We focus on the use of temporal features to characterize at which point of the execution an activity occurs. In this respect, a main difference with respect to previous work is that the process structure is taken into account when computing the time interval between an activity and its predecessor. We argue that this is a more accurate way of computing the time intervals than considering the interval between subsequent events in a trace. Hence, we expect that this representation can better support the classifier in detecting temporal relations among activities.

Summarizing, the main contributions of the present work are:

- We introduce a novel predictive process monitoring approach based on the use of graph data structures to encode control-flow information explicitly representing parallel constructs, coupled with the use of a deep learning architecture tailored to work with graphs;

- We introduce an approach to enrich an instance graph to provide a *multi-perspective* representation of a process execution. In particular, we discuss how to use structural information to encode the time perspective of the executed activities;
- We conduct a set of experiments on real-life event logs to show the effectiveness of the approach with respect to a set of state-of-the-art competitors.

A preliminary version of the approach presented in this paper has been described in Chiorrini et al. (2021). With respect to it, the present work considerably extends the theoretical framework, improves and extends the experimental comparison, and introduces a multi-perspective enrichment to the Instance Graphs.

The rest of this paper is organized as follows. Section 2 provides an overview of related work; Section 3 formalizes important concepts used thorough the paper; Section 4 illustrates the proposed approach; Section 5 discusses the experimental results; Section 6 draws some conclusions and delineates future work.

2 Related work

Predictive process monitoring made its appearance as a process mining task in the first decade of the 2000s (Castellanos et al., 2006; Van Der Aalst et al., 2010), receiving increasing attention in the latest years (Di Francescomarino et al., 2018; Teinemaa et al., 2019; Marquez-Chamorro et al., 2018). Three kinds of predictions can be considered (Di Francescomarino et al., 2018; Teinemaa et al., 2019): prediction of (typically continuous) measures of interest like the remaining execution time, overall duration, or cost of an ongoing case (Van Der Aalst et al., 2010, 2011), prediction of categorical values like the final outcome or class of risk of a case (Teinemaa et al., 2019; Becker et al., 2014), predictions related to the sequence of next activities that will be performed (Lakshmanan et al., 2015). In Polato et al. (2018), 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, whereas model-agnostic approaches only consider traces contained in the event log. 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, 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. Lakshmanan et al. (2015) 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 Unuvar et al. (2016), which proposes different encoding models to represent the parallel branch each activity belongs to, derived from the overall process model using token-replay principles. Polato et al. (2018) 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 Ceci et al. (2014), which employs sequential pattern mining techniques to derive partial process models that are then used to train classification or regression models. Becker et al. (2014) 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 (see Rama-Maneiro et al. (2021) for a recent survey). 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 (Evermann et al., 2017b; Tax et al., 2017; Camargo et al., 2019). LSTM trained with a Generative Adversarial Nets learning scheme has also been proposed (Taymouri et al., 2020), tackling the lack of sufficient training data that often impact performances. An alternative approach is that of Pasquadibisceglie et al. (2020) 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 Venugopal et al. (2021) experiments demonstrate that they can achieve good performance on some datasets. Other approaches like reinforcement learning or transformers have also been experimented (Chiorrini et al., 2020; Philipp et al., 2020). 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, Di Francescomarino et al. (2017) 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. A different strategy to take the process structure into account within the next-activity classification task consists in using graphs, which provide a convenient means to represent processes (van Dongen & van der Aalst, 2004; van der Aalst et al., 2003). A proposal to directly process graphs to predict the next activity has been done by Venugopal et al. (2021). This approach has some similarities with the present proposal. 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 Venugopal et al. (2021), the present paper adopts a different, instance-specific, graph model in the form of Instance Graph, managing also non-fitting traces. Furthermore, in Venugopal et al. (2021) the network architecture is composed of a single graph convolutional layer followed by two fully connected layers, while in the present paper a variation of the Deep Graph Convolutional Neural Network (DGCNN) of Zhang et al. (2018) 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 Venugopal et al. (2021), whereas the Instance Graphs adopted in this work can present the same activity more than once.

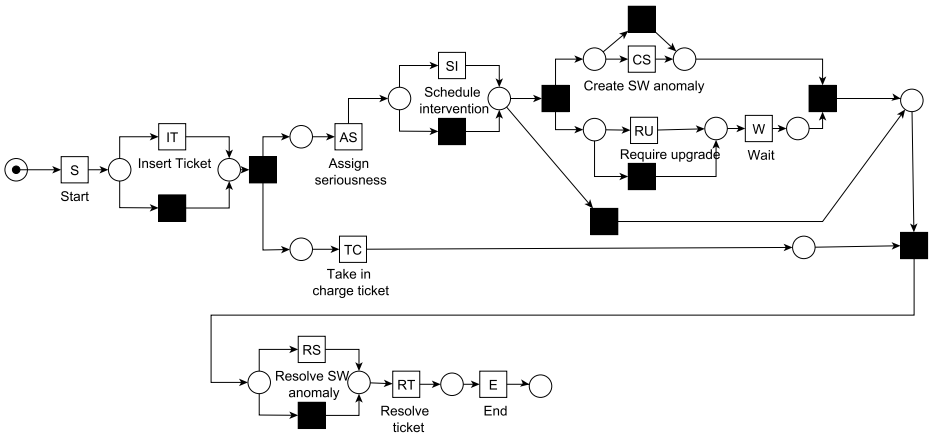


Fig. 1 Petri net mined with the Inductive Miner from the Helpdesk event log. Transition labels are displayed below each transition, while inside each square is the corresponding acronym

3 Preliminaries

In this section, we introduce some core definitions used throughout the paper.

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 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.[1]

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 $e = (a, D, c, i, t) \in A_L \times (V \not\rightarrow U) \times C \times \mathbb{J} \times H$ 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 $i \in \mathbb{N}$. A case corresponds to a single process execution; the number i identifies the position of the event within the sequence of events that occurred within a case. 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.

[1] https://data.mendeley.com/datasets/39bp3vv62t/1

Table 1 shows an excerpt of the event log for the Helpdesk process mentioned above. Through this paper, 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 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 1. The prefix of length 3 of σ_1 is p_3=⟨(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)⟩. 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* (Diamantini et al., 2016). 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

Table 1 Excerpt from the HelpDesk event log

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
...	...	...

labelled transitions, where the first one generates one of the input tokens for the second one, disregarding possible (hidden) transitions occurring in between.

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 graph prefix $p_k(g_j)$ is a subgraph of g_j involving only k nodes of g_j, i.e., nodes included in the corresponding trace prefix.

Example 1 Consider σ_1 and the set CR derived from the Petri net in Fig. 1. Figure 2a shows the IG corresponding to the trace, while Fig. 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 paper.

4 Methodology

This work introduces 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 3 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) 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,

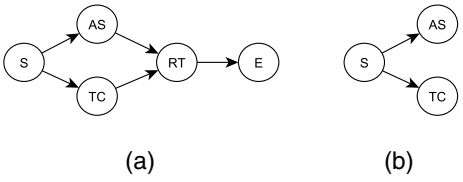


Fig. 2 IG for σ_1 (a) and its prefix of length 3 (b)

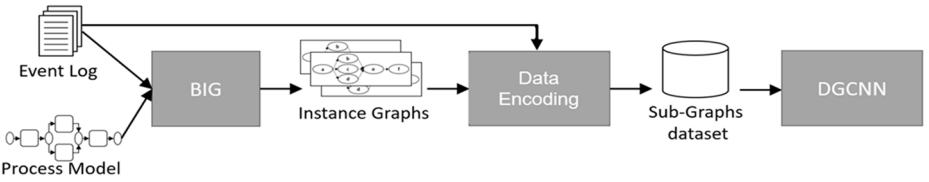


Fig. 3 The BIG-DGCNN methodology pipeline

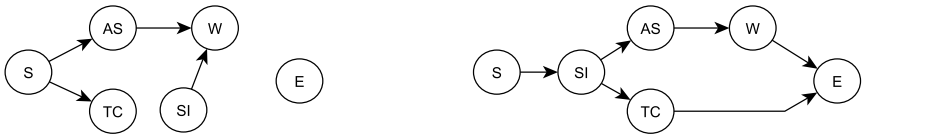
even in the presence of anomalous behaviors the approach returns instance graphs without structural anomalies and that provide a high-quality model for the corresponding process behaviors. The set of instances graphs is then used to train the graph neural network. For the classifier, among the various architectures proposed in the literature, we chose to adopt the Deep Graph Convolutional Neural Network (DGNN) (Zhang et al., 2018). In the following, we will refer to our methodology as Multi-BIG-DGCNN. The following subsections delve into each step of the approach.

4.1 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 paper we refer to the Building Instance Graph (BIG) algorithm proposed in Diamantini et al. (2016), 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 (see Section 2). In the presence of noncompliant events, however, this procedure generates anomalous, low-quality IGs. As an example, let us consider the following trace[2] $\sigma_2 = \langle(1, S), (2, SI), (3, AS), (4, TC), (5, W), (6, E)\rangle$. This trace is not compliant with respect to the model in Fig. 1 since the activity SI is executed before AS and the activity RT is missing. Figure 4a 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 Diamantini et al. (2016) 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,

[2]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

Fig. 4 IG repair examples

hence, IGs) are recognized in the event log by means of a conformance checking technique (Adriansyah et al., 2011). 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 4b 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 event log. 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.[3] It is worth noting that the repaired graphs do not fulfil Definition 5 with respect to the original casual 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.

4.2 Data encoding

This step aims to build 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.

[3]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 Diamantini et al. (2016) for additional details.

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

The building of the prefix-IG set is realized 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 Section 3, whose IG g is reported in Fig. 4b. Table 2 shows two prefix-IGs extracted by g, of length 2 and length 3, respectively.

4.2.2 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 $S' = \{(p'_i(g_j), a_l)\}$, where $p'_i(g_j) = (E_p, W_p, Val\_M)$ is a *Multi-Perspective Enriched* prefix-IG of lenght i of a graph $g_j = (E_j, W_j), i \in [2, \|E_j\| - 1]$.

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 (Tax et al., 2017; Pasquadibisceglie et al., 2020). However, thanks to the use of instance graphs in place

Table 2 Prefix-IG of lenght 2 and 3 extracted from the IG in Fig. 4b

	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

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 consecutive 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:

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

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 3 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 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 to the graph.

Table 3 Enriched prefix-IG of lenght 2 and 3 extracted from g

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

4.3 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 Zhang et al. (2018). The DGCNN is composed of three sequential stages. First it has several graph convolutional layers which extract the features from the nodes local substructure and 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:

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

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 to 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 to 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 (1) are stacked as follows:

$$Z^{k+1} = f\left(\tilde{D}^{-1}\tilde{A}Z^kW^k\right) \tag{2}$$

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. In the present paper we simplify the architecture leaving only one 1-D convolution layer with dropout (Srivastava et al., 2014) 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 (Wu et al., 2021). For further information we refer the interested reader to Zhang et al. (2018).

5 Experiments

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.

5.1 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 paper on the basis of citations and place of publication. On the basis of these criteria we selected:

- Venugopal et al. (2021) for MLP,
- Venugopal et al. (2021) for GCNN (specifically the Laplacian binary),
- Pasquadibisceglie et al. (2020) for the CNN,
- Tax et al. (2017) for the RNN (LSTM to be specific).

We highlight that for Tax et al. (2017) we had to reimplement the code since it was too outdated w.r.t. the used python modules. Also, for the GCNN proposed in Venugopal et al. (2021), 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 (Zhang et al., 2018). 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.

5.1.1 Dataset

For our experiments, we selected some of the benchmark datasets commonly used in literature, whose characteristics are reported in Table 4.

The *Helpdesk* dataset (Verenich, 2016) contains traces from a ticketing management process of the help desk of an Italian software company.

The *BPI12* dataset (van Dongen, 2012) 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.

Table 4 Overview of benchmark dataset

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
Prepaid	2099	18246	29	1	21	9

$|\sigma|$ represents the trace length

The *BPI20* dataset (van Dongen, 2020) 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* (PrePaid) 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.

We tested all the methods using the same 67%-33% train-test split (of chronologically ordered traces) for every dataset.

5.1.2 Parameter settings

The presented methodology involves two algorithms requiring the setting of parameters: the infrequent Inductive Miner (iIM) (Leemans et al., 2014), 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[4]). 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 (Kingma & Ba, 2015) as optimization algorithm and trained the network for 100 epochs with an early stopping. We used as loss function the categorical cross entropy, a fixed batch size of 64 and a fixed dropout percentage of 0.1. For all datasets, we 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 $\{10^{-2}, 10^{-3}, 10^{-4}\}$

[4]Here we refer to the state-of-the-art notion of fitness proposed by Adriansyah et al. (2011)

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 PrePaid. 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 (Fey & Lenssen, 2019) with torch version 1.10.0 or tensorflow 2.5 (Abadi et al., 2015), on an NVIDIA GeForce GTX 1080 GPU, a Intel(R) Core(TM) i7-8700K CPU@3.70GHz, and a 32 GB RAM.

5.1.3 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 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) (Brazdil & Soares, 2000). 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.

5.2 Results

Table 5 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

Table 5 Comparison results; measured accuracy and F score

Approach		Dataset						
		Helpdesk	BPI12W	BPI12	RfP	TP	ID	PrePaid
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%

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 Prepaid, 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 6 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

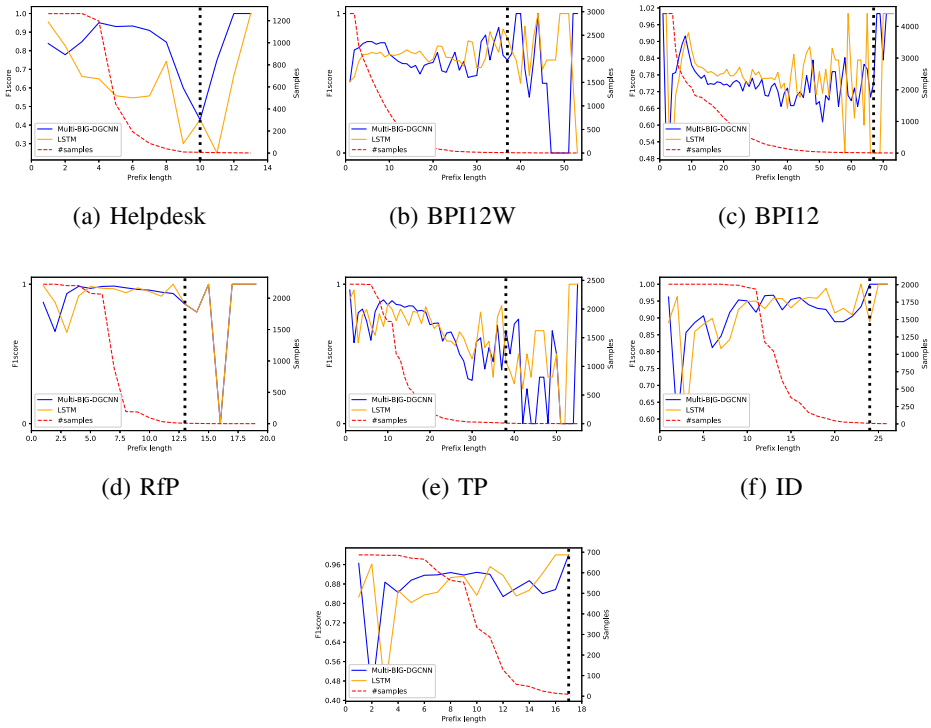
Table 6 Literature comparison, rankings

Approach	Accuracy			F1		
	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

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, 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 5 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 4.2.2). Therefore, we can reasonably assume that differences in performance



(a) Helpdesk (b) BPI12W (c) BPI12

(d) RfP (e) TP (f) ID

(g) Prepaid

Fig. 5 F1 score of Multi-BIG-DGCNN and LSTM on the tested datasets plotted against the prefix lengths, together with the number of samples

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 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 Fig. 5 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 5.

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 need 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, this nodes are also the least informative ones for the longer, less frequent prefixes, with the result of a drop in performances.

6 Conclusions

The paper has presented BIG-DGCNN, 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 and resource information when available, it demonstrates to favourably compare to other approaches. Further analysis of the performance trend with respect to the

prefix length enlighten an interesting difference with respect to the LSTM architecture, that is the decay of performance on longer prefixes. This suggests to investigate further improvements of the approach, namely to train different networks for the different prefixes, to implement a GAN learning scheme in order to deal with the limited number of training samples for longer prefixes, or to adopt resampling or other imbalanced learning techniques.

Author Contributions Conceptualization: all authors; Methodology: all authors; Formal analysis and investigation: Andrea Chiorrini; Writing - original draft preparation: Andrea Chiorrini, Claudia Diamantini, Laurea Genga; Writing - review, editing and final approval: all authors; Supervision: Claudia Diamantini.

Funding No funding was received for conducting this study.

Data Availability All datasets used in this paper to support the findings are publicly available. Links are reported in the bibliography.

Declarations

Consent for Publication All authors agree with the content and give explicit consent to submit.

Conflict of Interests The authors have no financial or proprietary interests in any material discussed in this article.

References

Abadi, M., Agarwal, A., Sutskever, I., & et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, software available from tensorflow.org. Accessed 15 Sept 2022.

Adriansyah, A., van Dongen, B. F., & van der Aalst, W.M. (2011). Conformance checking using cost-based fitness analysis. In *2011 IEEE 15th international enterprise distributed object computing conference* (pp. 55–64). IEEE.

Appice, A., Di Mauro, N., & Malerba, D. (2019). Leveraging shallow machine learning to predict business process behavior. In *2019 IEEE international conference on services computing (SCC)* (pp. 184–188). IEEE.

Becker, J., Breuker, D., Delfmann, P., & et al. (2014). Designing and implementing a framework for event-based predictive modelling of business processes. pp 71–84.

Brazdil, P. B., & Soares, C. (2000). A comparison of ranking methods for classification algorithm selection. In R. López de Mántaras, & E. Plaza (Eds.) *Machine Learning: ECML 2000* (pp. 63–75). Berlin: Springer.

Camargo, M., Dumas, M., & González-Rojas, O. (2019). 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*, (Vol. 11675 pp. 286–302).

Castellanos, M., Salazar, N., Casati, F., & et al. (2006). Predictive business operations management. *International Journal of Computational Science and Engineering*, *2*(5-6), 292–301.

Ceci, M., Lanotte, P. F., Fumarola, F., & et al. (2014). Completion time and next activity prediction of processes using sequential pattern mining. In *International conference on discovery science* (pp. 49–61). Springer.

Chiorrini, A., Diamantini, C., Mircoli, A., & et al. (2020). 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*.

Chiorrini, A., Diamantini, C., Mircoli, A., & et al. (2021). Exploiting instance graphs and graph neural networks for next activity prediction. In *Process mining workshops, Lecture Notes in Business Information Processing*.

Di Francescomarino, C., Ghidini, C., Maggi, F. M., & et al. (2017). An eye into the future: leveraging a-priori knowledge in predictive business process monitoring. In *International conference on business process management* (pp. 252–268). Springer.

Di Francescomarino, C., Ghidini, C., Maggi, F. M., & et al. (2018). Predictive process monitoring methods: Which one suits me best? In M. Weske, M. Montali, I. Weber, & et al. (Eds.) *Business Process Management* (pp. 462–479). Cham: Springer International Publishing.

Diamantini, C., Genga, L., Potena, D., & et al. (2016). Building instance graphs for highly variable processes. *Expert Systems with Applications*, *59*, 101–118.

van Dongen, B. (2012). BPI Challenge 2012. https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f. https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204.

van Dongen, B. (2020). BPI challenge 2020. https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d613b51.

van Dongen, B. F., & van der Aalst, W. M. P. (2004). Multi-phase process mining: Building instance graphs. In P. Atzeni, W. Chu, H. Lu, & et al. (Eds.) *Conceptual Modeling – ER 2004* (pp. 362–376). Berlin: Springer.

Evermann, J., Rehse, J. R., & Fettke, P. (2017a). Predicting process behaviour using deep learning. *Decision Support Systems*, *100*, 129–140.

Evermann, J., Rehse, J. R., & Fettke, P. (2017b). Predicting process behaviour using deep learning. *Decision Support Systems*, *100*, 129–140. Smart Business Process Management.

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR workshop on representation learning on graphs and manifolds*.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd international conference on learning representations (ICLR 2015)*.

Lakshmanan, G., Shamsi, D., Doganata, Y., & et al. (2015). A Markov prediction model for data-driven semi-structured business processes. *Knowledge and Information Systems*, *42*(1), 97–126.

Leemans, S. J. J., Fahland, D., & van der Aalst, W.M.P. (2014). Discovering block-structured process models from incomplete event logs. In G. Ciardo, & E. Kindler (Eds.) *Application and theory of Petri nets and concurrency* (pp. 91–110). Cham: Springer International Publishing.

Maggi, F. M., Francescomarino, C. D., Dumas, M., & et al. (2014). Predictive monitoring of business processes. In *International conference on advanced information systems engineering* (pp. 457–472). Springer.

Marquez-Chamorro, A., Resinas, M., & Ruiz-Cortes, A. (2018). Predictive monitoring of business processes: A survey. *IEEE Transactions on Services Computing*, *11*(6), 962–977.

Metzger, A., & Neubauer, A. (2018). Considering non-sequential control flows for process prediction with recurrent neural networks. In *2018 44th Euromicro conference on software engineering and advanced applications (SEAA)* (pp. 268–272). IEEE.

Pasquadibisceglie, V., Appice, A., Castellano, G., & et al. (2020). Predictive process mining meets computer vision. In *Business process management forum (BPM'20), Lecture Notes in Business Information Processing* (pp. 176–192).

Pasquadibisceglie, V., Appice, A., Castellano, G., & et al. (2021). A multi-view deep learning approach for predictive business process monitoring. *IEEE Transactions on Services Computing*.

Philipp, P., Jacob, R., Robert, S., & et al. (2020). Predictive analysis of business processes using neural networks with attention mechanism. pp 225–230.

Polato, M., Sperduti, A., Burattin, A., & et al. (2018). Time and activity sequence prediction of business process instances. *Computing*, *100*(9), 1005–1031.

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

Srivastava, N., Hinton, G., Krizhevsky, A., & et al. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

Tax, N., Verenich, I., La Rosa, M., & et al. (2017). Predictive business process monitoring with LSTM neural networks. In *Advanced information systems engineering. CAiSE 2017. Lecture Notes in Computer Science* (vol. 10253 pp. 477–492).

Taymouri, F., Rosa, M. L., Erfani, S., & et al. (2020). Predictive business process monitoring via generative adversarial nets: The case of next event prediction. In D. Fahland, C. Ghidini, J. Becker, & et al. (Eds.) *Business Process Management* (pp. 237–256). Cham: Springer International Publishing.

Teinemaa, I., Dumas, M., Rosa, M., & et al. (2019). Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Transactions on Knowledge Discovery from Data, 13*(2).

Unuvar, M., Lakshmanan, G. T., & Doganata, Y.N. (2016). Leveraging path information to generate predictions for parallel business processes. *Knowledge and Information Systems*, *47*(2), 433–461.

van der Aalst, W., van Dongen, B., Herbst, J., & et al. (2003). Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, *47*(2), 237–267.

Van Der Aalst, W., Pesic, M., & Song, M. (2010). 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) 6051 LNCS:38–52.

Van Der Aalst, W., Schonenberg, M., & Song, M. (2011). Time prediction based on process mining. *Information Systems*, *36*(2), 450–475.

Venugopal, I., Tollich, J., Fairbank, M., & et al. (2021). A comparison of deep learning methods for analysing and predicting business processes. In *Proceedings of international joint conference on neural networks, IJCNN*.

Verenich, I. (2016). Helpdesk. https://doi.org/10.17632/39bp3vv62t.1.

Wu, Z., Pan, S., Chen, F., & et al. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24.

Zhang, M., Cui, Z., Neumann, M., & et al. (2018). An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*.