Computing and Informatics, Vol. 43, 2024, 181–212, doi: 10.31577/cai_2024_1_181

PREDICTION OF COMPLEX EVENT GRAPHS WITH NEURAL NETWORKS

László Kovács, Erika Baksáné Varga, Péter Mileff

Institute of Informatics
University of Miskolc
Egyetemváros
3515 Miskolc, Hungary
e-mail: {laszlo.kovacs, erika.b.varga, peter.mileff}@uni-miskolc.hu

Abstract. A key problem domain inside Robotic Process Automation is the automatic discovery of workflow process schemes. Considering current process mining technologies, graph-based approaches dominate the industry. On the other hand, the conventional methods suffer from low time efficiency and varying accuracy. Machine learning-based methods can provide better efficiency, but they have significant limitations considering schema flexibility. The paper presents a novel neural network-based schema induction model for the discovery of event patterns containing parallel and optional sequences of different actors. This model can process more complex event graphs and situations than the conventional methods. The performed analysis and test results show the unique power of this approach in process schema mining.

Keywords: Event graph mining, neural network classifier, process mining, sequence prediction

1 INTRODUCTION

Robotic process automation (RPA) is one of the key emerging technologies to automate routine business processes using software tools. RPA systems are used for increasing efficiency and consistency [1] by replacing human resources with software robots. Based on the investment cost-profit relationship, RPA is usually applied to processes that are frequent enough to discover statistical rules but infrequent or irregular for traditional process automation [2]. One of the key components of RPA

systems is the discovery of possible frequent event sequence scenarios which can also be used in many different problem domains of data mining like computational linguistics or biology [3].

The application of frequent pattern mining algorithms on event logs generates event sequence patterns that are characteristic of the investigated business processes. Traditional pattern mining algorithms, however, suffer greatly from longer running times and low accuracy when applied on large datasets [4]. Regarding the analysis of large or irregular event logs, the application of neural network tools is the leading approach. For the prediction of sequences, the family of recurrent neural networks dominates over other neural network architectures. Based on recent research experiments, the long short-term memory (LSTM) and gated recurrent unit (GRU) recurrent networks provide the best accuracy on sequences [5]. Current works usually focus on the application of special attention modules in the network which can be used to enhance long-term dependencies in the training sequences.

Considering the current event sequence mining algorithms, the output of the methods is a single sequence, containing the items with the highest probability at each prediction step. To uncover more possible alternative sequences in the event graph, the baseline model is extended with a beam-search method. The beam search algorithm selects several good items as the next event instead of the selection of a single best event. The studies in recent years show that beam search techniques usually provide much better accuracy [6]. The beam search method is used in [7] to generate a probability tree that describes the network of alternative events based on a given input (see Figure 1).

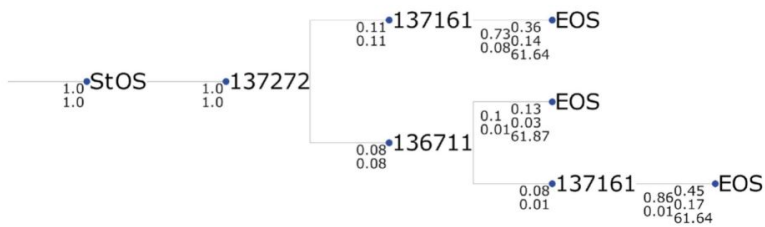


Figure 1. Sample probability event tree [7]

The generated probability event tree can provide more information than conventional event sequences, but real event graphs may have much more complex structures. In many application areas, the event graph pattern schema may contain loops or parallel sequences of different actors. Based on our literature analysis, no neural network-based sequence mining algorithm exists that can cover these complex schema structures.

The main goal of this paper is to present a novel neural network model for the discovery of event patterns containing parallel sequences of different actors. This

model can process more complex event graphs and situations than the methods based on event tree structures. This can be the case when considering a collaboration scenario, where different processes run parallel and these processes are synchronized to start or to finish the specific activities. All the current workflow modeling languages apply different synchronization events including fork, join, or parallel split when a single thread of execution is split into two or more branches. Thus, we argue that efficient event pattern mining algorithms should be able to discover not only sequences and OR-based hierarchies, but more complex structures like parallel sequences. As the performed test results show, the proposed network architecture with the proposed training and prediction methods can cover parallelism in the investigated event graph.

The paper starts with an overview of the related concepts and results in process modeling and process mining in Section 2. Section 3 presents the description and methodology of the proposed neural network model for mining parallel sequences. The technical details of the neural network architecture and that of the training and prediction processes are presented in Section 4. The description of the test framework, including the generation of training data, together with the test results can be found in Section 5. Finally, Section 6 summarizes the key features and the benefits of the proposed method.

2 OVERVIEW OF PROCESS MINING METHODS

2.1 Process Modeling

Organizations have processes that define the way they operate. These workflows are composed of a series of tasks which employees perform regularly to achieve specific goals. Formalizing these processes is the key factor of efficiency and replicability.

As a first approach, conceptual process models are used to visually represent the actions that capture, manipulate, store, and distribute data within business processes. These have in common that processes are described in terms of activities and the ordering of activities is modeled by causal dependencies. It is represented with a graph built up of nodes and directed edges between the nodes indicating the chronological order of the process flow. In addition, a process model may also describe temporal properties, or specify the creation and use of data to model decisions and prescribe the way that resources interact with the process [8].

The Workflow Patterns Initiative [9, 10] conducted a systematic analysis of the constructs used by the existing process modeling notations and workflow languages. As a result, a large collection of patterns was identified. These patterns cover all workflow perspectives, for example, there are control-flow patterns, data patterns, or resource patterns. Our investigations focus on the control-flow perspective [11], where 43 patterns were identified and classified into 8 classes. The basic control flow patterns are similar to those proposed by the Workflow Management Coalition [12]:

- In sequential routing, a task in a process is enabled after the completion of a preceding task in the same process.
- Branch and merge (XOR, or exclusive choice with simple merge) corresponds to deciding on which path to choose from several branches. In conditional routing, when the incoming branch is enabled, the thread of control is immediately passed to exactly one of the outgoing branches. When merging the flow routes, the activated incoming branch should be completed before the thread of control is passed forward.
- Fork and join (AND, or parallel split with synchronization) is applied when a single thread of execution is split into two or more branches which are triggered concurrently, and synchronized at some future time.
- Split and join (OR, or multi-choice with synchronizing merge) corresponds to activating one or more paths among the outgoing control flows. When merging the flow routes, all the activated incoming branches should be completed before the thread of control is passed forward.

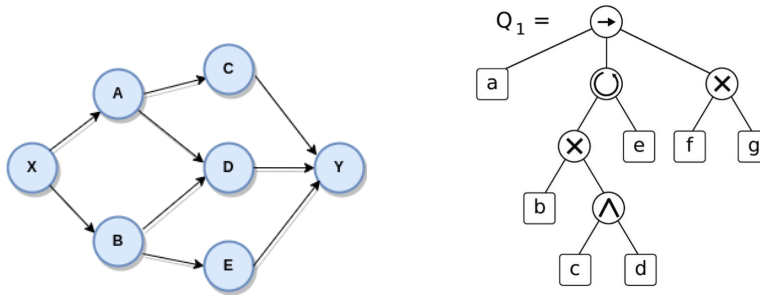
The models mentioned above have special notations for the elements that affect the control flow. These elements interconnect activities in the conceptual process model graph, but there is no sign of them in the event log, which is the storage format of the events which happened in the information system [8]. For this reason, a special representation is needed for process mining algorithms.

One approach is using Causal Nets [13]. In these graphs, nodes are defined by a three-element tuple. The first element is the activity represented by the node, the second element is a list of this node's possible input bindings and the third element is a list of the node's possible output bindings.

Another approach is the use of Process Trees [14], which were designed to solve the deadlock and livelock problems of graph-based process notations. Process Trees represent block-structured models with a hierarchical process notation, where the (inner) nodes are operators, such as sequence and choice, and the leaves are activities. The leaf nodes in a process tree correspond to event nodes, while the non-leaf nodes are operator nodes describing the execution order of the events. The model provides the following operator nodes:

- event sequence ($\rightarrow$),
- parallel execution (AND) ($\wedge$),
- non-exclusive choice (OR) ($\vee$),
- exclusive choice (XOR) ($\times$), and
- event loop ($\circlearrowleft$).

A sample process tree is presented in Figure 2. One limitation of the process tree model is that not every event graph can be converted into an equivalent tree structure.



a) Sample event graph b) Sample process tree

Figure 2. Schema representation formats [15]

2.2 Process Mining

In the information systems, activities that have been executed sequentially, are stored in log files. An event log is basically a structured table where each record corresponds to an event, which refers to an activity and is related to a particular case of a process. The columns include all the data that can be automatically captured during the execution of an activity. The most important attribute of an activity is its timestamp, as this is the basis for ordering the events. The standard format for storing event logs is XES (eXtensible Event Stream) [16]. In XES logs, events are grouped under process instances, called traces. In multi-actor and distributed systems, however, various objects interact in a process and one event may involve a mixture of objects, which means that the event cannot be classified into a single process instance. To tackle this problem, an object-centric event log format (OCEL) was proposed [17].

Event logs can be used to extract knowledge about real processes. In [8], three types of process mining tasks are studied:

1. process discovery, which produces a process model from an event log without using any apriori information;
2. conformance checking, where a process model is compared with an event log; and
3. process enhancement, where the comparison results in changing the original model.

Finding frequent patterns in event sequences is also of interest in many domains. For example, Laxman et al. developed an algorithm for sequence prediction over long categorical event streams [18]. Karoly and Abonyi [19] and Weiss [20] propose methods to extract temporal patterns in industrial alarm management systems. Analyzing study paths for the prediction of student dropout [21] and identifying efficient learning patterns in e-learning environments [22, 23] are key concepts in

educational data mining. In these examples, frequent patterns are selected based on their utility. In [24], the authors argue that also the cost perspective should be taken into account when mining event sequences.

The drawbacks of the traditional frequent pattern mining solutions come to the surface when dealing with massive datasets [25]. The algorithms have problems with running time and accuracy [4], and their output data is difficult to interpret and handle [26]. To efficiently mine frequent patterns, a tree-based representation is proved to be more compact and practically more usable [27]. For this purpose, Lin et al. proposed the FUSP-tree structure [28], which gave rise to numerous variations of incremental tree-based pattern mining algorithms [29, 30, 31, 32].

The incremental approaches in sequential pattern mining derive patterns recursively. Chen [33] introduces a directed acyclic graph representation, which allows for pattern growth along both ends of the detected patterns, thereby producing fewer levels of recursion and faster pattern growth. Patel and Patel [34] combines a graph-based approach with clustering of patterns to avoid the recursive reconstruction of intermediate trees. Singh et al. [35] use a graph-based approach to extract frequent sequential web access patterns, while Dong et al. [36] present a new weighted graph structure and a method to find variable-length sequential patterns.

Finding the structure in a sequence data can also be considered a grammatical inference problem and hence can be modeled by finite state automata. In [37], the navigation patterns of web users are captured, where a navigation session is modeled as a hypertext probabilistic grammar, which is a restricted form of probabilistic regular grammar. Hingston [38] also uses finite state automaton for sequence mining, but in this case, the resulting grammar is stochastic. Jacquemont et al. [39] use a generalized representation of the original sequences in the form of probabilistic automata, and make efforts in controlling the number of false discoveries by integrating statistical constraints. A highly investigated domain recently is the automated inference of formal specifications of software systems, and many researchers have proposed various FSA-based approaches [40, 41, 42].

Deep learning-based techniques are specifically designed for processing massive datasets, and recurrent neural networks (RNNs) were the first to be widely used in sequential data mining. However, traditional RNNs consisting of sigma or tanh cells are not suitable for learning the relevant information of input data when the input gap is large. By introducing gate functions into the cell structure, long short-term memory (LSTM) networks can handle the problem of long-term dependencies [43, 44]. Jamshed et al. [45] created a combination of convolutional neural network (CNN) and long short-term memory (LSTM) to infer customer behavior and purchasing patterns in terms of time. CNN is used to reveal the frequent itemsets, and LSTM is used to identify the time interval among these itemsets. LSTM also proved to outperform other prediction models in price prediction [46]. The sequence-to-sequence learning problem was first addressed by Sutskever et al. [47] when using LSTM to improve machine translation.

The seq2seq method has had several applications since then. Karatzoglou et al. [48] use it to predict future human movement patterns to improve mobile location-

based services. Rebane and Karlsson [49] compare the method to traditional models in cryptocurrency prediction. Baumel et al. [50] apply the method to query-focused summarization. Wu et al. [51] integrate IoT data logs and analyze them with a seq2seq-based method to support the management of IoT systems.

Abonyi et al. [7] expanded the seq2seq algorithm so that its output is a probability tree, that describes the predicted alternative courses of events, instead of a single, most probable sequence. As trees do not allow for modeling complicated structures, like parallelism and conditional forking, our NN-based approach aims to produce event graphs that contain complex control flow elements as well.

3 MINING OF EVENT GRAPHS WITH PARALLEL SEQUENCES

In the process tree modeling approach, parallel execution (AND node) relates to the case when the ordering of the component events is arbitrary, there is no adjacency dependency among the elements. For example, the event sequences

$$(a, b, c) \quad (b, c, a) \quad (a, c, b) \quad (b, a, c) \quad (c, a, b) \quad (c, b, a)$$

can be mapped to this schema:

$$\wedge(a, b, c).$$

In our investigation, we use a slightly different approach. Parallelism denotes parallel workflows of different actors and resources, and the split and join control nodes denote an artifact-level dependency among the different branches. For example, we consider a workflow to produce a mobile phone. In this process, the production of the different components can be executed in a parallel way. A synchronization join node denotes the case when the next assembly step requires the availability of all components produced in the preceding steps. These control nodes can be automatically discovered if the event log contains an artifact attribute, too. The artifact attribute identifies the target, i.e. the object of the given action. Using this parameter we can discover the artifact level dependency between the different actions of the event log.

Another consideration supporting our decision to focus on this kind of parallelism instead of the conventional approach is that our goal is to discover valid execution paths for the automation of the workflow process. In this sense, having a set of equivalent paths, only one valid path is enough to find, it is not necessary to discover all valid execution paths.

Table 1 shows a sample event log related to this approach. Based on the event log, we can discover one join-split synchronization node, where the incoming sequence relates to actor U1, while the outgoing sequences are executed by U2 and U4.

In this paper, the investigation focuses on discovering and predicting complex event-graph structures, including among others parallelism and conditional forking, using neural network tools. The method provides a unique, novel approach, as it

Trace	Actor	Action	Time	Input	Output
1	U1	A1	01.45	O1	O2
1	U1	A3	02.10	O2	O2
1	U2	A5	02.15	O2	O4, O5
1	U3	A8	02.18	–	O3
1	U3	A15	02.18	O3	O3
1	U4	A15	02.31	O2	O8
1	U4	A15	02.45	O8	O8

Table 1. Sample event log with artifact identification

is suitable for managing also complex schema structures, unlike the conventional approaches.

3.1 Discovery of the Synchronization Nodes

In the investigated model, the events of the event logs are identified by the following parameters:

- trace id;
- action (event type);
- event time;
- actor, resource;
- input artifacts;
- output artifacts.

Besides the actor events, the extended input event graph for the training process contains also synchronization control nodes which describe the adjacency relationship among the event sequences. We assume that every control node has an input set of event sequences and an output set of event sequences. Similarly to Petri-nets, the join control node is triggered only when all of the input sequences are finished. If the transition is triggered, all output sequences will start the execution.

In the preprocessing phase of the proposed method, the engine determines the related synchronization nodes first. Node mining is based on the following considerations:

- Event nodes are processed in temporal order, and the current event is denoted by e_i.
- We take the set of output artifacts of e_i and store them in out_i.
- We determine the set of adjacent events $E_j = \{e_j\}$ where the input artifacts correspond to some elements of out_i, and there is no other intermediate event processing these artifacts.

- If the actors of the matching e_i and e_j pair are different actors, a synchronization node e_s is needed between them.
- We add actor of e_i (a_i) to the input-actors of e_s and the output actor set of e_s is extended with a_j.
- If we find an actor not present in the actor set of e_s and being active at e_s with an artifact-level dependency with an event in the output-actor list of e_s, then it will be included in the input-actor set of e_s. A similar method can be used to extend the output-actor list of e_s.
- The previous two steps are repeated until we get a closure of the input-output artifact relationship at e_s.

Having the input event log, the preprocessing module discovers the related join-split synchronization nodes. A synchronization node is given with the following attributes:

- trace id;
- event type;
- event time;
- input actors;
- output actors.

Example 1. We assume that an event vector contains the event ID, the action ID, the actor ID, and the time point when the event starts. Let us take the following event log:

```
EventT(1,1,'A',1)
EventT(2,2,'A',3)
EventT(3,3,'A',5)
EventT(4,4,'B',10)
EventT(5,5,'C',11)
EventT(6,6,'B',15)
EventT(7,7,'C',18)
EventT(8,8,'D',21)
EventT(9,9,'D',31)
```

The related action repository contains the following entries:

```
ActionT('1',{},{'o1'})
ActionT('2',{'o1'},{'o1'})
ActionT('3',{'o1'},{'o2'})
ActionT('4',{'o2'},{'o3'})
ActionT('5',{'o2'},{'o4'})
ActionT('6',{'o3'},{'o5'})
ActionT('7',{'o4'},{'o6'})
```

```
ActionT('8',{'o5','o6'},{'o7'})
ActionT('9',{'o7'},{'o8'})
```

In the description, the first argument is the action ID, the second is the set of incoming artifacts, and the third parameter denotes the set of outgoing artifacts.

The output of the algorithm for synchronization node discovery generates two nodes:

```
{3} {4, 5} : 5 10
{6, 7} {8} : 18 21
```

The first synchronization node is linked to input event 3 and output events 4 and 5. The time window is the interval between 5 and 10. At the second synchronization node, there are two incoming events (6,7) and one outgoing event (8). The corresponding time window (18,21). The corresponding event graph is presented in Figure 3.

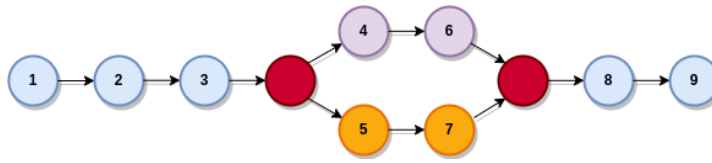


Figure 3. Generated event graph for Example 1

Definition 1 (Event graph structure)**.** An event graph is defined as

$$\sigma = (N_\sigma, \rightarrow_\sigma),$$

where

- $n \in W \times A$ is an event where $A(n)$ denotes the actor of the event and $T(n)$ denotes the timestamp of the event;
- $c \in C$ is a control event, and every c has a timestamp denoted by $T(c)$;
- $E_\sigma = (e_1 e_2 \ldots e_k : e_i \in W \times A, \forall i, j : A(n_i) = A(n_j))$: the event sequence nodes;
- $C(c)$: the control event nodes;
- $N_\sigma = E_\sigma \cup C_\sigma$: the nodes in the graph instance;
- $\rightarrow_E \subseteq E_\sigma \times C_\sigma$: the edges from actor events to control events;
- $\rightarrow_C \subseteq C_\sigma \times E_\sigma$: the edges from control events to actor events;
- $\rightarrow_\sigma = \rightarrow_E \cup \rightarrow_C$: the edges in the graph;
- $x \rightarrow_E y \Rightarrow y \rightarrow_C x$: there is no cycle in the graph.

The control nodes in the graph instances are:

- c^*: synchronization (parallel) node, where all the input sequences should be finished to start all output sequences in parallel;
- c^S: start event;
- c^E: end event.

The graph schema is generated with the aggregation of graph instances containing similar nodes. The only difference between these graph instances is the set of the applied control nodes. In the case of the graph schema we use a branching control node as well:

- c^+: synchronization (optional) node, where all the input sequences should be finished to start one of the output sequences.

Regarding the event graphs, we assume some important integrity rules.

Definition 2 (Event graph consistency)**.** The graph is consistent if there are no such parallel event sequences that belong to the same actor.

This property means that an actor can work only at a single event sequence at the same time, i.e. actors can not work parallel on more sequences.

To cope with the complexity of event graphs, the method uses a two-level approach. At the top level, we focus only on the main structure of the graph using only the control node components. The event sequence nodes are replaced here with a single node denoting only the actor of the actions.

Definition 3 (The reduced graph model)**.** The graph model π-graph is a reduction of the σ graph eliminating the details of the actor level sequences, i.e.

$$\pi = (N_\pi, \rightarrow_\pi)$$

is given by

- $N_\pi = E_R \cup C_\sigma$,
- $E_\pi = \{e\}$ where $A(e) \in A$,
- $\rightarrow_E \subseteq E_\pi \times C_\sigma$,
- $\rightarrow_C \subseteq C_\sigma \times E_\pi$,
- $\rightarrow_\pi = \rightarrow_E \cup \rightarrow_C$.

In the model, we assign two sets for each synchronization node: a set of input sequence actors and a set of output sequence actors. These sets are denoted by $in(c) \subseteq A$ and $out(c) \subseteq A$, respectively.

Working with consistent graphs, for every control event c and for every actor a, the input (as well as the output) sequence list contains a maximum of one occurrence of the sequences related to a.

To manage the complexity of the graph, we approximate it with a sequence, which is the sequence of the control nodes (s). In this sequence, the positions of the

nodes correspond to their time-based ordering. Thus, the first event in the sequence is the event executed first, while the last event is the event executed at the end.

Example 2. Let us take the following σ graph given in Figure 4.

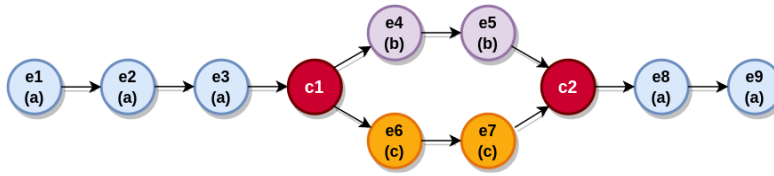


Figure 4. Schema graph for Example 2

The conversion into π graph results in the graph shown in Figure 5.

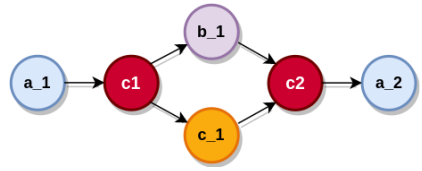


Figure 5. Generated π graph for Example 2

We introduce a mapping between the graph and the sequence form in the following way.

Definition 4 (Conversion algorithms)**.**

- Algorithm $Seq(\pi)$: Having an event graph, every control event c has a timestamp. Based on that timestamp value, we can order the c elements. If two control events have the same timestamp then their order is arbitrary.
- Algorithm $Grp(s_\pi)$: Having a s_π event sequence, we can construct the event graph using the following steps. We take every c event as a node of the output graph. We construct an edge from c_1 to c_2 if

$$\exists a \in A : a \in out(c_1), a \in in(c_2),$$

$$\neg\exists c_3 : a \in in(c_3) \text{ or } a \in out(c_3).$$

As shown in the next example, not every sequence is valid for generating a corresponding graph.

Example 3. Taking the sequence:

$$e_0(g), e_1(a), e_2(b), e_3(a), e_4(b), e_5(a), e_6(a), e_7(g),$$

where the artifact-level relationships among the actions are the following:

- $in(e_0) = \{t_1\}$, $out(e_0) = \{t_2, t_3, t_4\}$,
- $in(e_1) = \{t_2\}$, $out(e_1) = \{t_5\}$,
- $in(e_2) = \{t_3\}$, $out(e_2) = \{t_6\}$,
- $in(e_3) = \{t_4\}$, $out(e_3) = \{t_7\}$,
- $in(e_4) = \{t_6\}$, $out(e_4) = \{t_8\}$,
- $in(e_5) = \{t_5\}$, $out(e_5) = \{t_9\}$,
- $in(e_6) = \{t_7\}$, $out(e_2) = \{t_{10}\}$,
- $in(e_7) = \{t_8, t_9, t_{10}\}$, $out(e_2) = \{t_{11}\}$.

The corresponding graph (Figure 6) contains two parallel sequences assigned to the same actor (a), which means that the graph is inconsistent.

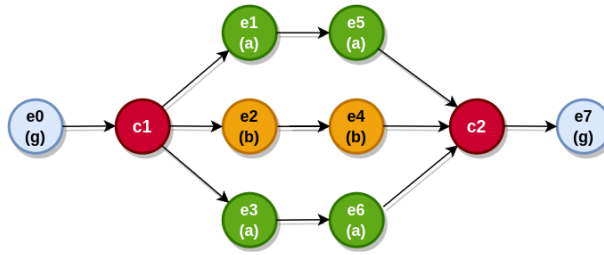


Figure 6. Generated event schema graph for Example 3

To control the validity of the sequences for a given event graph, we use the following validation model. We introduce a set X containing the actors being active at the current time point. Initially, X is empty. Taking the next event from the s_π event sequence, we perform the following operations (where c denotes the current event):

$$X' = (X \setminus in(c)) \cup out(c).$$

Theorem 1. For every s_π sequence, where the following properties are met for every c event:

$$in(c) \subseteq X,$$
$$(out(c) \setminus in(c)) \cap X = \emptyset.$$

$Grp(s_\pi)$ yields a consistent event graph. In the formula, X denotes the set of active actors at the given time point.

Proof. We use mathematical induction to show the validity of Theorem 1. Initially, s_π contains a single control event c^S, while $in(c^S)$ and X are both empty sets, thus

the conditions

$$in(c) \subseteq X,$$
$$(out(c) \setminus in(c)) \cap X = \oslash$$

are met, and the generated section which contains the sequences in $out(c^S)$ is consistent. Assuming that the statement is valid for the first $i-1$ control events, let us take the next i^{th} control event c_i. If we assume that the graph becomes inconsistent, it means that there exists an actor a which runs parallel event sequences after c_i. In this case

$$a \in out(c_i),$$
$$a \in X,$$
$$a \notin in(c_i).$$

But these are in contradiction with the conditions

$$in(c) \subseteq X,$$
$$(out(c) \setminus in(c)) \cap X = \oslash.$$

Consequently, the graph must stay in a consistent state. □

Theorem 2. For every consistent π event graph:

$$Grp(Seq(\pi)) = \pi.$$

Proof. Considering the control event nodes, it is trivial that the π, $Seq(\pi)$ and $Grp(Seq(\pi))$ graphs contain the same set of control events. Let us assume that there is an edge related to a in the π graph, where the source event is c_1 and the output event is c_2. In this case, in $Seq(\pi)$:

$$a \in out(c_1), \quad a \in in(c_2), \quad c_1 \to c_2.$$

As the graph is consistent, no other sequence of a is active in the time period between c_1 and c_2. Thus, no such c_3 exists, where

$$c_1 \to c_2 \to c_3,$$
$$a \in in(c_3) \text{ or } a \in out(c_3).$$

This means that $a \in X$ for the time period (c_1, c_2) is in the $Seq(\pi)$ sequence. As a consequence, $Grp(Seq(\pi))$ will also contain an edge between c_1 and c_2 related to a.

If there is no edge between c_1 and c_2 related to a in the π graph, then the following cases may happen:

$$a \notin out(c_1) \text{ or } a \notin in(c_2),$$

or

$$\exists c_3 : c_1 \to c_3 \to c_2 \text{ where } a \in out(c_2) \text{ or } a \in in(c_3).$$

In the first case, $Grp(Seq(\pi))$ does not contain an edge between c_1 and c_2 related to a. Considering the second option, c_3 is in $Seq(\pi)$ between c_1 and c_2, thus the sequence starting from c_1 cannot be linked to c_2. Similarly, the sequence ending at c_2 cannot be linked to c_1.

Thus, $Grp(Seq(\pi))$ is equivalent with π. □

Regarding the prediction of the π graph, we decompose it into the prediction of sequences, and the graph is constructed from the component sequences. The proposed graph prediction algorithm consists of the following steps:

1. Prediction of the main level s_π sequence.
2. Construction of the $Grp(s_\pi)$) graph.
3. Prediction of the actor level event sequences for every edge in the $Grp(s_\pi)$) graph.
4. Construction of the detailed σ graph.

We can easily verify that the resulting event graph will be consistent.

The prediction of sequences is a standard problem in machine learning and the most widely used prediction method is the neural network approach. The dominating tool applied to this problem is the LSTM/GRU neural network family.

In our proposal, we focus on a slightly different approach. We use a novel compound neural network, containing more MLP units for the sequence prediction task. The main motivations in the selection of MLP are the following:

- MLP is the core element in many prediction toolsets.
- We can construct a flexible, well-parameterizable network system using MLP building blocks.

4 NEURAL NETWORK ARCHITECTURE AND THE PREDICTION PROCESS

4.1 Architecture for Event Graph Prediction

In the presented graph model, the event graph is decomposed into several event sequences. Based on this decomposition, graph prediction is performed by a multi-neural network architecture where each sequence type is processed by a separate neural network. Thus, in the case of event graph induction, the architecture contains

$1+|A|$ neural networks. There is a single network for the main level s_π sequence and there are separate neural networks for each actor type. As mentioned previously, an actor corresponds to a role or position in the business model. This means that during the training process, the event graph should be decomposed into sequences of different types. Each sequence type will be processed by a dedicated neural network Λ_i.

In the prediction of the sequences, the initial input is a prefix sequence that contains the event steps preceding the sequence section to be predicted. In conventional cases, this prefix sequence consists of elementary events. In our case, this approach is not suitable, as due to the merging process several prefix sequences can be found for a given sequence section in the graph.

To solve this problem, we use an extended prefix sequence model, where the components are sets instead of single event values:

$$(\{e_{11}, e_{12}, \dots\}, \{e_{21}, e_{22}, \dots\} \dots \{e_{m1}, e_{m2}, \dots\}).$$

The predicting neural network will merge these elements into a single description vector to use as an initial state for the prediction process.

Example 4. Having the following σ graph (Figure 7) and considering the event e_7, the prefix sequence has the following form in event level:

$$(\{e_5\}, \{e_1, e_2, e_3\}, \{e_4, e_5, e_6\}).$$

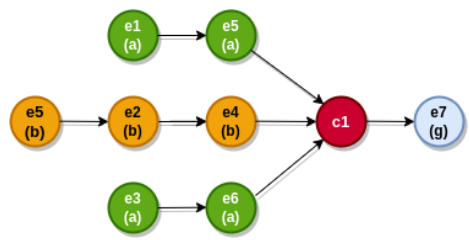


Figure 7. Event graph for Example 4

Assuming that there are 10 different action types (e_0–e_9), the binary (one-hot encoded) prefix part is equal to the following vector:

$$0000010000011100000000000111000.$$

4.2 Neural Network for Sequence Prediction

Considering the neural network for sequence prediction, we have tested five variants, two baseline models, and three novel variants. The two baseline models are the widely used LSTM and MLP models. The proposed novel modifications relate to

the extension of the prefix sequence length without the extension of the input size of the classification network. The proposed method applies a preparation step to reduce the input vector. We developed two methods for size reduction:

- simple union-based reduction,
- neural network-based reduction.

In the case of union-based reduction, every segment of the result feature vector is the union-based compression of a (usually longer) input feature sequence. Thus, having an input sequence

$$sec_i = sg_{i,1}, \ldots, sg_{i,m},$$

where sq_j denotes a segment, a fixed-size sequence of atomic events. The output feature sequence is

$$sec_o = sg_{o,1}, \ldots, sg_{o,n},$$

where $|sec_i| > |sec_o|$, contains segments created as the union of some input segments:

$$sg_o = \cup_l sg_{i,l}.$$

We assume that the integration domain for different output segments are disjoint segment groups. The compression process is characterized by a mapping vector

$$(l_1, l_2, \ldots, l_m) \quad \text{with} \quad \sum l_i = |sec_i|,$$

where l_i denotes the length of an input sequence domain in the input feature vector.

The second compression variant applies a separate neural network module instead of a single union operator. In this case

$$sg_{o,i} = \Lambda_i\{sg_{i,l}\}.$$

The system applies separate neural networks for every segment in the resulting feature vector. In our version, the last n most relevant elements of the prefix (nearest to the event to be predicted) will be delegated into the main input vector without any reduction. For the other elements, which denote earlier events that are further away from the event to be predicted, longer sections are taken and aggregated into a single abstract event.

The aggregation is executed with a standard MLP neural network which is a component of the main network. This means, that the optimizations of the aggregation networks are performed as part of the optimization of the whole network. The architecture of the proposed engine is presented in Figure 8.

Considering the efficiency of the proposed models, we performed a comparison test on benchmark sequences. The investigated network models are:

- MLP: baseline MLP,
- LSTM: baseline LSTM,

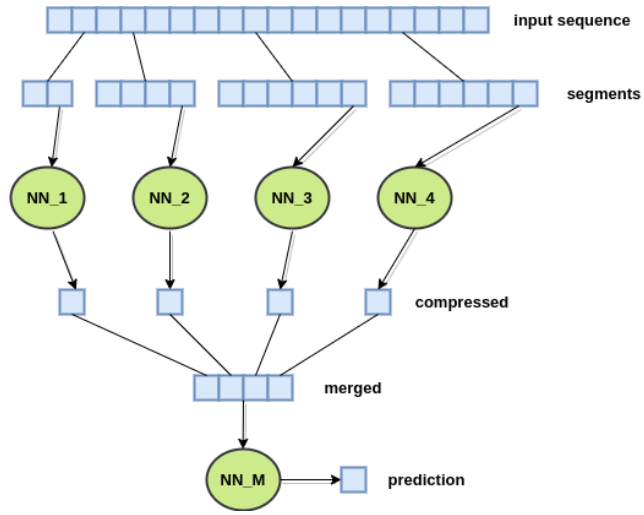


Figure 8. NN-based architecture for input sequence reduction

- BE-MLP: MLP with union-based reduction,
- BE-LSTM: LSTM with union-based reduction,
- HN-MLP: MLP with NN-based reduction.

We have used the following benchmark datasets from XES process mining competitions [52]:

- pdc_2016_1.xes (Process Discovery Contest 2016),
- pdc_2016_9.xes (Process Discovery Contest 2016),
- pdc_2017_5.xes (Process Discovery Contest 2017),
- pdc_2019_2.xes (Process Discovery Contest 2019)

and a uniform random generation file (load_random.xes).

The results of the comparison tests are presented in Table 2. The main conclusion is that the BE_MLP network type is a good choice for our architecture as

- it provides one of the best accuracy, and
- it has the lowest execution cost.

4.3 Neural Network for Sequence Graph Processing

The prediction of the event graph requires an ensemble of sequence prediction modules as the different agents cooperating in the event graph may have very different processing models. Thus we use different neural network units to model the different

Dataset	LSTM	MLP	NH-MLP	BE-MLP	BE-LSTM
pdc_2016_1.xes	63.5	63.4	63.4	63.9	64.1
pdc_2016_9.xes	82.0	82.5	81.2	81.8	82.6
pdc_2017_5.xes	62.4	62.8	63.7	64.8	64.2
pdc_2019_2.xes	64.6	65.2	63.9	66.3	65.6
load_random	58.2	57.5	56.6	58.7	58.0

Table 2. Accuracy comparison of sequence prediction NNs

agents. Besides the actor-level models, a main-level model – the level of synchronization events – is also included in the framework. The proposed architecture is presented in Figure 9.

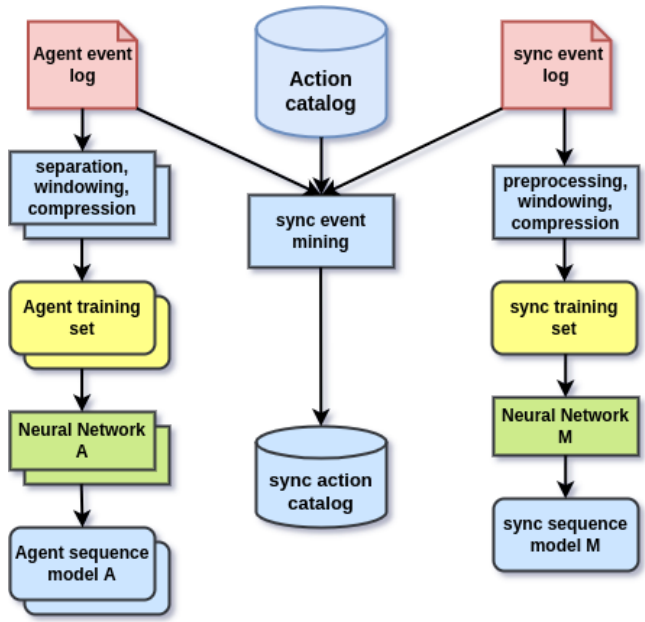


Figure 9. Architecture of the proposed MLP network system

In the architecture, the main inputs are the agent-level event logs and the action catalog which describes the artifact-level dependency among the actions. Based on these data, the system performs first the mining of synchronization events generating a catalog of synchronization events that includes, among others, the input and output actor lists. The engine will construct an event log for the synchronization events, and a sequence model for the synchronization events is built up using a neural network unit (Neural Network M). In the next phase, the engine separates the event log by the agents, and it generates the agent-level sequence model using the proposed neural network model with feature vector compression.

Considering the data representation formats, the external input is given by a list of lists, as every item in the prefix sequence is given by a set of events. The input prefix format uses one-hot encoding. Thus, for every position we have a fixed-size vector where the length is equal to the number of different event types. In a single section, the value at a given position for event e is 1, if e is in the list for the given position in the prefix sequence.

The outputs of the aggregation MLP modules are also one-hot encoded vectors. These vectors are concatenated into a single vector. Thus, the output of the main prediction MLP module is a one-hot encoded vector where every event type has a marker position calculated by the aggregation networks.

4.4 Prediction Process

After having the neural network models for both the main level and the actor level, we can use this compound architecture to predict event graphs. The overview of the prediction process is presented in Figure 10. The prediction is based on the following algorithm for the main level.

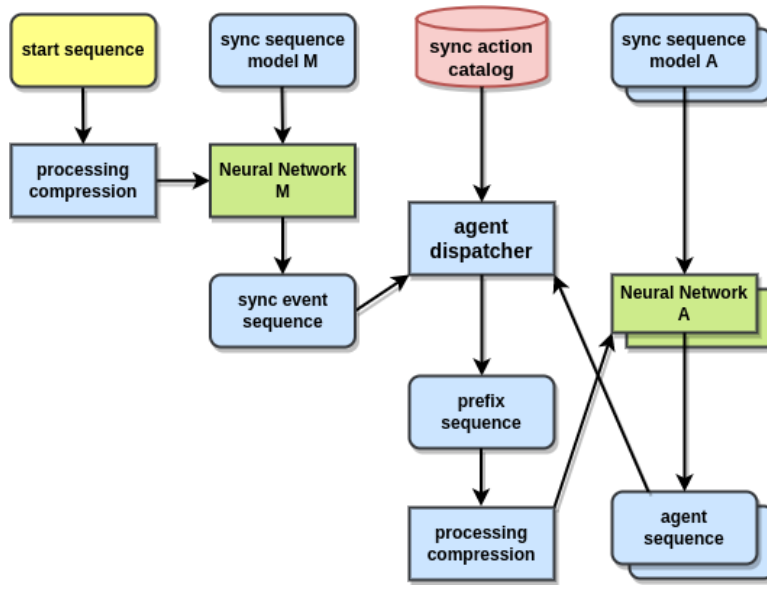


Figure 10. Architecture of the NN-based prediction process

Algorithm 1 (Prediction process)**.** First, we take an empty prefix sequence and predict the elements of the main sequence. In every iteration:

- the prefix is extended with the event predicted in the previous step,

- the engine predicts the next event based on the new prefix.

If the end symbol is the winner, or the length of the sequence exceeds a given limit, the prediction process terminates.

As shown earlier, the engine performs a conditional prediction, as the winner event at the main level should meet the following conditions to generate a consistent graph:

$$in(c) \subseteq X,$$
$$(out(c) \setminus in(c)) \cap X = \varnothing.$$

Thus, the engine selects the event which meets the consistency condition and its fitness value is maximal.

Another special characteristic of the proposed method is that, unlike the conventional approaches, the engine generates more candidates as output, not only the best one. In the standard approach, the engine selects the category with the highest rank in the output vector. In our approach, a quality threshold is used to determine the top candidates. Only those candidate events are selected where the fitness value (generated by the neural network as output) meets the condition

$$\varphi \geq \mu \, \varphi_{max}.$$

Here, φ_{max} is the fitness value of the best category and μ is the threshold value. If there is more than one winner event for the main sequence, the engine will add a c^+ synchronization event to the sequence.

In the case of actor-level sequences, the algorithm differs from the previous one in the following aspects:

- the initial prefix sequence is usually not empty, it is equal to the aggregation of the preceding sequences;
- there is no need for explicit c^+ node since every branching means an optional branch here (parallel branches may not occur at this level).

5 EVALUATION TESTS

For the evaluation of the efficiency of the proposed architecture, we have used open benchmark datasets, as well as synthetic random sequences generated by our sequence generator module.

5.1 Test Data Generation

To generate synthetic datasets with parallelism, we developed a framework to construct a workflow schema that can be used to generate matching event sequences.

Both modules were developed in the Python programming environment. The schema model language consists of the following parameters and node types:

- A = $[a_1, \ldots, a_n]$: set of actors;
- E = $[e_1, \ldots, e_n]$: set of user event types;
- C = $[c_S, c_E, c_{OR}, c_{AND}]$: set of control event types, namely the start event, the stop event, the OR and AND branching events;
- T = $[t_1, \ldots, t_n]$: execution time parameters of user events;
- P = $[p_1, \ldots, p_m]$: branching probabilities at OR-branching nodes.

The framework supports some additional features to validate the consistency of the model and to perform some corrections to guarantee the validity of the model. Automatic adjustment of the probability values is one of these model cleaning methods. The system also generates a graphical view of the schema to support manual verification.

5.1.1 Test Case A

In the following example, a sample schema is presented. The key parameters of the schema are set to the following values:

- A: a, b, c,
- E: $e1$, $e2$, $e3$, $e4$, $e5$, $e7$, $e8$, $e9$, $e10$,
- T: $e1 : [2, 3]$, $e2 : [1, 4]$, $e3 : [2, 3]$, $e4 : [4, 5]$, $e5 : [1, 2]$, $e7 : [1, 2]$, $e8 : [3, 5]$, $e9 : [1, 2]$, $e10 : [1, 2]$.

The execution time values are given with an interval where we used a uniform distribution. The connections of the event nodes are presented in Figure 11.

The test generator module generates a list of events for each of the given actors. In the following output sample, the symbols a, b, c denote the actors, and elements starting with e denote the user-defined events, while symbols starting with C mark the control events.

```
a:C0,e1,e2,e4,e2,e1,C3
b:C3,e5,e4,C4
c:C3,e9,e8,C4
a:C4,e7,e10,C-1
!
```

These kinds of lists are the primary input for the schema discovery engine.

5.1.2 Test Case B

In the second sample, a larger and more complex schema graph is constructed. In the model, three actors are defined: a, b and c. The list of available events, which

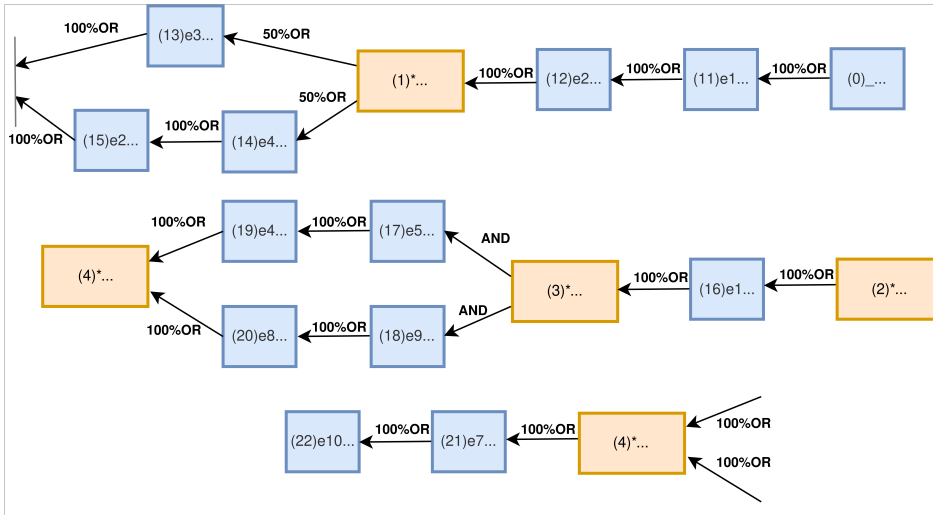


Figure 11. Sample workflow schema A

are given by their IDs and the related execution time interval, is as follows:

$$e1,(1,2) \quad e2,(3,5) \quad e3,(1,4) \quad e4,(2,3) \quad e5,(4,5)$$
$$e6,(1,3) \quad e7,(2,4) \quad e9,(1,4) \quad e10,(3,4) \quad e11,(2,4)$$

The graph structure of the schema is presented in Figure 12. For the training phase, 1 000 cases were generated for schema B.

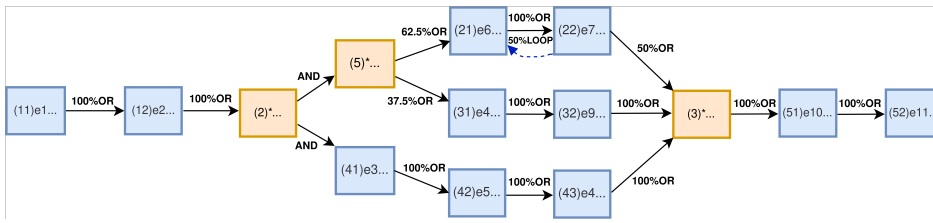


Figure 12. Sample workflow schema B

5.2 Test Results for Case A

The dataset for sample schema A contains the event sequences of 500 experiments. Based on this dataset, the engine can construct training sub-sequences. The framework generates 4 neural network units with similar structures (see Figure 13). In the training phase, the following validation accuracy values were achieved for the different neural network units:

- Main: 0.7980,
- Actor 'a': 0.9564,
- Actor 'b': 1.0000,
- Actor 'c': 1.0000.

```
Model: "model_187"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_562 (InputLayer)          [(None, 42)]         0
__________________________________________________________________________________________________
input_563 (InputLayer)          [(None, 28)]         0
__________________________________________________________________________________________________
input_564 (InputLayer)          [(None, 56)]         0
__________________________________________________________________________________________________
dense_1496 (Dense)              (None, 126)          5418        input_562[0][0]
__________________________________________________________________________________________________
dense_1498 (Dense)              (None, 84)           2436        input_563[0][0]
__________________________________________________________________________________________________
dense_1500 (Dense)              (None, 168)          9576        input_564[0][0]
__________________________________________________________________________________________________
dense_1497 (Dense)              (None, 42)           5334        dense_1496[0][0]
__________________________________________________________________________________________________
dense_1499 (Dense)              (None, 14)           1190        dense_1498[0][0]
__________________________________________________________________________________________________
dense_1501 (Dense)              (None, 14)           2366        dense_1500[0][0]
__________________________________________________________________________________________________
concatenate_187 (Concatenate)   (None, 70)           0           dense_1497[0][0]
                                                                 dense_1499[0][0]
                                                                 dense_1501[0][0]
__________________________________________________________________________________________________
dense_1502 (Dense)              (None, 140)          9940        concatenate_187[0][0]
__________________________________________________________________________________________________
dense_1503 (Dense)              (None, 14)           1974        dense_1502[0][0]
==================================================================================================
Total params: 38,234
Trainable params: 38,234
Non-trainable params: 0
```

Figure 13. Structure of a neural network module

Regarding the schema generation from the training set, the engine first generates the sequence of the control events. During the generation of the candidate sequences, a weight threshold $\mu = 0.90$ was applied. The engine generates two sequences: (1) and (1, 2, 3, 4). From these two variants, the second one corresponds to a real sequence. Considering the agent-level sequences for the second control sequence, we get the following results:

```
C0 {'a': ['e1', 'e2', 'e4', 'e2', 'e1', 'C-1'],
 'b': [], 'c': []}

C3 {'a': [], 'b': ['e5', 'e4', 'C-1'],
 'c': ['e9', 'e8', 'C-1']}

C4 {'a': ['e7', 'e10', 'C-1'], 'b': [], 'c': []}
```

The list contains the sequences starting at the different control events. For example, at control events $C0$ and $C4$ only one sequence starts which is executed by actor a. In the list, symbol $C-1$ denotes the end control event.

As the results show, the engine could discover and re-generate the source schema correctly, because the output is equivalent to the input in Figure 11.

5.3 Test Results for Case B

In the first processing phase, the engine determined the occurrences of synchronization events. The module processed the traces in the log and generated a list of synchronization events for every trace. The output list contains two types of event sequences:

```
[(1, 'e2')] [(2,v 'e4'), (3, 'e3')] : 5.5 8.0
[(4, 'e9'), (6, 'e4')] [(7, 'e10')] : 15.0 18.5
```

and

```
[(1, 'e2')] [(2, 'e6'), (3, 'e3')] : 5.5 7.5
[(4, 'e7'), (6, 'e4')] [(7, 'e10')] : 15.0 18.5
```

The last two numbers denote the calculated time window of the current synchronization event. Based on the generated synchronization event sequences, as input training set, the constructed neural network model M will predict the following synchronization sequence:

```
['EMPTY', 'C2', 'C3', 'EOS']
```

In the next phase, the engine predicts the agent-level sequences in the following form:

```
a : ['e1', 'e2', 'EOS']
b : ['e3', 'e5', 'e4', 'EOS']
c : ['e4', 'e9', 'EOS']
a : ['e10', 'e11', 'EOS']
```

After performing more trace generation experiments, the action list of agent c may vary. Another typical output is the action chain containing actions 'e6' and 'e7':

```
c : ['e6', 'e7', 'e6', 'e7', 'e6', 'e7', 'e6', 'e7',
 'e6', 'e7', 'e6', 'e7', 'e6', 'e7', 'EOS']
```

If we consider also the weight ratio between the best one and the second one, we can see that most of the steps are unambiguous, except at the start of the event sequence for c. The higher the ratio, the stronger the unambiguity. The next list shows these ratio values at the different steps:

```
y e4  , weight ratio: 1.464207
y e9  , weight ratio: 234.37918
y EOS , weight ratio: 237.77512
```

Considering the resulting graph for a single selection case (only the best candidate is selected as the next event), we can see that it contains a parallel execution section, where both agents b and c are active (Figure 14).

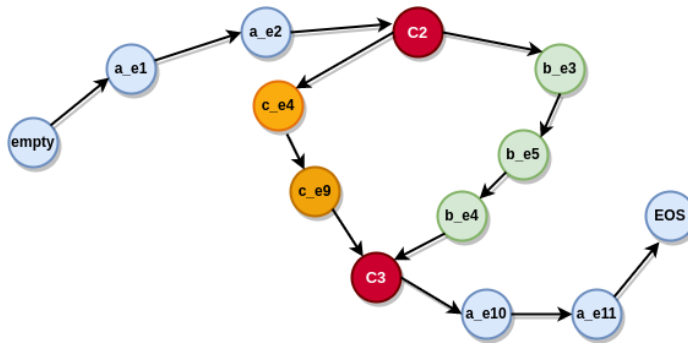


Figure 14. Generated schema graph for single selection

If we use a multi-selection prediction approach where we allow alternative routes, we get the result graph presented in Figure 15.

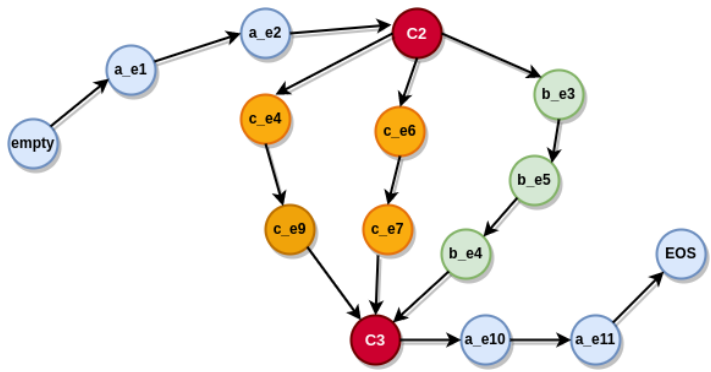


Figure 15. Generated schema graph for multi-selection

We can see that the method builds up the same schema graph we have used to generate the event logs. Thus, the proposed method is suitable for discovering also parallel and optional sequences in the schema graph, unlike the previous engines.

To demonstrate the differences between the proposed and the standard models, we give also the output of the baseline alpha miner method implemented in the PM4PY package (Figure 16). As the alpha miner uses a very different approach and

uses only a smaller set of information, it can not present the same complexity as the proposed method.

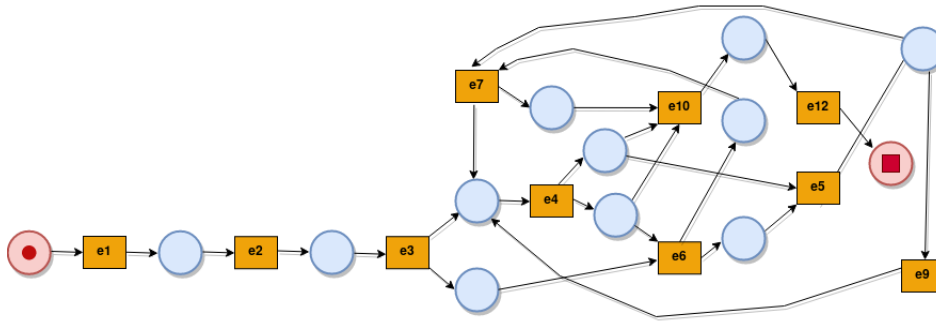


Figure 16. Output of alpha miner method

6 CONCLUSION

The paper presents a novel neural network model for the discovery of event patterns containing parallel and optional sequences of different actors. The proposed model can process more complex event graphs and situations than the conventional methods. Our new approach introduces synchronization nodes into the event sequences in such a way that these synchronization events can be discovered automatically. The schema mining engine performs two-level processing. First, it performs sequence discovery at the synchronization level, then on the level of actors. We have developed a technique for the compression of the prefix sequences which improves the accuracy of the standard sequence prediction methods. The efficiency of our neural network-based method is demonstrated with some benchmark tests and examples.

REFERENCES

[1] Kirchmer, M.—Franz, P.: Value-Driven Robotic Process Automation (RPA). In: Shishkov, B. (Ed.): Business Modeling and Software Design (BMSD 2019). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 356, 2019, pp. 31–46, doi: 10.1007/978-3-030-24854-3_3.

[2] van der Aalst, W. M. P.—Bichler, M.—Heinzl, A.: Robotic Process Automation. Business & Information Systems Engineering, Vol. 60, 2018, No. 4, pp. 269–272, doi: 10.1007/s12599-018-0542-4.

[3] Wang, X.—Li, J.: Detecting Communities by the Core-Vertex and Intimate Degree in Complex Networks. Physica A: Statistical Mechanics and Its Applications, Vol. 392, 2013, No. 2, pp. 2555–2563, doi: 10.1016/j.physa.2013.01.039.

[4] Belhadi, A.—Djenouri, Y.—Lin, J. C. W.—Cano, A.: A General-Purpose Distributed Pattern Mining System. Applied Intelligence, Vol. 50, 2020, No. 9, pp. 2647–2662, doi: 10.1007/s10489-020-01664-w.

[5] Weytjens, H.—De Weerdt, J.: Process Outcome Prediction: CNN vs. LSTM (with Attention). In: Del Río Ortega, A., Leopold, H., Santoro, F. M. (Eds.): Business Process Management Workshops (BPM 2020). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 397, 2020, pp. 321–333, doi: 10.1007/978-3-030-66498-5_24.

[6] Scheidl, H.—Fiel, S.—Sablatnig, R.: Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm. 2018, pp. 253–258, doi: 10.1109/ICFHR-2018.2018.00052.

[7] Abonyi, J.—Károly, R.—Dörgö, G.: Event-Tree Based Sequence Mining Using LSTM Deep-Learning Model. Complexity, Vol. 2021, 2021, Art. No. 7887159, doi: 10.1155/2021/7887159.

[8] van der Aalst, W.: Process Mining: Data Science in Action. Springer, Berlin, Heidelberg, 2016, doi: 10.1007/978-3-662-49851-4.

[9] van der Aalst, W. M. P.—Ter Hofstede, A. H. M.—Kiepuszewski, B.—Barros, A. P.: Workflow Patterns. Distributed and Parallel Databases, Vol. 14, 2003, No. 1, pp. 5–51, doi: 10.1023/A:1022883727209.

[10] Russell, N.—van der Aalst, W. M. P.—ter Hofstede, A. H. M.: Workflow Patterns: The Definitive Guide. The MIT Press, 2016.

[11] Russell, N. C.—ter Hofstede, A. H. M.—van der Aalst, W.—Mulyar, N. A.: Workflow Control-Flow Patterns – A Revised View. Technical Report. 2006.

[12] Workflow Management Coalition Terminology & Glossary. Workflow Management Coalition, 1999, www.wfmc.org.

[13] van der Aalst, W.—Adriansyah, A.—van Dongen, B.: Causal Nets: A Modeling Language Tailored Towards Process Discovery. In: Katoen, J. P., König, B. (Eds.): CONCUR 2011 – Concurrency Theory. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6901, 2011, pp. 28–42, doi: 10.1007/978-3-642-23217-6_3.

[14] Buijs, J. C. A. M.—van Dongen, B. F.—van der Aalst, W. M. P.: A Genetic Algorithm for Discovering Process Trees. 2012 IEEE Congress on Evolutionary Computation, 2012, pp. 1–8, doi: 10.1109/CEC.2012.6256458.

[15] Arriagada-Benítez, M.—Sepúlveda, M.—Munoz-Gama, J.—Buijs, J. C. A. M.: Strategies to Automatically Derive a Process Model from a Configurable Process Model Based on Event Data. Applied Sciences, Vol. 7, 2017, No. 1, Art. No. 1023, doi: 10.3390/app7101023.

[16] 1849–2016 – IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. IEEE, 2016, doi: 10.1109/IEEESTD.2016.7740858.

[17] Ghahfarokhi, A. F.—Park, G.—Berti, A.—van der Aalst, W.: OCEL Standard. Process and Data Science Group, RWTH Aachen University, 2020, http://ocel-standard.org/.

[18] Laxman, S.—Tankasali, V.—White, R. W.: Stream Prediction Using a Generative Model Based on Frequent Episodes in Event Sequences. Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08), 2008, pp. 453–461, doi: 10.1145/1401890.1401947.

[19] Károly, R.—Abonyi, J.: Multi-Temporal Sequential Pattern Mining Based Improvement of Alarm Management Systems. 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2017, pp. 3870–3875, doi: 10.1109/SMC.2016.7844838.

[20] Weiss, G. M.: Industry: Predicting Telecommunication Equipment Failures from Sequences of Network Alarms. In: Kloesgen, W., Zytkow, J. (Eds.): Handbook of Knowledge Discovery and Data Mining. Oxford University Press, Inc., 2002, pp. 891–896.

[21] Csalódi, R.—Abonyi, J.: Integrated Survival Analysis and Frequent Pattern Mining for Course Failure-Based Prediction of Student Dropout. Mathematics, Vol. 9, 2021, No. 5, Art. No. 463, doi: 10.3390/math9050463.

[22] Kinnebrew, J. S.—Biswas, G.: Identifying Learning Behaviors by Contextualizing Differential Sequence Mining with Action Features and Performance Evolution. Proceedings of the 5th International Conference on Educational Data Mining (EDM 2012), 2012, pp. 57–62.

[23] Taub, M.—Azevedo, R.—Bradbury, A. E.—Millar, G. C.—Lester, J.: Using Sequence Mining to Reveal the Efficiency in Scientific Reasoning During STEM Learning with a Game-Based Learning Environment. Learning and Instruction, Vol. 54, 2018, pp. 93–103, doi: 10.1016/j.learninstruc.2017.08.005.

[24] Fournier-Viger, P.—Li, J.—Lin, J. C. W.—Truong Chi, T.—Uday Kiran, R.: Mining Cost-Effective Patterns in Event Logs. Knowledge-Based Systems, Vol. 191, 2020, Art. No. 105241, doi: 10.1016/j.knosys.2019.105241.

[25] Truong-Chi, T.—Fournier-Viger, P.: A Survey of High Utility Sequential Pattern Mining. High-Utility Pattern Mining: Theory, Algorithms and Applications, Springer, Cham, Studies in Big Data, Vol. 51, 2019, pp. 97–129, doi: 10.1007/978-3-030-04921-8_4.

[26] d'Aquin, M.—Jay, N.: Interpreting Data Mining Results with Linked Data for Learning Analytics: Motivation, Case Study and Directions. Proceedings of the Third International Conference on Learning Analytics and Knowledge (LAK '13), 2013, pp. 155–164, doi: 10.1145/2460296.2460327.

[27] El-Hajj, M.—Zaïane, O. R.: Non-Recursive Generation of Frequent K-Itemsets from Frequent Pattern Tree Representations. In: Kambayashi, Y., Mohania, M., Wöß, W. (Eds.): Data Warehousing and Knowledge Discovery (DaWaK 2003). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2737, 2003, pp. 371–380, doi: 10.1007/978-3-540-45228-7_37.

[28] Lin, C. W.—Hong, T. P.—Lu, W. H.—Lin, W. Y.: An Incremental FUSP-Tree Maintenance Algorithm. 2008 Eighth International Conference on Intelligent Systems Design and Applications (ISDA 2008), IEEE, Vol. 1, 2008, pp. 445–449, doi: 10.1109/ISDA.2008.126.

[29] Liu, J.—Yan, S.—Wang, Y.—Ren, J.: Incremental Mining Algorithm of Sequen-

tial Patterns Based on Sequence Tree. Vol. 138, 2012, pp. 61–67, doi: 10.1007/978-3-642-27869-3_8.

[30] Dhakshayani, J.—Sivasathya, S.—Sharmiladevi, S.—LourduMarie Sophie, S.: A Survey on Incremental Pattern Mining and Their Techniques. International Journal of Emerging Technologies and Innovative Research, Vol. 5, 2018, No. 10, pp. 284–290, http://www.jetir.org/papers/JETIR1810946.pdf.

[31] Rizvee, R. A.—Arefin, M. F.—Ahmed, C. F.: Tree-Miner: Mining Sequential Patterns from SP-Tree. In: Lauw, H. W., Wong, R. C. W., Ntoulas, A., Lim, E. P., Ng, S. K., Pan, S. J. (Eds.): Advances in Knowledge Discovery and Data Mining (PAKDD 2020). Springer, Cham, Lecture Notes in Computer Science, Vol. 12085, 2020, pp. 44–56, doi: 10.1007/978-3-030-47436-2_4.

[32] Liu, X.—Zheng, L.—Zhang, W.—Zhou, J.—Cao, S.—Yu, S.: An Evolutive Frequent Pattern Tree-Based Incremental Knowledge Discovery Algorithm. ACM Transactions on Management Information Systems (TMIS), Vol. 13, 2022, No. 3, Art. No. 30, doi: 10.1145/3495213.

[33] Chen, J.: An UpDown Directed Acyclic Graph Approach for Sequential Pattern Mining. IEEE Transactions on Knowledge and Data Engineering, Vol. 22, 2010, No. 7, pp. 913–928, doi: 10.1109/TKDE.2009.135.

[34] Patel, A.—Patel, A.: Graph Based Approach and Clustering of Patterns (GACP) for Sequential Pattern Mining. International Journal on Computer Science and Engineering, Vol. 3, 2011, No. 4, pp. 1501–1509.

[35] Singh, D. K.—Sharma, V.—Sharma, S.: Graph Based Approach for Mining Frequent Sequential Access Patterns of Web Pages. International Journal of Computer Applications, Vol. 40, 2012, No. 10, pp. 33–37, doi: 10.5120/5003-7285.

[36] Dong, W.—Lee, E. W.—Hertzberg, V. S.—Simpson, R. L.—Ho, J. C.: GASP: Graph-Based Approximate Sequential Pattern Mining for Electronic Health Records. In: Bellatreche, L., Dumas, M., Karras, P., Matulevičius, R., Awad, A., Weidlich, M., Ivanović, M., Hartig, O. (Eds.): New Trends in Database and Information Systems (ADBIS 2021). Springer, Cham, Communications in Computer and Information Science, Vol. 1450, 2021, pp. 50–60, doi: 10.1007/978-3-030-85082-1_5.

[37] Borges, J.—Levene, M.: Data Mining of User Navigation Patterns. In: Masand, B., Spiliopoulou, M. (Eds.): Web Usage Analysis and User Profiling (WebKDD 1999). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1836, 1999, pp. 92–112, doi: 10.1007/3-540-44934-5_6.

[38] Hingston, P.: Using Finite State Automata for Sequence Mining. Australian Computer Science Communication, Vol. 24, 2002, No. 1, pp. 105–110.

[39] Jacquemont, S.—Jacquenet, F.—Sebban, M.: Mining Probabilistic Automata. Machine Learning, Vol. 75, 2009, No. 1, pp. 91–127, doi: 10.1007/S10994-008-5098-Y.

[40] Beschastnikh, I.—Brun, Y.—Schneider, S.—Sloan, M.—Ernst, M. D.: Leveraging Existing Instrumentation to Automatically Infer Invariant-Constrained Models. Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11), 2011, pp. 267–277, doi: 10.1145/2025113.2025151.

[41] Krka, I.—Brun, Y.—Medvidovic, N.: Automatic Mining of Specifications from

Invocation Traces and Method Invariants. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014), 2014, pp. 178–189, doi: 10.1145/2635868.2635890.

[42] Le, T. D. B.—Le, X. B. D.—Lo, D.—Beschastnikh, I.: Synergizing Specification Miners Through Model Fissions and Fusions (T). 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015), 2016, pp. 115–125, doi: 10.1109/ASE.2015.83.

[43] Gers, F. A.—Schmidhuber, J.—Cummins, F.: Learning to Forget: Continual Prediction with LSTM. 1999 Ninth International Conference on Artificial Neural Networks (ICANN 99), Vol. 2, 1999, pp. 850–855, doi: 10.1049/CP:19991218.

[44] Gers, F. A.—Schmidhuber, J.: LSTM Recurrent Networks Learn Simple Context-Free and Context-Sensitive Languages. IEEE Transactions on Neural Networks, Vol. 12, 2001, No. 6, pp. 1333–1340, doi: 10.1109/72.963769.

[45] Jamshed, A.—Mallick, B.—Kumar, P.: Deep Learning-Based Sequential Pattern Mining for Progressive Database. Soft Computing, Vol. 24, 2020, No. 22, pp. 17233–17246, doi: 10.1007/s00500-020-05015-2.

[46] Ji, S.—Kim, J.—Im, H.: A Comparative Study of Bitcoin Price Prediction Using Deep Learning. Mathematics, Vol. 7, 2019, No. 10, Art. No. 898, doi: 10.3390/math7100898.

[47] Sutskever, I.—Vinyals, O.—Le, Q. V.: Sequence to Sequence Learning with Neural Networks. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 27 (NIPS 2014). Curran Associates, Inc., 2014, pp. 3104–3112, https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

[48] Karatzoglou, A.—Jablonski, A.—Beigl, M.: A Seq2Seq Learning Approach for Modeling Semantic Trajectories and Predicting the Next Location. Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'18), 2018, pp. 528–531, doi: 10.1145/3274895.3274983.

[49] Rebane, J.—Karlsson, I.: Seq2Seq RNNs and ARIMA Models for Cryptocurrency Prediction: A Comparative Study. Proceedings of SIGKDD Workshop on Fintech (SIGKDD Fintech'18), 2018.

[50] Baumel, T.—Eyal, M.—Elhadad, M.: Query Focused Abstractive Summarization: Incorporating Query Relevance, Multi-Document Coverage, and Summary Length Constraints into seq2seq Models. CoRR, 2018, doi: 10.48550/arXiv.1801.07704.

[51] Wu, P.—Lu, Z.—Zhou, Q.—Lei, Z.—Li, X.—Qiu, M.—Hung, P. C. K.: Bigdata Logs Analysis Based on seq2seq Networks for Cognitive Internet of Things. Future Generation Computer Systems, Vol. 90, 2019, pp. 477–488, doi: 10.1016/j.future.2018.08.021.

[52] IEEE CIS Task Force on Process Mining. www.tf-pm.org/competitions-awards/discovery-contest [accessed 2022-01-20].

László Kovács is Professor at the University of Miskolc, Hungary. He is the Head of the Department of Information Technology and Leader of the Research Group on Machine Learning, a specialist in knowledge modeling and data mining. He obtained his Ph.D. degree in technical sciences from the University of Miskolc. His main teaching areas cover DB systems, data mining, and ontology management. His research interests include soft computing, concept set management, heuristic optimization, ontology modeling, statistical grammar induction, and rough set models. He has about 250 publications with 450 citations.

Erika Baksáné Varga is Associate Professor at the Institute of Informatics, University of Miskolc, Hungary. She received her Ph.D. degree in computer science from the University of Miskolc in 2011. She has academic experience in teaching procedural and object-oriented programming, data analysis and data mining. Her research interests include teaching methodologies of programming, data and process modeling, data analysis and data mining, ontological modeling, NLP, and text mining.

Péter Mileff is a senior software developer and Associate Professor at the Institute of Informatics, University of Miskolc, Hungary. He obtained his Ph.D. in computer science in 2008. He has more than 10 years of teaching experience in the fields of SW development and building and designing SW systems. He has a high-level professional and research experience in the field of computer visualization and game development. He has 15 years of industrial experience in the design and development of digital payment systems and platform-independent technologies.