Graph Autoencoders for Business Process Anomaly Detection

Siyu Huo[1(✉)], Hagen Völzer[2], Prabhat Reddy[1], Prerna Agarwal[3], Vatche Isahagian[4], and Vinod Muthusamy[5]

[1] IBM Research, Yorktown Heights, NY, USA
{siyu.huo,Prabhat.Reddy}@ibm.com
[2] IBM Research, Rueschlikon, ZH, Switzerland
hvo@zurich.ibm.com
[3] IBM Research, Gurgaon, HR, India
preragar@in.ibm.com
[4] IBM Research, Cambridge, MA, USA
vatchei@ibm.com
[5] IBM Research, Austin, TX, USA
vmuthus@us.ibm.com

Abstract. We propose an approach to identify anomalies in business processes by building an anomaly detector using graph encodings of process event log data coupled with graph autoencoders. We evaluate the proposed approach with randomly mutated real event logs as well as synthetic data. The evaluation shows significant performance improvements (in terms of F1 score) over previous approaches, in particular with respect to other types of autoencoders that use flat encodings of the same data. The performance improvements are also stable under training and evaluation noise. Our approach is generic in that it requires no prior knowledge of the business process.

1 Introduction

Anomaly detection is an unsupervised machine learning technique that has become very popular with the recent advances in AI. An anomaly detector automatically learns correlations, i.e., regularities in its structured input data and flags irregular data as anomalies, where the notion of an anomaly depends on the dataset and use case.

Business process data have many different aspects, such as the activities, their ordering, their duration and waiting times, the acting resources and roles, the business objects and associated values, states, milestones, decisions and process outcomes. Parts or combinations of these data as well as abstractions, transformations and aggregations thereof, such as KPIs, sliding windows, rolling averages etc. can be presented to an anomaly detector which would all result in different notions of anomaly.

In line with recent studies [24,26,27], in this paper we consider process activities, their ordering, and their business object attributes in relation to the following anomaly detection use cases:

A. Polyvyanyy et al. (Eds.): BPM 2021, LNCS 12875, pp. 417–433, 2021.
https://doi.org/10.1007/978-3-030-85469-0_26

- Log errors: When the process event log is distorted because parts of it has been collected manually, i.e., subject to human error, or it has been recorded or transmitted by an unreliable mechanism, then an anomaly detector can be used to detect and correct such perturbations, provided they occur with limited frequency.
- Exception analysis: Finding and inspecting rare exceptional cases can help to understand deviations from expected behavior. This is similar to variant analysis and conformance checking [3,35] in traditional process mining [2]. However, the traditional concept of variant is very fine-grained such that normal behaviour can distribute over a large number of variants and exceptional behavior may not always be easily distinguished from rare but normal variants. Conformance checking on the other hand requires a carefully handcrafted specification of normative (or normal) behavior, either as an imperative (BPMN) model [4,7,8] or, in a more declarative form, as a set of rules. For more complex processes, creating and maintaining such a specification over time may require substantial effort, whereas anomaly detection requires no such specification.
- Process drift: Exceptions can also be detected at run time, and when multiple anomalies occur in succession, they could indicate changes in external conditions, process drift, or unwanted behavior.

Anomaly detectors for the above use cases can be partially evaluated with real event logs that are perturbed with limited random mutations to see how well the anomaly detector identifies these mutations [23,24,27].

A popular type of anomaly detector is an *autoencoder*, which is an artificial neural network that learns an efficient representation of the input data, i.e., an encoding or embedding, together with a decoding that reproduces the input data from that internal representation in a way that minimizes the reproduction error. A threshold on the reproduction error identifies anomalous input data [42]. The learned encoding can be seen as a form of dimensionality reduction of the input data.

Earlier work [24,26,28] has applied *multilayer perceptron-based autoencoders*, *variational autoencoders* and *LSTM-based autoencoders* to business process anomaly detection and showed that they outperform other methods, such as t-STIDE [41], OC-SVM [39], HMM [18], and Likelihood [27] in terms of accuracy, noise endurance, and generalizability. However, their absolute performance is still limited leaving substantial room for improvement.

In this paper, we show that the performance indeed can be substantially improved by enriching the autoencoder input data representation with activity relationships, i.e., edges between different events of a trace. Thus, the autoencoder input becomes a graph, and we then present the graph to a *graph autoencoder* [20] with edge-conditioned convolutions (ECC) [40]. We evaluate the performance of our graph autoencoder on both synthesized and real-life event logs from the Business Process Intelligence Challenge (BPIC) against several earlier methods.

2 Background

To better explain the idea of this paper, we introduce some basic notations and concepts, partially borrowed from [1].

Definition 1 (Universes). *Let U_E be the set of all* event identifiers*, let U_C be the set of all* case identifiers*, let U_A be the set of all* activity identifiers*, let U_T be the totally ordered set of all* timestamp identifiers*, and let $U_F = \{U_{f_1}, ..., U_{f_k}\}$ be the identifier collections of event associated* features *with k categories: $\{f_1, ..., f_k\}$.*

We assume that events are characterized by various properties. For example, an event has a timestamp, corresponds to an activity, is performed by a particular resource with several attributes, etc. Given the focus of this paper, we assume that events should at least contain activity and timestamp properties, and there is a function $\pi_A : U_E \rightarrow U_A$ that assigns to each event an activity from the finite set of process activities, and also a function $\pi_T : U_E \rightarrow U_T$ that assigns a timestamp to each event.

Definition 2 (Sequence, Trace, Event Log). *Given a set A, a finite sequence over A of length n is a function $\sigma \in A^* : \{1, ..., n\} \rightarrow A$, typically written as $\sigma = \langle a_1, a_2, ..., a_n \rangle$, where $\sigma(i) = a_i$. For any sequence σ, we define $|\sigma| = n$. A* trace *is a finite non-empty sequence of events $\sigma \in U_E^*$ such that each event appears at most once and time is non-decreasing, i.e., for $1 \leq i \leq j \leq |\sigma|$, $\sigma(i) \neq \sigma(j)$ and $\pi_T(\sigma(i)) \leq \pi_T(\sigma(j))$. Let C be the set of all possible traces. An* event log *is a set of traces $L \subseteq C$ such that each event appears at most once in the entire log, and each trace in the log represents the execution of one case assigned with a case identifier (case ID) by a function $\pi_C : U_E \rightarrow U_C$.*

Definition 3 (Directed Graph). *A directed graph $G = (V, E)$ consists of a nonempty set of* nodes *V and a set of directed* edges *$E \subseteq V \times V$, and for a directed edge $e_{u,v} = (u, v), e_{u,v} \in E$, we call u the* tail *node of e and v the* head *node of e. $N_d(v)$ defines* neighbor *(namely predecessor) nodes of v, which returns the nodes that directly connect to v with incoming edge towards v. For example, for a simple graph with three nodes $u \rightarrow v \leftarrow w$, $N_d(v)$ returns the node set $\{u, w\}$.*

2.1 Autoencoders and Anomaly Detection

An autoencoder is a type of artificial neural network for unsupervised learning, which contains two main components: an *encoder* and a *decoder* [6]. The encoder takes an input vector $x \in [0, 1]^d$ and maps it to a hidden representation $h \in [0, 1]^{d'}$ by a deterministic mapping function $f_\phi : [0, 1]^d \rightarrow [0, 1]^{d'}$ parameterized by ϕ. Symmetrically, the decoder takes the encoder output h and maps it to $z \in [0, 1]^d$ by a mapping function $g_\psi : [0, 1]^{d'} \rightarrow [0, 1]^d$ parameterized by ψ. f_ϕ and g_ψ here can be corresponding typical neural networks such as multi-layer perceptrons, recurrent neural networks, etc. Each input x is thus first encoded

into h and then decoded into z. During training, the parameters are optimized to minimize the observed *reconstruction loss* $L(x, z)$ of input x and its reconstruction z through backpropagation:

$$\begin{aligned} \phi^\star, \psi^\star &= \arg\min_{\phi,\psi} L(x, z) \\ &= \arg\min_{\phi,\psi} L(x, g_\psi(f_\phi(x)) \end{aligned} \tag{1}$$

where the reconstruction loss function is traditionally defined as the mean square error:

$$L(x, z) = \frac{1}{d}\sum_{i=1}^{d}(x^{(i)} - z^{(i)})^2 \tag{2}$$

An autoencoder based anomaly detector uses reconstruction loss as the anomaly score, i.e., a data point with relatively high reconstruction loss is considered to be an anomaly. During the training phase, the autoencoder learns f_ϕ and g_ψ through data training split based on Eq. (1), and then a reconstruction loss threshold θ is chosen to classify anomalies during the validation phase based on another independent validation data split. (Our setting details are described in Sect. 4.2.) Finally, the model inference relies on f_ϕ, g_ψ and θ obtained from the training and validation phases described as above.

2.2 Feature Encoding for Business Process Event Logs

Machine learning always requires an adequate feature extraction and engineering for the target data. In the business process literature, [21] converts the event log into vectors using a method that is similar to continuous bag of words (CBOW), which is a natural language processing (NLP) method for document encoding. These vectors include different encoding levels: activity level, trace level, and the entire log level. These vectors, or their embeddings, are then fed into different neural networks designed for purposes such as anomaly detection, trace clustering, and process comparison. As all event logs record the executed process activities and their ordering, [17] extracts the event ordering from the log. However, these methods do not leverage other business process object attributes.

The greater the variety of process data attributes included in the input vector, the more types of anomalies can be detected by the autoencoder. For example, besides the activity name, a delayed process activity could result in an unusual ending timestamp, which is a numerical attribute in the log. Also several business data attributes are often related to each other, such as the credit score and loan amount in a loan application. Usually it is quite challenging to extract only useful attribute features without any prior domain knowledge about the business process. Some earlier work [24,26] treats the time series event log as flat structural data, and directly applies one-hot or dummy encoding on categorical features and re-scales numerical features to generate process encoding vectors. These encoding vectors are then concatenated in time series order and fed into the neural network. In the business process anomaly detection literature, the

primary difference between existing autoencoder approaches and our work is that we encode both, structural information as well as various kinds of business process attributes by using a graph encoding on them.

2.3 Graph Neural Networks

Over the past few years, there has been a surge of approaches that seek to learn the representations of graph nodes, or entire (sub-)graphs based on Graph Neural Networks (GNN), which extend well-known network architectures, including recurrent neural networks (RNNs) and convolutional neural networks (CNNs), to graph data [10,12,13,15,16,22,25,38]. Most of the existing graph neural networks are instances of the Message Passing Neural Network (MPNN) framework [14] for node aggregation. In a directed graph G, the forward pass consists of two phases, a message passing phase and a readout phase. At the time step t, the message passing function S_t and vertex update function U_t describe how the hidden state h_v^t for each node in the graph is updated:

$$s_v^t = \sum_{u \in N_d(v)} S_t(h_v^{t-1}, h_u^{t-1}, e_{u,v}) \tag{3}$$

$$h_v^t = U_t(h_v^{t-1}, s_v^t) \tag{4}$$

Then the readout phase computes a feature vector for the whole graph using a readout function R according to:

$$\hat{y} = R(\{h_v | v \in G\}) \tag{5}$$

The message functions S_t, vertex update functions U_t, and readout function R are all learned differentiable functions, and associated parameters can be learned by back-propagation based on an error function, such as an error function on a graph classification prediction score $\hat{y}$ where y is the classification label.

3 Method

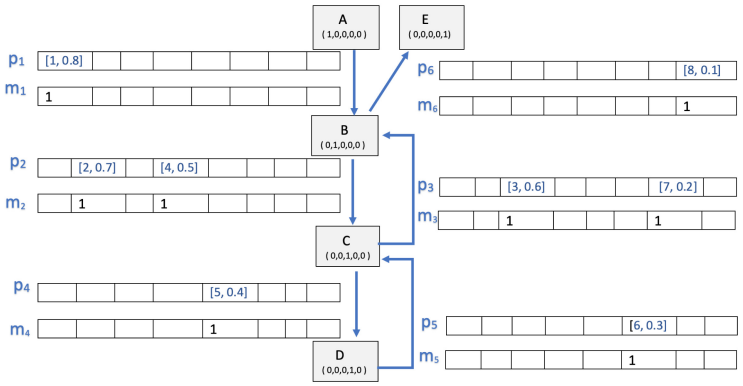
3.1 Graph Construction on the Process Event Log

The Graph-based representation of business process data has been previously used to improve process discovery [32], build Bayesian networks [33,37], and generate likelihood graphs of causally dependent event attributes [27]. To represent the structural process information of an event trace in the log, we build a directed graph $G = (V, E)$ on it as follows (similar to a Directly-Follows-Graph in process mining). We treat activity names, which are a required property of the events in the log, as nodes V in the graph, and edges E correspond to every pair of adjacent events in the time ordered trace. Thus a trace with n activities results in $n - 1$ directed edges. Since the node is identified by its activity name, an edge that is formed by two adjacent activity names may contain several

duplicates in a trace. To leverage this situation, we propose a positional embedding method by using the specific location of the adjacent events in a vector p to represent edge positional information and associated activity attributes, and the vector p is treated as an edge feature vector. This way, duplicate edges are represented by digits in different locations within the same positional vector. Following the graph construction steps below, a trace with n activities potentially results in $k \leq n-1$ positional vectors p corresponding to the unique k edges. The overall graph construction diagram can be represented by k label vectors $m \in \{0,1\}^{t_m-1}$, which record activity occurrence information and are targets of the autoencoder's decoding output.

CaseID	Activity	Timestamp	Variable1	Variable 2
0	A	3/1/2021	1	1
0	B	3/2/2021	2	1.8
0	C	3/3/2021	4	2.5
0	B	3/4/2021	7	3.1
0	C	3/5/2021	11	3.6
0	D	3/6/2021	16	4
0	C	3/7/2021	22	4.3
0	B	3/8/2021	29	4.5
0	E	3/9/2021	37	4.6

(a) An example event log.



(b) The constructed graph corresponding to the example event log.

Fig. 1. A graph construction example based on the example event log.

The following are the steps of the graph construction algorithm and the feature extraction for one trace with n activities:

1. Compute the maximum trace length $t_m = max\{|\sigma| : \sigma \in U_E\}$ over the whole events U_E in the log, and obtain an event attribute feature embedding with dimension $d_e = d_n + d_c$ by concatenating activity numerical features (rescaling

with dimension d_n) and categorical features (one-hot encoding with dimension d_c) of event attributes (excluding the activity name columns). Meanwhile, track all existing activities as nodes and assign them an initial node embedding h_v^0 (one-hot encoding).

2. Generate $n-1$ activity-activity pairs based on occurrence order, and maintain this order. For example, trace (A, B, C, B, C) ends up in 4 ordered activity-activity pairs (A, B), (B, C), (C, B), and (B, C), which define edges in the graph.
3. Within one activity-activity pair at position $i \in [1, n-1]$ based on the second step, treat the activity that happened first as u, and the other as v. Also compute the edge feature vector based on the event attribute feature embedding distance (subtraction) of u and v, and put this computed vector in index i of the positional vector p which has dimension $d_p = (t_m - 1) \times d_e$ (each computed vector takes d_e space in p initialized with all 0, and $t_m - 1$ is the number of activity-activity pairs for the longest trace). Finally, build a corresponding binary edge label vector $m \in \{0, 1\}^{t_m - 1}$, setting position i to 1 and all other positions to 0.
4. Loop over the third step until the end of the trace. It will end up with $n-1$ positional vectors p and edge label vectors m associated with each edge. As we mentioned, there could be duplicate edges (as the example in step 2), and we deal with this by aggregating (sum) p and m with names of u and v as keys (e.g. 'AB', 'BC', 'CB'). Finally we end up with $k \leq n-1$ positional vectors p as final edge features and edge label vectors m associated with each unique edge that appeared in the trace. The initial node features h_v^0 are specified by one-hot encoding of the activity names.

Figure 1 shows an example of building a graph based on an example event log with 5 nodes and 6 edges. Each node comes with its initial embedding based on its name, and each edge is associated with a positional vector p_i and a label vector m_i. The first element in each cell of p_i stands for the feature computed from the "Variable 1" column, the second is for the "Variable 2" column. The blank space in each cell is filled with $\{0\}^d$ where d is the same dimension for all cells in the vector. In the example, d is 2 (equal to d_e) for p and 1 for m.

Looking more closely at this example, in the constructed graph view, the activity E is more closely related to and affected by A (connected through B). However from the perspective of a flat sequence (such as with LSTM or RNN), E is the farthest event from A. In a hypothetical process, it is possible that event A is a starting event, E is the proceeding event, B corresponds to a "check status" activity, and events C and D happen when B "fails"; otherwise E would directly follow B if A happened. Intuitively, in such a process E should be closely related to A, but the flat sequence where B is near the end of the sequence can not preserve such process logic. Meanwhile in the case where the sequence is very long, the sequence encoder (such as an LSTM or RNN) could suffer from the vanishing gradient problem [30]. These are some of the factors we have considered and lead us to believe that the graph encoding, which takes more

structural process information into account, is a better choice than sequence encoding.

3.2 Encoding by the Graph Autoencoder with ECC

To make full use of our graph construction and the converted business process features, we apply edge-conditioned convolution (ECC) filters to obtain a better graph encoding. The ECC filters in the graph autoencoder transform each node representation computed from the previous layer of the neural network and combines them with their transformed neighbor nodes representation conditioning on edge features. Let $l \in \{1, ..., l_m\}$ be the layer index in a feed-forward neural network, h_v^l, $h_{e_{u,v}}^l$ be the vector representation for the node v and edge $e_{u,v}$ at layer l (layer 0 is the input layer). The node representation updating through ECC in the graph can be formulated as:

$$h_v^l = \phi\Big(F^l(h_v^{l-1}; w) + \sum_{u \in N_d(v)} F_e^l(h_{e_{u,v}}; w_e) h_u^{l-1} + b^l\Big) \tag{6}$$

where ϕ is the activation function, $F^l : \mathbb{R}^{d_{l-1}} \rightarrow \mathbb{R}^{d_l}$, and $F_e^l : \mathbb{R}^{d_p} \rightarrow \mathbb{R}^{d_l \times d_{l-1}}$ denote a parameterized feature transformation neural network for the node feature and the edge feature, and their corresponding parameters w, w_e. The learnable random initialized bias term $b^l \in \mathbb{R}^{d_l}$ can be learned through backpropagation during training.

The graph reconstruction, i.e., readout, can be treated as an edge label prediction or link prediction similar to the approach introduced in [20]. Based on Eq. (5), the graph reconstruction edge-associated probability of recreating binary edge labels m is formulated as:

$$m' = \text{sigmoid}\left(F_r(h_u^{l_m} \oplus h_v^{l_m}; w_r)\right) \tag{7}$$

where $u, v \in V, e_{u,v} \in E$, and $F_r : \mathbb{R}^{2d_{l_m}} \rightarrow \mathbb{R}^{t_m - 1}$ denotes neural network processing concatenated vector of h_u and h_v with trainable parameter w_r. $\oplus$ denotes vector concatenation.

Finally, instead of Eq. (2), in this paper the reconstruction loss (error) of the target graph is defined by the average of the binary cross entropy loss:

$$L(m', m) = \frac{1}{k(t_m - 1)} \left(-\sum_{i=1}^{k} \sum_{j=1}^{t_m - 1} m_i^{(j)} \log(m'^{(j)}_i) + (1 - m_i^{(j)})(1 - \log(m'^{(j)}_i)) \right) \tag{8}$$

The autoencoder training objective is to minimize the output from (8) through backpropagation. Furthermore, Eq. (8) is used as the anomaly scoring function which is consistent with the approaches in the literature on anomaly detection.

4 Experiments

4.1 Simulated Anomalous Data Sets

As we are not aware of a standard process mining anomaly detection benchmark, we rely on fully or partially simulated data used in the literature. Note that we do not devise our own datasets but borrow the datasets and preprocessing steps from existing work. We consider three types of simulated datasets containing labeled anomalous cases. These anomalous cases are introduced in three ways described below. The dataset details are shown in Table 1. Training, validation, and testing are randomly sampled based on the provided ratios in the table and corresponding data set. Note that the sum may not total to 100% since we used the remainder to create noise cases (mentioned below) based on the original normal and anomalous data.

Table 1. Statistics of event logs we used in our experiments. The 'N' and 'A' in the Test set refer to normal and anomalous cases, respectively. The ratios are counted as a portion of the number of traces. The number of attributes count does not include the activity name.

Event log	No. traces	Max. trace length	No. activities	No. attributes	Training	Validation	Test
BPIC2012	13,087	95	24	2	50%	20%	15%N, 15%A
BPIC2013	7,554	34	11	2	50%	20%	15%N, 15%A
Loan	10,000	63	19	2	25%	10%	15%N, 15%A
BPIC2017	14,289	100	26	1	32%	12%	6%N, 6%A
Large	5,000	14	43	4	32%	12%	6%N, 6%A
Huge	5,000	13	55	4	32%	12%	6%N, 6%A

4.1.1 Log Attributes Anomalies

To evaluate the detection of anomalies in a distorted or wrongly recorded log, we follow [24] by using simulated anomalies injected into two public real event logs, from the BPI challenges of 2012 and 2013 (BPIC2012 and BPIC2013). We use the same original data with the corresponding process to introduce anomalies[1] as described in [24]. Every anomalous trace is simulated by randomly choosing a proportion of $L/2$ activity columns and replacing them with random activities sampled from U_A, and choosing a proportion of $L/2$ activity duration columns and replacing them with uniformly random sampled values ranging from the minimum and maximum activity duration values observed in this column. Thus it will create a proportion of L anomalous values for one anomalous trace. In our experiments we set $L = 0.5$ for these two datasets.

[1] Available at BPIC2012 and BPIC2013 data sets with injected anomalies.

4.1.2 Activity Ordering Anomalies

In this case, an anomaly is caused by a variation of the activity ordering [36] or concept drift [9]. During process execution, a change in the activity ordering may happen due to a change in the environment, and it can be seasonal or hourly but only takes up a small portion of all executed cases. This type of anomaly is sampled by a certain pattern or distribution instead of random error. We use the same simulated loan access data set from [23]. The event logs in this work are simulated by the Apromore [34] platform for a loan access process. Anomalies here are introduced as changes of the activity ordering in mainly 3 categories: *Insertion ("I")*, e.g., add/remove fragment or duplicate fragment in the control flow; *Resequentialization ("R")*, e.g., synchronize two fragments or make two fragments sequential in the control flow; and *Optionalization ("O")*, e.g., change the branching frequency or make a fragment skippable in the control flow. The details of these categories are described in [23]. In our experiment, we use six composite operations ("IOR", "IRO", "OIR", "ORI", "RIO", "ROI") to form anomalous traces. For example, the operation "IRO" is obtained by first adding a new activity ("I"), then making this activity in parallel with an existing activity ("R") and finally skipping the latter activity ("O"). Each composite operation of the six forms have the same amount of anomalous traces.

4.1.3 Composite Anomalies

We also evaluated our anomaly detector against the combination of random attribute changes and changes in the activity ordering using the existing three data sets: BPIC2017, 'Large', and 'Huge'[2] simulated by [27], where anomalies are injected by the following 6 types of operations:

1. Skip: A sequence of up to 3 events is skipped.
2. Insert: Up to 3 random activities are inserted in random places in the trace.
3. Rework: A sequence of up to 3 events is repeated.
4. Early: A sequence of up to 2 events is moved backward in the trace.
5. Late: A sequence of up to 2 events is moved forward in the trace.
6. Attribute: An attribute value is mutated in up to 3 events in the trace.

It should be mentioned that in this case the above operations are directly applied to the event log without considering any underlying process model, thus they are not simply mixtures of the previous two anomalous types. For example, the Rework operation applied on a log may not result in an anomalous case if the control flow permits a loop over these events, such as a process model with a 'check' and 'resubmit' control loop until the submission is satisfied.

Meanwhile during training and validation, in order to test the impact of noise as mentioned in the literature [26], we introduce a noise ratio $r \in [0, 0.5]$ to all data sets mentioned above, where r is the proportion of anomalous cases manually added into training and validation sets.

[2] Available at: BPIC2017, 'Large', and 'Huge' data sets with injected anomalies.

4.2 Experiment Settings and Results

We compare our *Graph Autoencoder (GAE)* with three baseline autoencoders: the original *multilayer perceptron-based autoencoder (AE)* [24,26–28], a *variational autoencoder (VAE)* [24] and a *LSTM-based autoencoder (LSTMAE)* [24]. Note that a modified version of the latter was also used in [27] as an encoding network. We choose the F1 score, where correctly detected anomaly cases are treated as true positive cases, to measure their detection performance.

For the configurations of the three baseline models, we consider the neural network layer settings used in [24]. For AE and VAE we use two encoding dense layers [31] and two decoding dense layers, and for LSTMAE we use one encoding LSTM layer and one decoding LSTM layer. These layer sizes are adapted to input dimensions. Regarding the graph neural network configurations, initially we use one ECC layer ($l = 1$) for Eq. (6) and 1-layer dense layer for F, F_e and F_r for Eqs. (6) and (7). The hidden embedding $h_v^{l_m}$ is set to be $2\times$ h_v^0 for each dataset.

For the training of the neural networks, we use the Adam optimizer [19] with a learning rate of 0.001, the training batch size is 8 and the training epoch is 100 with early stopping [11] to avoid overfitting. During the validation phase, we choose the anomaly threshold θ based on the reconstruction error as in [26], but we use the average reconstruction error on the validation data instead of the training data since the validation data can better represent data which has not been used to train the model. Meanwhile we tune the model hyper-parameters to maximize the F1 score on the validation data. To summarize, we use the training set to train the autoencoder and the validation set to determine an appropriate θ and tune model hyper-parameters.

Table 2. F1 score for different autoencoders and datasets with training and validation set noise ratio $r = 0$.

	BPIC2012	BPIC2013	Loan	BPIC2017	Large	Huge
GAE	**0.95**	**0.67**	**0.98**	**0.72**	**0.93**	0.86
LSTMAE	0.64	0.37	0.86	0.63	0.90	**0.88**
AE	0.52	0.32	0.89	0.56	0.83	0.81
VAE	0.51	0.27	0.83	0.52	0.73	0.69

First we evaluate our proposed method and the other baselines on the six synthesized data sets mentioned above, with noise $r = 0$. Table 2 shows the results on the test sets. We observe that our GAE method performs better in 5 out of 6 cases than the other three baseline approaches in the non-noise settings. There are two main reasons for this improvement. Firstly, the GAE captures extra structural process information and relations among activity occurrences, which are key components in process data, in addition to process attribute features, and treats these activities as nodes (functional central components) within the neural network. The other three baseline approaches do not exploit that structure

and treat the data as simple flat event series data. Secondly, for our case-level anomaly detection task, the objective function for the GAE, which is to recreate the edge label, becomes easier than for the other three autoencoders, which try to recreate the data itself with mixed features. Some data sets such as BPIC2012 and BPIC2017 contain long traces that result in large encoding feature vectors for AE, VAE and LSTMAE to recreate, and this can cause inefficient training and limiting their performance. The GAE however simplifies the task by recreating abstract structural information (i.e., binary edge labels) instead of data attributes themselves. Thus it is easier for the GAE training to converge.

In practice training and validation sets will have some degree of noise as mentioned above, and in order to test the effects of noise on the performance, assuming that in a normal system anomalous cases are rare [5,29], we set the noise parameter r to be $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ for the six data sets. Figure 2 shows the F1 score under these different noise ratio settings, where we can observe that due to uncertainty and inconsistency introduced by noise in training and validation, model performance becomes generally worse. However the GAE approach still performs better in most situations (23 out of 30 experimental points) than the other baselines, suggesting that GAE is more robust to noisy data.

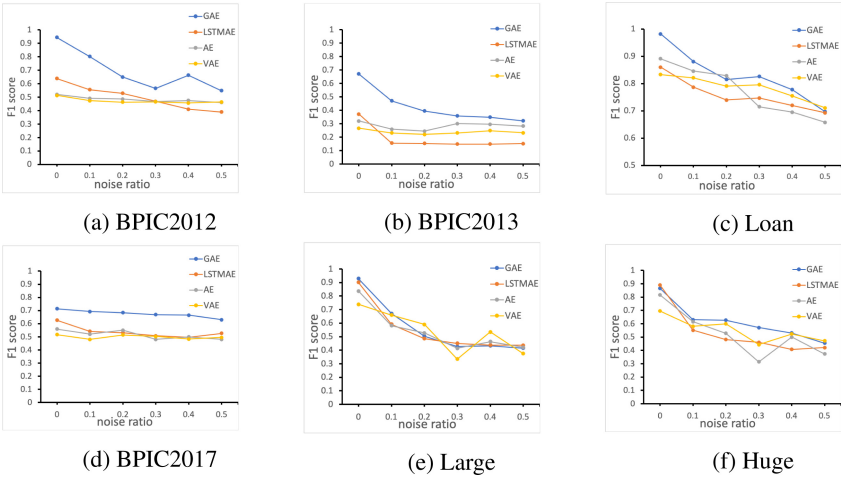
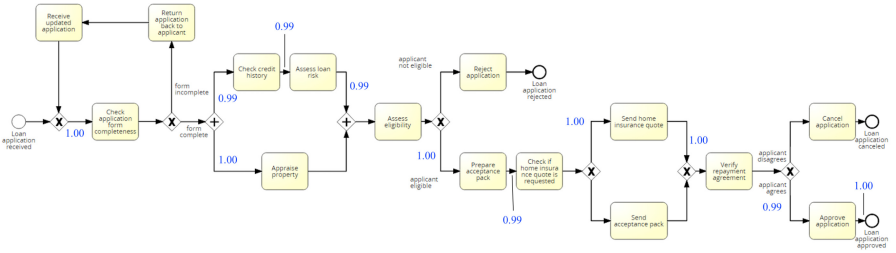


Fig. 2. F1 score for the six data sets with varying training/validation noise ratio r.

4.3 Anomaly Example and Diagnostic Information

In addition to identifying an anomaly, sometimes it is useful to explain or give insights on the nature of the anomaly. To that end, in this section, we provide an example that illustrates the information that our approach produces.

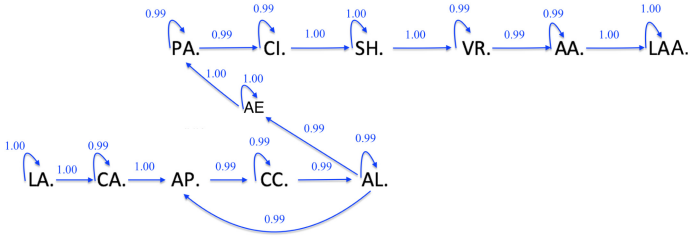
Figure 3a shows a BPMN model that was used by [23] to create the 'Loan' event log mentioned above. This model represents 'normal' process behaviour



(a) A process model representing normal behavior annotated with edge anomaly scores from a normal trace.

CaseID	Activity	Timestamp	EventType
b1_101	Loan application received	2004-02-10T11:30:00.000+00:00	assign
b1_101	Loan application received	2004-02-10T11:30:00.000+00:00	complete
b1_101	Check application form completeness	2004-02-10T11:30:00.000+00:00	assign
b1_101	Check application form completeness	2004-02-10T12:11:43.834+00:00	complete
b1_101	Appraise property	2004-02-10T12:11:43.834+00:00	assign
b1_101	Check credit history	2004-02-10T12:11:43.834+00:00	assign
b1_101	Check credit history	2004-02-10T12:45:07.845+00:00	complete
b1_101	Assess loan risk	2004-02-10T12:45:07.845+00:00	assign
b1_101	Appraise property	2004-02-10T12:46:02.233+00:00	complete
b1_101	Assess loan risk	2004-02-10T13:05:47.283+00:00	complete
b1_101	Assess eligibility	2004-02-10T13:05:47.283+00:00	assign
b1_101	Assess eligibility	2004-02-10T13:07:04.188+00:00	complete
b1_101	Prepare acceptance pack	2004-02-10T13:07:04.188+00:00	assign
b1_101	Prepare acceptance pack	2004-02-10T13:52:51.236+00:00	complete
b1_101	Check if home insurance quote is requested	2004-02-10T13:52:51.236+00:00	assign
b1_101	Check if home insurance quote is requested	2004-02-10T14:10:44.533+00:00	complete
b1_101	Send home insurance quote	2004-02-10T14:10:44.533+00:00	assign
b1_101	Send home insurance quote	2004-02-10T14:19:59.779+00:00	complete
b1_101	Verify repayment agreement	2004-02-10T14:19:59.779+00:00	assign
b1_101	Verify repayment agreement	2004-02-10T14:41:08.538+00:00	complete
b1_101	Approve application	2004-02-10T14:41:08.538+00:00	assign
b1_101	Approve application	2004-02-10T14:52:57.812+00:00	complete
b1_101	Loan application approved	2004-02-10T14:52:57.812+00:00	assign
b1_101	Loan application approved	2004-02-10T14:52:57.812+00:00	complete

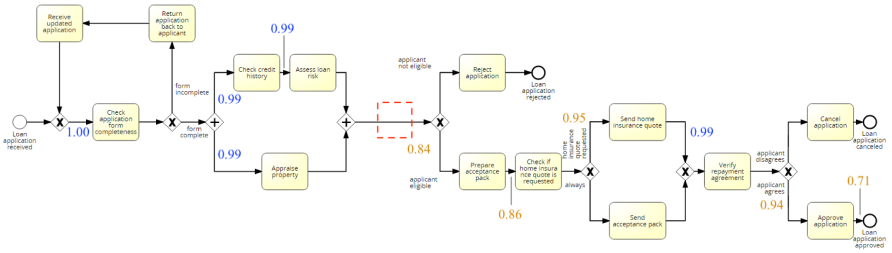
(b) Event log of normal trace `b1_101`, taken from the 'Loan' data set [23].



(c) GAE generated graph with edge anomaly scores for normal trace `b1_101`.

Fig. 3. An example of a normal case from the 'Loan' data set.

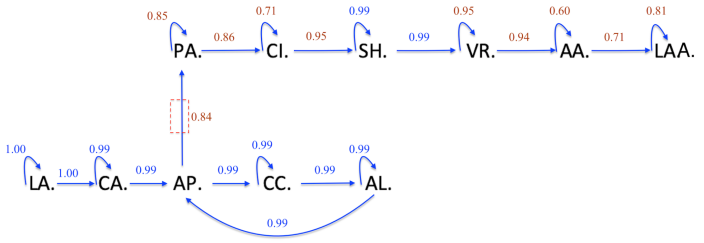
(ignore the blue numerical annotations for now). Figure 3b shows an example trace that is generated from this BPMN model and is included in one subset of the 'Loan' event log. The corresponding graph that is constructed by our GAE is shown in Fig. 3c. It is classified as a 'normal' trace with low reconstruction error.



(a) A process model representing anomalous behavior annotated with edge anomaly scores from an anomalous trace.

CaseID	Activity	Timestamp	EventType
r1_10	Loan application received	2004-05-03T14:00:00.000+00:00	assign
r1_10	Loan application received	2004-05-03T14:00:00.000+00:00	complete
r1_10	Check application form completeness	2004-05-03T14:00:00.000+00:00	assign
r1_10	Check application form completeness	2004-05-03T14:09:02.570+00:00	complete
r1_10	Appraise property	2004-05-03T14:09:02.570+00:00	assign
r1_10	Check credit history	2004-05-03T14:09:02.570+00:00	assign
r1_10	Check credit history	2004-05-03T14:11:19.888+00:00	complete
r1_10	Assess loan risk	2004-05-03T14:11:19.888+00:00	assign
r1_10	Assess loan risk	2004-05-03T14:11:49.985+00:00	complete
r1_10	Appraise property	2004-05-03T14:52:26.357+00:00	complete
r1_10	Prepare acceptance pack	2004-05-03T14:52:26.357+00:00	assign
r1_10	Prepare acceptance pack	2004-05-03T15:20:39.245+00:00	complete
r1_10	Check if home insurance quote is requested	2004-05-03T15:20:39.245+00:00	assign
r1_10	Check if home insurance quote is requested	2004-05-03T15:20:56.809+00:00	complete
r1_10	Send home insurance quote	2004-05-03T15:20:56.809+00:00	assign
r1_10	Send home insurance quote	2004-05-03T15:29:23.537+00:00	complete
r1_10	Verify repayment agreement	2004-05-03T15:29:23.537+00:00	assign
r1_10	Verify repayment agreement	2004-05-03T15:43:44.163+00:00	complete
r1_10	Approve application	2004-05-03T15:43:44.163+00:00	assign
r1_10	Approve application	2004-05-04T10:26:41.331+00:00	complete
r1_10	Loan application approved	2004-05-04T10:26:41.331+00:00	assign
r1_10	Loan application approved	2004-05-04T10:26:41.331+00:00	complete

(b) Event logs of anomalous trace `r1_10`, taken from the 'Loan' data set



(c) GAE generated graph with edge anomaly scores for anomalous trace `r1_10`.

Fig. 4. An example of anomalous case.

The reconstruction error has been proposed in earlier work [26] as additional diagnostic information. However, note that the reconstruction error is unbounded and hence its interpretation is less intuitive than a probability. Since we use explicit graph edges in our model, we can obtain, for each edge of the trace, a prediction probability that can be used as an edge-specific anomaly score, based on Eqs. (7) and (8). These probabilities are shown in Fig. 3c adjacent to

the edges. (Here, we use the first 2 letters to represent the activity name, e.g., "AE" stands for "Assess eligibility".) As expected, these probabilities are very high for a normal trace of our highly regular process. Figure 3b shows, for easier reference, the same probabilities played back in the original process model based on a matching of the graph edges.

Figure 4a shows a process model that can be seen as a concept drift with respect to the model in Fig. 3a: the activity 'Assess eligibility' has been removed. The traces generated from the process model in Fig. 4a can hence be considered as anomalies with respect to the normal behavior represented by Fig. 3a, Fig. 4b shows such an anomalous trace generated from the process model in Fig. 4a. The corresponding GAE graph and its edge anomaly scores are depicted in Fig. 4c. A drop in the anomaly score, marked in brown, now occurs on various edges, first between the activities "Access Loan Risk", "Appraise property" and "Prepare acceptance pack". This corresponds to the missing edges from the former two activities to the latter in the training set, where the flow has to go through the activity "Assess eligibility". Again, Fig. 4a shows, for easier reference, the same probabilities played back in the process model.

It is interesting to observe that some edges after this first anomalous point in the control flow are also affected with low scores, and the reason is that our objective function in the GAE is to predict a edge label m, which contains the absolute activity location information in a trace, based on positional vector p. Therefore, a case of delayed or early execution would also be punished with a low score. For example, "Approve application" happens earlier than usual (even if it correctly directly follows "Verify Repayment") and hence we observe a lower score on the edge toward it.

Also it should be mentioned that the GAE computes the embedding of each event type as a node embedding and makes predictions based on them. Neighboring events should have similar graph embeddings [16] in general since they are updated and synchronized closely with each other. Meanwhile edge predictions rely on events which are very close to each other, thus missing one event does not bring those scores sharply down to 0, and we can observe that edge anomaly scores are around 15% lower than the normal for respective edges.

In summary, our GAE setup detects anomalous process cases by taking into account both event ordering and the timestamps and activity durations in a trace. The edge prediction probability scores also reflect detected anomalies.

5 Conclusion

In this paper, we propose a new graph autoencoder approach for anomaly detection for business process event logs. As opposed to existing methods, we construct a graph for each log trace and apply a graph encoding in order to capture structural process information. Experimental results on six data sets with three types of anomaly settings demonstrates the advantages of our approach over other autoencoder based approaches, which do not use that structural process

information. Also, experiments show our graph autoencoder is robust to a certain level of noise during training and validation, which we believe is beneficial for use in practice.

References

1. Aalst, W.V.: Process mining: data science in action (2016)
2. Aalst, W.V., Adriansyah, A., Medeiros, A.K.A.D., Arcieri, F., Baier, T.: Process mining manifesto. Business Process Management Workshops (2011)
3. Aalst, W.V., Medeiros, A.K.A.D.: Process mining and security: detecting anomalous process executions and checking process conformance. Electron. Notes Theor. Comput. Sci. **121**, 3–21 (2005)
4. Aalst, W.V., Weijters, A., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**, 1128–1142 (2004)
5. Angiulli, F., Pizzuti, C.: Fast outlier detection in high dimensional spaces. In: PKDD (2002)
6. Bengio, Y.: Learning deep architectures for AI. Found. Trends Mach. Learn. **2**, 1–127 (2007)
7. Bezerra, F., Wainer, J.: Algorithms for anomaly detection of traces in logs of process aware information systems. Inf. Syst. **38**, 33–44 (2013)
8. Bezerra, F., Wainer, J., Aalst, W.V.: Anomaly detection using process mining. In: BMMDS/EMMSAD (2009)
9. Bose, R.P., Aalst, W.V., Žliobaitė, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Networks Learn. Syst. **25**, 154–171 (2014)
10. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. CoRR abs/1312.6203 (2014)
11. Caruana, R., Lawrence, S., Giles, C.L.: Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In: NIPS (2000)
12. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: NIPS (2016)
13. Duvenaud, D., et al.: Convolutional networks on graphs for learning molecular fingerprints. In: NIPS (2015)
14. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 1263–1272. PMLR, 06–11 August 2017
15. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, 2005, vol. 2, pp. 729–734 (2005)
16. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS (2017)
17. Hinkka, M., Lehto, T., Heljanko, K., Jung, A.: Structural feature selection for event logs. ArXiv abs/1710.02823 (2017)
18. Jain, R., Abouzakhar, N.: Hidden Markov model based anomaly intrusion detection. Presented at the (2012)
19. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2015)
20. Kipf, T., Welling, M.: Variational graph auto-encoders. ArXiv abs/1611.07308 (2016)

21. Koninck, P.D., Broucke, S.V., Weerdt, J.: act2vec, trace2vec, log2vec, and model2vec: representation learning for business processes. In: BPM (2018)
22. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. CoRR abs/1511.05493 (2016)
23. Maaradji, A., Dumas, M., Rosa, M., Ostovar, A.: Fast and accurate business process drift detection. In: BPM (2015)
24. Nguyen, H.C., Lee, S., Kim, J., Ko, J., Comuzzi, M.: Autoencoders for improving quality of process event logs. Expert Syst. Appl. **131**, 132–147 (2019)
25. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. ArXiv abs/1605.05273 (2016)
26. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Analyzing business process anomalies using autoencoders. Mach. Learn. **107**(11), 1875–1893 (2018)
27. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Binet: multi-perspective business process anomaly classification. Information Systems, p. 101458, October 2019
28. Nolle, T., Seeliger, A., Mühlhäuser, M.: Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders. In: DS (2016)
29. Ord, K.: Outliers in statistical data: V. barnett and t. lewis, 1994, 3rd edition, 584 pp., [uk pound]55.00, isbn 0-471-93094-6. International Journal of Forecasting 12, 175-176 (1996)
30. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: ICML (2013)
31. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. ArXiv abs/1912.01703 (2019)
32. Pegoraro, M., Uysal, M.S., Aalst, W.V.: Discovering process models from uncertain event data. ArXiv abs/1909.11567 (2019)
33. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. BPM (2014)
34. Rosa, M., Reijers, H., Aalst, W.V., Dijkman, R., Mendling, J., Dumas, M., García-Bañuelos, L.: Apromore: an advanced process model repository. Expert Syst. Appl. **38**, 7029–7040 (2011)
35. Rozinat, A., Aalst, W.V.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**, 64–95 (2008)
36. Sarno, R., Sari, P.L.I., Ginardi, H., Sunaryono, D., Mukhlash, I.: Decision mining for multi choice workflow patterns. Presented at the (2013)
37. Savickas, T., Vasilecas, O.: Business process event log transformation into bayesian belief network. In: ISD (2014)
38. Scarselli, F., Gori, M., Tsoi, A., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Networks **20**, 61–80 (2009)
39. Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., Platt, J.C.: Support vector method for novelty detection. In: NIPS (1999)
40. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. Presented at the (2017)
41. Warrender, C., Forrest, S., Pearlmutter, B.A.: Detecting intrusions using system calls: alternative data models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344), pp. 133–145 (1999)
42. Zhou, C., Paffenroth, R.C.: Anomaly detection with robust deep autoencoders. Presented at the (2017)