Detecting Anomalies in Hybrid Business Process Logs

Stephen Pauwels
University of Antwerp
Middelheimlaan 1
Antwerp, Belgium
stephen.pauwels@uantwerpen.be

Toon Calders
University of Antwerp
Middelheimlaan 1
Antwerp, Belgium
toon.calders@uantwerpen.be

ABSTRACT

Checking and analyzing various executions of different Business Processes can be a tedious task as the logs from these executions may contain lots of events, each with a (possibly large) number of both numerical and categorical attributes. We developed a way to automatically model the behavior captured in log files with dozens of these attributes. The advantage of our method is that we do not need any prior knowledge about the data or the attributes. We introduce a new algorithm that is able to learn a model of a log file starting from the data itself. The learned model can then be used to detect anomalous executions in the data. To achieve this we extend Dynamic Bayesian Networks with numerical attributes and functional dependencies to better model the normal behavior found in log files. The model is capable of scoring events and cases, even when previously unseen values or new combinations of values appear in the log file. An important benefit of our model is the ability to give a decomposition of the score that indicates the root cause of the anomalies. We also conducted a comparison with other state-of-the-art algorithms for detecting anomalies in Business Processes which shows that our approach outperforms other algorithms.

CCS Concepts

•Information systems → Business intelligence;

Keywords

Anomaly Detection, Probabilistic models, Event log and Workflow data

Table 1: Example Log file containing normal (black) and anomalous (red) cases. The normal events are used for training the model.

Time	ID	Type	Activity	UID	UName	URole	tID
0	0	System	Log in	001	User1	employee	1
1	1	System	Logged in	001	User1	employee	1
1	2	Request	Create	001	User1	employee	1
2	3	Request	Send Mail	001	User1	employee	1
3	4	System	Log in	002	User2	manager	2
4	5	System	Logged in	002	User2	manager	2
6	6	Request	Create	001	User1	employee	2
7	7	Request	Send Mail	001	User1	employee	2
8	8	Request	Disapproved	002	User2	manager	2
9	9	System	Log in	003	User3	employee	3
10	10	System	Logged in	003	User3	employee	3
10	11	Request	Create	003	User3	employee	3
11	12	Request	Approved	002	User2	manager	1
12	13	Request	Send Mail	003	User3	employee	3
17	14	Request	Approved	004	User4	sales	3
18	15	System	Log in	001	User1	manager	4
19	16	System	Logged in	001	User1	manager	4
20	17	Request	Create	001	User1	manager	4
21	18	Request	Approved	001	User1	manager	4
21	19	Request	Send Mail	001	User1	manager	4

1. INTRODUCTION

We propose a way of detecting different types of anomalous behavior in *Business Processes* (BPs). A BP is a series of ordered structured activities in order to perform a task [27]. Such a sequence of events that together form an instantiation of a BP is called a *case* of the business process. In order to monitor a BP, activities are logged in a log file. This file consists of different events and every line in the log file represents a single event. Often log files already indicate which events belong together in the same case. If not, we can apply a clustering algorithm as described in [17] for identifying the different cases. In this paper, we assume that events have already been grouped into cases. Events often contain extra information about the executed activity like duration, resource, documents used, department, sensor-data, ... These extra attributes can be either categorical or numerical.

Example 1. *The log file in Table 1 was generated by a Business Process where an employee needs to log in to a system to create a request. This request is then sent to his or her manager who can approve or reject the request. The log contains 7 attributes: Time, (event)ID, Type, Activity, UID, UName and URole. We also keep track of the case to which an event belongs. In total we have 4 users, each with a unique ID and Name. Every user has a role from a limited set of roles. For the sake of simplicity we have only captured a subset of all possible actions that can or should occur.*

In the context of Business Processes, the detection of anomalous behavior is an important problem. Therefor, in this paper we describe an anomaly detection system that can find deviating cases. This is done by learning the structure and parameters of a model that reflects the normal behavior of a system. Our model does this by taking both the categorical and numerical attributes into account, together with the different relations between the attributes. Attributes in a BP influence each other within events and between different events within a single case, therefor giving useful extra insights in the log file. We also exploit different possible structures of relations between attributes. Our model uses more information than existing techniques from the BP domain [25]. Besides missing or wrongly ordered activities, there can be constraints that enforce that two activities must be performed by the same person or that a person needs to have a certain role to perform an action. Our model captures these relations too.

Diagrams like BPMN [14] are a great tool for human understanding of a Business Process. For applications such as anomaly detection, BPMN models are, however, insufficiently powerful as they lack the ability to easily express joint probability distributions over multiple attributes; they focus on a single perspective (i.e. the resource-activity perspective). Therefore, in order to take advantage of all possible relations between attributes in a log file, we create a model based on Dynamic Bayesian Networks (DBNs) [22]. DBNs are extensions of Bayesian Networks that are able to incorporate the sequential nature of business processes. DBNs link events to their predecessors in order to find relations between these events rather than only relations within one event.

This paper is an extended version of [19]. In [19] we introduced the Extended Dynamic Bayesian Networks which are an extension of the Dynamic Bayesian Networks where we solved some of the shortcomings of DBNs when it comes to modeling the allowable sequences of a business process:

- DBNs are not able to handle unseen values in a way appropriate for business process logs.
- The case where a value always occurs together with another value describes a common structure in log files. We can model these relations in a DBN but only implicit, which may lead to less effective structures.

In this extension we further extended the EDBN in the following way:

- We also take the duration between attributes into account, and are thus able to detect anomalous execution times
- Our model is now able to handle numerical attributes which may depend on other numerical or categorical attributes.

The structure of our paper is as follows. Section 2 describes existing approaches for detecting anomalies in BP logs. Section 3 introduces the model for describing normal behavior in log files. We then use this model in Section 3.5 in order to discover anomalies in BP logs. The learning of the structure of the model is described in Section 4. We evaluate our new method in Section 5.

2. RELATED WORK

The problem we are interested in is that of finding anomalous sequences (cases) within a large database of multivariate sequences (BP logs). Different techniques have been proposed to solve this problem (partially) both in the anomaly detection field [2, 9, 30], as in the process mining field [5, 6, 15, 21]. Some of these techniques use signatures of known anomalies that can occur in the system. It is clear that these systems cannot recognize new types of anomalies and are too limited for our purpose. We are interested in model-based anomaly detection techniques, such as Markov Chains that represent normal behavior of a system.

A first type of algorithms works on a database of univariate sequences; i.e., they only take the activity perspective into account. Bezerra et al. [5] investigated the detection of anomalies in a log file using existing Process Mining algorithms in order to build a model of the process. Then they use conformance checking to detect deviating traces of activities.

Other algorithms work on databases consisting of multivariate sequences. Bertens [2] uses MDL to identify multivariate patterns that help to detect and describe anomalies. A code table consisting of mappings between encodings and frequently occurring patterns is first generated by their algorithm called DITTO [3]. An anomaly score is then defined by the negative log of the fraction of the length of the encoded sequence given the code table over the length of the sequence.

Nolle et al. [15] propose unsupervised anomaly detection methods based on neural networks in business process event logs. They explicitly divide the log in the control flow and a data perspective. These two perspectives are then used as inputs for their neural network that predicts both perspectives for the next event. The use of neural networks makes it possible to reduce the impact of noise in the dataset, where other methods need a training dataset without anomalies as a reference set. The major downsides of this method are that it can only handle a few attributes in the data perspective that is not able to cope with unseen values (both in the activity and data perspective).

Bohmer et al. [6] introduce a probabilistic model that is able to score events in the BP logs. First a *Basic Likelihood Graph* is constructed where all activities are nodes and the edges between nodes indicate the probability that given the previous activity, a certain activity happens next. In the next phase this graph is extended by adding context attributes such as *resource* and *weekday* between two activities that correspond to the resource that performed the previous action on a particular weekday. Using this graph it is possible to compute a baseline-score given the occurrence of a particular activity. This baseline-score is compared to the actual score given to an execution case by the model. To score a case, Bohmer et al. use the data in the graph with the corresponding probabilities. Besides data present in the graph, the model is also able to deal with new values. They do not describe and test the use of more attributes in detail, but their model can be extended in a straightforward way to other attributes as well.

In the field of detecting anomalies in Business Processes not much work has been done on exploiting the temporal perspective. Rogge-Solti et al. [21] proposed a method for detecting temporal anomalies in Business Processes. They propose a Bayesian model that can be inferred from the Petri net representation of a BP. Their main idea for finding

Table 2: Summary of Related Work in comparison with our proposed method

	Multi-Dim	Sequential	Continuous Time	Method
Our method	✓	✓	✓	DBN
Bezerra [5]		✓		Process Mining
Nolle [16]	✓	✓		Neural Networks
Bertens [2]	✓	✓		MDL
Bohmer [6]	✓	✓		Probabilistic Model
Rogge-Solti [21]		✓	✓	Parzen-Windows

outliers in the duration of activities is based on a hypothesis test, in which they determine the probability that a particular observation x was taken from a learned distribution. Their method is based on a method that uses Parzen-Windows for network intrusion detection [31]. The biggest drawbacks of this method are runtime when trying to analyze large log files and the fact that it cannot cope with new activities that occur in the log file.

A summary of the different techniques is found in Table 2.

3. EXTENDED DYNAMIC BAYESIAN NETWORKS

In this section we extend Dynamic Bayesian Networks to create a model which is more flexible and powerful when dealing with log files. To do so we first introduce Bayesian Networks, next we explain how we incorporate the sequential nature of our log file in the model, and finally we discuss the different types of relations that together make up the joint distribution expressed by the Extended Dynamic Bayesian Network.

3.1 (Dynamic) Bayesian Networks

A *(Dynamic) Bayesian Network* ((D)BN) represents a joint distribution for a set of random variables $(X_1, \ldots, X_n)$. It does so by capturing the relations between the random variables. A DBN represents these relations in a *directed acyclic graph* (DAG). Every variable is represented by a vertex in the DAG. When a variable X influences another variable Y there exists an edge in the DAG from X to Y. We call X the *parent* of variable Y, a single variable can have multiple parents. We denote the parents of a node Y as $Pa(Y)$ and the actual values of the parents as $v\_Pa(Y)$. The joint distribution of a BN with variables $X_1, \ldots, X_n$ is:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} (X_i | Pa(X_i)) \qquad (1)$$

The joint distribution thus gets decomposed into different factors, one factor for every variable in the model. This factor can be specified in different ways depending on the type of variable. For our model we use Conditional Dependencies, Functional Dependencies and Kernel Density Estimates.

3.2 K-contextlog

In order to incorporate the sequential aspect of the log in our model we create a *k-contextlog*, where every event also contains the information of its k previous events. But first we formally define a (BP) log.

Definition 1. *We assume that $\mathcal{A} = \{A_1, \ldots, A_n\}$, an ordered set of attributes, is given. For each attribute A_i a set of allowed values $dom(A_i)$ is also given. Furthermore we have $\mathcal{A}_c \subseteq \mathcal{A}$ and $\mathcal{A}_n \subseteq \mathcal{A}$ containing the categorical variables respectively the numerical variables, with $\mathcal{A}_c \cup \mathcal{A}_n = \mathcal{A}$ and $\mathcal{A}_c \cap \mathcal{A}_n = \emptyset$.*
An event e is a triplet $(ID, desc, T)$ with ID a (unique) identifier, desc an event description and T the timestamp of the event. An event description is a tuple $(a_1, \ldots, a_n)$ with $a_i \in dom(A_i)$; $desc.A_i$ denotes a_i. We use $e.A_i$ as a shorthand notation for $e.desc.A_i$ and $e.T$ for the timestamp of the event.
A case $C = \langle e^1, \ldots, e^i \rangle$ is a sequence of events. A log L is a set of cases, where all events in the cases have a different identifier.

We use X (uppercase) for attributes and x (lowercase) to denote a value of attribute X.

The k-contextlog consists, in contrast to the original log, out of the k-contexts of the events, where the k-context of an event is defined as follows:

Definition 2. *The k-context of an event e is a new event containing both the attributes from event e together with the attributes of the k events before e. When there is no such preceding event for e we use the value* None *as a placeholder to indicate that no previous events were present. Alongside the attributes we also keep track of the duration between every two events. We denote the k-context of an event e as $C_k(e)$.*

Example 2. *For the log in Table 1, the* 2-context *of the event with ID 3 is the tuple (System^2, Logged in^2, 001^2, User1^2, employee^2, Request^1, Create^1, 001^1, User1^1, employee^1, Request, Send Mail, 001, User1, employee) and $dur^2 = 0$ and $dur^1 = 1$. We use the subscript i to indicate the i-th timestep before the current event. For the current event we omit these subscripts.*

In the remainder of this paper we always consider the k-contextlog without explicitly talking about which attribute originates from which time step. Since events are only influenced by preceding events we only consider $Y \in Pa(X)$ with Y is not later in time than X.

3.3 Relations between attributes

DBNs model the joint probability over a set of random variables as described in Section 3.1. In our extended model we use three different dependencies for describing the behavior of a BP log. First we introduce the Conditional and

Table 3: CPT based on the example from Table 1

URole	*UName*	$P(UName \mid URole)$
employee	User1	0.5
employee	User3	0.5
manager	User2	1
sales	User4	1

Functional Dependencies which are only used for categorical attributes. Next we introduce Kernel Density Estimations for modeling dependencies between numerical and categorical attributes.

3.3.1 Conditional Dependencies

A DBN models the joint probability distribution over a number of random variables (attributes) by making use of the conditional dependencies between the attributes. These conditional dependencies are represented using Conditional Probability Tables (CPTs). An example of such a CPT is given in Table 3.

Definition 3. *A $CPT(X \mid \mathcal{Y})$ is a table where each row contains the conditional probability for a value of X given a combination of values of $\mathcal{Y}$. There is one row for each combination of values.*

3.3.2 Functional Dependencies

The following example indicates the problems we have when using only Conditional Probabilities for describing BP log files:

Example 3. *Consider the situation where every User has a particular Role and certain activities can only be executed by certain roles. The attribute Role depends on attributes User and Activity in this example. When building a single CPT we have to add a row for every possible combination of values for User and Activity, resulting in a large table with all probabilities equal to 1. Also, when a new user is added to the system, all combinations with this user would have to be added to the CPT.*

This observation certainly is not new, and in the literature several proposals exist to deal with large CPTs on the one hand [4, 13, 11] and new values on the other [8]. In this paper, however, we have chosen to use so-called *Functional Dependencies* (FDs) to deal with these problems. We explain the reasons for this choice after the formal definition of functional dependencies.

Definition 4. *Given a log L, a Functional Dependency $A \rightarrow B$ holds in L if for all events $e, f \in L$ holds that if $e.A = f.A \neq None$, then $e.B = f.B$ for attributes A and B.*

A functional dependency between attributes X and Y can be represented by a function $FD_{X\rightarrow Y} : a\_dom(X) \rightarrow a\_dom(Y)$, $FD_{X\rightarrow Y}(x) = y$, with x and y the respective values for attributes X and Y. The set $a\_dom(A)$ is defined as follows:

Definition 5. *Let L be a log over $\mathcal{A}$ and $\{A_{i_1}, \dots, A_{i_k}\} \subseteq \mathcal{A}(L)$. We define the active domain $a\_dom(A_{i_1}, \dots, A_{i_k}) = \{(e.a_{i_1}, \dots, e.a_{i_k}) \mid \exists C \in L : e \in C\}$ as the set containing all values that occur in the log for the given attributes.*

Example 4. In the log in Table 1, $UID \rightarrow URole$ is a Functional Dependency. Every value of UID maps to a single value of *URole*. A particular value in *URole* can however occur together with multiple values of UID. We have the following mappings in our log:

$$\{001 \mapsto employee,\ 002 \mapsto manager,$$
$$003 \mapsto employee,\ 004 \mapsto sales\}$$

A first benefit of using FDs is that they are well-studied and several highly efficient methods for listing all (approximate) functional dependencies exist [29]. They also ensure a more easy learning phase for the CPTs as some edges are already added by the FDs and should not be examined again by the Bayesian Net learning algorithm. Another benefit is the compactness of the model. Every FD is kept in a separate table, making it also possible to give a better, more detailed explanation of which particular FD has been violated.

Besides FDs we could also choose to use Decision Trees (DTs) [4], Association Rules (ARs) [13] or probabilistic modeling languages (like ProbLog [11]). These first two approaches have the disadvantage that they work on value-level. ProbLog, however, does allow for expressing functional dependencies, thanks to its use of variables. The advantage of FDs as compared to ProbLog is that ProbLog is a general purpose probabilistic modeling language, and learning ProbLog programs is a harder task than learning FDs. An interesting avenue for future work is to mine, next to functional dependencies, other specialized patterns and use ProbLog as a language to express all patterns together and use its powerful inference mechanism to exploit them.

FDs allow for enforcing constraints on unseen values, unlike ARs and DTs. Indeed, suppose that we discover an FD *UID* $\rightarrow$ *UName*. Such rule would allow for spotting the inconsistency of two events with the same *UID* but different *UName*, even if they were never observed before.

3.3.3 Kernel Density Estimations

When we want to extend our model to numerical attributes we can no longer use the Conditional and Functional Dependencies as defined above, as the number of possible values is infinite. Thanks to the decomposability of the total probability we can introduce a new way of scoring the conditional probabilities for numerical attributes without having to change how we score the categorical attributes. The model, however, has to take the type (categorical or numerical) of the attributes into account.

Often numerical methods assume that the data always follows the same kind of distribution (for example a gaussian distribution), however in real-life applications this is often not the case. We therefor use a more general approach.

A method that is able to describe a distribution without making assumptions on how the data looks like is Kernel Density Estimation (KDE) [7]. In our case we use KDE with a gaussian kernel and bandwidth α. We denote it as $KDE(x; X, \alpha)$, where x is the value we want to score, X is the set of values used to build the distribution and α is the bandwidth of the kernels used. The α parameter is determined during the learning phase, as will be explained in

Section 4. One of the advantages of KDE is that it can also be used for multidimensional distributions. As such we can use them to get the conditional probability for a numerical attribute, given its numerical parents, because

$$P(X|Pa(X)) = \frac{P(X, Pa(X))}{P(Pa(X))} \tag{2}$$

where both $P(X, Pa(X))$ and $P(Pa(X))$ can be expressed by KDEs.

Besides numerical parents, a numerical attribute also can have categorical parents. Given Y a set of numerical parents and Z a set of categorical parents, if we want the conditional probability for X given both Y and Z we have:

$$P(X|\ Y, Z) = \frac{P(X \cap Y \cap Z)}{P(Y \cap Z)} \tag{3}$$

Without changing the correctness we can add $P(Z)$ to both the nominator and denominator. We then have:

$$\frac{P(X \cap Y \cap Z)}{P(Y \cap Z)} = \frac{P(X \cap Y \cap Z)}{P(Z)} * \frac{P(Z)}{P(Y \cap Z)} = \frac{P(X \cap Y|Z)}{P(Y|Z)} \tag{4}$$

$P(X \cap Y)$ and $P(Y)$ can be calculated using KDE. The conditioning on Z can be done by partitioning the training data according to Z and train different KDEs for every partition. Next we also train a general version of the KDEs without having a conditioning on Z in case we encounter an unseen combination of values for the categorical parents.

For attribute X with numerical parents $Y_1, \ldots, Y_l$ and categorical parents $Y_{l+1}, \ldots, Y_m$ we then have:

$$P(x|y_1, y_2, \ldots, y_m) = \frac{KDE\big((x, y_1, \ldots, y_l); (X \times Y_1 \times \ldots \times Y_l)\big|_{Y_{l+1},\ldots,Y_m}, \alpha\big)}{KDE\big((y_1, \ldots, y_l); (Y_1 \times \ldots \times Y_l)\big|_{Y_{l+1},\ldots,Y_m}, \alpha\big)} \tag{5}$$

When an attribute has no numerical attributes the KDE in the denominator is set to 1.

An example Kernel Density Estimation for a given set of datapoints can be found in Figure 1.

In the remaining of this article we always use the notation $P(X|Pa(X))$ when referring to the conditional probability for both the categorical and numerical case, although the way of calculating them is different for both types of variables.

3.4 Unseen Categorical Values

BP logs often encounter values that have not been seen before (eg. a new customer or employee). These unseen values do not indicate a possible anomaly in the data as long as they do not brake any of the already known dependencies in the log. When using the CPTs and FDTs as introduced in the previous section we would simply assign 0 to these values. Smoothing can be used to overcome this problem

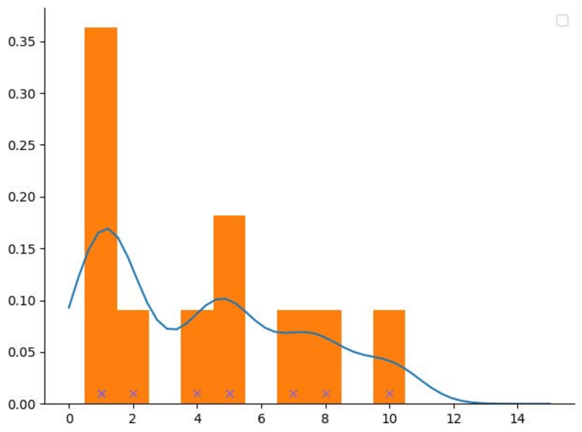


Figure 1: Example of a KDE estimation given the datapoints [1,1,1,1,2,4,5,5,7,8,10]. The crosses indicate the datapoints, the bars indicate the histogram and the curve indicates the estimated density.

but may be inappropriate for attributes that allow for new values to appear frequently. We thus want a way of indicating which attributes are more likely to encounter unseen values. To do so we use a known technique used in the area of Probabilistic Databases (PDBs) as presented by Ceylan et al. [8]. They return an interval of probabilities for a given query that contains known facts (seen values) and unknown facts (unseen values). Instead of calculating for all unseen values a different probability we calculate the probability of encountering a new value in the log for every attribute. When a new value is encountered we simple multiply the score by this probability, when a known value is encountered we multiply by 1 minus the probability. We use the same principle for unseen combinations of parent values for the Categorical CDs as explained in Section 4.

Example 5. *The attribute* URole *will never take a new value as these are fixed within the organization, while* UName *can contain new values when a new user is added to the system.*

3.5 Detecting anomalies

To find anomalous sequences of events we use a score-based approach. The score is obtained by calculating the probability for a case $\langle e^1, \ldots, e^s \rangle$ given a model M. We normalize the result using the s-th root, with s the number of events in the case. This normalization makes sure that longer cases are not penalized.

$$Score(\langle e^1, \ldots, e^s \rangle) = \sqrt[s]{P(\langle e^1, \ldots, e^s \rangle)} \tag{6}$$

Sequences with a high score thus have a high probability of occurring and are most likely to represent normal behavior, whereas low scores indicate higher chances of being an anomaly. We return an ordered list of cases, sorted by their scores. The idea is that a user can only handle the first k anomalies detected. Since we can score any sequence of events, we do not have to wait for a complete case before we can score it. The model can thus be used to detect anomalies in ongoing cases.

4. LEARNING THE MODEL

We build our model using a reference dataset. Ideally this dataset only contains normal executions of the process(es). But we show that our learning algorithm also produces a reliable model when the reference dataset contains a small amount of noise or anomalies. Recall that the timing aspect is incorporated by using the k-contextlog, so we can just assume we are working with a multivariate dataset. We only use the specific sequential aspect for determining the allowed causal relations between attributes, that is no $Y \in Pa(X)$ with Y later in time than X.

All dependencies present in our model should only indicate a causality relation; events in the present cannot influence events in the past. Therefore edges that do not represent a causal relation are blacklisted. This blacklist is created by adding all edges that do not end in the *current* time step.

The complete algorithm can be found in Algorithm 1. In the remaining of this section we explain the important steps in more detail.

```
Function LearnEDBN
    Data: k-log, $\mathcal{A}$, FDThreshold
    Result: The learned EDBN
    FDR = $\{X \rightarrow Y : X \in \mathcal{A}_c^*, Y \in \mathcal{A}_c^0, U(X|Y) > FDThreshold\}$
    blacklist = $\{X \rightarrow Y : X \in \mathcal{A}^i, Y \in \mathcal{A}^j \ with \ i \geq j > 0\}$
    whitelist = FDR
    G(V, E) = LearnBayesianNetwork($\mathcal{A}$, FDR, blacklist)
    $\mathcal{FD}$ = ConstructFDFunctions(FDR)
    $\mathcal{CPT}$ = ConstructCPTables($E \setminus FDR$)
    $\mathcal{CPK}$ = ConstructCPKernels(E)
    NV = $\{X \mapsto \frac{|a\_dom(X)|}{|\ L\ |} : \forall X \in \mathcal{A}_c\}$
    NR = $\{X \mapsto \frac{|a\_dom(Pa(X))|}{|\ L\ |} : \forall X \in \mathcal{A}_c\}$
    VIOL = $\{X \times Y \mapsto \frac{|\{e \in L : FDR_{X \rightarrow Y}(e.X) \neq e.Y\}|}{|L|} : \forall (X,Y) \in FDR\}$
    return EDBN(G(V, E \ FD), FDR, $\mathcal{CPT}$, $\mathcal{CPK}$, $\mathcal{FD}$, NV, NR, VIOL)
```
Algorithm 1: Algorithm for learning the structure and parameters of EDBNs

In a first step the algorithm searches for Functional Dependencies among the categorical attributes. In order to discover them, the Uncertainty Coefficient [20] is applied to all allowed combinations of attributes in the k-contextlog, which is defined as follows for the categorical attributes X and Y:

$$U(X|Y) = \frac{I(X;Y)}{H(X)} \ ,$$

with $H(X)$ the *entropy* [23] of X and I(X;Y) the *Mutual Information* [10] given as:

$$I(X;Y) = \sum_{y \in a\_dom(Y)} \sum_{x \in a\_dom(X)} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

$$H(X) = - \sum_{x \in a\_dom(X)} p(x) log(p(x))$$

The Uncertainty Coefficient is the normalized form of Mutual Information. It gives information about how much the values of an attribute depend on another attribute. We use it to determine what attributes are related to each other and how much they relate to each other. The measure ranges from 0 (no correlation between the two attributes) to 1 (full determination of X by Y, thus indicating the existence of a Functional Dependency) [28]. If $U(X|Y) > threshold$, we will assume that the FD $Y \rightarrow X$ holds. This threshold has to be chosen according to the amount of noise in the data. A higher threshold means a more strict Functional Dependency is used that is less able to cope with noise.

In order to deal with the unseen values for an attribute $A \in \mathcal{A}_c$ we introduce $new\_value(A)$, $new\_relation(A)$ and $violation(X, Y)$ as follows:

$$new\_value(A) : \frac{|a\_dom(A)|}{|\ L\ |},$$

$$new\_relation(A) : \frac{|a\_dom(Parents(A))|}{|\ L\ |}$$

$$violation(X,Y) : \frac{|\{e \in L : FD_{X \rightarrow Y}(e.X) \neq e.Y\}|}{|\ L\ |}$$

This choice reflects the main idea as proposed by Ceylan et al. [8], where they add unseen tuples to the Probabilistic Database, each with a certain probability, possibly depending on the values of other attributes. We consider all unseen values as equally likely and the probability they receive should reflect only the behavior of the attribute itself, therefor we assign to every unseen value of an attribute the probability of encountering a new value for this attribute in the database.

Example 6. *When considering the log from Table 1 the unseen probability for attribute Type is equal to 0.1, indicating that new values are not often encountered in the log. While attribute UID has an unseen probability of 0.2, indicating that this attribute does encounter more unseen values and should therefor less penalize the total score when a new value has to be scored.*

4.1 Learning the Bayesian Network Structure

With a standard Bayesian Network learning algorithm we can discover the Conditional Dependencies present in the data. It is possible to use any learning algorithm that uses data to learn the network structure. We use a greedy algorithm that tries to optimize a model score. This score takes three aspects into account. The first aspect is the prior distribution over all different models, the second is the score for the current found network structure and the last one is a penalty for overcomplicated networks. We thus want a good network that is not overly complex, without this penalty for the structure we would end up with a complete graph. We can drop the prior distribution, who favors more simple models, from the score as we include this aspect in the penalty part of the score. An important property of the model score used is its decomposability. When using a score that can be decomposed into the contribution of every single attribute we can use a different way of scoring categorical and numerical attributes. A categorical attribute can only have categorical parents, whereas a numerical attribute can have both categorical and numerical parents. A numerical attribute thus needs a different way of scoring than a categorical attribute.

Since dependencies between categorical attributes are either conditional or functional we do not want the BN learning algorithm to label already found functional dependencies as conditional. Therefor we add these edges to a whitelist that is past on to the learning algorithm. The learning algorithm should always include the edges from the whitelist in the model. This way the FDs are taken into account for calculating the score of a network but are not added a second time by the algorithm. One possible problem with this approach is that FDs do allow for cycles, but CDs do not. Therefor we consider a cycle of FDs as a single variable in the BN learning algorithm. Since the FDs fully determine each others values we can do this without loosing any information from the data. This way we do not introduce cycles in the whitelisted relations in the BN learning algorithm and are still able to learn dependencies to and from this cycle of FDs. An important constraint on both FDs and CDs is that nodes can only influence nodes in the current time step. Otherwise dependencies between attributes can be accounted for twice when scoring a case.

After running the greedy algorithm we have found the Conditional and Functional Dependencies that define the structures present in our data. We can then combine them into one single model. The next step in building the model is filling in all Conditional Probability Tables, constructing the Functional Dependency functions and learning all the Kernel Density Estimations. As a last step we learn the different unseen values probabilities for all categorical attributes.

To score the quality of a model, we use a combination of likelihood and a penalty term to discourage overly complex models. The total score of a model is the sum of the score for each of the factors.

$$Score(M) = \sum_{A \in \mathcal{A}} Score(A) - penalty(A) \tag{7}$$

We now describe the score for each of the factor types separately.

4.1.1 Categorical Attributes

For the categorical attributes we use the Akaike Information Criterion (AIC) [1] as the score for the current network structure. The score for a categorical attribute is defined as follows:

$$2 * k - 2 * ln(L), \tag{8}$$

with k the number of distinct parent configurations of a variable (the complexity aspect) and L equal to:

$$\sum_{x \in a\_dom(X)} freq(x) * p(x) \tag{9}$$

4.1.2 Numerical Attributes

Recall that for numerical attribtues, we represent the distribution using Kernel Density Estimation. For a numerical attribute A with categorical parents $\mathcal{C}$ and numerical parents $\mathcal{N}$, its score was decomposed as

$$P(A|Pa(A)) = P(A|\ \mathcal{C} \cup \mathcal{N}) = \frac{P(A, \mathcal{N}|\ \mathcal{C})}{P(\mathcal{N}|\ \mathcal{C})} \tag{10}$$

$P(A, \mathcal{C}|\ \mathcal{C})$ and $P(\mathcal{N}|\ \mathcal{C})$ are subsequently represented by a KDE for each instantiation of the categorical attributes $\mathcal{C}$. Hence, for each $\overline{c} \in a\_dom(\mathcal{C})$, we estimate the distribution $p(A, \mathcal{N}|\ \mathcal{C} = \overline{c})$ with a KDE based on the set of numerical values $\{(a, \overline{n})|\ \exists c \in L, \exists e \in \mathcal{C} : e.A = a, e.\mathcal{N} = \overline{n}, e.\mathcal{C} = \overline{c}\}$.

In theory this means that we can never represent $P(A, \mathcal{N}|\ \mathcal{C} = \overline{c})$ more compactly than by enumerating the complete dataset. However, it is possible to sample the found KDE and only use this sample for determining the KDE without loosing much of the accuracy of the KDE. Silverman et al. [24] showed the minimum sample size needed to obtain a good approximation given the dimensionality of the data. Therefor the complexity penalty is independent of the attribute, but only depends on the dimensionality. The series of minimum sample sizes is the following:

$$[4, 19, 67, 223, 768, 2790, 10700, 43700, 187000, 842000] \tag{11}$$

Where the first element denotes the sample size for a distribution of 1 dimension, the second for 2 dimensions, etc. Since we have multiple KDEs when categorical parents are present, we multiply the minimum sample size with the number of categorical parents.

The overal score for attribute A in a log L is hence:

$$\sum_{c \in L} \sum_{e \in c} log\Big(p\big(e.A|Pa(A)\big)\Big) - penalty(A), \tag{12}$$

where $p(e.A|Pa(A))$ is estimated using the KDE. To avoid that $e.A$ itself is used to estimate $p(e.A|Pa(A))$ we can use the leave-one-out cross-validation log-likelihood. But since we are using large datasets to train the model we can use the log-likelihood itself without sacrificing any performance.

5. EXPERIMENTS

We perform several experiments to illustrate the effectiveness of our EDBN algorithm for anomaly detection.

1. We illustrate that our algorithm can achieve **high accuracy** on datasets with **tons of attributes** of **mixed categorical and numerical type**.
2. We **compare** our **KDE-based approach** to deal with numerical attributes with the standard **discretization** approach, showing that KDEs slightly outperform discretization.
3. We compare our EDBN with the **state-of-the-art** techniques proposed by Bohmer et al. [6] and Nolle et al. [15]. We show that we clearly outperform the Likelihood Graphs proposed by Bohmer and on most datasets also performed better than the techniques introduced by Nolle.
4. Lastly, a **qualitative analysis** is performed on the BPI 2018 Challenge. In this experiment we use the EDBN scores to **detect Concept Drift**, and exploit the decomposable nature of our score for **Root Cause Analysis**.

Table 4: Mean AUROC values for different combinations of anomalies for the Synth dataset over 10 runs.

		Test							
	% Anomalies	0.1	0.5	1.0	2.5	5.0	10.0	25.0	50.0
Training	0.0	0.96	0.95	0.93	0.93	0.94	0.94	0.94	0.94
	0.5	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
	1.0	0.87	0.91	0.90	0.90	0.90	0.90	0.90	0.90
	2.5	0.91	0.92	0.90	0.90	0.90	0.90	0.90	0.90

Table 5: Mean AUROC values for different combinations of anomalies for the Synth-duration dataset over 10 runs.

		Test							
	% Anomalies	0.1	0.5	1.0	2.5	5.0	10.0	25.0	50.0
Training	0.0	0.93	0.94	0.93	0.93	0.92	0.93	0.93	0.93
	0.5	0.91	0.92	0.91	0.91	0.92	0.92	0.92	0.92
	1.0	0.88	0.92	0.92	0.91	0.91	0.91	0.90	0.91
	2.5	0.90	0.92	0.91	0.90	0.90	0.90	0.91	0.91

For these experiments we used a multitude of datasets, listed in table 6. In order to fully evaluate our proposed model.

A serious complication in our experimental validation is the absence of large business process logs with labeled anomalies. To alleviate these problems, we use a combination of synthetic datasets, real non-sequential datasets, and one real-life log in which we use our anomaly detection technique indirectly to detect known concept drifts.

All code to reproduce the shown experiments can be found on our GitHub repository[1] (including code used from Nolle et al.).

5.1 Evaluation

In this first part of our experiments we test the basic anomaly detection capabilities (both categorical anomalies and duration anomalies) and want to test how well our model performs when more anomalies are present in the training and test data. We use the area under the Receiver Operating Characteristic curve (AUROC) to measure the quality of the predictions. We choose this method because our model returns an anomaly score for every case, which we then sort from most to least anomalous. The AUROC value gives an indication of the ROC curve itself, without having to plot graphs for every experiment. A score of 1 indicates a perfect curve, meaning no false positives and false negatives occurred during the testing phase, a score of 0.5 means the detection was just random.

We built a data generation tool that allows us to create log files containing different relations between events. In order to do so we first create a model of sequential activities with depending attributes. The model is based on a BP for shipping goods. Goods can have a value and an extra insurance can be taken. Goods with an extra insurance need a different workflow from goods without extra insurance. The data consists of 13 attributes. We create one model for normal execution and one model for anomalous execution, where we explicitly changed the order of events or use the wrong flow of events according to the insurance chosen. Next we introduce extra attributes where some of these attributes depend on other attributes. For the anomalous cases we added random values on random places and changed the order of activities.

We generated multiple set-ups with a variable number of anomalies in both training and test data. We added anomalies in our training data to check and show that our approach does not require a flawless log file as training data but is able to deal with a small amount of unexpected behavior in the data. To minimize the impact of the random generation of the data we run every test 10 times and report the mean AUROC value of all runs. The AUROC-scores for different amounts of anomalies in both training and test data can be found in Table 4. This test shows that our algorithm is able to find the relations mentioned in Section 3, even when the training set contains a small amount of noise or anomalies.

In order to test the detection of duration anomalies we use a synthetic dataset with an easy process, meaning few variations in activities. In order to determine the duration between two events we use Gaussian distributions, where we use unique Gaussians for every combination of activities. Furthermore we also added a user that performed the activity and a random attribute. The results can be found in Table 5. These results show that our model performs evenly well when detecting temporal anomalies as when detecting categorical anomalies in datasets containing multiple attributes.

5.2 Evaluate Numerical Attributes

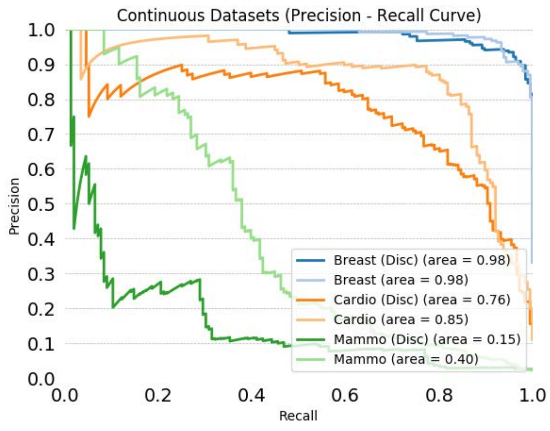
Next we want to test the the performance of our KDEs in comparison with a discretization method. Since there exist no business process log containing both numerical attributes and labeled anomalies, we use three existing non-sequential datasets containing labeled anomalies. These datasets are the Breast Cancer Wisconsin (breast)[2], the Cardiotocogra-

[1]https://github.com/StephenPauwels/edbn

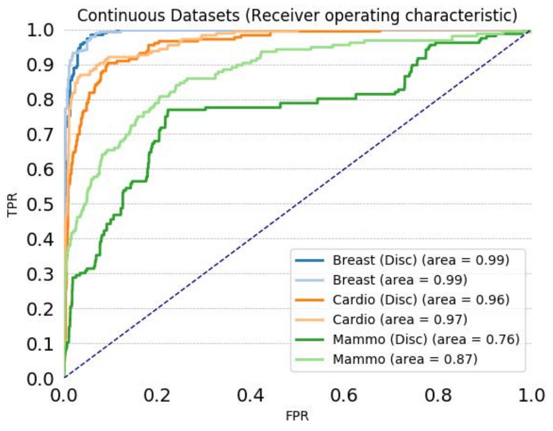
[2]http://odds.cs.stonybrook.edu/breast-cancer-wisconsin-original-dataset/

Table 6: Summary of synthetic datasets used for comparison.

Name	#Activities	#Cases	#Events	#Attr.	#Attr. Values
Synth	6	10,000	50,000 - 60,000	13	2 - 100
Synth-Dur	7	10,000	50,000 - 60,000	4	2 - 1,000
Breast	-	-	683	9	1 - 10
Mammo	-	-	11,183	6	numerical
Cardio	-	-	1,831	21	numerical
P2P	27	5,000	48,477	3	13 - 140
Small	41	5,000	55,058	3	13 - 141
Medium	65	5,000	39,956	3	13 - 140
Large	85	5,000	61,789	3	13 - 141
Huge	109	5,000	46,919	3	13 - 140
Gigantic	152	5,000	38774	3	13 - 141
Wide	63	5,000	39,678	3	13 - 140
BPIC18	41	43,809	2,514,266	22	1 - 225,270



(a) Precision-Recall curve



(b) ROC curve

Figure 2: Results for the continuous datasets

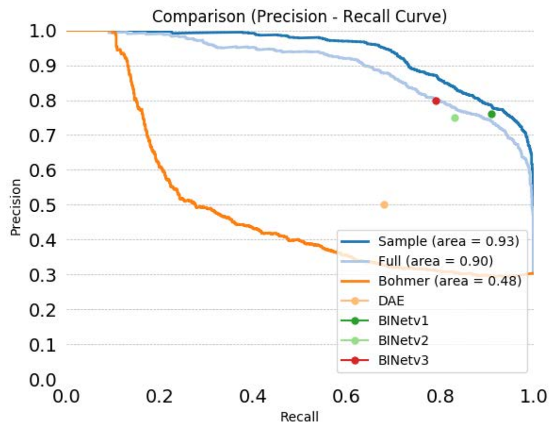
phy (cardio)[3] and Mammography (mammo)[4]. We tested these datasets both using discretization and without. The discretized version generates a model containing only categorical variables, while the original version generates a model containing numerical variables. Figure 2 shows the ROC curve and the Precision-Recall curve for the continuous datasets. We can see that the darker curves (corresponding to the discretized version) are always below the light curves. Meaning that the model using numerical variables performs better in detecting anomalies. The difference for the breast dataset is very small because of only a limited amount of attributes which have only integer values instead of floating point values. Although the KDEs outperform the discretization this experiment does show that even with a discretized approach our EDBN method also performs well.

5.3 Comparison

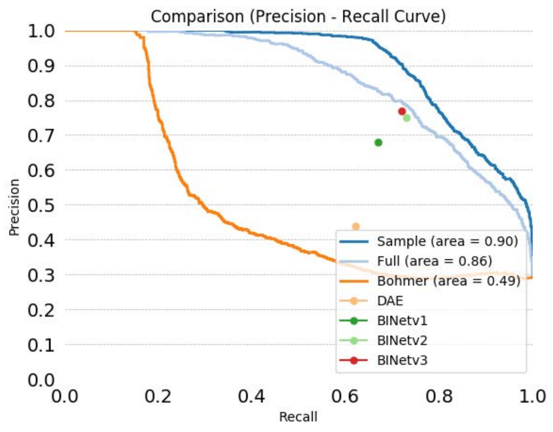
In order to compare with state-of-the-art methods we have chosen to compare with the Likelihood graph [6] and neural network methods [15]. This comparison cannot be performed in a straightforward way. This because every method works in a different manner. The Likelihood graph and neural networks all return for every case if it is an anomaly or not, while our method just returns a score for every case. For the Likelihood graph we were able to easily transform the binary classification into scores indicating how likely it is that the cases are anomalous. The Likelihood graph calculates the likelihood for the ongoing case and compares this with a baseline score in order to indicate if a case is an anomaly. But since the baseline score is based on the last event seen so far we can ignore this baseline because all our cases are considered done and all have the same end activity that indicates the end of the case. The lower the ongoing likelihood the more likely it is that the case contains an anomaly. Due to the way the neural networks work, and their complex way of determining if a trace is anomalous or not we were not able to transform this method. In the comparison we thus will return the Precision-Recall curve for our method and the Likelihood graph, and indicate the precision-recall results of the neural networks as single points in these graphs.

Another difference between the methods is the use of training and testing datasets. The Likelihood graph needs a clean

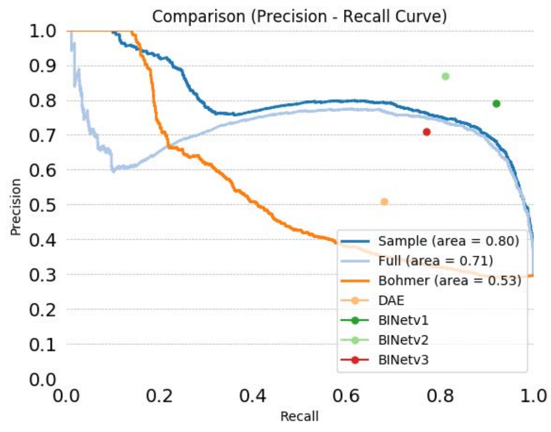
[3]http://odds.cs.stonybrook.edu/cardiotocogrpahy-dataset/
[4]http://odds.cs.stonybrook.edu/mammography-dataset/



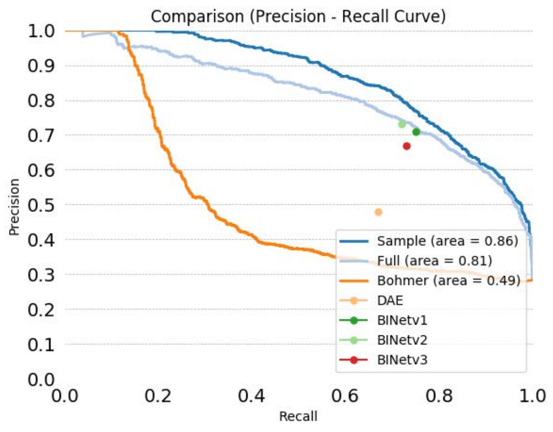
(a) Small



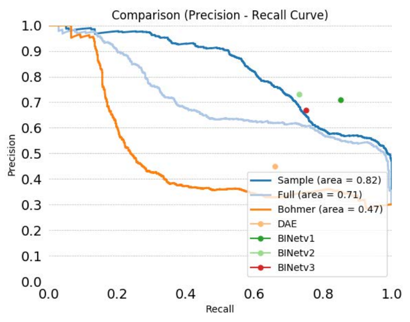
(b) Medium



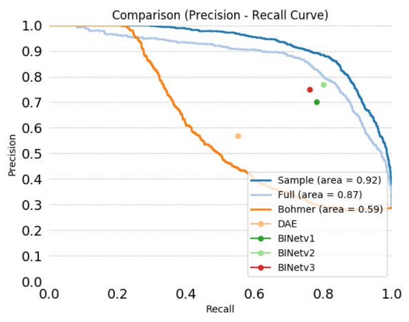
(c) Large



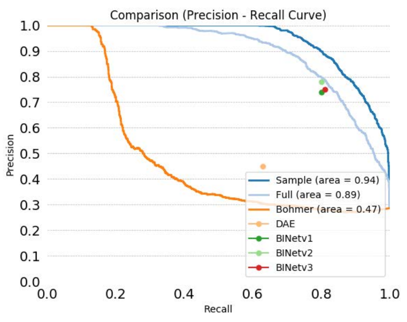
(d) Huge



(e) Gigantic

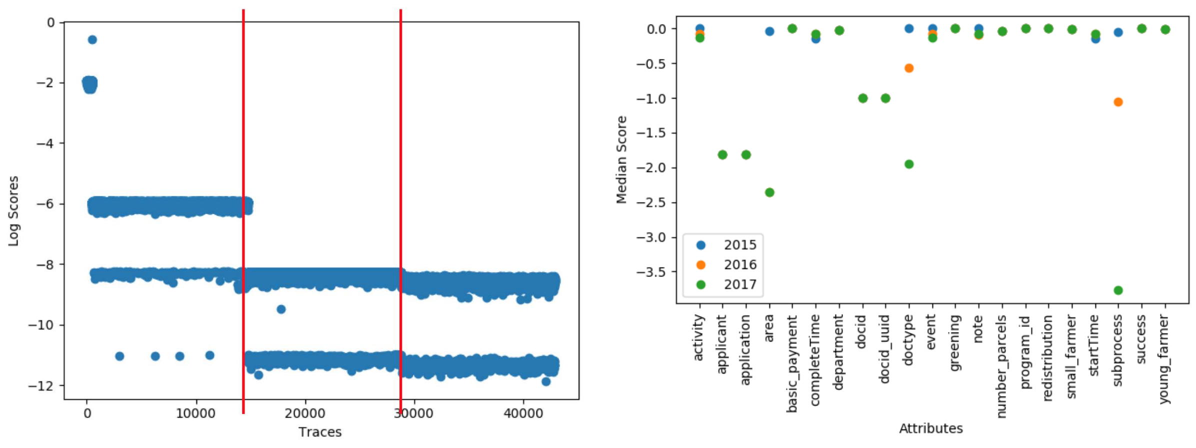


(f) Wide



(g) p2p

Figure 3: Comparison of precision-recall values.



(a) Case scores **(b)** Median values of the decomposed scores for every year

Figure 4: Concept drift detection

training dataset and a different testing dataset, as every case that occurs in the training dataset is assumed to be normal and will never be flagged as anomalous. The neural networks use the same dataset for both training and testing. This mainly because they cannot cope with the occurrence of unseen values. In the previous section we showed that our model is capable of dealing with a training dataset that contains anomalies. In order to be able to compare with both the Likelihood graph and the neural networks we always include two variants of the EDBN results. The first is a variant were we use a clean training set, as for the Likelihood graph. For the second variant we use the same dataset for training and testing, as in the neural network methods.

To prevent any bias towards our own model we used the datasets generated by Nolle et al. [15]. Because the Likelihood Graph was optimized in a setting with 3 attributes (activity, resource, weekday) we use the datasets containing only these three attributes, as described in Table 6. The results for the 7 datasets can be found in Figure 3. Which show that in most datasets we outperform the other algorithms. It also indicates that our algorithm does perform best when a clean training dataset is used, but that the performance does not diminish much when using the full dataset for training. The Likelihood Graphs score worst, indicating that they are not able to deal with more complex dependencies between the attributes and events in a log file.

5.4 Root Cause Analysis in Concept Drift

To further show the usefulness of EDBNs we show how we use them to detect deviations in the form of Concept Drift and use the decomposability of the score to find the cause(s) of the drifts. The data we used is from the BPI Challenge 2018 [26] and consists of applications for agricultural grants over a period of three years. Between the years changes may occur as legislation and documents change. For a more detailed analysis of this data we refer our work in [18]. For testing purposes we remove all information about the date and year events occurred. We only maintain the order in which events and traces occur.

The basic workflow that is being followed is to train the model on the first cases of the data. Next we score all cases in the data using the trained model. Instead of using the formula introduced in Section 3.5 we use the mean score of all event-scores in the case. We made this choice because we want to find the degree of deviation of a case from the reference training set rather than the indication that something is wrong. To detect drifts we plot all these case-scores in a single graph as shown in Figure 4a. Using the Kolmogorov-Smirnov statistical test we can, in detail, determine the possible drift points. The found drift points are indicated with red lines on the graph. To explain the drifts we decompose the scores per attribute and plot the median value for every drift period as in Figure 4b. This graph shows large differences between drift periods in the attributes *area*, *doctype* and *subprocess* which is in line with the expectations that are given as part of the dataset.

This qualitative experiment shows the correctness of the learned model in a more semantic way. And shows that EDBN can be used for solving other problem settings.

6. CONCLUSION

In this paper we further extended Dynamic Bayesian Networks by introducing the use of numerical variables in order to deal with hybrid log files generated by process-aware information systems, that is logfiles containing both categorical and numerical data. We further investigated and solved some of the shortcomings of standard DBNs for analyzing these log files. Next to Conditional Dependencies and Functional Dependencies we added Numerical Dependencies based on Kernel Density Estimation. Next we described our algorithm for creating models that reflect the multidimensional and sequential nature of log files. We conducted different types of experiments: the first experiment confirmed that our algorithm achieves high performance in different settings with different amounts of anomalies in both training and test sets. Next we explicitly tested our model with numerical anomaly datasets, showing that the KDE methods performs better than discretizing the numerical attributes. We also compared our approach with existing solutions where we showed that we were able to perform better than other state-of-the-art methods. Previously proposed methods either needed to train on the same data as the one which was tested (Nolle et al.) or needed to train using a clean training set (Bohmer et al.). In the evaluation we showed that we created a more hybrid method that is both able to learn using a clean or dirty training dataset. Finally we showed the broader range of problems that could be solved using our extended model by taking advantage of the decomposability of the total anomaly score.

7. REFERENCES

[1] H. Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[2] R. Bertens. *Insight Information: from Abstract to Anomaly.* Universiteit Utrecht, 2017.

[3] R. Bertens, J. Vreeken, and A. Siebes. Keeping it short and simple: Summarising complex event sequences with multivariate patterns. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 735–744. ACM, 2016.

[4] A. Beygelzimer, J. Langford, Y. Lifshits, G. Sorkin, and A. Strehl. Conditional probability tree estimation analysis and algorithms. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 51–58. AUAI Press, 2009.

[5] F. Bezerra, J. Wainer, and W. M. van der Aalst. Anomaly detection using process mining. In *Enterprise, business-process and information systems modeling*, pages 149–161. Springer, 2009.

[6] K. Böhmer and S. Rinderle-Ma. Multi-perspective anomaly detection in business process execution events. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 80–98. Springer, 2016.

[7] T. Cacoullos. Estimation of a multivariate density. *Annals of the Institute of Statistical Mathematics*, 18(1):179–189, 1966.

[8] I. I. Ceylan, A. Darwiche, and G. Van den Broeck. Open-world probabilistic databases. In *Description Logics*, 2016.

[9] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection for discrete sequences: A survey. *IEEE TKDE*, 24(5):823–839, 2012.

[10] T. M. Cover and J. A. Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[11] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its application in link discovery. 2007.

[12] R. Hofmann and V. Tresp. Discovering structure in continuous variables using bayesian networks. In *Advances in neural information processing systems*, pages 500–506, 1996.

[13] J. Luo and S. M. Bridges. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. *International Journal of Intelligent Systems*, 15(8):687–703, 2000.

[14] B. P. Model. Notation (bpmn) version 2.0. *OMG Specification, Object Management Group*, pages 22–31, 2011.

[15] T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser. Binet: Multi-perspective business process anomaly classification. *arXiv preprint arXiv:1902.03155*, 2019.

[16] T. Nolle, A. Seeliger, and M. Mühlhäuser. Binet: Multivariate business process anomaly detection using deep learning. In *International Conference on Business Process Management*, pages 271–287. Springer, 2018.

[17] S. Pauwels and T. Calders. Mining multi-dimensional complex log data. *Benelearn*, 2016.

[18] S. Pauwels and T. Calders. Detecting and explaining drifts in yearly grant applications. *arXiv preprint arXiv:1809.05650*, 2018.

[19] S. Pauwels and T. Calders. An anomaly detection technique for business processes based on extended dynamic bayesian networks. In *Proceedings of the 2019 ACM Symposium on Applied Computing*, 2019.

[20] W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

[21] A. Rogge-Solti and G. Kasneci. Temporal anomaly detection in business processes. In *International Conference on Business Process Management*, pages 234–249. Springer, 2014.

[22] S. J. Russell and P. Norvig. Artificial intelligence: a modern approach (3rd edition), 2009.

[23] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, Jan. 2001.

[24] B. W. Silverman. *Density estimation for statistics and data analysis.* Routledge, 2018.

[25] W. M. Van der Aalst and A. K. A. de Medeiros. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science*, 121:3–21, 2005.

[26] B.F. (Boudewijn). Van Dongen and F. (Florian). Borchert. Eindhoven university of technology. dataset., 2018.

[27] M. Von Rosing, H. Von Scheel, and A.-W. Scheer. *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM*, volume 1. Morgan Kaufmann, 2014.

[28] J. White, S. Steingold, and C. Fournelle. Performance metrics for group-detection algorithms. *Proceedings of Interface 2004*, 2004.

[29] H. Yao and H. J. Hamilton. Mining functional dependencies from data. *Data Mining and Knowledge Discovery*, 16(2):197–219, 2008.

[30] N. Ye et al. A markov chain model of temporal behavior for anomaly detection. In *Procs of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, volume 166, page 169. West Point, NY, 2000.

[31] D.-Y. Yeung and C. Chow. Parzen-window network intrusion detectors. In *Object recognition supported by user interaction for service robots*, volume 4, pages 385–388. IEEE, 2002.

ABOUT THE AUTHORS:

Stephen Pauwels is a PhD student in Computer Science at the University of Antwerp (UA) since 2016. He received his Master degree in Computer Science: Data Science cum laude in 2015. Since 2015 he is a Teaching Assistant at the UA in the research group Adrem Data Lab. His research domain consists of analyzing and modeling of multi-dimensional Business Log Files. He won the academic track of the BPI Challenge 2018, held together with BPM 2018, with a report about detecting Concept Drift.

Toon Calders received his PhD in Computer Science in 2003 from the University of Antwerp. In 2006 he joined the Eindhoven University of Technology as an assistant professor, where he left in 2012 to become associate professor at the Université libre de Bruxelles. In 2016 Toon Calders rejoined the University of Antwerp as a full professor. The research interests of Toon Calders are situated in machine learning, data mining, and artificial intelligence. More specifically, he carried out research projects on integrating pattern mining in database systems, on fairness in machine learning, on stream mining, and on dynamic network analysis. He published over 80 papers in data mining and machine learning conferences such as ACM SIGKDD, ECML/PKDD, ACM PODS, IEEE ICDM, ACM WSDM, pVLDB, SIAM SDM and journals ACM Transactions on Database Systems, Machine Learning, and Data Mining and Knowledge Discovery.