An Anomaly Detection Technique for Business Processes based on Extended Dynamic Bayesian Networks

Stephen Pauwels
University of Antwerp
Antwerp
stephen.pauwels@uantwerpen.be

Toon Calders
University of Antwerp
Antwerp
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 attributes. We developed a way to automatically model the behavior captured in log files with dozens of attributes. The advantage of our method is that we do not need any prior knowledge about the data and the attributes. The learned model can then be used to detect anomalous executions in the data. To achieve this we extend the existing Dynamic Bayesian Networks with other (existing) techniques to better model the normal behavior found in log files. We introduce a new algorithm that is able to learn a model of a log file starting from the data itself. The model is capable of scoring events and cases, even when new values or new combinations of values appear in the log file, and has the ability to give a decomposition of the given score, indicating the root cause for the anomalies. Furthermore we show that our model can be used in a more general way for detecting Concept Drift.

CCS CONCEPTS

• **Information systems → Business intelligence**;

KEYWORDS

Anomaly Detection, Probabilistic models, Event log and Workflow data

ACM Reference Format:
Stephen Pauwels and Toon Calders. 2019. An Anomaly Detection Technique for Business Processes based on Extended Dynamic Bayesian Networks. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus* , 8 pages. https://doi.org/10.1145/3297280.3297326

1 INTRODUCTION

We propose a way of detecting anomalous behavior in Business Processes (BPs). A BP is a series of structured activities in order to perform a task [33]. 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 [23] for identifying the different cases.

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

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

Example 1.1. The log file in Table 1 is generated by a Business Process where an employee needs to login 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 consists of 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 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 takes all attributes and relations between attributes into account, in contrast to existing techniques from the Business Process domain [30]. Our model provides us with a lot more useful information since log files created by an autonomous system often consist of many attributes. Attributes can influence each other within an event and between different events. 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.

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04.
https://doi.org/10.1145/3297280.3297326

Diagrams like BPMN [19] 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 and 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) [26]. DBNs are an extension of Bayesian Networks that are able to incorporate discrete time. DBNs link events to their predecessors in order to find relations between these events rather than only relations within one event.

In this paper we identify and alleviate two important shortcomings of DBNs when it comes to modeling the allowable sequences in a log:

- 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.

To achieve this we extend the formalism of Dynamic Bayesian Networks to incorporate the aspects that are typical for log files. We show that the extended Dynamic Bayesian Networks perform well for detecting anomalies.

The structure of our paper is as follows. Section 2 describes existing approaches to this (or similar) problems. Section 3 introduces the model for describing normal behavior in log files. We then use this model in Section 3.4 in order to discover anomalies in cases of events found in log files. The construction of the model is described in Section 4. We will 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 discrete multivariate sequences (BP logs). Different techniques have been proposed to solve this problem both in the anomaly detection field [2, 10, 36], as in the process mining field [5, 6, 21]. Some of these techniques use signatures of known anomalies that can occur in the system. It is clear that these systems cannot recognize a new type of anomaly and are too limited for our purpose. We are interested in techniques that build a model, 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 of multivariate sequences. Bertens [2] uses MDL to identify multivariate patterns that helps 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]. The anomaly score is defined by dividing the length of the encoded sequence given the code table on the whole dataset by the length of the sequence.

	Univariate	Multivariate	Method
Our method		✓	DBN
Bezerra [5]	✓		Process Mining
Nolle [21]		✓	Neural Networks
Bertens [2]		✓	MDL
Bohmer [6]		✓	Probabilistic Model

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

Nolle et al. [21] propose an unsupervised anomaly detection method 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 architecture that predicts both perspectives for the next event. The use of these 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 cannot handle a lot of attributes in the data perspective as the learning phase becomes infeasible and 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 data. 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 with the actual score given to an execution case by the model. To score an actual case, Bohmer et al. use the data in the graph with the corresponding probabilities to score the entire case. 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. A summary of the different techniques can be 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. In order to do so we first formally define a log file.

Definition 3.1. 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.

An event e is a pair $(ID, desc)$ with ID an identifier and desc an event description. 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 $e.desc.A_i$.

A case $C = \langle e^1, \ldots, e^i \rangle$ is a sequence of events. A log L is a set of cases, where events in the cases have different identifiers.

3.1 History and Context of an event

To be able to incorporate the time aspect we introduce the *k-history* and *k-context* of an event.

Definition 3.2. The k-history of an event e^i is defined as a tuple $\mathcal{H}_k(e^i) = (x^k \cdot \ldots \cdot x^1)$ where $\cdot$ denotes concatenation and where

$$x^l = \begin{cases} e^{i-l}.desc & \text{if } i - l > 0 \\ (None_1, \ldots, None_{|\mathcal{A}|}) & otherwise \end{cases}$$

None is a special dedicated value that should not occur in the log. We use $\mathcal{H}_k(e^i).A^l$ to denote the value of attribute A from the l-th event before e^i in the case that is, $x^l.A$.

Definition 3.3. The k-context of an event e is defined as $C_k(e) = (\mathcal{H}_k(e) \cdot e.desc)$. We use the notations:

$$C_k(e).A := e.A$$
$$C_k(e).A^l := \mathcal{H}_k(e).A^l$$

Example 3.4. For the log in Table 1, the 2-history *of the event with ID 3 is the tuple (User-Actions^2, Logged in^2, 001^2, User1^2, employee^2, Request^1, Create Request^1, 001^1, User1^1, employee^1). The* 2-context *of this event is the tuple (User-Actions^2, Logged in^2, 001^2, User1^2, employee^2, Request^1, Create Request^1, 001^1, User1^1, employee^1, Request, Send Mail, 001, User1, employee).* Were we use the subscript i to indicate the i-th timestep before the current event. For the current event we omit these subscripts.

3.2 Conditional Probability Tables and Functional Dependencies

In Dynamic Bayesian Networks, the relations within the model are represented using Conditional Probability Tables (CPTs).

Definition 3.5. A $CPT(X \mid Y)$ is a table where each row contains the conditional probability for a value of X given a combination of values of Y.

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

Example 3.6. 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 the User and the 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 a lot of 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, 12, 18] and new values on the other [9]. In this paper, however, we have chosen to use so-called functional dependencies to deal with these problems. We will explain the reasons for this choice after the formal definition of functional dependencies.

Definition 3.7. Given a log L. A Functional Dependency $A^{t_1} \rightarrow B^{t_2}$ holds in L if for all events $e, f \in L$ holds that if $C(e).A^{t_1} = C(f).A^{t_1} \neq None$, then $C(e).B^{t_2} = C(f).B^{t_2}$ for attributes A and B and time steps t_1 and t_2, with $t_1 \leq t_2$ and t_2 equal to the current timestep.

A Functional Dependency (FD) 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. $a\_dom(A)$ is defined as follows:

Definition 3.8. Let L be a log over $\mathcal{A}$ and $\{A_{i_1}, \ldots, A_{i_k}\} \subseteq \mathcal{A}(L)$. We define the active domain $a\_dom(A_{i_1}, \ldots, A_{i_k}) = \{(e.a_{i_1}, \ldots, e.a_{i_k}) \mid \forall T \in L : e \in T\}$ as the set containing all values that occur in the log for the given attributes.

Example 3.9. 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:

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

A first benefit of using FDs is that they are well-studied and several highly efficient methods for listing all (approximate) functional dependencies exist [35]. They also ensure a more easy learning phase for the CPTs as some relations are already found and should not be checked again. Another benefit is the compactness of the model. A CPT where we set all probabilities to 1 would explode when an attribute has multiple parents. 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] or standard Association Rules (ARs) [18] but lots of these approaches have the disadvantage that they work on value-level. ProbLog [12], 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 its powerful inference mechanism to exploit them.

FDs allow for enforcing constraints on unseen values, unlike ARs and DTs. Indeed, suppose that we discover a FD *user ID* $\rightarrow$ *user name*. Such rule would allow for spotting the inconsistency of two events with the same *user ID* but different *user name*, even if they were never observed before. An unseen value that satisfies all dependencies can be part of a correct event. To overcome the problem of assigning 0 to these values, as CPTs would do, smoothing can be used, but this may be inappropriate for attributes that allow for new values to appear frequently. Therefor we use a known technique used in the area of Probabilistic Databases (PDBs) as presented by Ceylan et al. [9]. They return an interval of probabilities for a given query that contains known facts (seen values) and unknown facts (unseen values). We show in Section 4 that our way of handling these values is closely related to their proposed solution.

3.3 Extending the Dynamic Bayesian Networks

Combining all these elements, we extend the definition of a DBN as follows:

Definition 3.10. An extended DBN (eDBN) with memory k over $\mathcal{A}$ is a tuple:

$(G, FDR, \mathcal{CPT}, \mathcal{FD}, new\_value, new\_relation, violation)$ where:

- $G(V, E)$ is a directed acyclic graph with $V = \mathcal{A}^k \cup \ldots \cup \mathcal{A}^1 \cup \mathcal{A}$ where $\mathcal{A}^i = \{A^i | A \in \mathcal{A}\}$ for $i = 1, \ldots, k$, and $E \subseteq V \times \mathcal{A}$. $\mathcal{A}^i$ represents the attributes of the ith event before the current event.
  E expresses dependencies of the attributes of the current event on the other attributes in its context.
- $FD \subseteq E$ denotes the set of functional dependencies.
- For each variable $A \in \mathcal{A}$, $Parents(A)$ denotes the set of variables $\{B \in V | (B, A) \in E \setminus FD\}$.
- $\mathcal{CPT}$ consists of a Conditional Probability Table $CPT(A|Parents(A))$ for each $A \in \mathcal{A}$
- $\mathcal{FD}$ consists of a Mapping $FD_{A \to B}$ for each $(A, B) \in FD$
- $new\_value(A)$ is a function $\mathcal{A} \to [0, 1]$ representing the probability of encountering an unseen value
- $new\_relation(A)$ is a function $\mathcal{A} \to [0, 1]$ representing the probability of encountering an unseen combination of parent values for the CPT.
- $violation(X, Y)$ is a function $\mathcal{A} \times \mathcal{A} \to [0, 1]$ representing the probability that $FD_{X \to Y}$ is broken.

Figure 1 shows a possible eDBN based on our example.

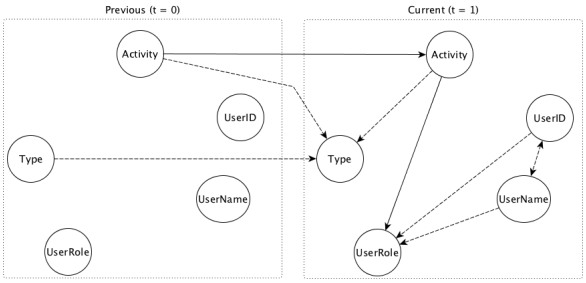


Figure 1: eDBN with conditional (full) and functional dependencies (dotted)

3.3.1 The joint distribution of an eDBN. An eDBN with memory k represents a joint distribution over sequences as follows:

$$P(\langle e^1, \ldots, e^m \rangle) = \prod_{e \in \langle e^1, \ldots, e^m \rangle} P(e | \mathcal{H}_k(e)) \quad (1)$$

$$= \prod_{e \in \langle e^1, \ldots, e^m \rangle} \prod_{A \in \mathcal{A}} P(e.A |\ C_k(e)|_{Parents(A)}) \quad (2)$$

The probability for an attribute in an event consists of three different parts. The first part checks for new values and is defined as:

$$value_A(x) = \begin{cases} 1 - new\_value(A) & \text{if } x \in a\_dom(A) \\ new\_value(A) & \text{otherwise} \end{cases} \quad (3)$$

The probability for the Conditional Dependency is given as follows:

$$Relation(x_i | Parents(X_i)) = \begin{cases} new\_relation(Parents(X_i)) & \text{if new combination of parent values.} \\ (1 - new\_relation(Parents(X_i))* CPT(x_i | Parents(X_i)) & \text{otherwise} \end{cases} \quad (4)$$

The probability for a Functional Dependency is expressed as follows:

$$FDM_{X,Y}(y|x) = \begin{cases} 1 - violation(X, Y) & \text{if } FD_{X \to Y}(x) = y \text{ or } x \notin a\_dom(X) \\ violation(X, Y) & \text{otherwise} \end{cases} \quad (5)$$

To incorporate all the new elements we introduced in our model we extend the way of determining the probability in contrast to original BNs.

$$P(e.A |\ C_k(e)|_{Parents(A)}) = value_A(e.A) \cdot Relation(e.A | Parents(A)) \cdot \prod_{(X,A) \in FDR} FDM_{X \to A}(e.A | C_k(e).X))$$

Example 3.11. The probability for an event e in the model given in Figure 1 is equal to:

$$value(Activity^1) Relation(Activity^1 | Activity^0) value(Type^1) \cdot FDM(Type^1 | Activity^0) FDM(Type^1 | Activity^1) \cdot FDM(Type^1 | Type^0) value(UID^1) FDM(UID^1 | UName^1) \cdot value(UName^1) FDM(UName^1 | UID^1) value(URole^1) \cdot Relation(URole^1 | Activity^1) FDM(URole^1 | UID^1) \cdot FDM(URole^1 | UName^1)$$

The value for the attribute $UserRole^1$ for the event with ID 2 is:

$$value(URole^1) Relation(URole^1 | Activity^1) * FDM(URole^1 | UID^1) FDM(URole^1 | UName^1) = (1 - 0.2) * (1 - 0.4) * 1 * (1 - 0) * (1 - 0) = 0.48$$

This score can be decomposed to find the root cause for any anomaly in the data. This will be further elaborated in future work.

3.4 Anomaly detection

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

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

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 a sorted 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 STRUCTURE AND PARAMETERS OF THE MODEL

We build our model using a reference dataset containing only normal executions of the process. Our experiments show that the performance of our algorithm is, however, not influenced when the dataset contains a small amount of noise. In order to incorporate the timing aspect we replace every event in the log with its *k-context*. We refer to this log as the *k-context log*.

We can use the k-context log as input for traditional Bayesian Network learning algorithms that have no specific knowledge about the different time steps to find the conditional probability tables. Afterwards we interpret the different attributes in their appropriate time slice. The complete algorithm for computing the structure can be found in Algorithm 1.

Function *LearnEDBN*
  Data: variables, FDThreshold
  Result: The learned eDBN
  V = $variables$
  FD = $\{X \rightarrow Y : X, Y \in V, U(X|Y) > FDThreshold\}$
  blacklist = $\{X \rightarrow Y : X \in V_i, Y \in V_j \ with \ i \geq j > 0\}$
  whitelist = FD
  G(V, E) = LearnBayesianNetwork(variables = V, blacklist, whitelist)
  FDM = ConstructFunctionalDependencyFunctions(FD)
  CPT = ConstructConditionalProbabilitiesTables($E \setminus FD$)
  NV = $\{X \mapsto \frac{|a\_dom(X)|}{|L|} : \forall X \in V\}$
  NR = $\{X \mapsto \frac{|a\_dom(Parents(X))|}{|L|} : \forall X \in V\}$
  VIOL = $\{X \times Y \mapsto \frac{|\{e \in L : FD_{X \rightarrow Y}(e.X) \neq e.Y\}|}{|L|} : \forall (X, Y) \in FD\}$
  return *eDBN(G(V, E \ FD), FD, CPT, FDM, NV, NR, VIOL)*

Algorithm 1: Algorithm for learning the structure and parameters of eDBNs

First the algorithm searches for Functional Dependencies in the data. In order to discover them, the Uncertainty Coefficient [25] is applied to the k-context log, which is defined as follows for the random variables X and Y:

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

with $H(X)$ the *entropy* [29] of X and I(X;Y) the *Mutual Information* [11] 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 (completely correlated attributes, thus indicating the existence of a Functional Dependency) [34]. 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.

For an attribute A in log L, $new\_value(A)$, $new\_relation(A)$ and $violation(X, Y)$ are defined 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. [9], where they add unseen tuples to their PDB, 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 every unseen value of an attribute the probability of encountering a new value for this attribute in the database.

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 its structure. We chose to use a Greedy algorithm that finds a local optimum for the Akaike Information Criterion (AIC) [1].

The relations 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.

We do not want the algorithm to find edges already labeled as FDs, therefor we add these edges to a whitelist. The Bayesian Net learning algorithm should always include the edges from the whitelist in the model. This way the learning algorithm takes advantage of the information we already know about these FDs.

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. These steps give us the structure of the eDBN-model. The next step in building the model is filling in all the different Conditional Probability Tables (CPTs) and constructing the Functional Dependency functions for all nodes.

5 EXPERIMENTS

To evaluate our extended model we use two different datasets. The first dataset is a synthetically generated multi-dimensional dataset. The second is the BPI Challenge 2015 (MUNIS) [31] data. This data consists of applications for building permits in 5 Dutch municipalities, we refer to each individual municipality as MUNI1

to MUNI5. Table 3 summarizes all the dataset used in this section. We use the same amount of attributes and anomalies as described by Bohmer et al. [6] to best compare our approaches.

We use the synthetic dataset to test the overall performance of our algorithm, where we try different ratios of anomalies in both training and test set. Next we perform a more in-depth comparison with the Likelihood Graphs proposed by Bohmer et al. [6] using the reduced subset of the MUNIS data. As a last experiment on anomaly detection we compare our approach to a variety of algorithms available in the ELKI - tool [28], using both the synthetic data and the reduced MUNIS data. The ELKI - tool contains most of the existing anomaly detection algorithms in a uniform way. The Area Under the Curve (AUC) is used to compare the algorithms. Finally we add an extra evaluation were we show the usefulness of the model to detect Concept Drift. All code used to perform the experiments and generate the datasets can be found on our GitHub repository.[1].

5.1 Testing with synthetic data

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 some extra attributes where some of these attributes depend on other attributes. For the anomalous cases we added random values on random places. 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 AUC value of all runs.

The AUC-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.

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

		Test							
	% Anomalies	0.1	0.5	1.0	2.5	5.0	10.0	25.0	50.0
Training	0.0	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94
	0.5	0.95	0.89	0.90	0.91	0.91	0.91	0.90	0.91
	1.0	0.95	0.90	0.90	0.90	0.90	0.91	0.91	0.91
	2.5	0.93	0.92	0.90	0.90	0.90	0.91	0.91	0.91

Table 4: AUC values for different combinations of anomalies.

5.2 Comparison

5.2.1 Comparison with Likelihood Graphs. In order to compare our approach to the solution presented by Bohmer et al. we first implemented the algorithm found in [6]. Next we generated data as described by Bohmer et al. starting from the reduced MUNIS data. Therefor we randomly split the original data in two equal data sets, one for training and one for testing. In the test data we introduced anomalies according to the description in Bohmer et al. The statistics for the generated files can be found in Table 5. Normal input will, however, never contain this many anomalies.

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. Since our method works with giving scores and sorting all anomalies, we slightly changed the way we compute the scores for the different activities within a case in order to best capture the ideas of the Likelihood Graphs. The lower the difference the more likely it is that this case contains an anomaly. We use both the precision/recall-curve and the ROC-curve to compare the two approaches. The results can be found in the graphs in Figure 2 and 3 for each of the five different municipalities. Since all five municipalities have different ways of performing the different processes we also created one file containing all data of all municipalities. Then we introduced anomalies in the same way as we did for the other files. This combined dataset allows us to test how well each approach can handle different processes in a single log file.

We can see that our method always outperforms the likelihood graphs using the MUNIS data. Especially in the case were we merged all five datasets. We can conclude that our model is better in scoring the different anomalies in the data, especially when the number of processes present in the data becomes larger.

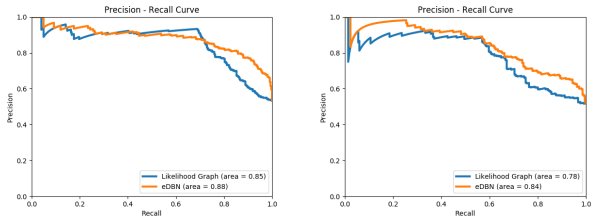
5.2.2 Comparison with other anomaly detection methods. We also tested our method against other anomaly detection methods (not necessarily methods that take into account the sequential or Business Process nature). We used the *k-context* format as input for all the algorithms. The best parameters were chosen after performing some experiments. We performed the experiments using the ELKI -

Dataset	# cases	Average case length	# Activities	# Attributes
Synthetic	10000	6	8	13
MUNI1	1199	43.5	398	3
MUNI2	832	53.3	410	3
MUNI3	1409	42.3	383	3
MUNI4	1053	44.9	359	3
MUNI5	1156	51.0	389	3
GRANTS	43809	57	41	23

Table 3: Description of the BPIC datasets

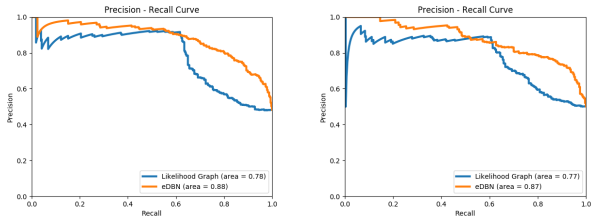
File	Training size	Test size	# Anomalies in Test set
MUNI1	589	610	291 (47.7%)
MUNI2	408	423	214 (50.5%)
MUNI3	723	686	356 (51.8%)
MUNI4	522	530	257 (48.4%)
MUNI5	595	561	283 (50.4%)

Table 5: Number of cases present in the different log files.



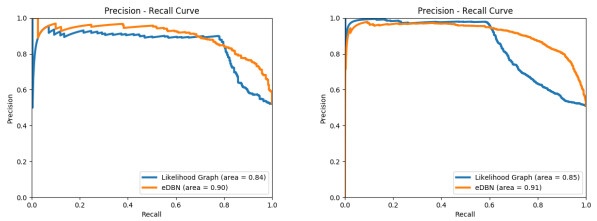
(a) MUNI1

(b) MUNI2



(c) MUNI3

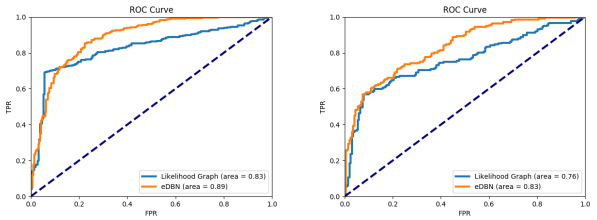
(d) MUNI4



(e) MUNI5

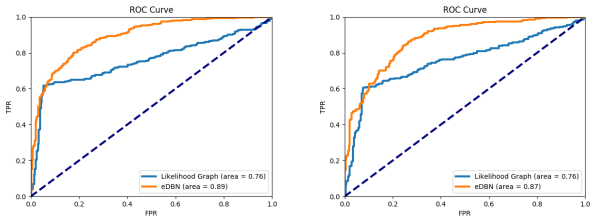
(f) Full MUNIS

Figure 2: Comparison of precision/recall-graphs.



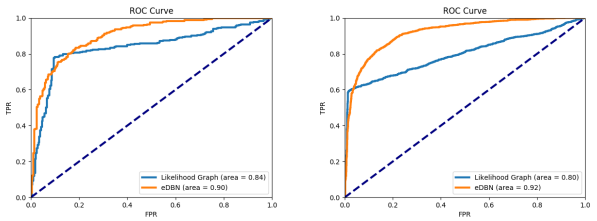
(a) MUNI1

(b) MUNI2



(c) MUNI3

(d) MUNI4



(e) MUNI5

(f) Full MUNIS

Figure 3: Comparison of ROC-graphs.

tool [28]. Since none of these methods uses a different (clean) training dataset we used the same file to generate our model as to test the model. The results can be seen in Table 6. We see that only ALOCI outperforms us on the MUNIS data, this is due to the fact that we used the same file for training and testing. Our method and Bohmer et al. performs best when having a clear training dataset. For some algorithms we were not able to get results for both datasets (due to runtimes and memory usage).

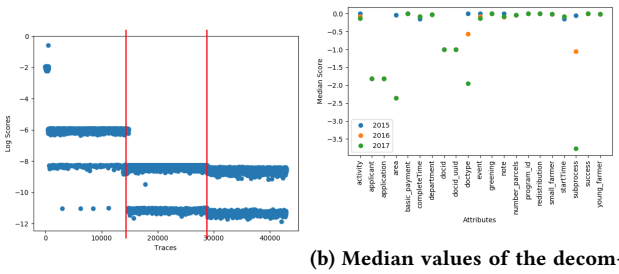
	AUC		
Method	Synth data	MUNIS data	Remark
eDBN	1.00	0.89	no FDs were found for MUNIS
eDBN without FD	0.69	0.83	
Bohmer et al. [6]	1.00	0.85	
FastABOD [15]	0.50	0.56	
LOF [8]	0.49	0.55	
SOD [13]	0.53	0.60	
Feature Bagging [17]	0.75	0.57	
SimpleCOP [37]	0.53	0.60	
LibSVMOneClass [27]	0.51	0.46	
COP [14]	0.81	0.63	
DWOF [20]	0.51	0.44	
OpticSOF [7]	0.53	0.46	
ALOCI [22]	-	0.86	
Bertens et al. [2]	-	-	not able to get a complete run

Table 6: Overview of results for different Anomaly detection techniques.

5.3 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 score to find the cause(s) of the drifts. The data we used is from the BPI Challenge 2018 [32] and consists of applications for agricultural grants over a period of three years. Between the years changes may occur as legislation and documents change. We refer to this dataset as GRANTS. For a more detailed analysis of this data we refer our work in [24].

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 3.4 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.



(a) Case scores

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

Figure 4: Concept drift detection

6 CONCLUSION

In this paper we extended Dynamic Bayesian Networks using other, existing techniques in order to create a new model that allows us to better and in more detail describe the structure and properties of a log file generated by process-aware information systems. As standard DBNs have shortcomings for analyzing these logs we added some elements to cope with these shortcomings. We added Functional Dependencies for a better description of the structure of a log file. Since DBNs cannot cope with unseen values we also improved the way our model deals with these unseen values. 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 compared our approach with existing solutions. Finally we showed the broader range of problems that could be solved using our extended model. In the future we would like to extend our model with other promising (existing) techniques like Inductive Logic Programming [16]. Also the incorporation of the time aspect in our model that can model the duration of activities and gaps between activities is an extension we like to investigate further.

REFERENCES

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

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

[3] Roel Bertens, Jilles Vreeken, and Arno Siebes. 2016. 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*. ACM, 735–744.

[4] Alina Beygelzimer, John Langford, Yuri Lifshits, Gregory Sorkin, and Alex Strehl. 2009. Conditional probability tree estimation analysis and algorithms. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 51–58.

[5] Fábio Bezerra, Jacques Wainer, and Wil MP van der Aalst. 2009. Anomaly detection using process mining. In *Enterprise, business-process and information systems modeling*. Springer, 149–161.

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

[7] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 1999. Optics-of: Identifying local outliers. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 262–270.

[8] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *ACM sigmod record*, Vol. 29. ACM, 93–104.

[9] Ismail Ilkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. 2016. Open-World Probabilistic Databases.. In *Description Logics*.

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

[11] Thomas M Cover and Joy A Thomas. 2012. *Elements of information theory*. John Wiley & Sons.

[12] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. (2007).

[13] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2009. Outlier detection in axis-parallel subspaces of high dimensional data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 831–838.

[14] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. 2012. Outlier detection in arbitrarily oriented subspaces. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE, 379–388.

[15] Hans-Peter Kriegel, Arthur Zimek, et al. 2008. Angle-based outlier detection in high-dimensional data. In *Procs of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 444–452.

[16] Nada Lavrac and Saso Dzeroski. 1994. Inductive Logic Programming.. In *WLP*. Springer, 146–160.

[17] Aleksandar Lazarevic and Vipin Kumar. 2005. Feature bagging for outlier detection. In *Procs of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 157–166.

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

[19] Business Process Model. 2011. Notation (BPMN) version 2.0. *OMG Specification, Object Management Group* (2011), 22–31.

[20] Rana Momtaz, Nesma Mohssen, and Mohammad A Gowayyed. 2013. DWOF: A Robust Density-Based Outlier Detection Approach. In *Iberian Conference on Pattern Recognition and Image Analysis*. Springer, 517–525.

[21] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. 2018. BINet: Multivariate Business Process Anomaly Detection Using Deep Learning. In *International Conference on Business Process Management*. Springer, 271–287.

[22] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B Gibbons, and Christos Faloutsos. 2003. Loci: Fast outlier detection using the local correlation integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*. IEEE, 315–326.

[23] Stephen Pauwels and Toon Calders. 2016. Mining multi-dimensional complex log data. *Benelearn* (2016).

[24] Stephen Pauwels and Toon Calders. 2018. Detecting and Explaining Drifts in Yearly Grant Applications. *arXiv preprint arXiv:1809.05650* (2018).

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

[26] Stuart Jonathan Russell and Peter Norvig. 2009. Artificial intelligence: a modern approach (3rd edition).

[27] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13, 7 (2001), 1443–1471.

[28] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. 2015. A Framework for Clustering Uncertain Data. *PVLDB* 8, 12 (2015), 1976–1979.

[29] C. E. Shannon. 2001. A Mathematical Theory of Communication. *SIGMOBILE Mob. Comput. Commun. Rev.* 5, 1 (Jan. 2001), 3–55.

[30] Wil MP Van der Aalst and Ana Karla A de Medeiros. 2005. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science* 121 (2005), 3–21.

[31] B.F. van Dongen. [n. d.]. (2015) BPI Challenge 2015. Eindhoven University of Technology. https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1.

[32] B.F. (Boudewijn) Van Dongen and F. (Florian) Borchert. 2018. Eindhoven University of Technology. Dataset. https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972

[33] Mark Von Rosing, Henrik Von Scheel, and August-Wilhelm Scheer. 2014. *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM*. Vol. 1. Morgan Kaufmann.

[34] JV White, Sam Steingold, and CG Fournelle. 2004. Performance metrics for group-detection algorithms. *Proceedings of Interface 2004* (2004).

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

[36] Nong Ye et al. 2000. 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*, Vol. 166. West Point, NY, 169.

[37] Arthur Zimek. 2009. Correlation clustering. *ACM SIGKDD Explorations Newsletter* 11, 1 (2009), 53–54.