Generating Process Anomalies with Markov Chains: A Pattern-Driven Approach

Jochem Veldman[1], Xixi Lu[1], Wouter van der Waal[1], Marcus Dees[2], and Inge van de Weerd[1]

[1] Utrecht University, Utrecht, the Netherlands
[2] UWV, the Netherlands

Abstract. Generating anomalies for process executions helps to train anomaly detection methods and evaluate their performance. Anomalous behavior tends to be diverse and very infrequent. Generating process anomalies can help compare detection models and select the suited ones. However, little research has been focused on generating anomalous behavior in a systematic and also stochastic way. In this paper, we built on the idea of training a Markov chain using an event log to capture regular process behavior. We then use a set of predefined *anomaly patterns* to adapt the Markov chain to generate anomalous traces. To evaluate the quality of our generated anomalies, we use them in the downstream task training a detection model. For each pattern, we vary the quantity of injected anomalous traces and their deviation rate. Unsurprisingly, the results show that the models trained with the generated anomalies have a significant improvement in detecting these anomalies. The AUC score increased from 0.63 to reaching a maximum of 0.98 or higher for all three patterns. This confirms our expectation that generating anomalies can help train and evaluate detection models.

1 Introduction

Monitoring business processes and detecting anomalies for early intervention can help prevent compliance issues. Accurate anomaly detection enables the prompt implementation of countermeasures. Anomaly detection is an essential task in data analytics that finds applications across a wide range of industries and is suited for various tasks [1,2]. In addition to the banking sector, anomaly detection is also applied in intrusion detection, bot detection, fake review detection, identifying terrorist activities, and medical diagnosis [3].

To evaluate the performance of these detection methods, it is necessary to have test data with *labeled anomalies*. However, labeling data instances is expensive and time-consuming, especially in the field of anomaly detection. In domains such as healthcare, banking, or insurance, highly trained experts are needed to manually determine if the instance is an anomaly [4]. Furthermore, anomalies are infrequent, leading to *imbalanced data* issues. For example, in most cases, the majority of individuals are legitimate users rather than fraudsters. Consequently, if labeled anomalies are available, they typically account for only a small percentage, such as 1% of the data [5,6,7]. The scarcity of labeled data for evaluation is a pervasive challenge within the scientific community.

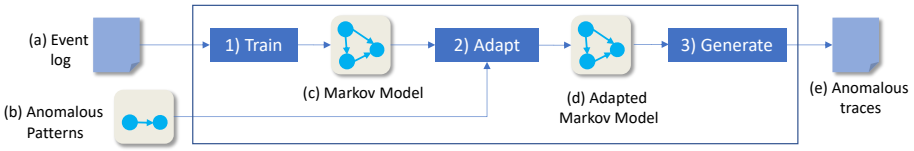

Fig. 1: An overview of the research problem and our proposed approach

Generative approaches have emerged as ways to generate additional data sets [8]. The generated anomalies can then be used to evaluate and compare the performance of detection methods.

In this paper, we propose a lightweight generative approach for process anomaly generation using Markov chains and process anomaly patterns. An overview of the approach is shown in Fig. 1. Assuming we have (a) an event log and (b) a set of patterns that denote potential anomalous behavior of interest. The set of patterns can be obtained by either using domain knowledge, using interactive pattern explorer [9], or selecting the patterns of interest from the repository we built based on a systemic literature review [10].

Given these two inputs, we first train a Markov chain on the input event log, which represents the normative behavior (Step 1). We then use (b) the anomaly patterns to adapt the Markov chain (Step 2). Finally, We use (d) the adapted Markov chain to generate anomalous traces (Step 3).

To evaluate this approach, we inject these generated anomalies into the training datasets to train supervised detection models. However, it is important to note that the test data remains untouched and contains the true anomalies. The underlying reason for this setting is that if the generated process anomalies are of high quality and closely resemble the true anomalies, injecting these generated anomalies should help train two detection models and improve their accuracy in detecting the true anomalies. Following this setting, we demonstrate the approach using a publicly available BPIC dataset. The results show that injecting the generated anomalous sequences helps improve the accuracy of supervised detection models.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. Next, we introduce the preliminaries in Section 3, followed by the approach in Section 4, the evaluation in Section 5, and the results in Section 6. Finally, we conclude the paper in Section 7.

2 Related work

We discuss ways to generate data described in the literature. Below we discuss four different streams of approaches with respect to process anomaly generation.

Earlier studies that propose detection methods tend to either use a random approach to add noise or introduce anomalies in an ad-hoc manner to evaluate the detection performance [11,12]. We argue that our approach complements such random approaches and allows the users to have more control over the generated anomalous traces.

Simulation-based approaches have also been studied and can be used to generate accurate anomalies. However, assuming a set of multiple different anomalous patterns

are of interest, the simulation models have to be manually (re)build or reconfigured. The effort to rebuild such simulation models highly depends on the simulation model/tool used. Therefore, we use the lightweight Markov models and propose (step 2) to adapt the Markov model automatically using the patterns.

Deep models or AI-based generative approaches, such as GAN [13], have also emerged. In [8], the authors have proposed to integrate Deep models with simulations models to generate traces. However, given our problem setting as shown in Fig. 1, users can train GANs on (a) the event log to generate normal traces, but they will have difficulty using the anomalous patterns to adapt GANs or make GANs to generate anomalous traces. Alternatively, one can train GANs on a small set of anomalous traces if such traces are available. Yet, the GANs trained on anomalies may ignore the normative process, whereas our approach largely maintains the distribution in the normative process (represented by the trained Markov model (c)).

3 Preliminaries

In this section, we briefly recall the preliminary concepts related to event logs and Markov chains.

Event logs Let $\mathcal{E}$ be the universe of event identifiers. Let A be a set of activities and $\alpha : \mathcal{E} \rightarrow A$ an event labeling function that returns the activity $\alpha(e)$ of event $e \in \mathcal{E}$. A trace $\sigma = \langle e_1, e_2, \cdots, e_n \rangle \in \mathcal{E}^*$ is a sequence of events. An event log L is a set of traces. We overload the labeling function α with a trace, i.e., $\alpha(\sigma) = \langle \alpha(e_1), \alpha(e_2), \cdots, \alpha(e_n) \rangle$. An example of an event log is listed in Table 1.

Table 1: Example log for Markov chain explanation

Session ID	CustomerID	Activity	Timestamp
Session 1	1	Start_application	2015-11-06 08:07:22.780
Session 1	1	Input_info	2015-11-06 08:07:40.767
Session 1	1	Send_application	2015-11-06 08:07:51.390
Session 1	1	Accept_offer	2015-11-06 08:08:06.003
Session 2	101	Send_application	2016-02-28 08:17:15.947
Session 2	101	Accept_offer	2016-02-28 08:18:31.454
Session 3	224	Input_info	2016-01-14 08:32:11.511
Session 3	224	Send_application	2016-01-14 08:34:12.123
Session 3	224	Accept_offer	2016-01-14 08:37:23.984
Session 4	7653	Start_application	2016-02-20 20:15:10.321
Session 4	7653	Input_info	2016-02-20 20:16:09.647
Session 5	63	Accept_offer	2016-02-20 20:16:09.647

Markov chain We follow the definition of Markov chain in [14]. Let $S = \{s_1, \cdots, s_n\}$ be the set of possible *states* in a Markov chain. For any states s_i, $s_j \in S$, let $\mathbb{P}(s_j \mid s_i) = p_{ij}$ be the *transition probability* from the current state s_i to a subsequent state s_j. A Markov chain is represented by a matrix $T = \{p_{ij}\}$ of transition probabilities, where $p_{ij} = \mathbb{P}(s_j \mid s_i)$, for all $s_i, s_j \in S$. We define a *Markov chain* $M = (S, T)$, where S is the set of states, and T is the transition matrix. For all states $s_i \in S$, $\sum_{s_j \in S} \mathbb{P}(s_j \mid s_i) = 1$. As in [14], we extend the set S with two special states - a start state ($\circ$) and an end state ($\bullet$). For instance, $\mathbb{P}(s_i \mid \circ)$ is the probability of the Markov chain starting in state s_i; $\mathbb{P}(\bullet \mid s_i)$ is the probability of the Markov chain ending in state s_i. Note that $\sum_{s_i \in S} \mathbb{P}(s_i \mid \circ) = 1$, $\sum_{s_i \in S} \mathbb{P}(\circ \mid s_i) = 0$, and $\sum_{s_i \in S} \mathbb{P}(s_i \mid \bullet) = 0$.

4 Approach

4.1 Log to Markov chain

Inspired by existing work on training Markov chain using event logs such as [15], the first step of our approach is to train a first-order Markov chain (S, T) from an event log L, where the states S are the set of activities A with the start and end states $\{\circ, \bullet\}$ added. For each trace in the log, we added a dummy "START" at the beginning of the trace and a dummy "END" at the end of the trace, to correspond to the start and end states, respectively.

To compute the transition probabilities T, we first calculate the frequency matrix of any two consecutive activities, i.e., $Freq(a, b) := \sum_{\sigma \in L} \#\{(e_i, e_{i+1}) \mid \alpha(e_i) = a \wedge \alpha(e_{i+1}) = b \wedge e_i, e_{i+1} \in \sigma\}$. For instance, considering the example log listed in Table 1, we obtain the frequency matrix listed in Table 2. Next, for each row, we compute the row total and divide the value in each cell by the row total. The final matrix of transition probabilities is listed in Table 3. In this paper, we only describe the basic algorithm where the last occurred activity is considered as the state; for a more general algorithm, we refer to [15].

4.2 Process Anomaly Patterns

We define process anomaly patterns as directed, labeled graphs $P = (A', \rightarrow)$. The process anomaly patterns of interest can be obtained by using domain knowledge or using interactive pattern explorer [9]. In addition, to support users creating anomaly patterns of interest, a systematic literature review was conducted to establish a repository of anomaly patterns. In total, 24 domain-specific fraudulent patterns and 14 anomaly patterns in process mining were found. A detailed description of each anomaly pattern is presented in [10]. Each of the domain-specific fraudulent characteristics can be used to create a process anomaly pattern $P = (A', \rightarrow)$ and used further by our approach to generate anomalous traces.

In the following, we explain three examples of process anomaly patterns, selected from the repository [10]. We focus on explaining these three patterns because they are used in the evaluation.

Repetition The repetition pattern is often described in the literature as an anomaly characteristic in process mining [10]. It consists of a single activity that can be performed more or less frequently than expected. This pattern manifests itself as $P = (A', \rightarrow)$

Table 2: Frequency matrix resulted from counting of all consecutive events

	Accept_offer	Input_info	Send_application	Start_application	END	sum
START	1	1	1	2	0	5
Accept_offer	0	0	0	0	4	4
Input_info	0	0	2	0	1	3
Send_application	3	0	0	0	0	3
Start_application	0	2	0	0	0	2

Table 3: Transition probabilities discovered using the example event log

	Accept_offer	Input_info	Send_application	Start_application	END
START	0,2	0,2	0,2	0,4	0
Accept_offer	0	0	0	0	1
Input_info	0	0	0,67	0	0,33
Send_application	1	0	0	0	0
Start_application	0	1	0	0	0

(a) repetition pattern

(b) direct-follow pattern

(c) Advanced repetition pattern

Fig. 2: Examples of the three patterns obtained and used in the evaluation.

where $A' = \{a\}$ consists of a single activity, and $\rightarrow = \{(a, a)\}$. An example of the repetition pattern can be seen in Figure 2a, which is a repetition of "../AANVRAGEN-WW".

Direct-follow The direct-follow relation consists of two consecutively occurring activities in a trace. Note that the directly-follows pattern is directed, i.e., when an activity a is directly followed by another activity b is considered an anomalous pattern, then this does not imply that b directly followed by a is also an anomaly. This pattern manifests itself as $P = (A', \rightarrow)$, where $A' = \{a, b\}$ consists of two activities, and $\rightarrow = \{(a, b)\}$. The direct-follow relation can be seen in Figure 2b.

Advanced repetition The advanced repetition pattern consists of three different activities. Activity a links to activities b and c. Activity b and c link back to activity a. All the links between the activities are direct links. This creates a loop between activity a and b and between activities a and c. We define this pattern as $P = (A', \rightarrow)$, where $A' = \{a, b, c\}$ is the set of three activities, and $\rightarrow = \{(a, b), (b, a), (a, c), (c, a)\}$.

Configured Anomaly Patterns The reason for these simple anomaly patterns is that they can be easily reused and adapted to different contexts where an event log is available. As we would like to use them to generate anomalous traces, the users should be able to configure the *deviation rate* and *the number of generated traces*. We define a configured anomaly pattern $P_{\delta,k} = (A', \rightarrow, \delta, k)$ where $(A', \rightarrow)$ is the pattern, $\delta : (A' \times A') \rightarrow \mathbb{R}$ is the deviation rate that maps each edge in the pattern to a real number, and $k \geq 1$ is the number of traces to be generated.

4.3 Using patterns for adapting Markov chains

Assuming we have a Markov chain (S, T) trained from the event log L and a configured $P_{\delta,k}$, for each $(a, b) \in \rightarrow$, we update the transition probability T by adding $\delta(a, b)$ to $T(a, b)$. After that, we (re)normalize each row such that the transition probabilities of a row sum up to 1.

For example, let us consider the repetition of "SEND_APPLICATION" as a configured anomaly pattern with a deviation rate of 10. Thus, we change this transition probability to T("SEND_APPLICATION", "SEND_APPLICATION") + 10, see Table 4. We overwrite this value with a new value 10. After that, we (re)normalize the probabil-

Table 4: The highlighted transition probability will be changed for the repetition pattern. If the deviation rate is 10, the cell value will be increased by 10.

	Accept_offer	Input_info	Send_application	Start_application	END
START	0,2	0,2	0,2	0,4	0
Accept_offer	0	0	0	0	1
Input_info	0	0	0,67	0	0,33
Send_application	1	0	0	0	0
Start_application	0	1	0	0	0

Table 5: After applying the deviation rate, the transition probabilities of the entire row will be recalculated; the cell value is divided by the row total.

	Accept_offer	Input_info	Send_application	Start_application	END
START	0,2	0,2	0,2	0,4	0
Accept_offer	0	0	0	0	1
Input_info	0	0	0,67	0	0,33
Send_application	0,09	0	0,91	0	0
Start_application	0	1	0	0	0

ities such that the sum of the probabilities at each row is 1, which resulted in 0.91, see Table 5.

4.4 Generating Anomalies using Markov chain

In step (3), we *generate* anomalous traces using the configured anomaly pattern and the adapted Markov chain as our inputs. We briefly sketch the algorithm. It begins with the starting activity. From the "START" row, the activities and their corresponding probabilities are considered. The follow-up activity is chosen randomly based on the given probabilities.

For example, assuming the adapted Markov chain in Table 5, this would mean that there is 20% that "INPUT_INFO" will be selected as the start activity. If this activity is indeed selected as the starting activity, an "INPUT_INFO" event is appended to a new trace. The algorithm then continues with the row "INPUT_INFO" and considers the subsequent activities and their probabilities, i.e., "SEND_APPLICATION" with a 67% probability and "END" with a 33% probability. Let's assume that the function selects "END" as the consecutive activity, then an event of "END" is appended to the trace. The activity "END" signifies that the generated sequence has reached its final state and is considered complete.

To ensure the generated trace is relevant while maintaining the stochastic nature, we check if the trace at least contains the activities of the anomaly pattern. If the trace meets this condition, it is added to the set of generated anomalous traces until the desired *number of generated anomalous traces* k has been reached.

5 Evaluation

We implemented the proposed approach in Python (version 3.9.7). For Python, we used Pandas, Numpy, and Sci-kit learn as our main libraries. The implementation was done in Jupyter Notebook and made publicly available[3]. The objective is to (1) show that our approach can used to generate anomalous traces and (2) investigate the effect on the existing detection methods when injecting the generated anomalous traces into the training set.

[3] https://github.com/JochemVeldmanUU/Thesis_anomaly_generation

5.1 Data description

We use a public dataset from the BPIC 2016 [16] which is provided by the dutch organization Uitvoeringsinstituut Werknemersverzekeringen (UWV). The UWV is a governmental organization responsible for employee-related social benefits in the Netherlands. In this evaluation, we use the click data of visits to the UWV website for logged-in clients. The original dataset contains no labels.

The event log provided covers the period from 6 November 2015 to 28 February 2016. The data consists of 7,174,934 rows and has a size of 1.06GB. A total of 781 unique activities (i.e., web pages) have occurred in 660,270 sessions (traces). The average number of activities per trace is 10.87, covering 26,647 unique users. The variables that have our interest are CustomerID, SessionID, TIMESTAMP, and URL_FILE. We use the session id as the case id and the URL_FILE as the activity. In addition, we selected a set of 50,000 traces due to the capacity limitation of the laptop used.

5.2 Interviews for Labeling Anomalies

To evaluate the quality of the generated process anomalies, we obtain the labels of true anomalies via interviews with the domain expert. It is important to note that this step to obtain labels is not a step in our approach (see Fig. 1); it is only needed for the evaluation we designed.

Two interviews were held with the UWV domain expert who provided the dataset. We obtained the three concrete rules to label anomalous traces. For each rule, we create a separate class label.

The first rule follows the repetition pattern. After the interview, it is concluded that when a trace includes more than fifty repetitions of the same activity "../AANVRAGEN-WW", then the trace is considered an anomaly, with no other activities visited in between. This labeling of the data results in 465 traces that are marked as an anomaly. The other traces are labeled as normal instances.

The second rule follows the direct-follow pattern. The labeling rule is decided that a trace is an anomaly if it has more than one occurrence of a direct-follows relation from activity "/XPSIMAGE/WDO212395" to activity "../AANVRAGEN-WW". This results in 87 traces that are labeled as anomalies.

The third rule follows the advanced repetition pattern. The labeling rule is defined as follows: a trace is an anomaly if the trace contains more than three non-consecutive executions of activity "../HOME/SERVICES/REQUESTS-WW". This results in 352 labeled anomalies, and the rest are labeled as normal instances.

5.3 Setup

After obtaining the labeled event log with true anomalies, we use this data set to evaluate our approach to generate anomalies. Assuming we do not have the labeling rules but only the following three anomaly patterns: (1) the repetition of the activity "../AANVRAGEN_WW", (2) the directly-follows pattern from the "/XPSIMAGE/WDO212395" activity to the "../AANVRAGEN_WW" activity, and (3) the advanced repetition pattern

between "../AANVRAGEN_WW", "HOME" and "DIENSTEN/OVERZICHT". The three patterns are shown in Figure 2.

We follow a standard pipeline for training and evaluating a supervised machine-learning model to detect anomalies. In the following, we explain the setup.

Generate anomalies We use our approach to generate anomalies. The three patterns are matched to the Markov model and used to manipulate the probability of the corresponding transition probabilities. For the *deviation rate* δ, we use values of $\{0, 0.2, 1, 5, 25\}$ for all relations in δ. For each value, we recalculate the probability such that the total probabilities sum up to one. Using the adjusted Markov model, we apply our approach and generate the anomalous traces. For the number of anomalous traces injected, we set k to the values of $\{0, 10, 100, 500, 1000, 2000\}$.

Trace encoding For encoding the traces to fit the detection technique, we use the simple bag-of-activities encoding, i.e., the activities as the features. For each trace, for each activity, we encode the occurrence frequency of the activity as the feature value.

We split the normal traces randomly into a training set (80% of the original) and a test set (20% of the original). From the true anomaly traces, *10* traces are included in the training data, to mimic the real-life imbalanced data issue with very few anomalous instances labeled. The rest of the true anomaly traces are added to the test data. Next, the generated anomalies are added to the training dataset at different rates. The test set is unchanged.

Training and evaluating the detection model We use a logistic regression model with default parameters and later a decision tree as the supervised detection methods. The max iteration is set to 1000. We evaluate the performance of the detection model using AUC, recall, and precision.

6 Results and Discussion

This section presents the obtained results. As aforementioned, the objective is to show that the approach helps to generate anomalies and to investigate the effect on the downstream anomaly detection techniques. The underlying rationale is that *if the generated process anomalies closely resemble true anomalous behavior*, then injecting these generated anomalies into the training data should lead to *improved detection accuracy* for the detection methods. This improvement is expected since we are to some extent leaking information to the detection methods. In the following, we examine the AUC performance of the detection model under different settings of δ and k.

6.1 AUC scores

Repetition pattern Table 6 lists the attained AUC scores for the repetition pattern. The column names represent the *deviation rate*, while the row names indicate the number of injected anomalous traces. The first row represents the baseline AUC score, thus without injecting the generated anomalies.

As can be seen in Table 6, the AUC score demonstrates a notable increase from 0.666 to 0.995, with an improvement of 0.329. In the setting where the deviation rate

Table 6: The AUC scores of the detection model trained with varying settings for the repetition pattern.

Number of anomalous traces injected	AUC score Deviation rate = 0	AUC score Deviation rate = 0.2	AUC score Deviation rate = 1	AUC score Deviation rate = 5	AUC score Deviation rate = 25
0	0.666	0.666	0.666	0.666	0.666
10	0.662	0.649	0.680	0.698	0.670
100	0.796	0.831	0.859	0.832	0.760
500	0.998	0.987	0.984	0.954	0.880
1000	0.995	0.997	0.993	0.978	0.917
2000	0.994	0.994	0.995	0.992	0.953

Table 7: The AUC scores of the model trained with varying settings for the direct-follow pattern.

Number of anomalous traces injected	AUC score deviation rate = 0	AUC score deviation rate = 0.2	AUC score deviation rate = 1	AUC score deviation rate = 5	AUC score deviation rate = 25
0	0.623	0.623	0.623	0.623	0.623
10	0.636	0.636	0.649	0.649	0.688
100	0.630	0.643	0.753	0.773	0.708
500	0.817	0.869	0.914	0.940	0.940
1000	0.907	0.946	0.972	0.998	0.985
2000	0.944	0.976	0.995	0.996	0.996

Table 8: The AUC scores for the advanced repetition pattern.

Number of anomalous traces injected	AUC score deviation rate = 0	AUC score deviation rate = 0.2	AUC score deviation rate = 1	AUC score deviation rate = 5	AUC score deviation rate = 25
0	0.577	0.577	0.577	0.577	0.577
10	0.569	0.572	0.560	0.566	0.556
100	0.621	0.607	0.588	0.577	0.560
500	0.823	0.640	0.612	0.596	0.570
1000	0.936	0.670	0.620	0.597	0.580
2000	0.980	0.710	0.636	0.613	0.572

Table 9: The optimal AUC, precision, and recall scores of our approach (M) compared with the oversampling technique (SO).

	AUC		Precision		Recall		F1	
Pattern	OS	M	OS	M	SO	M	SO	M
Repetition	0.888	**0.995**	**0.99**	0.93	0.78	**0.99**	0.87	**0.96**
Direct-follow	0.870	**0.996**	**0.89**	0.72	0.74	**0.83**	**0.81**	0.77
Advanced repetition	0.757	**0.980**	**0.92**	0.76	0.51	**0.97**	0.66	**0.85**

is 1 and 2000 anomalous traces are injected, the detection model achieves its highest AUC score. On average, injecting 2000 anomalous traces with different deviation rates yields an AUC improvement of 0.320 compared to the baseline. Interestingly, setting the deviation rate to a high value of 25 yields a less significant improvement.

Direct-follow pattern Table 7 shows the AUC scores attained for the direct-follow pattern. The AUC score for this pattern also shows a significant improvement of 0.373, reaching 0.996. This improvement is obtained when the deviation rate is set to 5 or 25. These two settings also achieve the highest AUC score. On average, these settings in the table achieve an increase of 0.358 in the AUC scores.

Advanced repetition pattern Table 8 lists the AUC scores attained for the advanced repetition pattern. The AUC scores for this pattern also show an increase of 0.403 (from 0.577 to 0.980), where the deviation rate is set to 0 and the number of injected anomalous traces to 2000. Interestingly, the performance improvement differs significantly between the deviation rate of 0 and the other deviation rates. In contrast to the previous patterns, the increase in AUC scores diminishes as the deviation rate is set to higher values. For instance, when the deviation rate is set to 25, four out of five obtained AUC scores are even lower than the baseline AUC of 0.577.

Overall, the improvements in AUC are as we expected. In addition, our observations indicate that when the deviation rate is set to a relatively low value, injecting a large number of generated anomalous traces significantly enhances the detection model's AUC performance, compared to not injecting any traces. We also computed the recall and precision, which exhibit similar trends but slightly less significant improvements compared to the AUC scores. The optimal recall and precision scores for each pattern are listed in Table 9.

6.2 Comparative experiments

We compare our results to a sampling approach. We use the training set that contains the true anomalies but no generated anomalies and the unchanged test set and used an oversampling technique. Additionally, we also use another detection model, i.e., decision tree, to investigate the generalizability of the effects.

Comparing to oversampling For the implementation of the oversampling method, we used the RandomOverSampler from the Imbalanced learn library. This function oversamples the minority class by randomly picking samples with replacements. The original training set includes 39,957 normal traces and *10* true anomaly traces. The oversampling function samples these 10 traces and increases the number of anomalous traces to 39,957 to balance the classes. The resulting oversampled training set was used to train a logistic regression model, which was subsequently evaluated using the test set.

Table 9 presents the results attained by the logistic regression trained using the oversampled data. The highest scores are highlighted in bold for comparison with our generative approach. When comparing oversampling with the generative approach, we observe that oversampling achieves a higher precision score but a lower recall score. However, when considering the F1 score, we see that our generative approach achieves a higher score for the repetition and advanced repetition scores. On the other hand, the oversampling technique achieves a better F1 score for the direct-follow pattern.

Considering the AUC scores, it is evident that the generative approach achieves better AUC scores for all three patterns compared to the oversampling technique. Hence, these findings suggest that injecting the anomalous traces helps the detection methods in effectively distinguishing between normal and abnormal behavior, leading to a notable increase in AUC scores. This result is also expected since we are using more complex models (i.e., the Markov chains) to capture the anomalous behavior than just sampling. It is worth noting that the lower precision or F1 score observed may be attributed to the cut-off threshold set for classifying anomalies in the detection methods. Adjusting the anomaly threshold could potentially improve the precision and F1 score.

Using the decision tree model We conducted a similar experiment using a Decision Tree model instead of a Logistic Regression model. The improvements in the performance of the Decision Tree model were less significant compared to the Logistic Regression model. However, the results still exhibited the same trend observed in the Logistic Regression model. In the AUC curves, we observed that increasing the number of injected anomalous traces led to improved performance. Additionally, increasing the deviation rate also resulted in a slight increase in performance, although this increase differed from that observed in the Logistic Regression model. Notably, the Decision Tree model performed well for the advanced repetition pattern when the deviation rate was increased, which was not the case for the Logistic Regression model.

6.3 Discussion

The evaluation results have shown that generating and injecting the anomalies helps to improve the detection accuracy significantly. For the repetition pattern, our approach helps to achieve an increase of an AUC from 0.666 to 0.995 and an increase in the F1

score from 0.50 to 0.96. For the direct-follow pattern, the AUC increases from 0.623 to 0.996, and the F1 score from 0.39 to 0.77. For the advanced repetition pattern, the injection of anomalies did not help as much as for the other two patterns. It only achieves good results when the deviation rate is 0. For this deviation rate, we achieve an increase in AUC from 0.577 to 0.980 and in F1 score from 0.26 to 0.85.

We observed that injecting more generated anomalies leads to an increase in both the AUC score and recall, while the precision decreases for all three patterns. However, the high AUC scores indicate that the detection model effectively distinguishes anomalies from normal behavior. This suggests that the cutoff threshold could be further optimized, for example, through the use of cross-validation.

Compared to injecting more generated traces, increasing the deviation rate has a less significant impact on the AUC scores obtained by the models for the repetition and direct-follow patterns. The resulting AUC scores related to these two patterns tend to converge to a similar value across different configurations with varying deviation rates. However, for the advanced repetition pattern, we observe a decrease in AUC when the deviation rate is increased. We attribute this to the complexity of the pattern and the simplicity of our features and detection model, which may not accurately learn the decision boundary. In future work, we suggest experimenting with other trace encoding techniques, such as encoding the transitions between activities.

A side effect that arose during the evaluation is the iterative nature of our pattern-driven approach, which uses anomaly patterns as a starting point and supports inspecting the potential anomalies and refining the patterns. During the first interview, we identified the three general anomaly patterns of interest. After the first interview, these patterns are instantiated with the concreted activities that occurred in the data set and analyzed. During the second interview, the instances that follow the patterns are discussed with the experts and used to decide on concrete labeling rules. This iterative process allowed for a collaborative and informed approach to refining the anomaly patterns based on expert insights, making this approach suited for domain-specific or data-specific adaptions.

We discuss the following limitations. (1) The anomaly patterns we investigated in the evaluation are rather simplistic. Our definition of process anomaly patterns is however general and can capture more complex patterns. Moreover, the Markov models can be extended with more refined state abstraction. We consider this study as a first step that confirms the potential of this direction. (2) The detection methods and the trace encoding used in the evaluation are simple, classic machine-learning techniques. For future work, we should investigate the effect of injecting generated anomalies on more advanced trace encoding and detection techniques. (3) One may argue that if the anomaly patterns are known, one can use the patterns to detect anomalies, which we agree with. However, our approach aims to generate anomalies, which can also be used to evaluate and compare the detection methods.

7 Conclusion

In this paper, we have presented a light-weighted approach for generating stochastic process anomalies using Markov models. We use the anomaly patterns to adapt the

Markov model. The adapted Markov model is used to generate anomaly traces, leveraging the stochastic nature of Markov models. Our evaluation demonstrates that injecting these generated anomalous traces improves the AUC of the detection models, achieving a 0.98 or higher, which confirms our expectation. This suggests that the generated anomalies can be used to further train or evaluate the detection methods.

For future research, an interesting direction would be to explore the diverse ways of training the Markov chain to capture data attributes or timestamps, expanding the scope of anomaly generation in process mining.

References

1. M. Ahmed, A. Mahmood, and M. Islam, "A survey of anomaly detection techniques in financial domain," *Future Gener. Comput. Syst.*, vol. 55, pp. 278–288, 2016.
2. M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS one*, vol. 11, no. 4, p. e0152173, 2016.
3. S. Alla and S. Adari, *Beginning anomaly detection using python-based deep learning*. Springer, 2019.
4. N. Tax, K. de Vries, M. de Jong, N. Dosoula, B. van den Akker, J. Smith, O. Thuong, and L. Bernardi, "Machine learning for fraud detection in e-commerce: A research agenda," *CoRR*, vol. abs/2107.01979, 2021.
5. G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 38:1–38:38, 2021.
6. A. Adewumi and A. Akinyelu, "A survey of machine-learning and nature-inspired based credit card fraud detection techniques," *Int. J. Syst. Assur. Eng. Manag.*, vol. 8, no. 2s, pp. 937–953, 2017.
7. I. Achituve, S. Kraus, and J. Goldberger, "Interpretable online banking fraud detection based on hierarchical attention mechanism," in *29th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2019, Pittsburgh, PA, USA, October 13-16, 2019*. IEEE, 2019, pp. 1–6.
8. M. Camargo, M. Dumas, and O. G. Rojas, "Discovering generative models from event logs: data-driven simulation vs deep learning," *PeerJ Comput. Sci.*, vol. 7, p. e577, 2021.
9. M. Vazifehdoostirani and et. al., "Interactive multi-interest process pattern discovery," in *BPM 2023*, ser. LNCS, vol. (accepted). Springer, 2023.
10. J. Veldman, "Generating process anomalies using a taxonomy of fraud characteristics and markov models for accurate detection," Master's thesis, Utrecht University, 2022. [Online]. Available: https://studenttheses.uu.nl/handle/20.500.12932/42635
11. M. Alizadeh, X. Lu, D. Fahland, N. Zannone, and W. van der Aalst, "Linking data and process perspectives for conformance analysis," *Comput. Secur.*, vol. 73, pp. 172–193, 2018.
12. X. Lu, D. Fahland, F. J. H. M. van den Biggelaar, and W. M. P. van der Aalst, "Detecting deviating behaviors without models," in *BPM Workshops*, ser. LNBIP, vol. 256. Springer, 2015, pp. 126–139.
13. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014, pp. 2672–2680.
14. D. R. Ferreira and D. Gillblad, "Discovering process models from unlabelled event logs," in *BPM*, ser. Lecture Notes in Computer Science, vol. 5701. Springer, 2009, pp. 143–158.
15. W. M. P. van der Aalst, M. H. Schonenberg, and M. Song, "Time prediction based on process mining," *Inf. Syst.*, vol. 36, no. 2, pp. 450–475, 2011.
16. M. Dees and B. B. van Dongen, "BPI Challenge 2016," 2016.