A Real-Time Method for Detecting Temporary Process Variants in Event Log Data

Sudhanshu Chouhan[1(✉)], Anna Wilbik[2], and Remco Dijkman[1]

[1] Eindhoven University of Technology, Eindhoven 5612 AZ, The Netherlands
{s.g.r.chouhan,r.m.dijkman}@tue.nl
[2] Maastricht University, Maastricht 6229 GT, The Netherlands
a.wilbik@maastrichtuniversity.nl

Abstract. During the execution of a business process, organizations or individual employees may introduce mistakes, as well as temporary or permanent changes to the process. Such mistakes and changes in the process can introduce anomalies and deviations in the event logs, which in turn introduce temporary and periodic process variants. Early identification of such deviations from the most common types of cases can help an organization to act on them. Keeping this problem in focus, we developed a method that can discover temporary and periodic changes to processes in event log data in real-time. The method classifies cases into common, periodic, temporary, and anomalous cases. The proposed method is evaluated using synthetic and real-world data with promising results.

Keywords: Process discovery · Fuzzy clustering · Process variant

1 Introduction

In flexible business processes, such as in hospitals and administrative offices, the executions of the activities are not always according to the defined process. In such processes, it is possible that the workplace employees deviate from the defined process and follow a different process per case. It is also possible that for a certain period of time they deviate from the defined process for most cases. For example, the employees may temporarily skip some process steps when there is a high workload. When the workload goes back to normal, they follow the normal process again. This temporary deviation from the defined process may cause temporal deviations in the event log data. Another example is that the rules and regulations pertaining to the processes may change with time, which can lead to a permanent shift in the way in which the process is normally executed. This permanent shift may induce a persistent deviation in the event log data. It may be interesting to remark that what is "normal" is usually not exactly clear, because there may be frequent deviations from the defined process flow as

A. Polyvyanyy et al. (Eds.): BPM 2021, LNCS 12875, pp. 197–214, 2021.
https://doi.org/10.1007/978-3-030-85469-0_14

well. There may even be deviations from the process flow that are anomalous, but are not considered deviations because there is no temporal aspect to them. Identification of temporary and periodic process variants introduced by such temporal and permanent deviations from the most common type of cases followed in the process can help a business get better understanding of their actual process as it changes periodically or over time. It also enables them to take appropriate actions, if necessary.

While there exist methods for detecting anomalous cases in business processes, these methods will not detect different variants of the process, as we will also show in the evaluation section of the paper. Nonetheless, the detection of such temporary process variants caused by temporal deviations is important. For example, because they may point to some problem that must be solved, some policy that employees use that should be made explicit in the process, or cases that must be filtered from the log before doing an analysis of the performance of the business process.

To fill this gap, this paper proposes a method to discover, in real time, temporary and permanent changes to the process from event log data, in addition to anomalies. The method classifies cases in an event log into four categories: *(i)* common cases (type of cases which are most followed in the process), *(ii)* temporary cases (type of cases which are followed temporarily in the process), *(iii)* periodic cases (type of cases which are followed at certain times in the process), and *(iv)* anomalous cases (type of cases which are anomalous). At the core of this method lies a clustering approach using Non-Euclidean Relational Fuzzy c-Means (NERFCM) supported by Correlation Cluster Validity (CCV). CCV is used to determine possible number of clusters existing in the event log data and NERFCM is used to form those clusters. In addition, the proposed method also includes a feature to forget a cluster when no new case falls in it for a defined period of time.

Against this background, the remainder of the paper is organized as follows: Sect. 2 presents a review of the literature related to this topic. Section 3 briefly discusses theoretical concepts involved in working of the proposed method. Section 4 details the proposed method. The evaluation of the proposed method is presented in Sect. 5. Section 6 provides conclusions and suggestions for future work.

2 Related Work

The roots of process mining can be traced back to about half a century ago [17,28,32] but it emerged only in the last decade [42,43]. Even after this rapid emergence, in the last decade, the topic of anomaly detection was not frequently researched [2,3]. In context of event logs, it is interesting to observe that after years of research, the literature still has not settled on a unified definition of anomaly. Despite not having a formal definition, the literature has developed an intuition and suggests on what can be considered anomalous; an anomaly is "some kind of unlikely or infrequently occurring behaviour" [7]. It is well known

that the analysis of the event logs is influenced by noise and irregular behaviour of a process [27], which can also be considered anomalous. The research done on the topic of anomaly detection in event logs in the last decade proposed using process discovery algorithms in order to mine a reference process model from business process event logs, and then use the discovered model for conformance checking to detect presence of anomalous behaviour.

Compared to the previous decade, there is a noticeable boom in the research outputs on this topic. The authors of [11] and [5] presented a frequency-based algorithm which finds less occurring and never occurring process executions and considers them anomalous. The authors of [12] presented a similar approach by using integer linear programming for detection and removal of infrequent behaviour observed in an event log. In [26] another approach is proposed using frequent pattern outlier factor which intends to use empirical rule of statistics to differentiate between normal and anomalous instances of a process. Moving forward from frequency-based algorithms, [6] presented a multi-perspective anomaly detection method which is based on likelihood of occurrence of execution events. [22] presented a similar approach of filtering out infrequent events based on expectation of occurrence of an activity.

Clustering algorithms from the domain of data science have also been applied and tested in the domain of process mining [19], for example, k-nearest neighbour [21,40], and use of density based clustering [41]. Use of neural networks has recently caught attention of researchers in process mining which has resulted in some of the best anomaly detection algorithms [29]). Other approaches for anomaly detection in event logs are as follows: dynamic threshold algorithm based on conformance threshold [4], based on Bayesian network [31,34], based on Markov model [1,18,24], based on association rule mining [8,35], based on correlation analysis [30], and based on Needleman–Wunsch algorithm [9].

The anomaly detection methods found in the related work can be distinguished into two types: *(i)* online methods (to detect anomaly in a running case) [29,41], and *(ii)* offline methods (to detect anomaly in historic event log) [5,6,31]. Moreover, in both the approaches, anomalies are detected as infrequent cases, or some kind of improbable combination of event attributes. We found that these methods do not give any indication of changes in the process overtime. In literature we also found research done on real-time detection of concept drift [25], process discovery [44], and conformance checking [45], but these methods try to discover the process and changes in the process overtime while not considering anomalous executions of the process. The method we propose can work both online and offline and is able to categorize the cases as common, periodic, temporary, and anomalous.

3 Background

The proposed method employs NERFCM clustering algorithm and CCV index, therefore in the next sub-sections we introduce them briefly.

3.1 Non-Euclidean Relational Fuzzy C-Means (NERFCM)

NERFCM is a clustering algorithm, an adaptation of k-means algorithm that generate fuzzy clusters (i.e. a cluster member can belong to more than one cluster) based on a dissimilarity matrix D between the data points. [20]. The dissimilarity matrix D expresses the pairwise distinction between two traces. In context of event logs, a trace is nothing but a concatenated sequence of activities occurred in a case. For example, in a case if three activities Activity_A, Activity_B, and Activity_C were performed one after another, then their respective trace could be 'abc'. If we consider other traces 'abcd' and 'acde', then computing distances among all the three traces we could obtain a dissimilarity matrix of order 3×3.

Typical distance types used to measure non-euclidean distances between two data points are Jaccard and Levenshtein distances [13]. NERFCM can handle such distances.

In addition to D, the NERFCM algorithm requires three other parameters as input: fuzzifier m, convergence criteria *epsilon*, and number of clusters c. For a specified number of clusters c and fuzzifier $m \in (1, \infty)$ the output of NERFCM is a fuzzy c-partition U which is an approximate local minimizer of a global squared-error type criterion, similar to k-means method. For more elaborate description of NERFCM algorithm please refer to [20].

Number of clusters c sets the number of clusters the input set of traces will be clustered into. This c is computed using a correlation cluster validity index as discussed in the following sub-section.

3.2 Correlation Cluster Validity (CCV)

NERFCM requires from a user a parameter, that is the number of clusters to be created, c. In order to determine number of clusters we are using Cluster Correlation Validity (CCV) [33]. CCV is an universal cluster validity measure that can be applied to partitions obtained by any relational or object data clustering algorithm (NERFCM in our case). The reason of choosing CCV over other validity indices such as Davies-Bouldin index, Xi-Beni Index [46] or Relational Xi-Beni [36] is that CCV is better at finding number of clusters in a dataset compared to all other validity indices [33].

The CCV index adopted in this method is Spearman CCV Index (v_{ccvs}); based on Spearman's Correlation Coefficient (CC), which quantifies the linear relationship between the $n(n-1)/2$ dissimilarity pairs with $i \neq j$ after ordering the elements of D and D(U) as vectors in $\Re^{n(n-1)/2}$. This is accomplished without actually ordering the elements using Eq. 1 [23,33].

$$v_{\mathrm{ccvs}}(\mathrm{D}, \mathrm{D(U)}) = 1 - \left(\frac{6 \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} (\mathrm{D_{ij}} - [\mathrm{D(U)}]_{ij})^2}{n^3 - n} \right) \tag{1}$$

where D is the input dissimilarity matrix (or reference matrix), and D(U) is the dissimilarity matrix between the partition matrix rows. CCV index can be used to evaluate and compare different partitions. A partition with a highest value of the index represents the best clustering. One can generate partitions for different number of clusters $c = 2, 3, \ldots$ and select one with the highest value of CCV index.

4 Proposed Method

This section presents our proposed method for finding periodic and temporary process variants in the event log data while simultaneously detecting anomalies. First we list the input parameters and then we provide a brief overview of the proposed method followed by an in depth step-by-step explanation on how the proposed method works.

4.1 Input Parameters

The proposed method takes the following parameters as inputs: *event_log* - the event log dataset, *distance_type* - for now Jaccard Distance only, *initial_cases* - number of cases for initial clustering, *merging_criteria* - if any two clusters have this much similarity then they will merge (range 100% similarity to 0% similarity), *forgetting_type* - Yes, if clusters are to be forgotten. No, if clusters are not to be forgotten, *forget_after* - number of days after which a cluster is to be forgotten or anomalies are to be saved. Two other input parameters are for NERFCM, m - fuzzifier (default value 2) and *epsilon* - convergence criteria (default value 0.0001) [20].

4.2 Overview of Proposed Method

This subsection provides a brief overview of the steps elicited in Algorithm 1. First, the user defines the number of initial cases (*initial_cases*) to be used to form initial clusters. A distance matrix D is computed for the *initial_cases* using the selected *distance_type*. Then CCV algorithm is applied on the selected number of cases. The result of CCV algorithm is the probable number of clusters c that exist in *initial_cases*. Next, the NERFCM algorithm is applied on the selected number of cases, and initial clustering is performed using D and c as input. The formed clusters are saved in *cluster_list*. At this stage the *cut_off_size_for_new_cluster* is also computed - it tells us how large a new cluster should be to qualify as a cluster (explained in *initialize_clusters()*). Once the initial clustering is done, when a new case arrives and it falls under the radius of any of the existing clusters then is added to that cluster, otherwise it is stored in *anomaly_list* (explained in *update_clusters()*). Simultaneously, it is checked if there are new clusters forming inside the *anomaly_list*. If at any point a cluster in *anomaly_list* becomes larger than the *cut_off_size_for_new_cluster*, then it is removed from the *anomaly_list* and added to the main *cluster_list* (explained in

form_new_clusters()). Next, if at any point in time the similarity between any two or more clusters in the *cluster_list* becomes greater than the *merging_criteria* then those clusters are merged (explained in *merge_clusters()*). If no new case is added to a cluster in *cluster_list* for *forget_after* days, then that cluster is removed from the *cluster_list* and is added to *cluster_list_forgotten* (explained in *forget_clusters()*). At last, if no new case is added to the *anomaly_list* for *forget_after* days, then all the cases are removed from the *anomaly_list* and are saved in *anomaly_list_saved* (explained in *save_anomalies()*). Then the algorithm waits for a new case to arrive and implements all the functions from *update_clusters()* to *save_anomalies()*. A detailed explanation of each step is provided in subsection 4.3.

4.3 Steps

The steps detailed inside the While loop in Algorithm 1 produce the following output: *cluster_list* - a list of all the formed clusters, *cluster_list_forgotten* - a list of all the clusters that were forgotten after *forget_after* days, *anomaly_list* - a list of all the cases that were detected anomalous, *anomaly_list_saved* - a list of all the cases that were saved as anomalies after *forget_after* days. Cases in these clusters are then categorized as common, periodic, temporary, and anomalous in the post analysis step.

Brief overview of the steps of the proposed method is presented in Algorithm 1, followed by detailed textual description of each step.

Algorithm 1: Steps included in the proposed method

```
initialize_clusters()
while True do
    update_clusters()
    if len_current_anomaly_list > len_last_anomaly_list then
        form_new_clusters()
    if similarity between any two clusters >= merging_criteria then
        merge_clusters()
    if no new case added to an existing cluster in forget_after day then
        forget_clusters()
    if no new case added to the anomaly_list in forget_after day then
        save_anomalies()
end
post analysis
```

initialize_clusters(): In order to form initial clusters, a number of initial cases need to be picked, i.e. the parameter *initial_cases*. The value of parameter *initial_cases* is dependent on the user and the dataset. For example *initial_cases* can be number of all the cases completed within one week from beginning of

the process. Then the distance matrix D is computed for the *initial_cases*, and using CCV initial number of clusters c is determined. After finding c, the value of parameter *cut_off_size_for_new_cluster* is computed as *initial_cases*$/c$; value is useful in implementing function *form_new_clusters()*. It is to be noted that the proposed method assumes that all the incoming cases are completed and contain an end-timestamp. It is important that the cases are complete because unless a case is completed, it cannot be assigned to any cluster. The end-timestamps are important to know the order of arrival of the cases.

Next, NERFCM algorithm is implemented using D and c to obtain a partition matrix U. It is to be noted that initialization of prototypes in NERFCM is not random which makes this method deterministic. Next, using D and U, c initial clusters are formed among the *initial_cases*. Since in a partition matrix each data point belongs to each partition with a certain degree of membership, therefore each trace is only kept in the cluster with which it has the highest degree of membership. After the creation of initial clusters, their respective *cluster_center* and *cluster_radius* are determined. For each cluster, the cluster member with the highest degree of membership to a cluster is selected as its *cluster_center*. For each cluster, the weighted average distance between the cluster center and each cluster member is computed; longest of all these distances is selected as *cluster_radius*. Finally, all the traces falling outside their respective cluster radius are added to *anomaly_list*.

update_clusters(): When a new completed case arrives, its activities are combined to form a trace (*newTrace*), and its similarity with all existing clusters is checked. If *newTrace* falls in any of the existing clusters then it is added to that cluster, else it is added to the *anomaly_list*.

form_new_clusters(): In this step, existing anomaly list is explored to find if there exist any clusters in the *anomaly_list*. For this purpose, similar process as *initialize_clusters()* is carried out but on the traces in *anomaly_list*. A distance matrix D_a is computed for all the traces in *anomaly_list* and using CCV initial number of clusters c_a is determined. After finding c_a, NERFCM algorithm is implemented using D_a and c_a to obtain a partition matrix U_a. Using D_a and U_a, c_a clusters are formed and saved in a temporary list of clusters *cluster_list_temp*. Similar to *initialize_clusters()*, each trace is only kept in the cluster with which it has the highest degree of membership. Next, cluster center and cluster radius are determined, and all the traces falling outside their respective cluster radius are added to a temporary list of anomaly *anomaly_list_temp*. Once all the temporary clusters are formed then the temporary clusters which are larger or equal in size to *cut_off_size_for_new_cluster* are added to the main *cluster_list* and the contents of these clusters are deleted from the main *anomaly_list*. The idea behind computation of *cut_off_size_for_new_cluster* is that if c clusters exist in *initial_cases*, then on average each cluster has *initial_cases*$/c$ traces; so, when a cluster formed in *anomaly_list* has equal or more traces than that average, then it can be considered as a valid cluster.

merge_clusters(): Any two clusters are merged on satisfaction of either of the two following conditions: *(i)* if distance between the two clusters is less than or equal to the parameter *merging_criteria*, or *(ii)* if the overall percentage of number of common elements in the two clusters exceed *1−merging_criteria*. In case two clusters are merged, then cluster center and radius is computed for the merged cluster. Also, the traces which do not fall under the new cluster radius are added to the *anomaly_list*.

forget_clusters(): If value of parameter *forgetting_type* is set to 'Yes' and if no *newTrace* is added to any existing cluster for *forget_after* days, then that cluster is removed from the main *cluster_list* and added to the *cluster_list_forgotten*.

save_anomalies(): If value of parameter *forgetting_type* is set to 'Yes' and if no *newTrace* is added to *anomaly_list* for *forget_after* days, then the cases existing in *anomaly_list* are removed from the *anomaly_list* and added to the *anomaly_list_saved*.

Once all the steps are completed, the algorithm wait for arrival of a new case. As soon as a new case arrives, it calls *update_clusters()* function and continues the while loop. When all the cases are processed, the user must go through post analysis of the output.

Post Analysis: Based on the performed clustering, the completed cases of an event log are categorized into the following four categories:

1. **Common Cases**: cases in main *cluster_list* are considered common since they were never forgotten or saved as anomalies.
2. **Periodic Cases**: cases in *cluster_list_forgotten* are considered periodic if they reappear again in main *cluster_list* or in *cluster_list_forgotten*.
3. **Temporary Cases**: cases in *cluster_list_forgotten* are considered temporary if they do not reappear in the main *cluster_list* or in *cluster_list_forgotten*; they were probably used for some special cases.
4. **Anomalous Cases**: at any given time, cases in *anomaly_list_saved* and *anomaly_list* are considered anomalous since they never belong to any cluster (whether forgotten or not forgotten).

5 Method Evaluation

In this section, first we discuss the event logs used to evaluate the method and the anomalies that are introduced to those event logs. Next we discuss the parameters selected to evaluate the method, followed by the results obtained from applying the proposed method on the selected event logs.

5.1 Event Logs

To assess the performance of an anomaly detection algorithm, it is necessary to know which traces in a process are anomalous and which traces are not anomalous. For this purpose we require synthetic event logs where we already know the process. We used PLG [10] to create six process models and their event logs with varying complexity of number of activities, breadth, and width. These event logs are the same as used in [29]. In addition to the synthetic event logs, we also used 9 real-life event logs from the Business Process Intelligence Challenge (BPIC) - BPIC12 [16], BPIC13 [37–39], and BPIC15 [15]. In the remaining of this paper, the event logs are referred by their names as defined in Table 1.

Table 1. Overview of event Logs

Name	Number of Logs	Number of Activities	Number of Cases	Number of Events
Small	1	41	5k	45.2k
Medium	1	65	5k	29.8k
Large	1	85	5k	55.6k
Huge	1	109	5k	40.6k
Gigantic	1	154–157	5k	31.5k
Wide	1	68–69	5k	30.4k
BPIC12	1	24	13k	262.8k
BPIC13	3	11–27	0.8k–7.5k	4k–81k
BPIC15	5	422–486	0.8k–1.4k	46k–62k
Alpha	1	78	3.5k	32k

To test the effectiveness of the proposed method, we needed to use an event log in which we know where common, periodic, and temporary cases occur. For this reason we created another synthetic event log which was created using combination of three event logs from Table 1, namely Large, Gigantic, and Huge. We named this synthetic event log 'Alpha'. The Alpha event log contains total 3500 cases: 2000 cases from Large, 1000 cases from Gigantic, and 500 cases from Huge. To induce periodicity in the Alpha event log, 500 cases from Gigantic are added at 14 days from the beginning of the event log, and the remaining 500 cases are added 14 days after that. Whereas, to introduce temporal nature in the event log, all of the 500 cases from Huge are added at 4 weeks from the completion of first case in the event log.

Moreover, random anomalies of type Rework, Skip, Early, Late, and Insert were introduced to the datasets, using the approach proposed by Nolle et al. [29] to make the event log dataset more realistic. The introduction of anomalies is done for the purpose of completeness.

5.2 Experiment Setup

To test the working of the proposed method, a set of values for the input parameters (Sect. 4.1) needed to be defined. For the first input parameter, *distance_type*,

we selected Jaccard Distance because it is computationally more efficient than Levenshtein Distance (linear versus quadratic time complexity) [14]. For *initial_cases*, we propose three variants - *small*, *medium*, and *large*. Variant *small* selects all the cases ending within one week from the starting date of the first case; for *medium* variant this range is till two weeks from the starting date of the first case; and for *large* variant it is four weeks. The values of fuzzifier m and converging criteria *epsilon* were set to their default values. For *merging_criteria*, we use three distinct values - 90%, 80% and 70% similarity. Both 'Yes' and 'No' values are tested for parameter *forgetting_type*. The value of *forget_after* is set to 7 days.

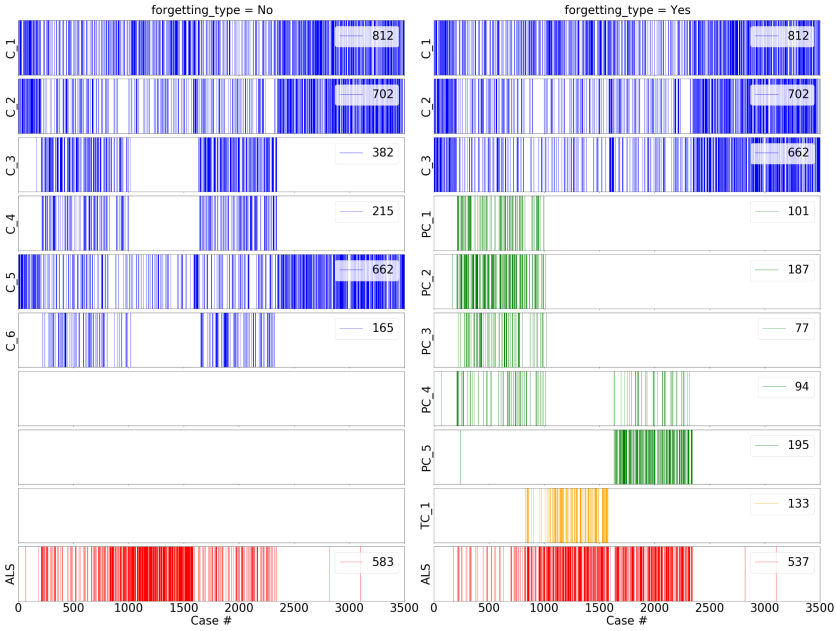
It is to be noted that the evaluation of the method is done mainly by varying values of the parameters *initial_cases*, *merging_criteria*, and *forget_type*. Considering the values of these parameters to be set as mentioned in the previous paragraph, we obtain 18 distinct combinations. Since it would be impractical to discuss results from all 18 combinations for each of 16 event logs ($18 \times 16 = 288$ combinations), therefore, for brevity, we only present in-depth results for the Alpha event log.

5.3 Results

Using the selected parameters, we receive a set of results for Alpha event log as shown in Fig. 1 and Table 2. Figure 1 shows a visual comparison between results obtained by setting the parameter *forgetting_type* as 'No' and 'Yes'. In the Fig. 1 each row represents a cluster, where cluster C_n represents common cases, cluster PC_n represents periodic cases, cluster TC_n represents temporary cases; where n is the number of cluster. For instance, C_1 shows the first cluster in the main *cluster_list*. The last row in both Fig. 1a and Fig. 1b shows the *anomaly_list_saved* (ALS). The horizontal axis represents the arrival of cases in the order of their time of completion. Each vertical bar in a cluster shows the assignment of case to that cluster. The legend in each cluster shows the number of cases that were added to that cluster. For instance, Fig. 1a shows that 812 cases were added to the first cluster, and 583 cases were added to the anomaly list.

In Fig. 1b, PC_1-PC_5 and TC_1 are the clusters which were forgotten from the main cluster list at some point in time since no new case was added to them. In the post analysis of the results, it is found that a similar cluster to PC_1 reappeared again in PC_2, PC_3 and part of PC_4. Also, part of PC_4 reappeared in PC_5. Since, all these reappearing clusters are similar to each other and they were forgotten after some time, therefore, by definition of Periodic Cases (in Subsect. 4.3) they are categorized as periodic cases. On the other hand, TC_1 is a cluster of cases which was forgotten after some time but no similar cluster ever reappeared in main clusters or forgotten clusters. For this reason, the cases in TC_1 are categorized as temporary cases. Furthermore, in Fig. 1a, all the periodic and temporary cases are included in the main cluster. Cases falling in these clusters makes up of periodic and temporary process variants.

In Table 2, the first thing we observe is that when value of parameter *initial_cases* is kept constant (e.g. small), the number of clusters formed at the



(a) Results S.No. 17 in Table 2 (b) Results S.No. 18 in Table 2

Fig. 1. Comparison of *forgetting_type* = No versus *forgetting_type* = Yes. Blue bars are common cases, Green bars are periodic case, Orange bars are temporary cases, and Red bars are cases marked anomalous. (Color figure online)

start of the run is always the same. For example, in S.No. 1 to S.No. 6, the number of clusters at start is always 9. This is always true even if other parameters are changed. The reason behind this consistency is that the initial clusters are formed using *initial_cases* (in *initialize_clusters()*); so as long as the value of *initial_cases* is unchanged, the number of clusters formed at the beginning will always be same.

Additionally, in comparison to using *forgetting_type* as 'Yes', the number of clusters formed in the end is always more when using *forgetting_type* as 'No'. This gives us a hint that in the entire duration of the event log generation, the process being followed is not always the same. Moreover, it indicates that there may exist periodic and temporary cases in the event log data. The presence of periodic and temporary cases is confirmed when exploring the results further. It can be observed that when the method is not set to forget clusters, it does not detect any periodic and temporary cases. To be noted that *forgetting_type* 'Yes' may not always detect periodic and temporary cases unless the data suggests so.

In Table 2, P, R, and F1 refer to Precision, Recall, and F1-Score calculated for detection of periodic, temporary and anomalous cases. It can be observed that precision for detecting periodic and temporary cases is high. Also, compared to small and medium variant of *initial_cases*, the recall for the large variant is

Table 2. Evaluation of the proposed method on Alpha event log with varying parameters

S. No.	init. cases	merge crit.	forget type	c		Periodic cases			Temporary cases			Anomaly cases		
				Start	End	P	R	F1	P	R	F1	P	R	F1
1	Small	90%	No	9	20	–	–	–	–	–	–	1.00	0.01	0.02
2	Small	90%	Yes	9	4	1.00	0.94	0.97	1.00	0.76	0.86	0.26	0.02	0.04
3	Small	80%	No	9	19	–	–	–	–	–	–	1.00	0.01	0.02
4	Small	80%	Yes	9	3	1.00	0.94	0.97	1.00	0.76	0.86	0.55	0.01	0.02
5	Small	70%	No	9	18	–	–	–	–	–	–	1.00	0.01	0.02
6	Small	70%	Yes	9	3	1.00	0.94	0.97	1.00	0.76	0.86	1.00	0.01	0.03
7	Med	90%	No	9	23	–	–	–	–	–	–	1.00	0.03	0.06
8	Med	90%	Yes	9	9	1.00	0.94	0.97	1.00	0.77	0.87	1.00	0.02	0.04
9	Med	80%	No	9	16	–	–	–	–	–	–	1.00	0.04	0.08
10	Med	80%	Yes	9	7	1.00	0.91	0.96	1.00	0.75	0.86	0.47	0.07	0.12
11	Med	70%	No	9	16	–	–	–	–	–	–	1.00	0.04	0.08
12	Med	70%	Yes	9	3	1.00	0.93	0.96	1.00	0.77	0.87	0.96	0.04	0.08
13	Large	90%	No	5	6	–	–	–	–	–	–	0.16	0.19	0.18
14	Large	90%	Yes	5	3	1.00	0.65	0.79	1.00	0.27	0.42	0.15	0.16	0.15
15	Large	80%	No	5	6	–	–	–	–	–	–	0.16	0.19	0.18
16	Large	80%	Yes	5	3	1.00	0.65	0.79	1.00	0.27	0.42	0.15	0.16	0.15
17	Large	70%	No	5	6	–	–	–	–	–	–	0.16	0.19	0.18
18	Large	70%	Yes	5	3	1.00	0.65	0.79	1.00	0.27	0.42	0.15	0.16	0.15

significantly small. Moreover, precision for anomalous cases is high for small and medium variants, and very small for the large variant. The reason for this that for the large variant the radii are larger since initial clustering was done on a large number of cases. So, the arriving anomalous cases may be considered not different enough and hence fall in the cluster.

The results also show that the anomalous cases introduced by us are not well detected by our method. The reason we found is that since they are too similar to an existing cluster center, thus they are added to an existing cluster. Please note that we used Jaccard distance, in which order of activities performed is ignored. For example, for our method, traces 'abcde' and 'bdcea' are same since Jaccard Distance between them is zero.

Similar results can be observed when the method was tested with other event logs considered in the study (Table 3).

Table 3 shows the results of the clustering performed on other event logs, including BPIC event logs. Since we do not know if periodic and temporary cases are present in these event logs, we cannot comment on precision, recall and F1-scores for these event log. For this reason we present how many cases were categorized as common, periodic, temporary, and anomalous (also in how many clusters). We take this opportunity to present the utility of this clustering method. For instance, considering BPIC12 event log: we form 5 clusters which consist of 12881 common cases. This tells us that there are 5 types of most common processes followed during the generation of this event log. These 5 types of processes are nothing but the cluster centers of those 5 clusters (as shown in Table 4). Other cases that lie in these 5 clusters have $(1 - cluster\_radius)\%$ of

Table 3. Evaluation of the proposed method on other event logs with parameters *initial_cases* = small, *merging_criteria* = 70%, *forgetting_type* = Yes, *forget_after* = 7 days

Name	c		Common cases (clusters)	Periodic cases (clusters)	Temporary cases (clusters)	Anomalous cases
	Start	End				
Small	2	2	4832 (2)	0 (0)	0 (0)	49
Medium	3	6	4814 (6)	0 (0)	0 (0)	48
Large	3	5	4946 (5)	0 (0)	0 (0)	36
Huge	3	10	4948 (10)	0 (0)	0 (0)	24
Gigantic	2	7	4867 (7)	83 (1)	0 (0)	51
Wide	4	4	4924 (4)	46 (1)	0 (0)	48
BPIC12	2	5	12869 (1)	0 (0)	0 (0)	217
BPIC13_1	7	1	1394 (1)	0 (0)	0 (0)	27
BPIC13_2	6	1	7373 (1)	0 (0)	0 (0)	175
BPIC13_3	5	1	668 (1)	40 (2)	0 (0)	142
BPIC15_1	4	0	0 (0)	138 (31)	863 (48)	183
BPIC15_2	4	4	77 (4)	105 (21)	589 (124)	63
BPIC15_3	2	3	199 (3)	0 (0)	933 (12)	277
BPIC15_4	4	5	96 (5)	137 (23)	774 (131)	49
BPIC15_5	3	3	202 (3)	197 (9)	588 (27)	172

similarity to their respective cluster centers. For example, trace 'abuucddddsd' only lies in the first cluster (with 66.67% similarity), whereas trace 'abuucuddsd' lies in both first and third cluster because its distance to both cluster centers is less than their cluster radius (0.33 and 0.42 respectively). We observed similar results while testing the method on all the other event logs but for brevity they are not discussed in this paper.

Table 4. Cluster centers for BPIC12 event log showing the most common type of cases followed in the process

Cluster	Trace	Case	Cluster radius
1	abuuuusu	173856	0.50
2	abcddddddegfhijdjkljlwlwwwww wwlwlllomnpl	176596	0.12
3	abcddabtd	178167	0.50
4	addefghijdjqtj	175976	0.50
5	abuucdudddddefghijdjjjkljlwlwwww wwwwwlwlsvl	174261	0.12

Overall, the proposed method is able to discover common, periodic, and temporary process variants in the event log data with a high precision by categorizing the cases as common, periodic, and temporary, while simultaneously marking infrequent cases as anomalous.

6 Conclusions

In this paper we presented a real-time method of discovering different process variants in the event log data which categorizes cases into four categories: common, periodic, temporary, anomalous. The method is able to produce at runtime an update on which type of cases are being executed at present by assigning cases to clusters. After testing on an artificially generated event log, it was found that the proposed method is able to categorize the event log data into the four categories with high precision.

Detecting different process variants in an event log data is an easy task if the entire event log data is known beforehand. It becomes challenging when the event log data is unknown. This is why we designed this method to work in real-time as soon as any new case is completed. Another reason to make the method process cases in real-time is that we wanted to capture the periodic and temporal nature in the event log data. Also, we wanted to capture the evolution of the clusters with time. Note that this method can also be used offline. A possible application of this method could be to use the obtained clusters in real-time process discovery and solving problem of spaghetti like models. This can be done by discovering the process model by using only the cluster representatives. Each cluster representative may also indicate a sub-process. Furthermore, we have information on periodic cases which can be used to discover periodic (sub-) process models.

As good as the method is in categorizing cases into the first three categories, we found out that it is not very efficient at detecting anomalies. Since many non-anomalous cases are categorized as anomalous, we suggest that the detected anomalies need to be further analysed by experts. This further analysis is necessary because the method only categorizes a case as anomalous if it is significantly different from rest of the cases executed prior to its completion. We realise that it is difficult to qualitatively assess the amount of the required expert input, and therefore we have identified a need for explanation of anomalies.

Moreover, all the forgotten clusters also need to be validated with the help of experts to understand whether that is how those cases were supposed to processed. This validation by experts is important because it is possible that the organisation recently made some changes to the process on purpose and they want to standardize those sub-processes. In this case, the process expert may mark a forgotten cluster as a main cluster. The same validation is also important for the detected anomalies.

Since the method proposed in this paper is able to cluster any new kind of case in real-time, it also provides a basis for providing explanation about when certain type of cases are used in a process. Our future work will be mainly focused on the explanation of anomalies.

One may argue that temporary cases can also be considered anomalous since they only happens once in a series and never again. We think it is a valid argument, but we believe the domain expert should have the freedom to make this decision.

A limitation of the proposed method is that it includes a uni-perspective anomaly detection method. As discussed earlier, the method discovers the structure in the event log data and also detects some anomalies as its byproduct. However, this is not a multi-perspective anomaly detection method since it only uses activities performed in each case. In future we want to include a multi-perspective view to the proposed method by providing weights to resources and other attributes associated with the activities.

Another limitation of this method is that when using Jaccard Distance, the method is good at finding the structure but detects less anomalies. In future work we would like to modify the method to work with Levenshtein Distance. In this work Levenshtein Distance is not used as it introduces two challenges: *(i)* It is difficult to decide on *merging_criteria* as, unlike Jaccard Distance, the Levenshtein Distance does not lie in the range [0, 1], which makes it difficult to define a global value of *merging_criteria*. *(ii)* As mentioned earlier in this paper, for larger data sets, using Levenshtein Distance increases computation time.

To overcome these challenges, we plan to use the study by Dolev et al. [14] to find a relation between Jaccard and Levenshtein distances and use that relation instead of computing Levenshtein distance every time a new case arrives.

As part of our future work we identify a need to make the parameter *forget_after* adapt to change in the frequency of arrival cases. The reasoning behind this need is explained in the following example. Let us assume value of the parameter *forget_after* is set to 7 days, and *cut_off_size_for_new_cluster* is 40. If we assume that generally we have an average of 50 cases arriving per day, but in summer time, because it is holiday period, we have an average of 5 cases arriving per day. It may be the situation that the cases completed in the summer time were added to the *anomaly_list* because they were different from the most common cases. As a result, all 25 cases execute in the last week were added to the *anomaly_list* and in the following 7 days they were saved as anomalies. Now even if the method detected a new cluster of 25 cases in the *anomaly_list*, it would not have qualified to be added to the main *cluster_list* because 25 is less than the *cut_off_size_for_new_cluster*, i.e. 40. For such situations, in a period of low frequency of arrival of cases, we think it is important that the method adapts to the situation and extends the window of *forget_after* days. This extension will give more chance to formation of a new cluster in such periods of low frequency of cases.

References

1. Armentano, M.G., Amandi, A.A.: Detection of sequences with anomalous behavior in a workflow process. In: Chen, Q., Hameurlain, A., Toumani, F., Wagner, R., Decker, H. (eds.) DEXA 2015. LNCS, vol. 9261, pp. 111–118. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22849-5_8

2. Bezerra, F., et al: Anomaly detection algorithms in business process logs. In: 10th International Conference on Enterprise Information Systems (2008)
3. Bezerra, F., Wainer, J., van der Aalst, W.M.P.: Anomaly detection using process mining. In: Halpin, T., et al. (eds.) BPMDS/EMMSAD -2009. LNBIP, vol. 29, pp. 149–161. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01862-6_13
4. Bezerra, F.L., et al.: A dynamic threshold algorithm for anomaly detection in logs of process aware systems (2012)
5. Bezerra, F., et al.: Algorithms for anomaly detection of traces in logs of process aware information systems. Inf. Syst. **38**(1), 33–44 (2013)
6. Böhmer, K., Rinderle-Ma, S.: Multi-perspective anomaly detection in business process execution events. In: Debruyne, C., et al. (eds.) OTM 2016. LNCS, vol. 10033, pp. 80–98. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48472-3_5
7. Böhmer, K., et al.: Anomaly detection in business process runtime behavior-challenges and limitations. arXiv:1705.06659 (2017)
8. Böhmer, K., Rinderle-Ma, S.: Association rules for anomaly detection and root cause analysis in process executions. In: Krogstie, J., Reijers, H.A. (eds.) CAiSE 2018. LNCS, vol. 10816, pp. 3–18. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91563-0_1
9. Bouarfa, L., et al.: Workflow mining and outlier detection from clinical activity D logs. J. Biomed. Inf. **45**(6), 1185–1190 (2012)
10. Burattin, A.: Plg2: multiperspective processes randomization and simulation for online and offline settings. arXiv preprint arXiv:1506.08415 (2015)
11. Chuang, Y.-C., Hsu, P.Y., Wang, M.T., Chen, S.-C.: A frequency-based algorithm for workflow outlier mining. In: Kim, T., Lee, Y., Kang, B.-H., Ślęzak, D. (eds.) FGIT 2010. LNCS, vol. 6485, pp. 191–207. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17569-5_21
12. Conforti, R., et al.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**(2), 300–314 (2016)
13. Dijkman, R., et al.: Linguistic summarization of event logs-a practical approach. Inf. Syst. **67**, 114–125 (2017)
14. Dolev, S., et al.: Relationship of jaccard and edit distance in malware clustering and online identification. In: 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA), pp. 1–5. IEEE (2017)
15. van Dongen, B.B.: Bpi challenge 2015 (May 2015). https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1, https://data.4tu.nl/collections/BPI_Challenge_2015/5065424/1
16. van Dongen, B.: Bpi challenge 2012 (April 2012). https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f, https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204/1
17. Gold, E.M.: Language identification in the limit. Inf. Control **10**(5), 447–474 (1967)
18. Gupta, N., Anand, K., Sureka, A.: Pariket: mining business process logs for root cause analysis of anomalous incidents. In: Chu, W., Kikuchi, S., Bhalla, S. (eds.) DNIS 2015. LNCS, vol. 8999, pp. 244–263. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16313-0_19
19. Han, H., et al.: Abnormal process instances identification method in healthcare environment. In: International Conference on Trust, Security and Privacy in Computing and Communications, pp. 1387–1392. IEEE (2011)
20. Hathaway, R.J., et al.: Nerf c-means: Non-euclidean relational fuzzy clustering. Pattern Recogn. **27**(3), 429–437 (1994)

21. Hsu, P.Y., et al.: Using contextualized activity-level duration to discover irregular process instances in business operations. Inf. Sci. **391**, 80–98 (2017)
22. Huang, Y., et al.: Filtering out infrequent events by expectation from business process event logs. In: 2018 14th International Conference on CIS, pp. 374–377. IEEE (2018)
23. Kendall, M.: Rank correlation methods (1948)
24. Linn, C., Werth, D.: Sequential anomaly detection techniques in business processes. In: Abramowicz, W., Alt, R., Franczyk, B. (eds.) BIS 2016. LNBIP, vol. 263, pp. 196–208. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52464-1_18
25. Maggi, F.M., Burattin, A., Cimitile, M., Sperduti, A.: Online process discovery to detect concept drifts in LTL-based declarative process models. In: Meersman, R., et al. (eds.) OTM 2013. LNCS, vol. 8185, pp. 94–111. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41030-7_7
26. Mardani, S., et al.: Fraud detection in process aware information systems using mapreduce. In: 2014 6th Conference on Information and Knowledge Technology (IKT), pp. 88–91. IEEE (2014)
27. Măruşter, et al.: A rule-based approach for process discovery: dealing with noise and imbalance in process logs. Data Min. Knowl. Disc. **13**(1), 67–87 (2006)
28. Nerode, A.: Linear automaton transformations. Proc. Am. Math. Soc. **9**(4), 541–544 (1958)
29. Nolle, T., et al.: Binet: Multi-perspective business process anomaly classification. Inf. Syst. (2019). https://doi.org/10.1016/j.is.2019.101458
30. Park, C.G., et al.: Temporal outlier detection and correlation analysis of business process executions. IEICE Trans. Inf. Syst. **102**(7), 1412–1416 (2019)
31. Pauwels, S., et al.: An anomaly detection technique for business processes based on extended dynamic bayesian networks. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, pp. 494–501 (2019)
32. Petri, C.: Kommunikation mit automaten (phd thesis). Institut für Instrumentelle Mathematik, Bonn, Germany (1962)
33. Popescu, M., et al.: Correlation cluster validity. In: 2011 IEEE International Conference on Systems, Man, and Cybernetics, pp. 2531–2536 (2011)
34. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 234–249. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_15
35. Sarno, R., et al.: Business process anomaly detection using ontology-based process modelling and multi-level class association rule learning. In: 2015 International Conference on Computer, Control, Informatics and its Applications (IC3INA), pp. 12–17. IEEE (2015)
36. Sledge, I.J., et al.: Relational generalizations of cluster validity indices. IEEE Trans. Fuzzy Syst. **18**(4), 771–786 (2010)
37. Steeman, W.: Bpi challenge 2013, closed problems (April 2013). https://doi.org/10.4121/uuid:c2c3b154-ab26-4b31-a0e8-8f2350ddac11, https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_closed_problems/12714476/1
38. Steeman, W.: Bpi challenge 2013, incidents (April 2013). https://doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee, https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_incidents/12693914/1
39. Steeman, W.: Bpi challenge 2013, open problems (April 2013). https://doi.org/10.4121/uuid:3537c19d-6c64-4b1d-815d-915ab0e479da, https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_open_problems/12688556/1
40. Sureka, A.: Kernel based sequential data anomaly detection in business process event logs. arXiv preprint arXiv:1507.01168 (2015)

41. Tavares, G.M., et al.: Anomaly detection in business process based on data stream mining. In: Brazilian Symposium on Information Systems, pp. 1–8 (2018)
42. Der Aalst, V., et al.: Workflow mining: a survey of issues and approaches. Data Knowl. Eng. **47**(2), 237–267 (2003)
43. Der Aalst, V., et al.: Business process mining: an industrial application. Inf. Syst. **32**(5), 713–732 (2007)
44. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Online discovery of cooperative structures in business processes. In: Debruyne, C., et al. (eds.) OTM 2016. LNCS, vol. 10033, pp. 210–228. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48472-3_12
45. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.P.: Online conformance checking: relating event streams to process models using prefix-alignments. Int. J. Data Sci. Anal. **8**(3), 269–284 (2017). https://doi.org/10.1007/s41060-017-0078-6
46. Wang, W., et al.: On fuzzy cluster validity indices. Fuzzy Sets Syst. **158**(19), 2095–2117 (2007)