Detecting Temporal Anomalies in Business Processes using Distance-based Methods

Ioannis Mavroudopoulos and Anastasios Gounaris

Aristotle University of Thessaloniki,Greece
{mavroudo,gounaria}@csd.auth.gr

Abstract. Outlier detection in process mining refers to either infrequent behavior in relation to the underlying business process models or to anomalous latencies of task execution (temporal anomalies). In this work, we focus on the latter form of anomalies and we propose distance-based methods. Compared to solutions relying on probability distribution analysis and based on the experimental evaluation presented, our proposal is shown to be capable of covering both trace and event outliers, and being more efficient and effective. More specifically, running times of our technique are lower by up to an order of magnitude, while we achieve significantly higher precision and recall.

1 Introduction

Nowadays a lot of businesses turn to Business Process Management (BPM) in order to improve their processes and become more efficient. BPM is the art and science of overseeing how work is performed in an organization to ensure consistent outcomes and to take advantage of improvement opportunities [16]. Thus the main focus of BPM is to improve the processes in a business. A business process is represented as a set of tasks and their flows, which are orchestrated to achieve a common business goal [14]. Since BPM execution is supported by a breadth of software tools, automated log collection is not just feasible but easy. These data can be processed using data mining techniques to provide more knowledge about the business than just monitoring [1].

In this work, we focus on finding anomalies (or equivalently, outliers) in the monitored data. An outlier is *"an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism"* [9]. The outlying data often contains useful information about the abnormal behavior, which is the reason that is used in applications such as intrusion detection, credit card fraud, and so on [2]. In business processes, an abnormal behavior can be related to a variety of issues, such as delay in a task execution, or wrong sequences of tasks in a trace.

Most studies in the field focus on finding outlier patterns in the sequence of events in a trace, e.g., [20, 7] are proposals that detect such structural anomalies. A complementary approach is to identify anomalies in the logged behavior of the task execution in terms of non-typical runtime. These anomalies are termed here as *temporal* ones, and as discussed in [18], it is important to check the behavior of a complete interdependent set of tasks to derive more insightful conclusions, e.g., whether it is (probably) a measurement error rather than actual abnormal task behavior.

The contribution of this work is to propose a distance-based outlier detection approach [2, 13, 12] to dealing with temporal anomalies in business processes. We present how this can be achieved, we discuss implementation issues and we thoroughly evaluate the proposed solution. According to our results, we can claim that our proposal is superior in terms of coverage, effectiveness and efficiency compared to the proposal in [18], which is based on probability distribution analysis. The running times of our technique are lower by up to an order of magnitude for practical settings, while we achieve significantly higher precision and recall.

The remainder of this work is structured as follows. In Section 2, we give the details of the problem and our solution. We present the experimental evaluation next. Section 4 discusses the related work and we conclude in Section 5.

2 A distance-based temporal outlier detection approach

In this section, first we formalize event logs. Then we describe the methods for discovering outlying traces and events. Following the terminology in [16, 1], a *task* is an atomic *activity*, and in this work, without loss of generality, when we refer to activities, we imply atomic ones; therefore the terms tasks and activities are used interchangeably.

2.1 Preliminaries

We follow the same terminology and event log definition as in many other works in business process mining, e.g., [7, 18]. More specifically, we assume that businesses have a mechanism to record the corresponding event logs in place in order to analyze their processes. An *event log* is composed of a set of traces. Each *trace* corresponds to a specific process instance execution and is identified by a unique *case* identifier. The instance execution is manifested as a recorded sequence of events. Each *event* records the execution of an *activity* in a particular trace(case).

Definition 1. *Event Log: let $A = \{a_1, a_2, \ldots, a_m\}$ be a finite set of activities (tasks) of size m. A log L is defined as $L = (E, C, \gamma, \delta, ts, \preceq)$ where E is the finite set of events, C is the finite set of Cases (Traces), $\gamma : E \rightarrow C$ is a surjective function assigning events to Cases, $\delta : E \rightarrow A$ is a surjective function assigning events to activities, ts records the timestamp denoting the finish of task execution, and $\preceq$ is a strict total ordering over events, normally based on execution timestamps.*

From the definition above, it is straightforward to calculate the latency of each task based on the difference between its timestamp and the timestamp of its immediate predecessor; process instance start and termination events are included in the logs. The latency includes both the task duration and any waiting time.[1]

2.2 Temporal trace outliers

To the best of our knowledge there is no method that finds temporal trace outliers. Here, we propose a method that finds outlier traces based on the number of executions and the mean time spend in every activity in a specific process instance. We do not pose any restriction on (i) the order the activities may be executed, (ii) whether they all appear in the same instance and (iii) on the times they are executed in a single instance, since all these factors may differ between process instances, e.g., [7, 19].

We group L entries by trace, i.e., case identifier. In order to define the distance between two traces, we first convert every trace to a vector of size $2m$. Each position to this vector corresponds to a distinct attribute.

Definition 2. *Trace Vector: given a trace t, the Trace Vector of this trace, is a vector $[n_0, ..., n_m, t_0, ..., t_m]$ where m is the number of different activities in the process to which L refers, $n_i,\ i = 1 \dots m$ is the number of executions of the same task in the same trace (e.g., due to loops) and t_i is the mean execution time of a_i.*

In the definition above, execution time should be interpreted as equivalent to task latency. In the next step, we normalize the data, so that no attribute dominates the distance calculation during the outlier detection process. We create a trace matrix $[t; t'; t''; ...]$, where each row is a trace vector and there are $2m$ columns. We then apply the z-score normalization Equation (1) to every attribute, so that its mean value becomes equal to 0 and its standard deviation is equal to 1 [2].

$$Z = \frac{X - mean(X)}{std(X)},\ X\ is\ a\ column\ of\ the\ trace\ matrix \quad (1)$$

Definition 3. *Trace Vectors Distance: given two trace vectors t_1, t_2, where $t_1 = [t^1_1, \dots, t^1_{2m}]$ and $t_2 = [t^2_1, \dots, t^2_{2m}]$, we define the distance between them as*

$$dist = \sqrt{\sum_{i=1}^{2m} \frac{(t^1_i - t^2_i)^2}{2m}} \quad (2)$$

The Equation (2) is similar to the traditional RMSE (Root Mean Square Error). We take the second power of their difference, so that the distance between two traces is dominated by the attributes for which they have the biggest difference.

[1] If the logs contain the start and end finish time of each task explicitly, then our approach to detecting latency anomalies can be applied to detecting anomalous task durations in a straightforward manner.

Algorithm 1 Find Outlier Traces

```
    traceVectors ← [ ]
    traces ← extract traces from L
    for all trace ∈ traces do
 4:     traceVectors append preprocess(trace)
    end for
    normalizedTraces ← normalize(traceVectors)
    mtree ← constructMTree(normalizedVectors)
 8: outlierFactors ← [ ]
    for all trace ∈ normalizedTraces do
        kneighbors ← mtree.kneirestneighbors(trace,k)
        outlierFactor ← 0
12:     for all neighbor ∈ kneighbors do
            outlierFactor ← outlierFactor + dist(trace,neighbor)
        end for
        outlierFactors append outlierFactor
16: end for
    sort outlierFactors
    return the top $\zeta$ traces
```

The pairwise distance between traces is used in a straightforward manner to detect outliers. Each trace is assigned an outlier factor, which is equal to the sum of distances from the k-nearest neighbors. As such, the traces that, when depicted in a euclidean space, are located in areas with low density, will be reported as outliers. Once we have calculated the outlier factor for every trace, we can report the top ζ outliers. Formally, the temporal trace outliers are defined as follows.

Definition 4. *Temporal trace outlier: given a normalized trace matrix, a (normalized) trace vector t is an outlier if the sum of the distances $dist(t, t')$ with the closest k other trace vectors t' is in the top ζ values, where k and ζ are parameters defined by the user.*

In real scenarios, there are thousands of traces, so we cannot afford to calculate the distance between every pair of traces due to the quadratic complexity of such a process. That is why we need a data structure called M-Tree to mitigate the performance impact of range queries. M-Tree is a tree data structure that is constructed using a metric and relies on the triangle equality for efficient range and k-nearest neighbors queries [6]. The pseudo-code is in Algorithm 1.

In the last step of the algorithm, instead of reporting the top ζ outliers, we can report the traces for which their outlier factor deviates more than x times the standard deviation from the mean value. Or we can resort to the outlier definition used in the subsequent section. However, our approach is more suitable for a high-dimensionality space, since $2m$ can grow large. This is because a key characteristic of the outlier definition above is that it does not rely on estimating the underlying probability distribution, which is notoriously difficult in multiple dimensions.

2.3 Temporal event outliers

The previous method of outlier detection refereed to traces as a whole. Event outliers, examine each event type separately, i.e., log entries in L are grouped by the activity they refer to through the function δ. The aim of temporal event outlier detection is to identify the event log entries, for which the corresponding execution latency is highly dissimilar compared to the rest of the executions of the same activity type.

This problem is first addressed in [18], where Rogge-Solti *et al* presented a method to find temporal anomalies, i.e. anomalies concerning the running time of an activity in a process, using probability distribution fitting. The key point is that, most commonly in real cases, the distribution of the task execution latencies does not follow a normal distribution; so the proposal in [18] leverages a more robust curve fitting method, which relies on the work in [23].

In this work we propose a method that uses the pairwise event log distances to determine the outliers. The distance between two task entries is defined as the absolute value of the difference between their execution latency. Then, the definition employed for trace outlier detection is transferred to events in a straightforward manner.

Definition 5. *Temporal event outlier: given a set of task execution latencies referring to the same activity $a_i \in A$, a logged task latency is an outlier if the sum of the distances with the closest k other task latencies is in the top ζ values, where k and ζ are parameters defined by the user.*

Alternatively, the following definition can be used, which is on the one hand the same as in the seminal work of [12, 13] but is more sensitive to the user-defined input parameters.

Definition 6. *Temporal event outlier (alternative definition): given a set of task execution latencies referring to the same activity $a_i \in A$, a logged task latency is an outlier if it has less than k neighbors, where another task latency is neighbor if its distance is less than or equal to R, and k and R are parameters defined by the user.*

Both definitions, for efficient implementation, require a data-structure, such as R-tree or M-tree to perform the range queries involved.[2] Another similarity is that they do not require data normalization, e.g., through Equation (1). Their main difference is that Definition 5, gives a list of ζ values and implies that the end user will post-process this list to check whether all the reported events are outliers indeed, or more outliers exist, e.g., through examining the relative differences of the sums. In Definition 6, there is no post-processing, but the correct setting of the R value, which is scenario-dependent, rests with the user.

[2] It is out of our scope in this work to compare R-tree vs M-tree.

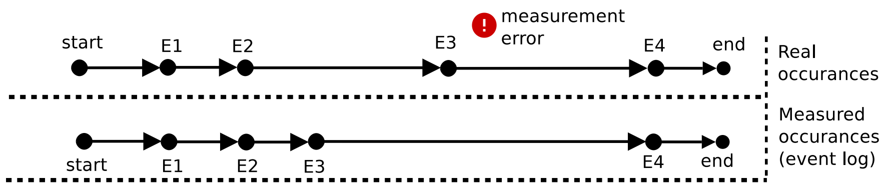


Fig. 1: Example of a measurement error from [18]. In the first row, the real execution times of each event are shown. After a measurement error at event E2, which also effected the execution time of E3, the second row shows the entries in the log file.

2.4 Temporal outliers of event pairs

Reporting individual temporal event outliers is hard to interpret because such temporal anomalies can be either true anomalies or measurement errors. In the latter case, the timestamp of an event has been falsely logged, causing an event to appear as outlier. This problem was first defined in [18], where the key observation is that measurements errors typically affect two consecutive tasks in the trace in a negatively correlated manner. More specifically, if there is no abnormal behavior in the real execution, but due to delayed (resp. early) recording, a task instance has a long (resp. short) latency in the logs, it is expected that the subsequent event will have a recorded short (resp. long) latency. This is (most probably) a measurement error and has to be distinguished from real outliers (see Figure 1).

The approach to identifying measurement errors in [18] consists of two components: (i) estimating the probability distribution function of the task latencies, as mentioned in the previous section, and (ii) employing a Bayesian Network to define the dependencies between activities; where the nodes in the Bayesian Network are the activities and the edges represent the succeeding-preceding relations. Here, we suggest to replace the first part with distance-based outlier detection techniques, extending the proposal in Section 2.3. More specifically, we introduce two techniques for detecting measurement errors (or better, probable measurement errors). The first one transfers the problem of outlying detection of event pairs to the problem of distance-based outlier detection of 2-dimensional points, so that Definitions 5 and 6 apply with simple extensions. The second one uses exactly the same setting as in Section 2.3.

Mapping consecutive events in a 2-dimensional space. First, we normalize the task latencies similarly to the approach in Section 2.2. Second, we take all the ordered pairs of consecutive events as they appear in the log file. For each such ordered pair, we keep only the ordered pair of their normalized latencies. Then, each ordered pair of latency values can be treated as a 2-dimensional point. Regardless of any probability distribution of the task latencies, for every activity, the mean latency is 0 and its standard deviation equals to 1. Overall,

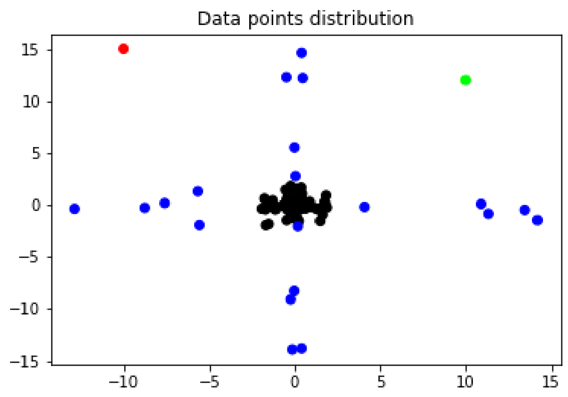


Fig. 2: Example data point distribution after normalization. The blue data points are true outliers. The green data point at the top-right is also a true outlier because its coordinates have the same sign. However, the red point at the top-left part is (probably) a measurement error.

the whole log set is transformed to a set of points. The size of this set is smaller than the number of log entries in L because we produce a point for each event in a trace apart from the start one.

Figure 2 shows an example of a set of log entries transformed to a set of points. We notice that most of the data points are near the center (0,0), but there are several points that are far from it. The data points that deviate significantly from the center, are reported as outliers using our distance-based techniques (either definitions), because they have a few points in their neighborhood. In the example, most of the outliers deviate only with regards to a single dimension; these are considered as true outliers. True outliers are also the outliers in the top-right and left-bottom part of such an illustration. However, any points in the top-left or right-bottom part, like the red point in the figure, are reported as (probable) measurement errors because the sum of their execution times is normal and there is a suspicion that it is simply due to delayed or premature recording of the finish timestamp of the preceding event.

We employ Definition 5 for outlier detection, using the euclidean distance. Instead of a M-tree, we can also use a R-tree (see also Algorithm 2). As previously, we report the top ζ points.

We use again as an outlying factor, the sum of distance for the k-nearest neighbors for every data point. In order to make fast knn queries we implement a tree data structure, named R-Tree, which is suitable for range queries and knn queries in the space. As a distance metric we are using the Euclidean distance. The m data points with the highest outlying factor, will be reported as outliers. In Algorithm 2 we show the procedure of detecting outlying pairs of events.

Algorithm 2 Temporal outliers of event pairs

```
Input traces
Output outliers
    eventPairs ← [ ]
    for all trace ∈ traces do
        for all $e_1, e_2$ ∈ trace do
            eventPairs append [normalized($e_1$.latency), normalized($e_2$.latency)]
5:      end for
    end for
    rtree ← constructRTree(eventPairs)
    outlierFactors ← [ ]
    for all pair ∈ eventPairs do
10:     kneighbors ← rtree.kneirestneighbors(pair)
        outlierFactor ← 0
        for all neighbor ∈ kneighbors do
            outlierFactor ← outlierFactor + euclidean-distance(pair,neighbor)
        end for
15:     outlierFactors append outlierFactor
    end for
    sort outlierFactors
    outliers ← top $\zeta$ pairs
```

The technique so far has produced the list of the consecutive logged events that are temporal outliers. The next step is the assessment of the type of outlierness. As already discussed, if these points have their coordinates with opposite signs, i.e., one positive and another negative, there are reported as measurement errors; otherwise they are reported as normal outliers. However, the normalized latencies may create a problem if the absolute mean latency of one task in the pair is much larger than the absolute mean latency of the other; in such a case, deviating the same amount of standard deviations in both dimensions does not counterbalance each other. Therefore, we denote as (d_1, d_2) the pair of the non-normalized latencies for each result item in Algorithm 2, we adopt the following definition and we run Algorithm 3.

Definition 7. *Measurement error: given a set of pairs of outliers as reported by Algorithm 2, the constituent outliers are reported as (probable) measurement errors if*

$$|(mean(d_1) - d_1) - (mean(d_2) - d_2)| \leq \tau \quad (3)$$

Outlier detection of pairs: a more efficient approach. In the previous technique, for each pair, we calculate the sum of distances from its k nearest neighbor. Even though with the use of R-Tree, we manage to reduce the running time, most of the computations were unnecessary, because they involve normal

Algorithm 3 OutliersOrMeasurementErrors

```
    Input outliers from Algorithm 2, threshold $\tau$
    Output normalOutliers, measurementErrors
    normalOutliers ← o ∈ outliers and (o.x same sign as o.y)
 2: measurementErrors ← [ ]
    for all pair ∈ outliers-normalOutliers do
 4:     if pair is measurement error based on Equation (3) then
            measurementErrors append pair
 6:     else
            normalOutliers append pair
 8:     end if
    end for
10: return normalOutliers and measurementErrors
```

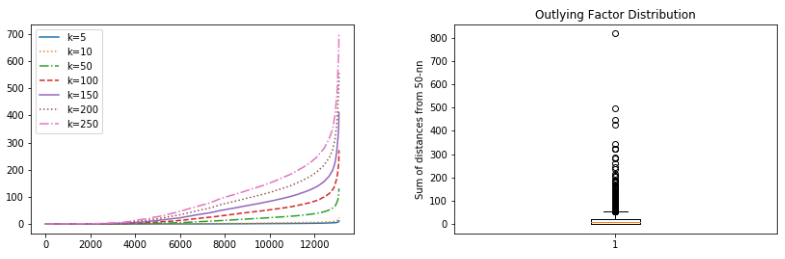
latencies. Based on this, we keep the Algorithm 3 the same, where outliers differentiate between *normal outliers* and *measurement errors*, but the input is not provided by Algorithm 2. Instead, we use the techniques in Section 2.3 to find isolated outliers. For each such outlier, we check whether its succeeding event is an outlier as well, and if this is the case, the pair is considered in Algorithm 3. As will be reported in the evaluation, avoiding to map the task pairs to a 2d space yields performance improvements while producing exactly the same results.

3 Evaluation

We used both real-world and synthetic datasets to evaluate the performance of the proposed methods. We start by presenting the datasets, followed by the evaluation of the trace outlier detection method. Then we compare the two pair-based methods that use distance and at the end we compare the best of these methods with the one proposed in [18]. All tests were conducted in a machine with 16GB of RAM and 3.2GHz CPU with 8 cores. The source code for all of the proposed distance-based outlier detection methods is publicly available on GitHub[3].

The real-world datasets are taken from the Business Process Intelligence (BPI) Challenges, and more specifically from the 2012 and 2017 ones. BPI12[4] is an event log of a loan application process. It consists of 13087 traces that contain a total number of 262200 events. The mean amount of events per trace is 20.03 and the minimum and maximum amount is 3 and 175, respectively. BPI17[5] is an event log, which also corresponds to a loan application of an Dutch financial institute. It includes 31509 traces, which contain over 1M (1202267) events in total. The mean, max and min number of events per trace for this dataset are 38.15, 10 and 180, respectively. We have also created a synthetic dataset (details

[3] https://github.com/mavroudo/BPM-outlierDetection-distance-based
[4] https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f
[5] https://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b



(a) (b)

Fig. 3: Sum of the distances from the k-nearest neighbor for different values of k (a) and sum value distributions for $k = 50$ (b)

will be discussed later).

3.1 Evaluation of Trace Outlier Detection

Our first experiment aims at verifying the effectiveness of our approach. For this, we use the read-world dataset BPI12. Figure 3(a) shows how the sum of distances from k-nearest neighbors is changing with different values of k. It also helps us identifying the most suitable value of k, which corresponds to the plot that is initially as parallel as possible with the horizontal axis, and then, after a sharp change, becomes parallel to the vertical one. Such a behavior allows a clearer distinction of outliers, and in our experiment is the plot for $k = 50$.

In Figure 3(b), the box-plot shows how the sums of distances of the 50 nearest neighbors are distributed for all the traces. As shown, the mean value is very low (actual value is 14.4) and there are only a few traces that have sum of distances more than 400. After executing the trace outlier detection method for $k = 50$ and a relatively large ζ value, we report as real outliers the traces with outlying factor greater than the mean value of all the sums plus 4 times the standard deviation; in this manner, we keep 52 outlying traces. Two examples are as follows:

1. Trace_2, which contains the activity "W_Wijzigen contractgegevens". This activity was only executed in 12 out of 13082 traces.
2. Trace_{856} because its mean execution time of the activity "W_Afhandelen leads" is 2327510.471secs. This activity has mean execution time 586 secs in the complete log.

The previous experiments referred to the effectiveness of the approach. Regarding the efficiency, even though that we use an M-Tree to reduce the time for k-nearest neighbors queries, as the dimensions increase, the running time for these queries increases in a quadratic manner, as shown in Figure 4.

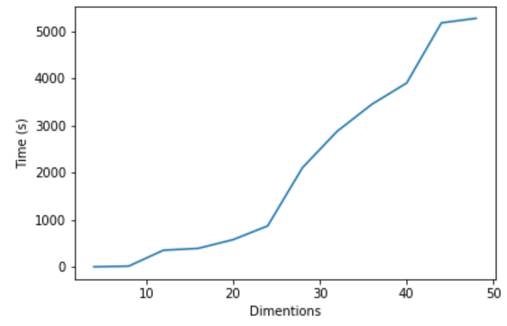


Fig. 4: Execution time for varying number of dimensions

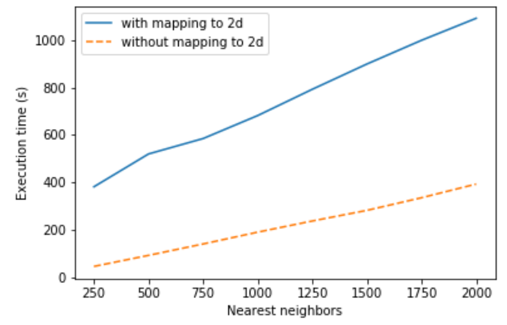


Fig. 5: Execution times of distance-based outlier detection with and without mapping event pairs to a 2-dimensional space

3.2 Evaluation of Event Outlier Detection

In Section 2.4 we proposed two different methods that use the distance to find anomalous pairs of events and then classifying them as true outliers or measurement errors. We conduct an experiment using the BPI12 dataset to compare the execution times between these two methods and we show the running times in Figure 5. From the figure, we can conclude that not mapping to a 2-dimensional space but directly relying on individual event outlier detection is more efficient by up to an order of magnitude for small k values. Therefore, next, we employ solely the second method from Section 2.4.

Comparison against [18]. To compare our distance-based outlier detection against the probability distribution-based in [18], we employ both BPI12 and BPI17 and also a synthetic data set. The synthetic dataset contains 4 activities with different distributions, namely (i) a combination of two normal distributions (see Figure 6), (ii) a combination of two alpha distributions, (iii) a combination

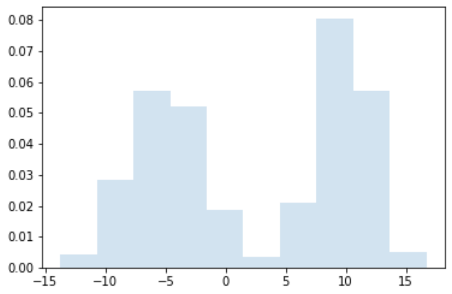


Fig. 6: Distribution of task execution latencies for the first activity in the synthetic dataset.

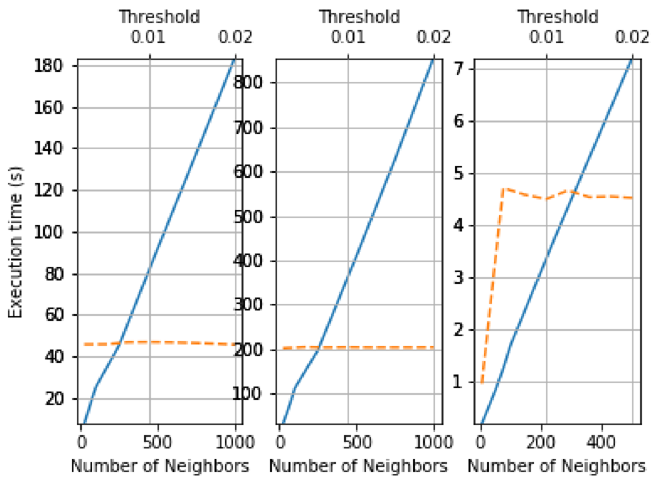


Fig. 7: Running times for the BPI12 (left), BPI17 (middle) and synthetic (right) datasets, respectively. The yellow (doted) line represents execution time based on threshold and the blue line represents execution time based on the number of neighbors

of two exponential distributions and (iv) a power lognormal distribution. It includes 526 traces of 8K events overall, where every trace contains between 5 to 25 events. The synthetic dataset is provided along with the source code.

The execution times for both methods are presented in Figure 7. We confirm that the time is linear dependent on k, i.e., the number of nearest neighbors that we take into account in the distance-based method. The execution time for the distribution fitting method does not depend on the input parameters (i.e., probability fitting threshold ranging from 0.001 to 0.2). For a value of k close

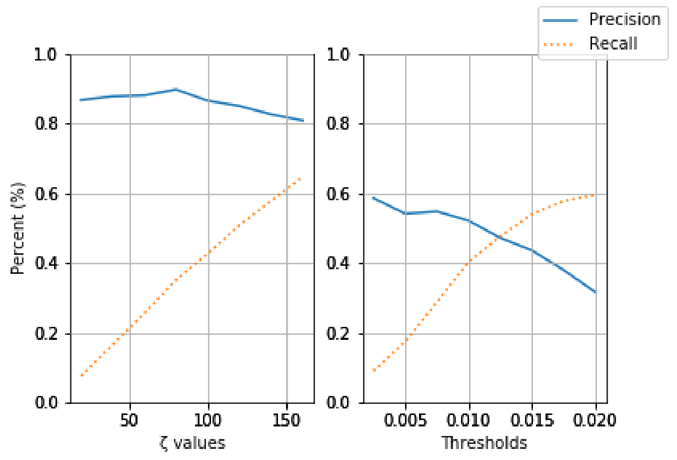


Fig. 8: Precision and recall as a function of ζ for the distance-based method with k=50 (left) and the probability fitting threshold for the technique in [18] (right).

to 50 that we have shown that make more sense, our method runs faster by an order of magnitude.

We test the effectiveness of each method in the synthetic dataset. Each event is classified as outlier or normal based on whether the probability density function is below 0.01. The precision and recall results appear in Figure 8. Our method has significantly higher precision and recall, whereas the distribution fitting method cannot approximate correctly the underlying distributions. E.g., for the distribution in Figure 6 (first activity in the synthetic dataset), it fails to correctly report outliers with task latency between 1 and 4 time units. Overall, Figure 8 shows that no matter how the technique in [18] is configured, there is better configuration of our technique in terms of precision and recall.

4 Related Work

There are several proposals about outlier detection in business processes. However, almost all of them focus on detecting outliers regarding the structure of the underlying process model. In this direction, outliers can be used in order to predict the failure of an ongoing process. In [11, 5], different approaches to predicting the next tasks of an active trace, and based on this prediction to determine if a trace will fail to execute properly, are presented. In addition, as a business process log trace is typically in the form of a sequence of tasks, outliers can be found in these sequences, e.g., [7, 15]. Anomaly detection methods that take into account both the structure of the model and the data attributes are known as multi-perspective, have been developed, such as the ones in [3, 17, 4]. However, none of these approaches deal with temporal anomalies in the way we do, and thus are not directly comparable to our approach. Temporal outlier detection, in the context of BPM, have been addressed in [18], and we have directly compared our solution against it in the previous section. The approach in [10], even though it calculates the distance between traces based on task duration, it only considers those traces that contain identical tasks. Hence, it cannot find global temporal outliers and does not deal with measurement errors.

In addition, a typical application is to clear the log dataset removing infrequent behavior in order to facilitate process discovery; process discovery aims to derive the underlying process model out of event logs. Another example of dealing with variations in the process model structure is to allow configurable models, as thoroughly covered in [19]. All the proposals above are orthogonal to our work, which focuses on temporal outliers.

From the outlier detection point of view, there exist several textbooks, e.g., [2]. The techniques based on statistics were the first to be proposed, e.g., through determining data values at the tails of a univariate distribution and the corresponding level of statistical significance. Distance-based outlier detection, which is leveraged in this work, is a representative of proximity-based anomaly detection. Proximity-based methods define a data point as an outlier if its neighborhood is sparsely populated. In distance-based techniques, a data point is considered as an outlier, if its distance from its k-th closest neighbor is longer than a predefined radius. Some of the advantages of these techniques are the linear scalability in the size of the dataset [12, 8], the ability to interpret the results and operate in a streaming and/or massively parallel environment, e.g., [22], and their wide applicability as reported in [21].

5 Summary

In this work, we advocate the usage of distance-based outlier detection methods for identifying anomalous behavior in event logs in terms of the running time of tasks. We explain the implementation details, and compared to the existing methods that rely on probability distribution approximation, our proposal is broader, in the sense that it applies to complete traces, more efficient, in the sense that runs faster, and more effective, in the sense that achieves higher precision and recall. Our implementation is provided as open-source.

Acknowledgment. The research work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant (Project Number:1052).

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. Aggarwal, C.C.: Outlier Analysis. Springer (2017)
3. Böhmer, K., Rinderle-Ma, S.: Multi-perspective anomaly detection in business process execution events. In: International Conference on Cooperative Information Systems (CoopIS) 2016 (October 2016), http://eprints.cs.univie.ac.at/4785/

4. Böhmer, K., Rinderle-Ma, S.: Mining association rules for anomaly detection in dynamic process runtime behavior and explaining the root cause to users. Information Systems **90**, 101–438 (2020)
5. Borkowski, M., Fdhila, W., Nardelli, M., Rinderle-Ma, S., Schulte, S.: Event-based failure prediction in distributed business processes. Inf. Syst. **81**, 220–235 (2019)
6. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. International conference on very large data bases (VLDB) (2001)
7. Conforti, R., Rosa, M.L., ter Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**(2), 300–314 (2017)
8. Dai, Q.Z., Xiong, Z.Y., Xie, J., Wang, X.X., Zhang, Y.F., Shang, J.X.: A novel clustering algorithm based on the natural reverse nearest neighbor structure. Information Systems **84**, 1 – 16 (2019)
9. Hawkins, D.: Identification of Outliers. Springer Netherlands
10. Hsu, P.Y., Chuang, Y.C., Lo, Y.C., He, S.C.: Using contextualized activity-level duration to discover irregular process instances in business operations. Information Sciences **391-392**, 80 – 98 (2017)
11. Kang, B., Kim, D., Kang, S.H.: Real-time business process monitoring method for prediction of abnormal termination using knni-based lof prediction. Expert Systems with Applications **39**(5), 6061 – 6068 (2012)
12. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: Proceedings of the 24rd International Conference on Very Large Data Bases. p. 392403 (1998)
13. Knorr, E.M., Ng, R.T.: Finding intensional knowledge of distance-based outliers (1999)
14. Kueng, P., Kawalek, P.: Goal-based business process models: Creation and evaluation. Business Process Management Journal **3** (09 1996)
15. de Lima Bezerra, F., Wainer, J.: Algorithms for anomaly detection of traces in logs of process aware information systems. Inf. Syst. **38**, 33–44 (2013)
16. Marlon, D., Marcello, L.R., Jan, M., Hajo A., R.: Fundamentals of Business Process Management (2019)
17. Nolle, T., Seeliger, A., Thoma, N., Mhlhuser, M.: Deepalign: Alignment-based process anomaly correction using recurrent neural networks (2019)
18. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. In: BPM. pp. 234–249 (2014)
19. Rosa, M.L., van der Aalst, W.M.P., Dumas, M., Milani, F.: Business process variability modeling: A survey. ACM Comput. Surv. **50**(1), 2:1–2:45 (2017)
20. Satyal, S., Weber, I., young Paik, H., Ciccio, C.D., Mendling, J.: Business process improvement with the ab-bpm methodology. Information Systems **84**, 283 – 298 (2019)
21. Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., Gunopulos, D.: Online outlier detection in sensor data using non-parametric models. In: VLDB. pp. 187–198 (2006)
22. Toliopoulos, T., Gounaris, A., Tsichlas, K., Papadopoulos, A., Sampaio, S.: Parallel continuous outlier mining in streaming data. In: 5th IEEE International Conference on Data Science and Advanced Analytics (DSAA) (2018)
23. Yeung, D.Y., Chow, C.: Parzen-window network intrusion detectors. In: Object recognition supported by user interaction for service robots. vol. 4, pp. 385–388 vol.4 (2002)