PErrCas: Process Error Cascade Mining in Trace Streams

Anna Wimbauer, Florian Richter, and Thomas Seidl

Ludwig-Maximilians-Universität München, Munich, Germany
a.wimbauer@campus.lmu.de, {richter, seidl}@dbs.ifi.lmu.de

Abstract. Efficient and quick detection of problems is an essential task in online process monitoring. Many anomaly detection approaches excel in finding local deviations. We propose a novel approach that tracks local deviations over multiple process instances and visualizes correlations of deviation points. PErrCas provides knowledge about current cascades of deviations to give process analysts a starting point for rational root-cause analysis if processes leave their in-control parameters. PErrCas monitors deviations online and maintains cascades of varying timespans. Hence, our approach avoids defining an observation window beforehand, which is a significant advantage due to its impracticability to predefine expected cascade properties in exploratory scenarios.

Keywords: Anomaly Detection · Cascades · Trace Streams

1 Introduction

Anomaly detection has multiple applications in process mining. The most prominent scenario is conformance checking, where misbehavior of process instances is measured against a reference process model by techniques like token replay or alignments. The identified anomalies represent structural non-compliances in comparison to previous or planned executions. Temporal deviations are another focus for process anomaly detection since detecting unexpected delays or speed-ups often provides a starting point for thorough investigations. Fraud, failures, or inefficient resource usage are only a few root causes for deviations.

While the research community has published a rich collection of techniques to detect various anomalies, most works focus explicitly on correlations within cases and neglect interferences between different cases. Whether it be customer journeys, production cycles, or sequences of administrative actions, cases are handled as independent process executions, and explanations for anomalies in a case are usually expected to be caused by previous events in the same case. However, cases share a resource pool containing staff, machinery, or infrastructure. Restricting a root cause analysis to singular cases might fail if another instance has caused an issue and subsequent cases are affected by its effects. We differ between local anomalies, isolated within a singular case, and global anomalies, which originate in a particular case of an event and spread through the process using common tie points between cases.

This work presents a novel online approach to identify process error cascades in a trace stream. Error cascades are typically not artificially implemented in processes. Since many processes contain a dynamic resource scheduling, e.g., the staff is assigned depending on current situations like workload or environmental influences, static cascade knowledge has limited value. Error cascades have two additional properties besides their various lifetimes, defined as the timespan between the actual event and the last moment that the cascade influences events.

Many cascades affect only structurally subsequent events according to the process. E.g., delayed transporters in logistic processes delay following transports, which might delay further transports waiting for the first segment. In specific processes, deviations may cause feedback in the process. Delays in production processes often cause previous and following actors to traverse into idle states. Depending on the process design, this allows preponing of cases in contrast to their scheduled execution. If the processes do not allow resources reallocation, previous actors also switch to a delay status.

The remaining important property of cascades is complexity. Typically, most cascades contain only a few correlated actions. Complex cascades with long correlation chains of affected actions are infrequent but provide valuable insights for later investigations. Large distances between root causes and detected deviations are typical scenarios where manual analysis fails to establish the causal connection.

2 Related Work

Correlations between different database objects have been extensively researched in the domain of sequential pattern mining[5]. Regarding sequential pattern mining on data streams, traditional SPM algorithms are required to overcome memory and performance restrictions and are therefore not always suitable to be applied on data streams directly. Marascu and Masseglia [9] propose an approximate algorithm called SMDS (Sequence Mining in Data Streams) primarily designed for Web usage data streams that can handle the complexity of streaming data. In their approach, user transactions are processed in batches. For each batch, the users are clustered based on their surfing behavior adding users to the most similar cluster or creating a new cluster. In [7], [14] research on online sequential pattern mining is continued. However, this research direction focuses on totally ordered sequences. Event-based processes allow concurrent executions of events, and anomalies are propagated non-linearly due to the process complexity. Moreover, we consider if two anomalies happen close in time to declare a correlation, while temporal intervals are usually neglected in sequence mining.

In the field of spatio-temporal data mining deep learning methods are used to learn traffic flow correlations to predict future traffic flow [6], [13]. Since those methods depend on the spatial features and processes mostly neglect spatial data while focusing on structural positions in the process, the approaches are not directly applicable for our use case. Even if event logs include spatial data, this information might not be relevant for the causal relationship between outliers.

In Liu et al. [8] the authors aim at finding causal interactions between traffic outliers by constructing outlier causality trees and running a frequent subtree mining algorithm on them. Toosinezhad et al. [12] applied these ideas for process mining. The authors are the first to solve a significant task, as the origins of process failures are not always found within the same process instance since the real world is interconnected. Their approach does not consider anomalies for each case individually but anomalies over various cases and their correlations. Hence, Toosinezhad et al. proposed a method to tackle this challenge and introduce a novel perspective of process anomaly detection.

As cases proceed in a process, their irregular behavior might disrupt the entire system causing further anomalies. Toosinezhad et al. divide the dataset into batches and construct one cascade graph per batch. The partitioning into specific intervals, like weeks, requires prior knowledge of certain cascade properties. Instead, we expand our cascades incrementally without batch restrictions. We create new cascades when incoming outlier events are not correlated to already identified cascades. Finally, we cluster the constructed cascades to give generalized cascade patterns, allowing quicker analysis by emphasizing the prominent structures.

The Performance Spectrum miner presented in [3] uses a descriptive analysis to reviel performance patterns. In [11] Senderovich et al. use both intra- and inter-case features to predict case properties. However, to the best of our knowledge, Toosinezhad et al. proposed the only work on detecting anomaly cascades in processes so far.

3 Preliminaries

The proposed method is applied to trace streams. A trace stream $S : \mathbb{N} \to \mathbb{N}$ is a mapping from natural numbers to the case identifier domain. Such a trace stream can be efficiently generated from an event stream, as already described in [10]. On case-level, each case contains finitely many events.

Definition 1 (Case-Level Event). *A case-level event e is a tupel e = (c, a, t) containing a case identifier $\#_{case}(e) = c$, an activity label $\#_{activity}(e) = a$ and a timestamp $\#_{time}(e) = t$. The case-level event may also contain additional attributes.*

Regarding intervals between case-level events, we define segment-level events. These are then aggregated into cascades which are modelled as graphs and represent the causal dependencies on the process level.

Definition 2 (Segment-Level Event). *A segment-level event s is a tupel s = (sn, c, st, et) containing a segment-name $\#_{segment}(s) = sn$, a case identifier $\#_{case}(s) = c$, a start-time $\#_{start}(s) = st$ and an end-time $\#_{end}(s) = et$. Every segment-level event s is composed of two case-level events e_i and e_j, where $\#_{case}(e_i) = \#_{case}(e_j) = c$, $(\#_{activity}(e_i), \#_{activity}(e_j)) = sn$, $\#_{time}(e_i) = st$ and $\#_{time}(e_j) = et$. It must hold that $\#_{time}(e_i) < \#_{time}(e_j)$ and there is no e_k with $\#_{case}(e_k) = c$ such that $\#_{time}(e_i) < \#_{time}(e_k) < \#_{time}(e_j)$.*

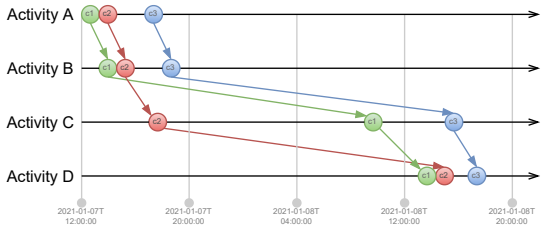


Fig. 1: Example process time line

Definition 3 (Error Cascade). *An error cascade is a directed graph $g = (V, E)$, where each node n in V represents a set of outliers $S = s_1, \cdots, s_k$ in one segment $\#_{segment}(s_1) = \cdots = \#_{segment}(s_k)$. There is an edge from node n_i to n_j, if outliers in n_j are correlated to preceding outliers in n_i.*

Each node has a *heat value* that gives information about the last time an outlier occurred in this segment. It is computed as an exponentially moving average to consider all past segment-level event outliers aggregated in this node. We declare a node as active if the time difference between the starting time of the current outlier and the heat value of the node is lower than a predefined *activity threshold* th_a. The activity threshold defines the time span in which we assume two outliers to be correlated. If the activity threshold is one day, an outlier can affect the process performance for one day. Henceforth, if the time difference between the heat value and a new outlier is greater than the activity threshold, a causal relationship between the outlier set of that node and the new outlier is impossible. We call a cascade active as long as at least one of its nodes is still active.

4 Online Cascade Mining

In this section, we define the three main steps of our method. Our approach operates on trace streams. We first scan for process segments that take an unusually long (or short) time for each incoming trace. We then check for each outlier if it is correlated to an already existing active cascade, in which case we add the outlier to the correlated cascade. If it is not correlated to an existing cascade, the outlier forms the start of a new cascade. These first two steps are performed on each trace consecutively. The last step is carried out in an offline phase once a set of cascades has accumulated. We cluster the cascades and compose all cascades in one cluster to a cascade pattern.

4.1 Outlier Segment-Level Events

For each incoming trace, we generate the segment-level events from consecutive case-level events and search for temorally deviating segment-level events. Fig. 1 shows an example process with four activities A, B, C, and D. Cases c_1, c_2 and

c_3 arrive shortly after one another and traverse through the process at different paces. Every circle on the timeline symbolizes a case-level event. It means, e.g., that case c_1 underwent activity A at 12:35 on the seventh of January 2021. Since there are four successive activites, we have three segment-level events per case: *A:B*, *B:C* and *C:D*. All three cases transition from activity A to activity B fairly quickly, then c_1 gets delayed in segment $B:C$. This leads to further delays of case c_2 in segment $C:D$ and case c_3 in segment $B:C$. We can already see that segment-level events *B:C* - c_1, *C:D* - c_2 and *B:C* - c_3 will be marked as outliers.

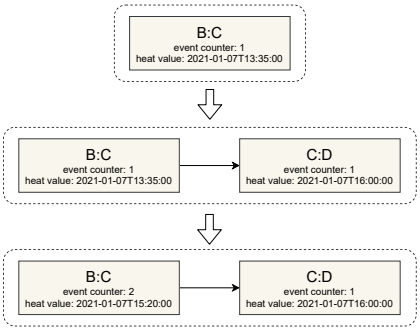
Formally we declare a segment-level event an outlier if its z-score $Z(s) = \frac{\Delta t - \mu_{segment}}{\sigma_{segment}}$ is higher than a certain outlier threshold th_o. With $\Delta t = \#_{end}(s) - \#_{start}(s)$ being the duration of the segment-level event. The mean $\mu_{segment}$, the variance $\sigma^2_{segment}$ and the number of events per segment $k_{segment}$ are stored for each segment and updated with every incoming segment-level event.

4.2 Error Cascade Construction

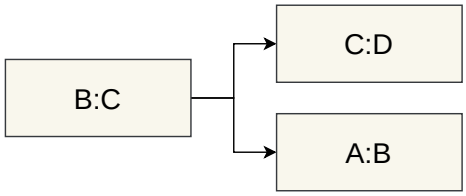
When a new outlier arrives, we check whether it correlates to any currently active cascades. In this case, it is "added" to this cascade. If an outlier is not correlated to an active cascade, a new cascade is started. Over time older cascades become inactive node by node, and new cascades are started and built up. If an outlier segment-level event s and a cascade fulfill one of the two following cases we assume that they are correlated.

1. Segment-level event s belongs to the same segment as a node n in the cascade and $\#_{start}(s) - \#_{heat}(n) < th_a$. The cascade already includes a set of outliers in the same segment that is still active, in a sense that the time difference between heat value and starting time of the outlier does not exceed the activity threshold. In this case, outlier s is added to node n by increasing the event counter by one and updating the heat value: $\#^{new}_{heat}(n) = \#_{start}(s) - [0.25 \cdot (\#_{start}(s) - \#^{old}_{heat}(n))]$
2. Segment-level event s and a node n in the cascade share a common activity and $\#_{start}(s) - \#_{heat}(n) < th_a$. Since outlier segment-level event s and the outliers of node n are close in time and overlap in their segments, we assume that the anomalous behaviour of s is correlated to the segment-level events aggregated in node n. In this case a new node n_{new} for segment $\#_{segment}(s)$ is appended to the cascade such that *event counter* $= 1$ and $\#_{heat}(n_{new}) = \#_{start}(s)$. An edge is added from n to n_{new} symbolizing the correlation between n and n_{new}.

If a segment-level event s is not correlated to a cascade, we assume that none of the preceding events are correlated to this outlier. As stated above, the new outlier event s then marks the start of a new cascade. We start a new cascade by generating a new cascade graph with one node. In the same way as a new node is added to an existing cascade, the first node of the new cascade has $\#_{segment}(s)$ as segment and *event counter* $= 1$ and $\#_{heat}(n) = \#_{start}(s)$. If the cascade still contains one node once it becomes inactive, we delete it and



(a) Example cascade building process



(b) Cascades going from *B:C* to *C:D* and from *B:C* to *A:B* were clustered together and composited to a cascade pattern

Fig. 2: Example Cascade Mining

regard the corresponding outlier (or outliers) as standalone. Fig. 2a describes the incremental cascade building process for our example. *B:C* is the first node of the new cascade, because outlier *B:C* - c_1 could not be added to an existing cascade. Next comes outlier *C:D* - c_2 which is correlated to node *B:C* because they overlap in activity C and are temporally close. A new node *C:D* with an edge from *B:C* to *C:D* is added to the cascade. The third outlier *B:C* - c_3 is correlated to both existing nodes. Since there is already an active node for segment *B:C* the outlier is added to this node by updating the event counter and heat value.

4.3 Cascade Patterns

In the first two steps, we process the traces and the outliers within these traces consecutively. Every time a specific time has passed, and a set of cascades could be collected within this period, the last step is carried out. We then cluster these cascades in an offline phase to search for patterns within the cascades, i.e., patterns of correlated segments. Alternatively, one of the various online clustering algorithms (see [15]) could be applied to every error cascade that is no longer active. This however is not in the scope of this paper.

We first cluster the cascade set by applying DBSCAN [4]. We chose the DBSCAN clustering algorithm [4], because it can find clusters of arbitrary shapes and can handle noise. The clustering provides a grouping into similar cascade graphs and filters out noisy or rare cascades simultaneously. To apply the algorithm, we define a distance measure within the cascade space. For the distance between two cascade graph we use the maximum common subgraph metric as presented in [1]. To get more representative clustering results, we assign an additional weight to every cascade. If a cascade weights 2, the clustering algorithm handles the cascade as if it was contained in the set twice. As weights, we choose the average number of segment-level event outliers that nodes in this cascade contain. Adding weights is necessary because there might be cascades that stay active for a long time. If correlated outliers come in at frequent intervals, we always add them to the same cascade. This continuously prolongs the cascades activity, and no new cascades with the same cascade pattern are generated.

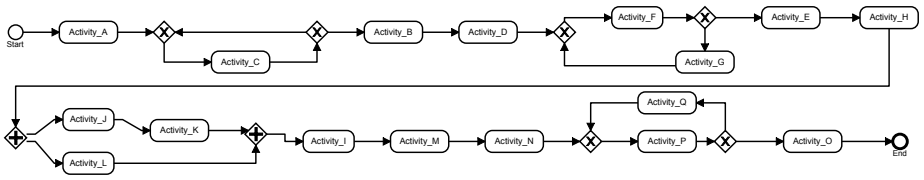


Fig. 3: Underlying process model for the synthetic data

Without adding any weights, DBSCAN would declare these cascade graphs as noise, even though they represent many segment-level event outliers.

Finally, we compose all cascade graphs within a cluster into one cascade pattern. Composing the cascades means we summarize all nodes and edges from the individual graphs in one graph, the cascade pattern. The cascade pattern provides a good overview of the various cascades in the respective cluster.

Clustering and composing the cascades aims at generating a relatively small, manageable and easy to interpret result set. Different cascade patterns represent distinct groups of outlier correlations. The compression is a significant advantage compared to [12], where the number of resulting frequent cascades tends to be very large, and there are often large groups of very similar frequent cascades.

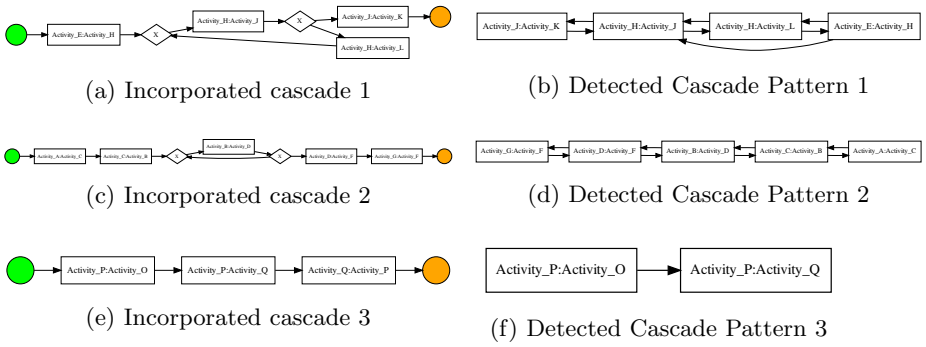
Getting back to our example, let us assume that we retrieved a few more cascades from *B:C* to *C:D*. Additionally, cascades from *B:C* to *A:B* were detected. These cascades were grouped into the same cluster by the clustering algorithm, and we compose these cascades into the cascade pattern shown in Fig. 2b. This cascade pattern visualizes in an intuitive way that delays in segment *B:C* were correlated to delays in both segment *C:D* and *A:B*. The final cascade pattern then forms a good basis for possible process improvements.

5 Evaluation

5.1 Synthetic Data

In the following we present our results from testing our method on synthetic and real life data. We first tested our approach on synthetic data, as this way, we could verify the results we obtained from our method. For DBSCAN clustering we use the following parmeters: $\epsilon = 0.4$ and $minPts$ is set to the 75%-quantile of the cascade weights, but at least 4. For the synthetic data we used the processes and logs generator PLG2 [2] to generate an eventlog, based on the process model shown in Fig. 3. We then spread all traces over one year and introduced noise by randomly delaying every event (normally distributed with $\mu = 30$, $\sigma^2 = 25$ minutes). Finally, we incorporated the three cascades shown in Fig. 4, by delaying events in the corresponding segments. The cascades occur 300, 50 and 12 times and have an approximate length of one day, one week and one month.

We tested our approach with different parameters, achieving the best results with an *activity threshold* of 1 day and an *outlier threshold* of 5. During the



(a) Incorporated cascade 1

(b) Detected Cascade Pattern 1

(c) Incorporated cascade 2

(d) Detected Cascade Pattern 2

(e) Incorporated cascade 3

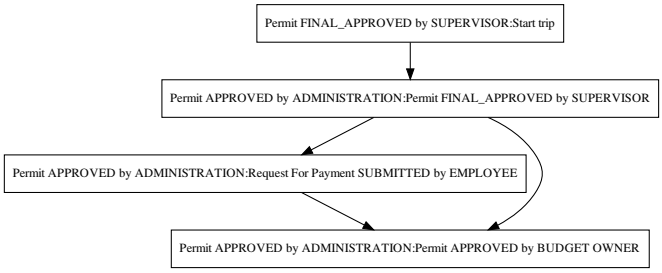
(f) Detected Cascade Pattern 3

Fig. 4: Induced and detected cascades in the synthetic log with $activity\_threshold = 1, outlier\_threshold = 5$ and $\epsilon = 0.4$

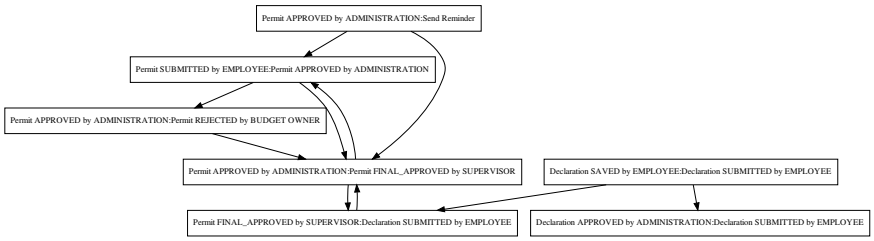
cascade detection phase 882 segment-level event outliers were detected and assigned to 167 cascades graphs. Out of these 167 graphs 121 were deleted before clustering because they contained only one node. In the end we received 46 cascade graphs, which were then grouped into 3 clusters (and some outliers) and composited to the 3 cascade patterns shown in Fig. 4. This complies with the number of cascades from the ground truth. Cascade 1 and 2 are nearly identical to the induced cascades and also have a maximum common subgraph (mcs) similarity of 1.00 with the ground truth cascade. Cascade 3 is missing its last segment node, which leads to a mcs similarity of 0.67.

To test our approach on datasets with different quality we increased noise in our dataset. As described earlier we first generated an event log without any noise (using PLG2) and induced the three cascades in a second step. To create synthetic logs with increasing noise, we introduced noise to the control flow of the initial event log using PLG2. To this end, we chose increasing parameters (0 to 40 promille) for the trace missing head, trace missing tail, trace missing episode, perturbed event order probability. We generated five logs for each noise parameter and averaged the results over these five logs, since the results varied due to randomness in the event log creation process.

The tested parameters and corresponding results are shown in Fig. 7. The F1-score was calculated by comparing each detected cascade pattern with the ground truth cascade it was most similar to. F1 nodes only considers correctly/wrong assigned nodes, whereas the total F1-score considers nodes and edges. The overall recall and the F1-score for nodes are significantly higher than the overall F1-score, which is mainly due to additional edges in the detected cascade patterns (Compared to the incorporated cascade patterns, the detected cascades contain more undirected instead of directed edges.). These additional edges are detected since the cascades were incorporated into the data in close intervals. Because of this, a cascade might still be active when delays of a subsequent cascade start. Fig. 7 shows that even though the quality of the results decreases slightly with increasing noise, it still stays at a pretty high level and our approach can deliver meaningful results.



(a) cascade pattern 1



(b) cascade pattern 2

Fig. 5: Exemplary cascade patterns retrieved from BPI 2020 dataset

To compare our results, we slightly adapted the method from [12] to our use case and implemented it using python. We tested the approach on our synthetic log with different parameters and achieved the best results (i.e. all cascades were detected, with minimum result set size) with $outlier\ threshold = 5$, $time\ interval = 60$ (batch length in days) and $minimum\ support = 3$ (for frequent subgraph mining). The resultset consisted of 153 cascades, where each cascade covered parts of the incorporated cascade patterns, and every cascade pattern was represented entirely by at least one frequent subgraph. Even though all incorporated cascade patterns were detected, the size of the result set was considerable, making it very difficult to interpret it. Furthermore, many frequent subgraphs differed from other subgraphs in only one node or edge and thus did not contribute any new valuable information. We observed that the size of the result set could vary significantly for different time intervals. At the same time, it is challenging to choose an appropriate time interval because it cannot be derived from the structure of the process. The size of the time interval defines the maximum duration of a cascade. However, this information is not given in a real-life cascade mining scenario, which means that by choosing a too small time interval, one might neglect longer-lasting cascades. At the same time, a smaller time interval might be desirable, as it leads to a smaller result set. Cascades of cascade pattern 3 (see Fig. 4e) have an approximate length of 30 days. This cas-

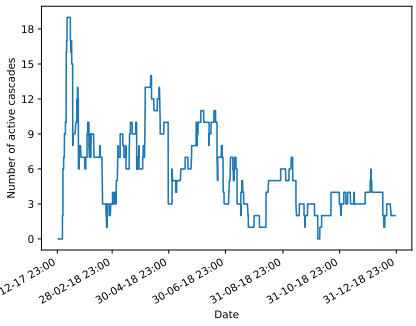


Fig. 6: Number of active cascades, travel permit log of BPI 2020 dataset

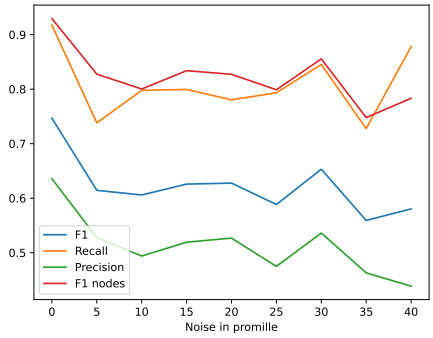


Fig. 7: Results on synthetic data with increasing noise

cade pattern was only detected entirely from a time interval of 30 days onward. For a bi-weekly interval, 3 of 106 frequent cascades had an mcs-similarity of 0.67 to cascade 3. For all the lower intervals, cascade 3 was not detected at all.

In conclusion, our approach yielded a far smaller result set (3 vs. 153 detected cascade patterns) that still contained the same amount of information. At the same time, we achieved good results even in a streaming scenario (compared to an event log), where we had to process traces consecutively.

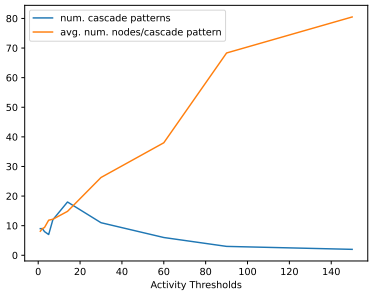
5.2 Travel Reimbursement Process

In addition to the synthetic data, we also tested our approach on real-world process data that was published for the BPI Challenge 2020[1]. The data was collected from the travel reimbursement process at TU/e in 2017 and 2018 and contained files for different subprocesses. Travel reimbursement is a process present in nearly every company and thus forms a good basis for our evaluation. For international trips, employees have to request a travel permit before starting the trip. At the end of the trip, they can request reimbursement of their costs. We chose this process for our tests because here, an array of delays can, in the worst-case, risk the entire trip. For our experiments we used the travel permits log, which contains the described process, and reduced it to traces in 2018.

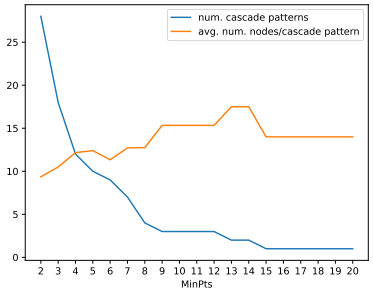
With an activity threshold of 7 days and an outlier threshold of 5, we detected 12 cascade patterns, showing two examples in Fig. 5. 528 segment-level event outliers were grouped into 124 cascades (+ 19 deleted cascades with one node). As Fig. 6 shows, the number of active cascades changed in waves and decreased over the year. A maximum number of 18 cascades was active at the beginning of the year, which might be due to many requests regarding trips later in the year.

Fig. 8 shows how the results vary for different parameters. We observed that with an increasing activity threshold, the number of detected cascade patterns decreases while the average number of nodes per cascade pattern increases (Fig.

[1] https://icpmconference.org/2020/bpi-challenge/
DOI: https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51



(a) Results for different activity thresholds with a constant $minPts$ of 4



(b) Results for different values of $minPts$ with a constant activity threshold of 7 days

Fig. 8: Parameter Sensibility (constant outlier threshold of 5)

8a). For a large activity threshold, e.g. 150 days, cascade nodes stay active for a very long time. New incoming outliers are declared correlated to existing cascades for a longer time, and no new cascades are started. This leads to larger cascades and thus also larger cascade patterns. New cascades are started more frequently for smaller activity thresholds, resulting in more cascades and fewer nodes per cascade. At this point, it needs to be mentioned that an activity threshold of 150 days or even 60 days is probably very unrealistic for this kind of process. The activity threshold resembles the time in which an anomaly can affect process performance. A proper value for the activity threshold can be picked in the context of the process structure, and in contrast to the time interval from [12] no prior knowledge of the cascades is needed.

The number of cascade patterns also decreases with an increasing $minPts$ (input parameter DBSCAN)(Fig. 8b). The $minPts$ parameter can be used as an importance regulator. The higher it is, the fewer cascade patterns are detected and the more cascades each pattern represents.

6 Conclusion

With our novel approach PErrCas, we are able to track correlated outliers over multiple process instances by continuously adding outliers to existing cascades and creating new cascades. We differentiate between two different correlations: accumulations of outliers in one segment and correlated outliers in different segments. The set of cascades can be analyzed in regular intervals to create cascade patterns and get an overall picture of the cascades. This continuous approach avoids defining an observation window beforehand. Instead, we consider how long an outlier can affect future process performance and track cascades as long they influence process performance. A useful extension of our work would be to discover a good candidate threshold for this automatically.

So far, our method only works on trace streams because we need entire traces to build segment-level events and detect outliers. Future work could examine how

error cascades can be detected in event streams. Another issue for future work is the correlation between outliers. We declare outliers to be correlated if they are close in time and their segments overlap. However, there are also many other ways in which two anomalies could be correlated.

References

1. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern recognition letters **19**(3-4), 255–259 (1998)
2. Burattin, A.: Plg2: Multiperspective process randomization with online and offline simulations. In: BPM (Demos). pp. 1–6. Citeseer (2016)
3. Denisov, V., Fahland, D., van der Aalst, W.M.: Unbiased, fine-grained description of processes performance from event data. In: International Conference on Business Process Management. pp. 139–157. Springer (2018)
4. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: kdd. vol. 96, pp. 226–231 (1996)
5. Fournier-Viger, P., Lin, J.C.W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. Data Science and Pattern Recognition **1**(1), 54–77 (2017)
6. Guo, S., Lin, Y., Feng, N., Song, C., Wan, H.: Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 922–929 (2019)
7. Laur, P., Symphor, J., Nock, R., Poncelet, P.: Mining sequential patterns on data streams: A near-optimal statistical approach. In: Proc. of the 2nd International Workshop on Knowledge Discovery from Data Streams (2005)
8. Liu, W., Zheng, Y., Chawla, S., Yuan, J., Xing, X.: Discovering spatio-temporal causal interactions in traffic data streams. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1010–1018 (2011)
9. Marascu, A., Masseglia, F.: Mining sequential patterns from temporal streaming data. In: Proc. of the 1st ECML/PKDD Workshop on Mining Spatio-Temporal Data (MSTD 2005). pp. 1–13. Citeseer (2005)
10. Richter, F., Maldonado, A., Zellner, L., Seidl, T.: Otoso: Online trace ordering for structural overviews. In: International Conference on Process Mining. pp. 218–229. Springer (2020)
11. Senderovich, A., Di Francescomarino, C., Ghidini, C., Jorbina, K., Maggi, F.M.: Intra and inter-case features in predictive process monitoring: A tale of two dimensions. In: International Conference on Business Process Management. pp. 306–323. Springer (2017)
12. Toosinezhad, Z., Fahland, D., Köroğlu, Ö., Van Der Aalst, W.M.: Detecting system-level behavior leading to dynamic bottlenecks. In: 2020 2nd International Conference on Process Mining (ICPM). pp. 17–24. IEEE (2020)
13. Wu, Y., Tan, H.: Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework. arXiv preprint arXiv:1612.01022 (2016)
14. Xu, C., Chen, Y., Bie, R.: Sequential pattern mining in data streams using the weighted sliding window model. In: 2009 15th International Conference on Parallel and Distributed Systems. pp. 886–890. IEEE (2009)
15. Zubaroğlu, A., Atalay, V.: Data stream clustering: a review. Artificial Intelligence Review **54**(2), 1201–1236 (2021)