Online anomaly detection using statistical leverage for streaming business process events

Jonghyeon Ko and Marco Comuzzi

Department of Industrial Engineering
Ulsan National Institute of Science and Technology (UNIST)
Ulsan, Republic of Korea
{whd1gus2,mcomuzzi}@unist.ac.kr

Abstract. While several techniques for detecting trace-level anomalies in event logs in offline settings have appeared recently in the literature, such techniques are currently lacking for online settings. Event log anomaly detection in online settings can be crucial for discovering anomalies in process execution as soon as they occur and, consequently, allowing to promptly take early corrective actions. This paper describes a novel approach to event log anomaly detection on event streams that uses statistical leverage. Leverage has been used extensively in statistics to develop measures to identify outliers and it has been adapted in this paper to the specific scenario of event stream data. The proposed approach has been evaluated on both artificial and real event streams.

Keywords: Process Mining, Online Anomaly Detection, Event Streams, Information Measure, Statistical Leverage

1 Introduction

Information logged during the execution of business processes is available in so-called event logs, which contain events belonging to different process instances (or *cases*). Each event is described by multiple attributes, such as a timestamp and a label capturing the activity in the process that was executed.

Event logs are prone to errors, which can stem from a variety of root causes [1,2], such as system malfunctioning or sub-optimal resource behaviour. For instance, sloppy human resources may forget to log the execution of specific activities in a process, or a system reboot may assign a different case id to all the new events recorded after rebooting. Errors in event log hamper the possibility of extracting useful process insights from event log analysis, and should therefore be fixed as early as possible [20].

To this end, the research field of event log anomaly detection (or event log cleaning) has emerged recently, providing methods to detect anomalies at trace level [1,2,9,10,13], i.e., concerning the order and occurrence of activities in a process, and at event level [15,16,18], i.e., concerning the value of attributes of events, using a variety of different approaches. Note that event log anomaly

detection is normally (process) model-agnostic, that is, it does not assume the existence of a process model or clean traces from which a model can be extracted. This aspect separates this research field from traditional process mining research on compliance checking [14].

In the specific case of online settings, i.e., event streams, while research has recently emerged in the field of online compliance checking, only the work by Tavares et al. [19] addresses the issue of anomaly detection. In particular, the authors propose a method to detect point anomalies specified by Principal Component Analysis (PCA). These point anomalies, however, do not normally reflect real-life anomaly patterns, such as inserting, skipping or switching events, commonly considered by event log anomaly detection in offline settings. Therefore, we argue that there is a lot of potential for new research in this area.

More in general, event log anomaly detection in online settings can be crucial for discovering anomalies in process execution as soon as they occur and, consequently, allowing to promptly take early corrective actions. The online settings, however, obviously introduce additional challenges to the design of an event log anomaly detection method. In particular, owing to the finite memory assumption of online settings [4,5,6,19,20], only a limited number of (recent) events are available at any given time to take a decision. This prevents to apply effectively some of the approaches that have been proposed in the literature for event log anomaly detection in offline settings. Probabilistic methods that detect anomalies after having created an intermediate model of frequent process behaviour [1,10,13] are hampered by the fact that only a limited number of events may be available to create such models. Online settings also prevent the application of machine learning reconstructive techniques for anomaly detection, e.g. [16,17]. These, in fact, normally rely on deep learning models, which require a high number of data points (complete process traces in this scenario) to be trained effectively. Also, any update of these models may require a long training time.

In this paper we propose an information-theoretic approach to online event log anomaly detection at trace level. Specifically, we devise an anomaly score based on statistical leverage [11]. The leverage is a relative measure of the information content of observations in a dataset that has been used extensively in statistics to develop observation distance measures and outlier detection techniques. Since leverage captures the information content of one observation in respect of all others in a dataset, the anomaly score proposed in this paper can always be calculated reliably based on the information available at any point in time, resulting in an anomaly detection method that does not require extensive amount of data to be executed effectively.

After having presented the related work (in Section 2), Section 3 presents a trace anomaly score based on the notion of statistical leverage. Then, we discuss how this score can be applied to anomaly detection of streams of events, addressing issues such as the grace period, the finite memory assumption, and the identification of anomaly detection thresholds. The proposed method is evaluated (in Section 4) on both artificial and real event logs injected with trace-level anomalies. Conclusions finally are drawn in Section 5.

2 Related work

While there is only limited literature regarding online event log anomaly detection, a number of recent contributions have focused on online conformance checking. To some extent, conformance checking can be seen as model-aware anomaly detection, since process models, given or extracted from clean traces, can be seen as signatures of positive behaviour to detect anomalies. As referred by [6], there are currently two research lines in online conformance checking: the prefix-alignment approach [20] and the model-based approach [4,5,6].

Conformance checking/alignment of streaming events tends to overestimate the computation of optimal alignments. In order to avoid this issue, [20] provides the first incremental/online conformance checking technique that uses prefix-alignment. Prefix-alignment [20] is characterised by high computational complexity and prevents to define a warming up period. Alternatively, Online Conformance Transition Systema (OCTS) [4,5] can partially check compliance on regions of a process. This technique also suffers from high computational complexity and prevents to consider the warm start scenario. In [6], the first solution to achieve a warm start with streaming events has been proposed by introducing weak order relations, that have reduced computational complexity.

Regarding event log anomaly detection, as mentioned in the Introduction, Tavares et al. [19] have first applied the online clustering algorithm DenStream [7] to detect anomalies on event streams. DenStream clusters cases into two groups, normal and anomalous, using histogram-based frequency of activities contained in each case. Since the histogram-based frequency ignores the sequence of events in traces, DenStream detects point anomalies in event logs defined by Principal Component Analysis (PCA) [21].

3 Research framework

There are two different elements in the proposed framework: the anomaly score and the anomaly detection method. The former (presented in Section 3.1) concerns the definition of a trace anomaly score based on statistical leverage. The latter (Section 3.2) concerns setting a threshold value above which a trace is considered anomalous based on its anomaly score.

3.1 Anomaly score

Statistical *leverage* [11] is a measure indicating how far away each observation is scattered from other observations in a dataset. It has been used as a key support measure for developing different observation distance metrics, such as Cook's distance, the Welsch-Kuh distance, and the Welsch's distance.

Given a matrix X, with $X \in \mathbb{R}^{J \times I}$, of a dataset with J observations and I numerical attributes (or variables), the leverage of the observations in X are the diagonal elements of the projection matrix $H = X(X^T X)^{-1} X^T$. Specifically, the leverage of the j-th observation in X is the diagonal element $h_{j,j} \in H$, which is

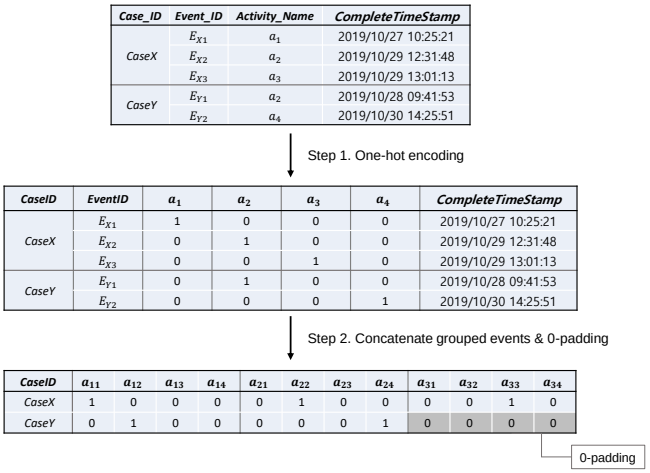


Fig. 1: One-hot encoding and 0-padding

comprised by definition between 0 and 1. The higher its leverage, the more likely an observation to be an anomaly.

Our objective is to detect anomalies at the level of occurrence and order of events in traces. Therefore, we can abstract an event log E as a set of J traces $\{\sigma_j\}_{j=1,\dots,J}$. Each trace is a sequence of events $e_{i,j}$ of variable length N_j, i.e., $\sigma_j = \{e_{1,j}, \dots, e_{N_j,j}\}$. Events are ordered in a trace by timestamp in ascending order and are defined by the activity that they represent, which is one in a set $A = \{a_1 \dots, a_K\}$ of K possible activity labels.

In order to define a leverage-based anomaly measure of traces in an event log, two pre-processing steps are necessary. The first one is an integer encoding step. This is necessary because the attributes of a dataset X must be numerical to calculate H, while the activity attribute in event logs is categorical. Second, events in an event log must be aggregated at trace level, such that the resulting matrix X has J rows, i.e., one for each trace. In conclusion, the projection matrix $H(E)$ can be calculated by considering an observation matrix $X(E)$ obtained from E applying the following pre-processing steps.

In the first pre-processing step (see Figure 1), similarly to [16], we apply one-hot encoding, that is, each event $e_{i,j}$ is encoded into a set K dummy attributes $d_{i,j,k}$ such that:

$$d_{i,j,k} = \begin{cases} 1 & \text{if } e_{i,j} = a_k \\ 0 & \text{otherwise} \end{cases}$$

Then, for trace-level aggregation, the one-hot encoded events are horizontally concatenated for each trace. Since traces have different length, for the traces shorter than the longest one(s) in E, i.e., with less events than $N^{max} = \max_{\sigma_j \in E}\{N_j\}$, zero padding is applied. For example, given a case consisting of

4 events and $N^{max} = 5$, the fifth event of this case is zero padded, therefore, $d_{5,j,k} = 0$, $\forall k$. Based on this pre-processing, an event log E is encoded into an observation matrix $X(E)$ with J rows (traces) and $I = N^{max} \times K$ columns (attributes).

Using $X(E)$, we can now define a first leverage-based anomaly score $\hat{l}(\sigma_j)$ by extracting the diagonal elements of $H(E) = X(E) \cdot (X(E)^T \cdot X(E))^{-1} \cdot X(E)^T$:

$$\hat{l}(\sigma_j) = h_{j,j}$$

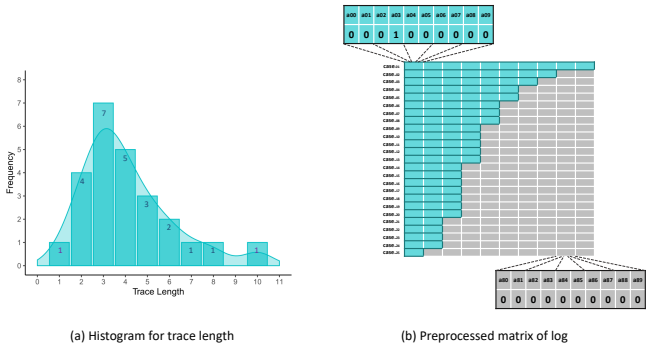


Fig. 2: Seesaw effect of zero-padding

This first anomaly score is likely to be biased by the zero-padded attributes in the aggregation pre-processing step. Normally, these zero-padded attributes should be treated as `null` values by any statistical method and therefore not considered in the analysis. However, this is not the case when calculating $\hat{l}(\sigma_j)$. The presence of 0-padded values, as shown in Figure 2, creates a a *seesaw* effect that increases the leverage of longer traces and decrease the one of shorter traces. Shorter traces, in fact, are more likely to be considered similar to each other, and therefore not anomalous, because they are encoded into a higher number of zero-padded values.

In order to counter this issue, we introduce a weighting factor w_j as a function of the trace length to increase/decrease the leverage $\hat{l}(\sigma_j)$ of shorter/longer traces σ_j. This weighting factor is calculated by first normalising the trace length N_j in the range $[0, 1]$. This is done by applying the Z transformation to normalize N_j to the average $mean[N_j]$, followed by the application of a sigmoid function. The sigmoid-based normalisation is generally used to improve the fit accuracy and decrease the computational complexity of the fitting model [12]:

$$sig(Z_j) = \frac{1}{1 + e^{-Z_j}}, \text{ with } Z_j = \frac{\{N_j - mean[N]\}}{stdev[N]} \tag{1}$$

The weighting factor w_j is then defined as:

$$w_j = \left[1 - sig(Z_j)\right]^{c(N^{max})} \tag{2}$$

The power coefficient $c(N^{max})$ is required to adjust the strength of the weighting factor for different event logs. Intuitively, if all traces in a log have similar length, then this adjustment factor should be low, approaching 1; if trace length variance is very high, then the adjustment should be higher.

To define an appropriate value of the power coefficient $c(N^{max})$, a relation between N^{max} and the anomaly detection performance bias should be first found. However, this relation can only be estimated and not optimised because trace length has no upper bound, which would lead to a non-finite state optimisation problem. Therefore, we have estimated the value of $c(N^{max})$ by fitting a non-linear regression using 6 real-life event logs[1] . To model a non-linear regression function, we use the values of $c(N^{max})$ that achieve the highest F1-score in anomaly detection using the 6 different logs in offline settings, i.e., considering all the traces in an event log at the same time in the observation matrix X. Under significance level 0.01, the non-linear equation in Eq. 3 has been fitted with two coefficient parameters a and b as in Table 1.

$$c(N^{max}) = \begin{cases} -2.2822 + (N^{max})^{0.3422} & \text{if } N^{max} > \frac{2.2822}{0.3422} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Table 1: Result of fitted non-linear regression model: $f(x) = a + x^b$

Parameter	Estimate	Standard Error	t value	p-value
a	-2.2822	0.3533	-6.46	0.0030
b	0.3422	0.0191	17.96	0.0001

In the end, using the estimated power coefficient $c(N^{max})$, we define the weighted *leverage*-based anomaly score as:

$$\hat{l}_w(\sigma_l) = w_j \cdot \hat{l}(\sigma_l). \tag{4}$$

3.2 Online anomaly detection

After having defined an anomaly score, the proposed anomaly detection method is complemented by the following four aspects: (i) grace period, (ii) finite memory usage, (iii) update of leverage scores, and (iv) anomaly threshold setting. These are described in detail next.

[1] These event logs belong to the ones made available by the Business Process Intelligence Challenge in 2012, 2013 and 2017

Grace period. Similarly to other online anomaly detection methods in the literature [8,19], for practical reasons it makes sense to begin taking decision on trace anomaly only after having received a sufficient number of events. For this purpose, we introduce the parameter Grace Period (GP), which specifies the minimum number of traces and events per trace that must be received before trace anomaly decisions can begin to be taken. In other words, the GP prevents to run the anomaly detection model at early stages, when a sufficient number of events has not been received yet. In this paper, the GP parameter is defined as the number of traces for which at least 2 events have been received. For example, if GP=100, the anomaly detection starts from the first event after having received at least the first 2 events of 100 different traces.

Finite memory usage. Another condition to be satisfied by an anomaly detection method in online settings is the one of finite memory usage. In principle, events may be infinitely received as time goes by. However, handling an infinite number of events would require infinite memory, which is impossible in practical settings [19,20]. Therefore, to calculate leverage using always a finite number of events received from a stream, we introduce the parameter windows size (W), defined as the number of recent cases that are considered to determine anomalies when a new event is received. More formally, at a given time t, let us refer to E_t as the set of events received until t. Now, if the number of (possibly incomplete) traces represented by events in E_t is more than W, then the earliest traces are removed from the set of traces considered to calculate the projection matrix H. Specifically, events of the trace whose first event is the earliest in E_t is first removed and so on until the number of traces represented in E_t is W.

Update of leverage scores. Each time a new event is received, the leverage of the case to which this event belongs is updated (see Figure 3). More in detail, let us assume that an event $e_{i,j}$ is received at time t. Then, after having possibly removed some cases represented by events in E_t to maintain the finite memory usage assumption, the leverage of the remaining traces $\hat{l}_w(\sigma_j)$ is calculated. The result obtained determines, based on the value of the considered anomaly detection threshold, whether the trace σ_j is considered anomalous after the arrival of event $e_{i,j}$, or not. This procedure is replicated each time a new event is received.

Anomaly threshold setting. The objective of anomaly detection is ultimately to determine whether traces are anomalous or not. Therefore, the problem of anomaly detection can be reduced to a binary classification problem. Based on the anomaly score $\hat{l}_w(\sigma_j)$, a decision should be made whether the trace σ_j is anomalous or not. This is normally done by setting the value of an anomaly detection threshold T for $\hat{l}_w(\sigma_j)$, such that a trace is anomalous if $\hat{l}_w(\sigma_j) > T$. In this work, we consider three constant thresholds and one variable threshold. We consider the constant values $T_{c1} = 0.1$, $T_{c2} = 0.15$, and $T_{c3} = 0.2$. These values are based on our experience in experiments in online and offline settings, where anomalous traces tend to have an anomaly score $\hat{l}_w(\sigma_j)$ greater than 0.1. As variable threshold, we consider the value $T_v = mean_{\sigma_j \subseteq E_t}[\hat{l}_w(\sigma_j)] + stdev_{\sigma_j \subseteq E_t}(\hat{l}_w(\sigma_j))$, which calculates

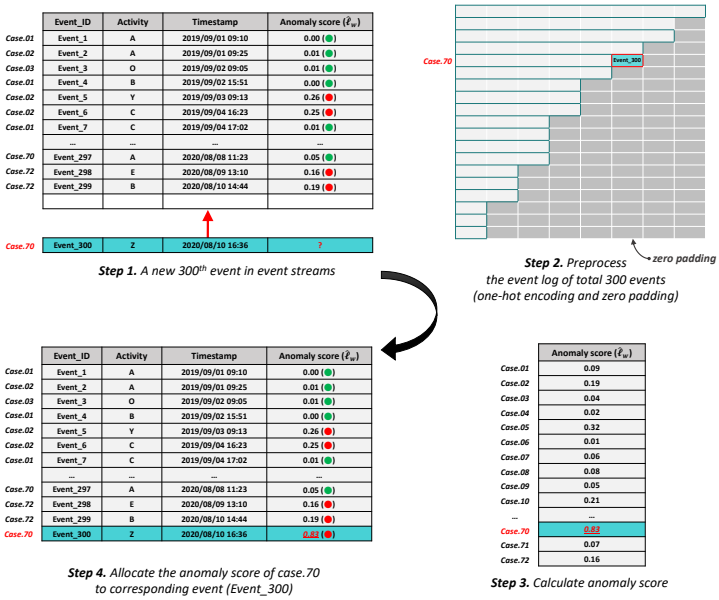


Fig. 3: Example of anomaly detection using a fixed anomaly threshold $T = 0.1$

the threshold based on the mean and standard deviation of the leverage scores of the traces considered in the observation matrix X. A similar principle to set the anomaly detection threshold is used by [16] for the timestamp anomaly detection threshold.

4 Evaluation

This section describes first the datasets that we used for evaluating the proposed online anomaly detection framework. Then, we present the evaluation metrics and experiment settings and, finally, we discuss the performance and computational cost of the proposed framework.

We consider two artificial logs used by [18] and one real-life log publicly available. The artificial logs are generated by simulating 2 process models (Small and Medium in [18]) using the PLG2 tool. Regarding the real log, we consider the *Helpdesk* event log, which contains events logged by a ticketing management system of the help desk of an Italian software company. These logs have been chosen because they have been considered by previous work in anomaly detection and they are also sufficiently small to control the running time of experiments. Descriptive statistics of these logs are reported in Table 2.

Evaluating an unsupervised approach of anomaly detection like the one that we propose requires event logs with labelled traces (normal v. anomalous), which are generally unavailable in practice. Therefore, a common practice in this research field is to inject anomalies using different types of anomaly patterns in

Table 2: Descriptive statistics of event logs (Log statistics are counted after injecting anomalies)

Type	Data	Statistics	Value
Artificial log	Small	-Number of cases	5,000
		-Number of events	44,811
		-Number of activities	20
		-Average # of cases per day	5,000
		-Average # of events per day	44,811
	Medium	-Number of cases	5,000
		-Number of events	29,683
		-Number of activities	32
		-Average # of cases per day	5,000
		-Average # of events per day	29,683
Real log	Helpdesk	-Number of cases	3,804
		-Number of events	13,901
		-Number of activities	9
		-Average # of cases per day	10.94
		-Average # of events per day	18.06

event log and creating labels during the anomaly injection process [1,2,3,16,18]. We consider the 5 anomaly patterns *Skip*, *Insert*, *Early*, *Late*, and *Rework* as defined in [18]: in *Skip*, a sequence of events is skipped in some cases; in $Insert$, one or more events are generated in random positions within existing traces; in $Early/Late$, timestamps of events are manipulated such that a sequence of events is moved earlier/later in a trace; in *Rework*, a sequence of events is repeated after its occurrence. Anomalies are randomly injected in an event log until 10% of the traces in the log have become anomalous. As far as performance measures are concerned, we consider the typical measures for classification problems obtained from the confusion matrix, i.e., precision, recall and F1-score. The datasets used in this paper and the code to reproduce the experiments discussed next are available at `https://github.com/jonghyeonk/OnlineAnomalyDetection`.

We set the GP to 1,000 cases and consider 3 values of window size W, i.e., $W \in [1000, 2000, 3000]$. A larger value of GP and W is likely to lead to better and more stable performance, while also implying a higher computational cost. The experiments are implemented in R on an Intel i7 Linux machine using a single CPU and 5GB memory limit.

Table 3 shows the performance of anomaly detection in event streams for different anomaly detection threshold values and different values of W. Note that the 4 columns T_{c1} to T_v report average performance measures calculated from the start of the streaming (after the GP condition has been reached). It can be noticed that the three constant thresholds show on average better performance than the variable threshold T_v. To better observe a trend of performance improvement as more events are received, the last two columns $T_v^{F:100}$ and $T_v^{L:100}$ show the average performance values calculated on the first 100 events received (after the GP condition has been met) and last 100 events received, respectively. The result shows a clear tendency of increasing performance. The performance is low at the initial stage, and it increases remarkably for the last 100 events received. Particularly in the case of the $Helpdesk$ log, the proposed framework could not detect any anomalous cases in the first 100 events, while the performance clearly

Table 3: Performance of online anomaly detection (average from the start of the stream and calculated only on first/last 100 events)

Data	Window size	Time cost (average sec)	Metric	Threshold					
				T_{c1}	T_{c2}	T_{c3}	T_v	$T_v^{F:100}$	$T_v^{L:100}$
Small	1,000	1.09	Precision	0.22	0.22	0.22	0.21	0.06	1.00
			Recall	0.63	0.62	0.59	0.62	1.00	0.63
			F1-score	0.33	0.32	0.32	0.31	0.11	0.77
	2,000	1.07	Precision	0.26	0.25	0.25	0.25	0.06	1.00
			Recall	0.61	0.60	0.57	0.61	1.00	0.63
			F1-score	0.36	0.36	0.35	0.35	0.11	0.77
	3,000	1.23	Precision	0.75	0.79	0.82	0.75	0.20	1.00
			Recall	0.57	0.55	0.53	0.57	0.17	0.63
			F1-score	0.65	0.65	0.64	0.65	0.18	0.77
Medium	1,000	1.42	Precision	0.16	0.17	0.17	0.15	0.50	1.00
			Recall	0.72	0.71	0.71	0.73	0.29	0.50
			F1-score	0.26	0.27	0.27	0.25	0.36	0.67
	2,000	1.46	Precision	0.37	0.45	0.52	0.26	0.50	1.00
			Recall	0.65	0.64	0.63	0.68	0.29	0.50
			F1-score	0.47	0.53	0.57	0.38	0.36	0.67
	3,000	2.06	Precision	0.28	0.30	0.31	0.25	0.50	1.00
			Recall	0.67	0.67	0.66	0.67	0.29	0.50
			F1-score	0.39	0.41	0.42	0.37	0.36	0.67
Helpdesk	1,000	0.34	Precision	0.06	0.06	0.06	0.06	0.00	0.50
			Recall	0.99	0.98	0.96	1.00	0.00	0.96
			F1-score	0.12	0.12	0.12	0.12	0.00	0.66
	2,000	0.37	Precision	0.08	0.08	0.09	0.08	0.00	0.51
			Recall	0.80	0.74	0.68	0.77	0.00	0.96
			F1-score	0.15	0.15	0.15	0.14	0.00	0.67
	3,000	0.39	Precision	0.09	0.11	0.13	0.10	0.00	0.61
			Recall	0.59	0.49	0.39	0.60	0.00	0.93
			F1-score	0.16	0.19	0.20	0.17	0.00	0.74

increases for the last 100 events. It should be noted that, as the number of events received increases, the performance of the proposed framework is likely to converge to the one showed by the average on the last 100 events. Regarding the window size W, the average time cost increases with the value of W. A larger window size also leads to better anomaly detection performance.

As an example, Figure 4 breaks down the performance of the proposed framework along time, counted as the number of events received, in the case of the Small event log with W=3000. It can be noted that the performance oscillates wildly until 20,000 events are received. After that, the performance tends to stabilise and, while the precision remains high, the recall also begins increasing more regularly. After 30,000 events are received, all the performance metrics appear to have become stable.

5 Conclusions

This paper has presented an approach to detect trace level anomalies in business process event streams using an anomaly score based on statistical leverage. A preliminary evaluation on artificial and real event logs also has been presented. The results obtained in this paper are important to determine the future work in this line of research.

First, the performance in the case of the *Helpdesk* log highlights an issue with anomaly threshold setting. The constant values chosen for the experiment

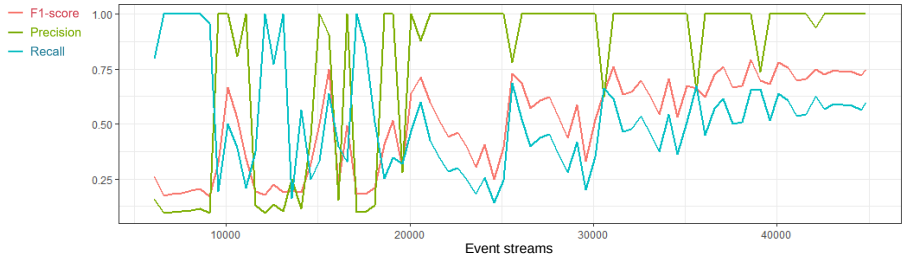


Fig. 4: Performance values as number of events received increases (Small log is applied, using T_v, W=3000)

(between 0.1 and 0.2) appear to be too low for this event log, which results in low precision and F1-score and only high recall. This points to the need for developing an advanced variable threshold that can adapt to the characteristics of different event logs.

Another limitation of the proposed approach, which also impacts the performance, is the fact that it does not distinguish between incomplete and completed traces when calculating the anomaly score. Therefore, many traces may be considered anomalous because they are incomplete, even though they will turn into normal at some point in the future as more events are received. A possible strategy to prevent this is to organise the events received into batches by different prefix length before calculating the anomaly score. This is inspired by [20] that, in the case of online compliance checking, addresses the issue of trace incompleteness using prefix-alignment.

Finally, considering word-embedding instead of one-hot encoding and zero-padding during pre-processing may be likely to reduce the size of the observation matrix X and, therefore, speed up the calculation of anomaly scores.

References

1. F. Bezerra and J. Wainer. Algorithms for anomaly detection of traces in logs of process aware information systems. *Information Systems*, 38(1):33–44, 2013.
2. K. Böhmer and S. Rinderle-Ma. Multi-perspective anomaly detection in business process execution events. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 80–98. Springer, 2016.
3. K. Böhmer and S. Rinderle-Ma. Multi instance anomaly detection in business process executions. In *International Conference on Business Process Management*, pages 77–93. Springer, 2017.
4. A. Burattin. Online conformance checking for petri nets and event streams. In *15th International Conference on Business Process Management (BPM 2017)*, 2017.
5. A. Burattin and J. Carmona. A framework for online conformance checking. In *International Conference on Business Process Management*, pages 165–177. Springer, 2017.

6. A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen, and J. Carmona. Online conformance checking using behavioural patterns. In *International Conference on Business Process Management*, pages 250–267. Springer, 2018.
7. F. Cao, M. Estert, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 328–339. SIAM, 2006.
8. P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.
9. L. Genga, M. Alizadeh, D. Potena, C. Diamantini, and N. Zannone. Discovering anomalous frequent patterns from partially ordered event logs. *Journal of Intelligent Information Systems*, 51(2):257–300, 2018.
10. L. Ghionna, G. Greco, A. Guzzo, and L. Pontieri. Outlier detection techniques for process mining applications. In *International symposium on methodologies for intelligent systems*, pages 150–159. Springer, 2008.
11. D. C. Hoaglin and R. E. Welsch. The hat matrix in regression and anova. *The American Statistician*, 32(1):17–22, 1978.
12. M. Klimstra and E. P. Zehr. A sigmoid function is the best fit for the ascending limb of the hoffmann reflex recruitment curve. *Experimental brain research*, 186(1):93–105, 2008.
13. S. J. Leemans, D. Fahland, and W. M. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *International conference on business process management*, pages 66–78. Springer, 2013.
14. S. J. Leemans, D. Fahland, and W. M. Van der Aalst. Scalable process discovery and conformance checking. *Software & Systems Modeling*, 17(2):599–631, 2018.
15. X. Lu, D. Fahland, F. J. van den Biggelaar, and W. M. van der Aalst. Detecting deviating behaviors without models. In *International Conference on Business Process Management*, pages 126–139. Springer, 2016.
16. H. T. C. Nguyen, S. Lee, J. Kim, J. Ko, and M. Comuzzi. Autoencoders for improving quality of process event logs. *Expert Systems with Applications*, 131:132–147, 2019.
17. T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser. Analyzing business process anomalies using autoencoders. *Machine Learning*, 107(11):1875–1893, 2018.
18. T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser. Binet: Multi-perspective business process anomaly classification. *Information Systems*, page 101458, 2019.
19. G. M. Tavares, V. G. T. da Costa, V. E. Martins, P. Ceravolo, and S. Barbon Jr. Leveraging anomaly detection in business process with data stream mining. *iSys-Revista Brasileira de Sistemas de Informação*, 12(1):54–75, 2019.
20. S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, and W. M. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics*, 8(3):269–284, 2019.
21. S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

Concept Drift Detection on Streaming Data with Dynamic Outlier Aggregation

Ludwig Zellner, Florian Richter, Janina Sontheim, Andrea Maldonado, and Thomas Seidl

LMU Munich, Germany
{zellner, richter, sontheim, maldonado, seidl}@dbs.ifi.lmu.de

Abstract. Many processes no matter what kind are regularly changing over time, adapting themselves to external and internal circumstances. Analyzing them in a streaming context is a very demanding task. Particularly the detection and classification of significant deviations is important to be able to re-integrate these possible micro-processes. Assuming a deviation of a certain process, the significance is implicitly given when a high number of instances contain this deviation similarly. To enhance a process the integration of or preventive measures against those anomalies is of high interest for all stakeholders as the actual process core gets discovered more and more in detail. Considering various areas of application, we focus on previously neglected but potentially significant anomalies like small changes in the disease process of a virus infection that has to be discovered to develop an appropriate reaction mechanism. We concentrate on non-conforming traces of a stream on which we compute a local outlier factor. This allows us to detect relations between traces based on changing outlier scores. Hence, hereby connected traces are clusters with which we achieve the detection of concept drift. We evaluate our approach on a synthetic event log and a real-world dataset corresponding to a process representing building permit applications which emphasizes the extensive applicability.

Keywords: Concept Drift Detection · Local Outlier Factor · Micro-Clusters.

1 Introduction and Motivation

Nearly every established process consists of deviations, which potentially contain valuable information impelling its context. E.g. by considering the Internet of Things as highly attractive for botnet attacks, it is a very interesting task to distinguish, detect and classify attack traces from operations of the ordinary process. The classification task to identify device-threatening traces is pursued by attack triage systems, in which our approach can provide outlier scores. A second example is the analysis of disease spreading processes. Regarding the regulations of the German public health department in the 2020‘s pandemic, symptomatic people have to keep health journals during the course of an infection to record

the timeline of symptoms. These journals are eventually gathered to construct the typical disease process. Fever, dry cough and fatigue count as common symptoms for Covid-19 and represent the aforementioned process. Extraordinary and more rare symptoms are: Loss of the senses of taste and smell, conjunctivitis and the so called covid toes. Having these symptoms occur for some patients, their disease process is non-conforming and a concept drift emerges. To extend the process, the course of treatment and its effects can also be incorporated. In general, the detection and classification of new types of infectious diseases can be done independently of the deeper understanding of its nature and, therefore, this procedure can also be mapped to various other diseases. Regarding business processes, in case of structural changes in the business itself, the question is, which deviations occur in reaction to initial changes. In particular, using a building permit application process, we apply our approach to detect deviating subprocesses as candidates for reintegration after a split of departments.

At large, process mining has its focus on conformance checking for a long time. Particularly considering an anomalous group of traces which nevertheless belongs to the main process, can help to enhance the main process by re-integrating this process deviation. The deviation itself qualifies for further analysis by consisting of multiple similar instances. Hereafter, we refer to an anomaly as a micro-cluster of non-conforming traces i.e. anomalous traces. This micro-cluster is defined by multiple related process instances. The relation between traces can manifest itself in different ways. Therefore, a specific distance between these instances is crucial, which will be defined for our specific case in Section 3.

In this work, we introduce a novel approach to classify non-conforming traces into micro-clusters on trace streams. Based on these clusters we then achieve a more fine-grained concept drift detection. Specifically, the basis of our approach is a stream of traces. As a combination of state-of-the-art methods from process mining and an established algorithm from the field of machine learning and data mining, our approach is very flexible due to its modularity. As mentioned before the field of application is very wide concerning current emergency cases but can also be applied to every other process which consists of interesting and critical deviations. An evaluation on a synthetic event log and the building permit application dataset from the BPI Challenge of 2015 demonstrates the considerable effects of our method.

In Section 2 we analyze the existing work related to clustering traces to groups of anomalies. Section 3 describes the background of our work. Our framework and our algorithmic methods are explained in Section 4. Sections 5 and 6 address the performed experiments and corresponding evaluation. We conclude our work with Section 7.

2 Related Work

The aggregation of outliers by leveraging local outlier factor serves as means to cluster data. In addition, *LOF* also incorporates concepts from density-based

algorithms like *reachability* or *core distance* from the field of data mining [2]. However, a major difference to its original field of application is the goal, which is not defined by finding local outliers, but is to find reductions of local outlierness, which eventually yield a new micro-cluster. First and foremost, the focus lies on finding groups of outliers, which define an anomaly of the underlying process, and detect concept drifts of different degree. Because this anomaly consists of a number of traces, it is highly probable, that there is a hidden micro-process driving those process instances. For that reason, it is an important task to analyze those micro-clusters and recognize the drift of the main process into one of those micro-processes. The underlying idea to our approach is trace clustering [5], which is based on an embedding of traces into vector space. The overall purpose is to split highly diverse processes into homogeneous subsets. Thus, the complexity is reduced, while the interpretability is increased.

In comparison to our approach Richter et al. [9] also use the idea of leveraging a reference model for distance computation between traces. They aim at clustering traces based on their distances to each other. However, the approach requires manual user knowledge to define the agglomerative clustering manually. It is not developed for stream application and does not provide further analysis like concept drift detection as we do.

With TESSERACT[8], concept drifts regarding the time dimension are focused. Interim-times between events of a stream are used to derive a drift indicator regarding sudden and incremental drifts. However, the change of completion times of an event often can have other reasons than indicating a drift. Therefore, we use the conformance of a trace by activity labels as a more direct indication.

Using statistical hypothesis testing Maaradji et al. [6] are detecting drifts between consecutive batches of traces by comparing the distribution of runs statistically. A subsequent paper by Ostovar et al. [7] focuses on the detection of drift within a trace rather than between traces. By using sliding windows and applying the G-test of independence this method even can be applied to event logs which are highly variable. In our approach high variability also is supported resulting in many micro-clusters which are given as means of drift categorization.

3 Preliminaries

In this section we describe the background of the methods, which will be harnessed in Sec. 4. These include a definition of trace streams, conformance checking, the method to compute distances between traces and the procedure behind the popular local outlier factor from the field of machine learning. *Case identifiers*, *activity labels* and *timestamps* are the minimal components of event logs. Their counterpart on streams can be defined similarly. In an event stream S an event e is emitted continuously in a specific order, duration and by reference to a certain *case identifier*. That means, that the context, in which the event is emitted, is known. However, the differences between an event log and an event stream is (1) the potentially *infinite* sequence of events and (2) the *incompleteness* of a case, which means that at a certain point in time a case does not have

to be necessarily completed, since a new event could possibly be executed in context of this case [11]. Additionally the timestamp at which an event is emitted depends on the stream. Thus, it can be neglected and we define $\#_{case}(e) = c$ and $\#_{act}(e) = a$ for $e = (c, a)$. Because of the different structure this type of data needs a different handling compared to conventional process discovery. Hence, we refer hereby to state-of-the-art methods in the literature and assume incoming data to be given as traces $\#_{trace}(c) = \hat{c}$ consisting of occurring events. Furthermore, we check every trace for conformance against the reference model and additionally against every micro-cluster model in each iteration. Thus, we utilize a preferably fast but efficient conformance checking method. We decide to apply token-based replay [10] as it has an advantage in speed compared to alignments [1]. Furthermore, to concentrate on the applicability of our approach, we focus on incremental and recurring drifts in Sec. 5 and Sec. 6.

To be able to provide every trace with a comparable distance property we utilize the reference model by following the approach of Richter et al. [9]. By interpreting the model as a map, we define the *geodesic distance* between two traces as the average number of edges on the shortest paths between every vertex of two traces. A transition is only counted as a vertex, if it is one of the transitions causing problems in the conformance checking procedure. The distance computation follows the approach in [9] and is defined by the following formula where X and Y are traces and x and y are transitions within:

$$dist(X, Y) = \frac{1}{|X| \cdot |Y|} \cdot \sum_{x \in X} \sum_{y \in Y} shortest\_path(x, y)$$

Essentially, every trace is reduced to its events, which cause problems on the reference model. Thereafter, the reduced traces are compared pairwise by computing the geodesic distance between every contained event. After we average the result we get the required distance between two traces, which is also known as average-linkage. The ulterior motive is, that for deviating traces one or multiple transitions can be determined, which lead to the low fitness value. These transitions posses different distances to each other when the missing transitions are filled up by the reference model and the amount of edges between the transitions are counted. Thus, traces deviating with similar problematic transitions have a lower distance as deviating traces with very different problematic transitions. This part as another module can be substituted by any other computation method as well, which possibly leads to different results. The main part of our novel method is the usage of the popular *Local Outlier Factor* developed by Breunig et al. [3] from the field of machine learning. This algorithm is constructed to detect local outliers in a dataset. Therefore, every trace is assigned a degree (LOF), which describes its outlierness with respect to the surrounding neighborhood. The problem of different types of outliers, i.e. global versus local is described in [3]. Local outlierness, therefore, is defined relatively to the neighborhood of a data point. We leverage this perspective in our work by analyzing traces with respect to their surroundings. These neighboring traces can be arbitrarily distant to each other. If the average reachability distance, described

below, between the neighbors is similar to the reachability distance of the trace itself, it gets reflected in a LOF of ~ 1.0. The LOF of a trace σ and the local reachability distance lrd is computed in the following way:

$$LOF_k(\sigma) = \frac{\sum_{\gamma \in N_k(\sigma)} \frac{lrd_k(\gamma)}{lrd_k(\sigma)}}{|N_k(\sigma)|}, \quad lrd_k(\sigma) = \frac{|N_k(\sigma)|}{\sum_{\gamma \in N_k(\sigma)} reach - dist_k(\sigma, \gamma)}$$

The variable k denotes the number of nearest neighbors surrounding σ, where the set of these neighbors is called k-neighborhood ($N_k(\sigma)$). and lrd refers to the local reachability distance: where *reachability–distance* is either the distance of γ to its k-th nearest neighbor or the real distance between γ and σ depending on which distance is greater. Analogously, if $LOF_k(\sigma) \gg 1.0$ the trace is an outlier.

4 Dynamic Outlier Aggregation

Our novel approach combines conformance checking from Process Mining with Local Outlier Factoring from the field of Data Mining to achieve the automated detection and classification of deviating traces in streams. The emerging micro-clusters are then used to detect concept drift on the aforementioned streams. This procedure differs from [9] to the extent that we aim for concept drift detection on streams. Richter et al. aim at clustering deviating traces by using a reference model given by the main process. The resulting micro-clusters represent deviations with increased potential due to its density. Furthermore, they do not provide means to the reader to automatically cluster or even classify new upcoming data. Trace clustering on deviating traces could also be achieved by our approach but it is not subject to this paper. After conformance checking of incoming traces to both the reference and micro-models, only the non-conforming traces are analyzed.

Our approach can be split into four modules (see Fig. 1), i.e. initial process discovery, conformance checking, LOF computation and micro-cluster aggregation. The former has to be done once for initiation to prepare a process model as a reference for further computations. The latter three steps are processed repeatedly after initiation. At first, in the initiation phase, a main model

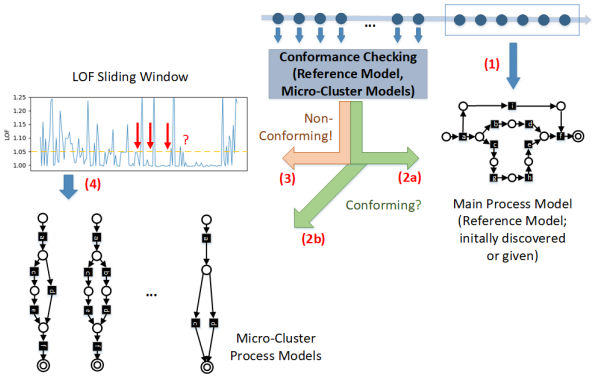


Fig. 1. Overview of our approach. Steps 1-4 denote the successive procedure to detect anomalous clusters.

is discovered, which serves as a reference model (Fig. 1 (1)). This reference model can be normative or declarative but the premise is that it does not yet hold traces, which actually should be considered non-conforming. Afterwards, in the iteration phase, the conformance of the incoming traces is checked and filtered by non-conformance, which means that we only analyze non-conforming traces in the following (Fig. 1 (2a and b)). Therefore, we use the popular method of token-replay as a conformance checking step. Here, the reference model is used to compute the fitness of a trace. For this purpose, the four counters *produced, consumed, missing* and *remaining* are used as described in Sec. 3 to determine, if a trace fits a model or not. Every conforming trace then is removed and the next steps are only based on the set of non-conforming traces. In the next step, we compute a local outlier factor (LOF) on every trace (Fig. 1 (3)). Gradually, the latest incoming traces are filtered in the aforementioned way and the local outlier factor of every stored trace is re-computed. Those computations serve as a snapshot, at which we decide, if there is a relation between two or more traces. This decision is based on the fact, that the scores of those traces drop below a given clustering threshold at a specific snapshot. For those corresponding traces a process model is discovered (Fig. 1 (4)). These micro-cluster models then are used as additional reference models, against which every incoming trace is checked before it is labelled non-conforming (Fig. 1 (2b)). Since the non-conforming traces are not removed from the set of local outlier factors, the computation of it has an increase in accuracy over time until it reaches the size of a sliding window. This sliding window is used to limit the complexity of re-computations. Alg. 1 depicts the described procedure in pseudocode. We assume that the streaming events are already gathered to traces. This means, that the procedure, in which we wait for a trace to start and end, is transferred to already existing efficient algorithms [8]. Furthermore, the outcome of this procedure does not have to be completely accurate since our approach can cope with inaccuracies arising in this step. In addition, it is assumed that the process model, which we use as a reference, is already discovered, solely relies on conforming traces and only has to be prepared for further usage. Another possibility is to provide a normative model by a domain expert. The iteration phase comprises repeating steps mainly consisting of the novel approach we call *Dynamic Outlier Aggregation*. This aggregation refers to the grouping of traces. Thus, if there is a number of traces, of which the LOF scores are below a constant threshold, which T exceeds a given constant number K, these traces are aggregated to micro-clusters. To compute the LOF for each trace we utilize a matrix holding all distances between every trace in the window. We start with the first trace having a distance of 0.0 to itself. For every trace being emitted afterwards by the stream we compute the distance between the current and every other trace in the sliding window based on their transitions causing problems on the reference model. The pairwise combination of the traces and the cartesian product between all these problematic transitions of a pair form the basis for further computation. The average of the overall hop count of the shortest paths between every aforementioned transition yields our resulting lower triangular matrix, which then is extended to a sym-

Algorithm 1 Dynamic Outlier Aggregation

Input: The emission interface of a stream SI of single traces i, a lower bound L and a sliding window size W, MinPts K for LOF computation and a threshold T, below which a trace is assigned to a micro-cluster

Output: A collection of micro-clusters represented by event logs and a LOF for each trace

$R \leftarrow discover\_reference\_model()$ ▷ Assumption: Traces are conforming
$MC \leftarrow \emptyset$ ▷ MC denotes the set of micro-clusters
$IBT \leftarrow \emptyset$ ▷ IBT denotes the set of non-conforming traces with LOF below threshold
$M \leftarrow \emptyset$ ▷ M denotes the distance matrix
while $|SI| > 0$ **do**
  Get next i
  if $!fitsR(i) \wedge !fitsM(i)$ **then** ▷ Check conformance with reference model and micro-cluster models
    $M \leftarrow compute\_distances(IBT, i)$
    $IBT \leftarrow token\_replay(R, i)$ ▷ Save activities with problems
    $M \leftarrow sym(M)$ ▷ Create symmetric matrix from M
    if $|rows(M)| > L$ **then**
      $IBT \leftarrow IBT[1:]$ ▷ Remove first element
      if $|rows(M)| > W$ **then**
        $M \leftarrow delete\_first\_row\_col(M)$
      end if
      ▷ Separate traces forming a micro-cluster from outliers
      $micro\_cluster \leftarrow separate(IBT, lof, T)$
      if $|micro\_cluster| \geq K$ **then**
        $MC \leftarrow MC \cup micro\_cluster$
      end if
    end if
  end if
end while

metric matrix for further computation. Besides, if in the next iteration step the number of distances exceeds W, the oldest trace is removed and so on. Thus, we can use the pre-computed distance matrix to return LOF for every trace. Because the LOF of every trace potentially changes in each iteration step, we have to separate the possible micro-clusters from the outliers. On that account, a threshold T is required, which indicates the affiliation of a trace to a micro-cluster. Hence, every trace, which possesses a LOF below T, is returned in a separate log. However, the connected trace and LOF is not yet removed from the matrix. The reason behind it is the increasing accuracy of LOF the more traces are located in a micro-cluster. Since the computational complexity of LOF is $\mathcal{O}(W^2)$, the overall complexity of *Dynamic Outlier Aggregation* is $\mathcal{O}(W^3)$ in a naive implementation. However, the theoretical complexity analysis based on stream data seems high, the actual complexity strongly depends on the window size, which covers only a small portion of the data. Thus, by choosing a fixed window size, which should be preferably small compared to the expected emitted data, the goal of our approach can be achieved in constant time ($\mathcal{O}(1)$).

5 Evaluation on a Synthetic Log

The proposed approach can be inspected and reproduced on GitHub[1]. The implementation is used to experiment with both synthetic and real-world logs. In

[1] https://github.com/zellnerlu/DOA

addition to the applicability on real-world data, we focus on the detection of recurring drifts as an example. Furthermore, the results of the usage within a fixed parameter set are compared. Moreover, the limitations of our approach are discussed.

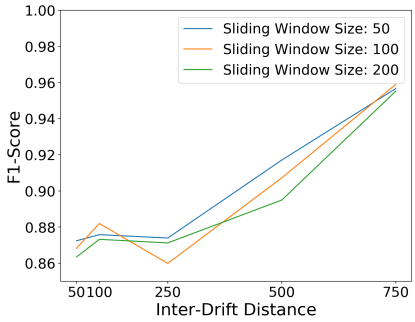
As a proof of concept we create an event log consisting of 1000 traces with the *Processes and Logs Generator* (PLG2) by Burattin [4] with which it is also possible to produce an evolution of a given process. Thus, we start with a rather small main process consisting of 6 activities and 2 XOR-gateways. Additionally, 3 deviation processes with the size of 1000 traces each are created by using the aforementioned option and the given parameters such as *depth, AND/XOR branches, AND/XOR weight etc.* are adjusted accordingly, such that the probability of occurrence is de- and increased. We inject the drifts, by splitting up and interleaving parts of the deviation processes with parts of the main process. Thus, an event log is produced, which consists of all 4 processes appearing successively. Here, we concentrate mainly on recurring drifts as an example to illustrate the process of detection. The scalability of our approach is evaluated by measuring the required time of the trace from comparison to the other traces in the sliding window until the clustering step. Furthermore, we utilize *F1-Score* as a quality measure, which is defined as the harmonic mean between precision and recall. Since our approach heavily relies on the creation of process models in different steps of the framework, we globally measure precision and recall instead of using a multi-class perspective.

5.1 Execution Times

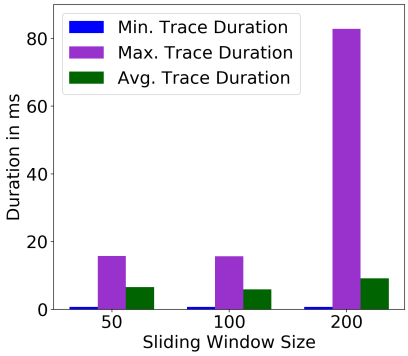
We perform the experiments on an Intel Core i7 with 3.2Ghz and 32GB RAM. The operating system is Windows 10. The measured time required for the distance computation to the other traces in the sliding window as well as the clustering procedure varies between 1ms and 20ms with an average duration of about 7ms. In Fig. 2b the application of a sliding window of size 200 shows an outlier in terms of maximum duration, which indicates the increasing time requirement with larger sliding windows. Nevertheless, this shows the applicability at least on trace streams and the approach also qualifies for an extension to event streams due to the fast processing.

5.2 Impact of Inter-Drift Distance and Sliding Window Size

At first we analyze the detection quality by varying the distance between drifts. This means, that we vary the size of the interleaved parts, namely inter-drift distance, by 5-75%, which means that 50, 100, 250, 500 and 750 traces of every event log are arranged in sequence. This is repeated until log completion. We choose LOF plots and Gantt Charts for visualization, where the x-axis shows the global trace identifiers, i.e. trace IDs within all emitted traces and the y-axis represents the LOF of a trace (LOF plot), i.e. the micro-cluster affiliation (Gantt Chart). It is also perceivable, that the LOF of some traces is highlighted with a certain color, i.e. the bars in the Gantt Charts are depicted with a certain color. This



(a) Analyzing the performance on various inter-drift distances by number of interleaved traces and the application of different sliding window sizes.



(b) Execution times by using different sizes of sliding windows. Min. Trace Duration is between 30 and 60 microseconds.

Fig. 2. Quality measure and execution time depending on the sliding window size

color represents the affiliation of the trace to a certain micro-cluster model. As it is shown in Fig. 2a, the resulting *F1-Score* increases in all three cases similarly, when increasing the inter-drift distance. This is due to the decreasing number of changes, which have to be detected. Thus, the probability of detecting *False Positives* decreases, as well. Fig 2a shows a continuously high detection quality, which also can be confirmed visually in Fig. 3. One can see, that micro-cluster

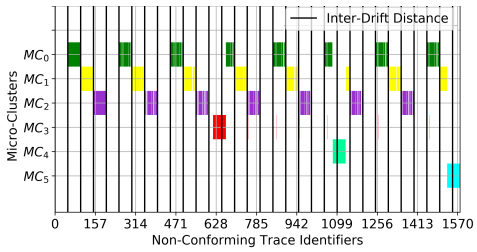


Fig. 3. Recurring drift detection on a a synthetic log with inter-drift distance of 50. The change points are emphasized with vertical black bars.

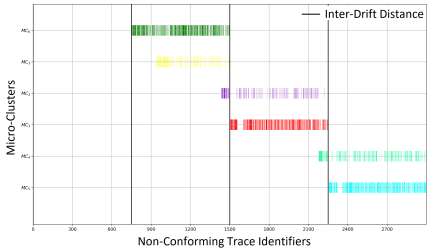


Fig. 4. Resulting Gantt Chart on an inter-drift distance of 750 and a sliding window size of 100.

0-2 are reliably representing the injected deviations from the main model. Interestingly, our approach regularly detects additional micro-clusters (3-5) beyond change points, which neither resemble recurring drifts nor were injected by design. This could be for reason of overly similar traces, which also neither do

fit to the reference model nor to one of the micro-cluster models. Nonetheless, the visualization shows a clear detection of recurring drifts, where false micro-clusters can be simply excluded and the remainder can be analyzed by a domain expert further on. When we increase the size of the sliding window by keeping a relatively small lower bound L, we get a result as shown in Fig. 4. Here, the recurring drifts are still detected very well, but two micro-cluster models, e.g. green and yellow or purple and red, share one injected cluster each. In the latter pair another red cluster besides the first one is created, because the purple cluster was initially created before the change point at trace id 1500. This issue brings some limitations to light. On the one hand, to estimate a fitting sliding window size is difficult as a minimum of $K \leq W$ of LOF scores have to drop below T to be detected as a cluster. Therefore, a high difference between K and W leads to the detection of very small micro-clusters and a low difference leads to a very low sensitivity against concept drifts. On the other hand, some test execution have to be made to estimate a fitting threshold. In the experiments it appears, that T is preferably set right below the average LOF. Additionally, the current implementation poses the problem that concept drift detection is constrained to the control-flow dimension. To include other perspectives like time or resources the approach has to be altered.

6 Evaluation on the Event Log of BPIC 2015

As it is already introduced in Sec. 1, we use data from the Business Process Intelligence Challenge 2015[2]. It consists of building permit applications of five Dutch municipalities with a total number of 5649 traces. Due to the four year period, in which changes were regularly made to the rules and regulations, an incremental concept drift is expected in the data. For the first experiment we use the first log with 1199 traces. Fig. 5 shows a snapshot of our analysis in LOF-plot format, which reveals the direction of the resulting Gantt-Chart. One has to be aware, that appearing traces in this plot are already classified non-conforming to both the reference and all other micro-cluster models. The colors show, that clusters are aggregated, which are temporally overlapping. This implies the affinity for each outlier aggregation to a different, successive cluster. Thus, incremental drifts are detected as new micro-clusters are consecutively created.

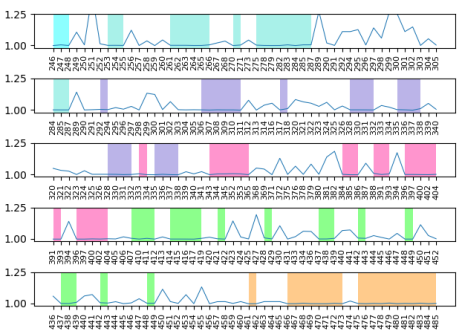
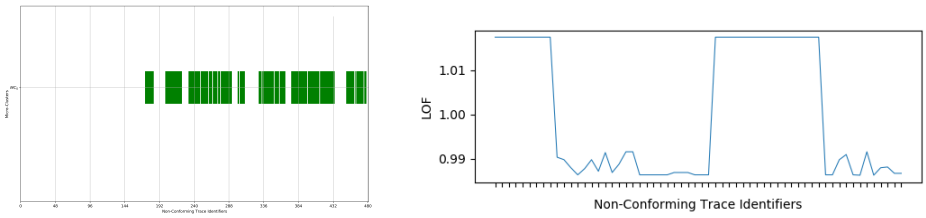


Fig. 5. Snapshot of five overlapping sliding window states. Every trace ID (x-axis) is assigned a LOF (y-axis). Background colors represent assignments to micro-clusters.

[2] https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1

Because BPIC 2015 is of high variability regarding the activity domain, traces inherently posses a rather high distance in between. Increasing the sliding window size does not lead to the expected micro-clusters in the data already holding an incremental drift but the loss in performance was relatively high. Thus, to show the applicability to on small recurring drifts, we extend the log to serve as a hybrid by repeatedly injecting data unrelated to the process. We randomly use 5-25 traces with an inter-drift distance between 25-100 to alter the log. Following this strategy, we simulate a changed process based on real-world data already comprising realistic noise and inherent drifts. The first 100 traces are used to create the initial reference model. We discover this model with a high dependency threshold to force many traces to be classified as non-conforming. In this setup, we set the sliding window size to 60, the lower bound to 35 and the offset to 0.005. The Gantt Chart in Fig. 6a marks all non-conforming traces,



(a) (b)

Fig. 6. Micro-cluster detection on hybrid log of BPIC 2015. The x-axis shows the non-conforming trace identifiers in both figures and the y-axis shows the micro-cluster affiliation (Fig. 6a) and the *LOF* (Fig. 6b).

which are aggregated to a micro-cluster and the aggregation is derived by high difference between the *LOF* of each trace in Fig. 6b. By simulating the basic process as a recurring drift in otherwise noisy data, we cluster the BPIC 2015 as one micro-cluster. An important aspect, which has to be considered is, that real noisy traces in the basic process also get filtered. This leads to a more pure version of the BPIC 2015 as a micro-cluster.

7 Conclusion

With our novel *Dynamic Outlier Aggregation* we detect concept drifts of different extent. Trace clustering of non-conforming traces is used as a first step. Afterwards, we focus on the detection of recurring drifts as an example. Using a sliding window approach allows us to discover changes of different magnitude in the underlying process. Depending on the size of the traces these parameters provide control over the results as we exchange processing time for accuracy. In respect of a process like the spread of a disease, changes happen in a very different time span. For example, we are constantly in need of knowledge about mutating viruses like influenza or other more recent ones. Thus, the origin and classification of a newly mutated kind is of high interest for the development of

a vaccine. Furthermore, if the sliding window is chosen with an appropriate size, and the underlying reference model consists of enough information of former viruses, even a rapid and significant change of a virus, is detected.

In future works, we will look at the application of this approach to single events instead of traces. The advantage of working on events instead of traces is the processing of incomplete traces with a higher rate. This involves the output of a micro-cluster probability for every emitted batch of connected events. In addition, we will be working on an appropriate solution to re-integrate clustered anomalies including a high amount of traces. This will also lead to great benefits for real-world applications, since unknown deviations are thoroughly analyzed, and the main process is purposefully extended.

References

1. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-based fitness in conformance checking. In: 2011 Eleventh International Conference on Application of Concurrency to System Design. pp. 57–66 (June 2011)
2. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. In: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data. p. 49–60. SIGMOD '99, Association for Computing Machinery, New York, NY, USA (1999)
3. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: Identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. p. 93–104. SIGMOD '00, Association for Computing Machinery, New York, NY, USA (2000)
4. Burattin, A.: PLG2: multiperspective process randomization with online and offline simulations. In: Azevedo, L., Cabanillas, C. (eds.) Proceedings of the BPM Demo Track Co-located with the 14th International Conference on Business Process Management, Rio de Janeiro, Brazil. vol. 1789, pp. 1–6 (2016)
5. Greco, G., Guzzo, A., Pontieri, L., Sacca, D.: Discovering expressive process models by clustering log traces. IEEE Transactions on Knowledge and Data Engineering **18**(8), 1010–1027 (Aug 2006)
6. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Fast and accurate business process drift detection. pp. 406–422 (08 2015)
7. Ostovar, A., Maaradji, A., La Rosa, M., ter Hofstede, A.H.M., van Dongen, B.F.V.: Detecting drift from event streams of unpredictable business processes. In: Comyn-Wattiau, I., Tanaka, K., Song, I.Y., Yamamoto, S., Saeki, M. (eds.) Conceptual Modeling. pp. 330–346. Springer International Publishing, Cham (2016)
8. Richter, F., Seidl, T.: Tesseract: Time-drifts in event streams using series of evolving rolling averages of completion times. Information Systems **84** (11 2018)
9. Richter, F., Zellner, L., Sontheim, J., Seidl, T.: Model-aware clustering of nonconforming traces. In: Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R. (eds.) On the Move to Meaningful Internet Systems: OTM 2019 Conferences. pp. 193–200. Springer International Publishing, Cham (2019)
10. Rozinat, A., Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**, 64–95 (03 2008)
11. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Event stream-based process discovery using abstract representations. Knowledge and Information Systems **54**(2), 407–435 (Feb 2018)

OTOSO: Online Trace Ordering for Structural Overviews

Florian Richter, Andrea Maldonado, Ludwig Zellner, and Thomas Seidl

Ludwig-Maximilians-Universität München, Munich, Germany
{richter, maldonado, zellner, seidl}@dbs.ifi.lmu.de

Abstract. Identifying structures in data is an essential step to enhance insights and understand applications. Clusters and anomalies are the basic building blocks for those structures and occur in various types. Clusters vary in shape and density, while anomalies occur as single-point outliers, contextual or collective anomalies. In online applications, clusters even have a higher complexity. Besides static clusters, which represent a persistent structure throughout the whole data stream, many clusters are dynamic, tend to drift and are only observable in certain time frames. Here, we propose OTOSO, a monitoring tool based on OPTICS. OTOSO is an anytime structure visualizer, that plots representations for density-based trace clusters in process event streams. It identifies temporal deviation clusters and visualizes them as a time-dependent graph. Each node represents a cluster of traces by size and density. Edges yield information about merging and splitting trace clusters. The aim is to provide an on-demand overview over the temporal deviation structure during the process execution. Not only for online applications, but also for static datasets, our approach yields insights about temporally limited occurrences of trace clusters, which are difficult to detect using a global clustering approach.

Keywords: Trace Clustering · Visualization · Operational Support · Anytime Clustering.

1 Introduction

The ongoing digitalization of industries and social systems creates a strong demand for analysis tools to transform data into useful insights. Especially early warning systems for already known issues or still uncovered problems are highly requested. However, without a thorough exploration of the data, those systems cannot be developed, since we need to know what we are looking for beforehand.

In online applications, the time for analysis is always very precious and never sufficient. Therefore, an in-depth analysis has to be postponed, as interesting and promising aspects have been identified. A more shallow high-level analysis is more suitable as a time-efficient first exploration.

In the field of clustering, DBSCAN[4] is a prominent technique for density-based clustering. However, finding good parameters to generate results that

leverage the data into a given story is very tedious. Restarting clustering algorithms with arbitrary parameters is very different from output-driven experimentation. Therefore, OPTICS[1] was proposed as an extension, that offers a two-dimensional visualization for any multidimensional dataset. In OPTICS plots, the structure of the data is abstracted and parameters for density-based clusterings are visually determined.

In an online process mining application, we need to increase the abstraction level even further. Anytime variants for DBSCAN and OPTICS have been proposed in literature already. However, the structure of an online process is not covered by observing an event stream and building an up-to-date process model. The time perspective provides clusters with a further dimension of volatility.

In the context of processes, we differentiate between the major behavior, the baseline process, and process variants with deviation behavior. During the process execution, the baseline stays mostly static and rarely tends to shift its behavior. In contrast, variants often traverse different lifecycles dynamically. They emerge at certain points in time, merge with other variants, separate again and disappear eventually. In some time intervals, variants can remain inactive and reappear seasonally or randomly later.

In this work, we propose OTOSO, an on-demand temporal structure visualization of event streams. It is based on OPTICS and developed to cope with dynamic structure transformations. OTOSO collects trace data from an event stream as temporal deviation signatures, generates temporary OPTICS plots and aggregates their information into a graph plot. This plot shows relations between baseline and variant clusters. In a quick analysis, structure changes are identified visually. Each cluster is represented as a node of a specific size at a point in time. Relations between clusters are indicated by edges between nodes. The whole plot can then be interpreted as a map, that show the dynamic changes of the process during the event stream.

2 Related Work

To the best of our knowledge, there is no direct competitor that proposes an anytime structure overview for event streams. However, there are related methods that have to be mentioned here. There is a plethora of published techniques regarding process discovery, conformance checking and clustering. Due to space constraints, we only mention works that have a focus on temporal perspectives or which work on event streams.

Event stream monitoring emerges as a required preprocessing step for anytime analyses. Works in this field mainly prepare intermediate data for process discovery[3, 9, 7] and conformance checking[2, 13]. These works propose methods to analyze event streams, which is the more complex task in comparison to trace stream analysis. The latter paradigm assumes that events are already grouped into traces, which is mostly a difficult requirement. In many practical scenarios, there is also a strong concurrency between cases. Cases can become inactive

or are stopped without any further information. An approach based on event streams has to come up with a heuristic to deal with the lack of information.

In the area of temporal anomaly detection, Rogge et al.[12] analyzed interim times between events by applying kernel density estimation to identify outliers in the temporal perspective. In [11], the authors identify such outliers of event pairs online by using hashing for event collecting and applying z-scoring to define an in-control area for unsuspicious event relations. In [10], this idea is leveraged on the trace level to detect collective trace anomalies using density-based clustering on temporal deviation signatures. We adapt the presented clustering technique for OTOSO.

The area of event stream concept drift detection contains more established works. In [6], Hassani elaborated the idea of [7] to detect work-flow-based concept drifts using different structural metrics on process models. In [8], the authors present a technique to change forecasting models according to changed environments due to concept drifts. However, we are not aware of any concept drift detection approaches taking the temporal perspective into account.

3 Preliminaries

An event stream $S : \mathbb{N} \to \mathbb{N} \times A \times \mathbb{N}$ is a mapping from natural numbers to the event domain. Each event $e = (c, a, t)$ consists of an case identifier $c \in \mathbb{N}$, an activity label $a \in A$ and a timestamp $t \in \mathbb{N}$. For case identifiers from another domain, there is typically a canonical translation into the natural numbers. The same holds for the timestamps. In the following, we will not distinguish between cases and case identifiers, as the context provides enough clarification.

Since OTOSO can also be applied to event logs, we define an event log as a finite multiset of events. Although an event log is mostly grouped by case identifiers, for OTOSO the log should be sorted by timestamp. Additional event attributes like resources are ignored in this work, although they might enhance the results in future works.

Next, we call tuples of two activities $(a_1, a_2) \in A^2$ relations. A relation (a_1, a_2) exists in a case c, if there are two events $e_1 = (c, a_1, t_1)$ and $e_2 = (c, a_2, t_2)$ with $t_1 < t_2$. We canonically define the mean μ and variance σ of all time intervals in a finite set of cases for a certain relation. Using z-scoring as follows we account for the imbalance between all different relations and define the temporal deviation signature as:

$$TDS^c(a_1, a_2) = \begin{cases} \frac{|t_2 - t_1| - \mu_{(a_1,a_2)}}{\sigma_{(a_1,a_2)}} & , e_1 = (c, a_1, t_1), e_2 = (c, a_2, t_2) \in c \\ 0 & , \text{otherwise} \end{cases}$$

In case of multiple occurrences of a relation, the average z-score is used. A distance is a positive-definite function, that is symmetrical and fulfills the triangle inequality. In the following, we use the Euclidian distance due to its popularity and will not go into detail about other functions in this work. For the

clustering step, we require a measure of density. Density is defined by a number of objects n in a certain area of radius ε. If an object, here a case represented by its temporal deviation signature, contains at least $MinPts$ many objects within a neighborhood $N_\varepsilon(c)$ of radius ε, this case is a core object. All cases within the neighborhood are at least border objects, if their neighborhood is not dense enough to be core objects themselves. All remaining cases are noise.

One of the most popular density-based methods is DBSCAN[4]. It selects objects and classifies them depending of their neighborhood as core, border or noise points. For a more in-depth description, we point to the corresponding work of Ester et al. A major drawback of DBSCAN is the difficulty to choose an appropriate value for the neighborhood distance ε. To overcome this issue, Ankerst et al. developed OPTICS[1]. Given $MinPts$, this method determines for each object its core distance, the minimal distance needed such that the ε-neighborhood contains $MinPts$ many objects. Derived from the core distance, the reachability distance between two objects is computed then. According to this distance, the processing order is depending on the nearest neighbor that has not been processed yet. This 2D reachability plot uses the ordering on the x-axis and the reachability distance on the y-axis. Since dense object clusters in the data space have low pairwise reachability distances, they are accumulated in the plot and clusters are identified as troughs in the reachability plot. Using a horizontal line as a density threshold, all troughs below this level represent clusters using the height as the according ε-value.

4 OTOSO

OPTICS visualizes the cluster structure of a static dataset. However, especially in process mining, process behaviors are dynamic and cluster structures are likely to change. To visualize not only a snapshot in a particular time frame, but the evolution of process variants and trace anomalies, we propose OTOSO, which is briefly summarized a visual time series of trace cluster structures. OTOSO consists of two phases. First, the event stream is observed and the necessary statistics are collected. By using a hashing data structure, the data is provided for the second module on-demand. At any point in time, the stored data can be queried as input for OPTICS to produce the current temporal cluster structure in the recent event stream. All those individual clustering snapshots are used to iteratively plot the clustering overview for the whole event stream.

4.1 Monitoring Temporal Deviations

OTOSO uses an event stream as input. In contrast to trace streams, the events have to be collected individually before case statistics can be extracted. A major problem of stream input is that we can never be sure that a case is still active. Therefore, we need an aging mechanism to discard old cases without the certainty that they are canceled or just paused and will be continued later.

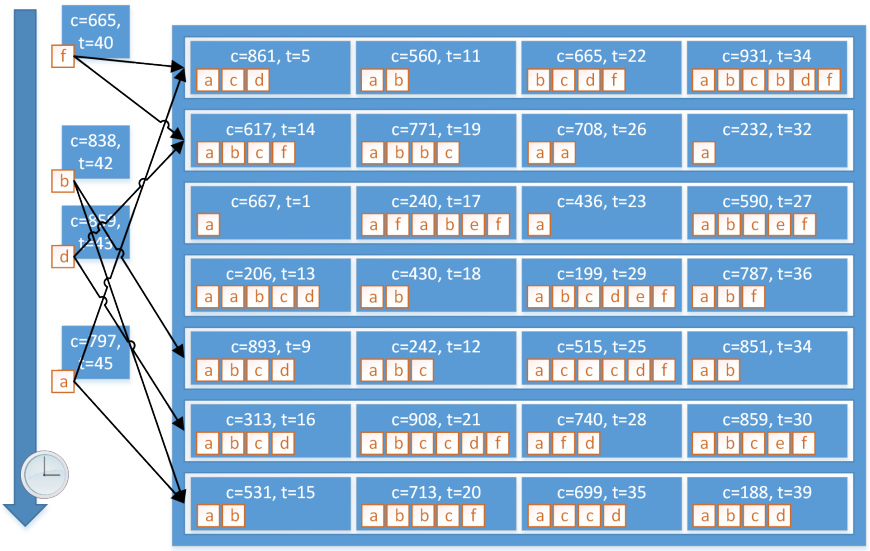


Fig. 1. Example hash table with $h = 7$ and $w = 4$. Each observed stream event has two potential rows to store it. Since the table is already full, either an event can be appended to its corresponding trace or an old trace has to be discarded. Do not be confused with the activity labels, since complete event information is stored.

We utilize Cuckoo-Hashing as it already provided a useful discarding technique for StrProM [7]. A hash table of height h is filled with case data, that is the last timestamp, the case identifier and all observed events. Two hash functions are applied on the case identifier to determine two potential hash table cells for each case. Instead of storing the case data directly in the hash table, we store a small and finite collection of cases in a cell. Technically, this width w of the table is implemented using arrays. Thus, the decaying factor can be adjusted without corrupting the operation complexity.

For each observed event, both hash functions are applied to identify all potential storage cells. If the case is already stored, it is updated by adding the event and setting the last-modified timestamp. In Fig. 1, the stream event in the top left corner belongs to case $c = 665$. A potential storage option is in the first hash table row. The case is already present in this row at the third position. We can update this cell by appending the event and updating the timestamp to $t = 40$. If the case has not been stored yet, we replace the case with the stored case, that has the oldest last-modified timestamp. The replaced case is the least recent one in this hash table cell. We try to insert it in the secondary position. Either, the secondary position has empty space, or we replace it again with the oldest case in this position. The procedure is recursively repeated until the secondary position has only more current entries and we discard the current item. Considering Fig. 1 again, the second stream event with $c = 838$ has storage options for row 5 and 7. Neither holds data for this case already. Using the first

option, we attempt to store this new case in the fifth row, depending on the first of both hash functions. The oldest case here is case $c = 893$. We replace it with the new case and try to re-insert $c = 893$. The timestamp $t = 9$ tends to be already deprecated, however, since there are older entries in the table, there might be a chance to discard it after a series of replacements. This would be the case, if the alternative storage position is in row 1 or 3. Otherwise, the already existing timestamps in the remaining rows are newer and case $c = 893$ is discarded.

With this strategy, the hash table is always a finite representation of the recent cases, however some older behaviors potentially survive in the data structure since the swap operations regard the table only partially. Another drawback is that events in the beginning of cases are represented excessively, as the chance to be discarded is increased for longer cases. Alternatively, the length of the case can be included in the discarding mechanism. Nevertheless, this gives older cases an advantage to be kept stored, since smaller and recent cases are discarded. To the best of our knowledge, a perfectly fair sampling for event streams is still an open research topic, so we accept the drawbacks and discard by recency only.

Regarding hash functions, there are various ways to implement a set of two functions. Most programming languages provide at least one built-in hash function. To derive a second one, it is mostly sufficient to reverse the case identifier and use the same function again. Another strategy splits the identifier in two chunks and uses the hash value for the first and for the second chunk to determine both positions. We did not perform an in-depth evaluation on this topic here.

4.2 Structure Analysis

The hash table provides at most $h \cdot w$ many cases at any point in time t. The cases do not have to be completed already. The complete hash table is processed to extract the case data and to generate the z-scored temporal deviation signatures for all cases, which is used as input for OPTICS to cluster the traces. The output gives an impression on the recent temporal trace clustering structure. For the stream structure overview, we extract all clusters depending on the chosen density parameters $(\varepsilon, MinPts)$. For each cluster C, we create a node at position $x = t$ and $y = \sum_{c \in C} \mathrm{core}_{\varepsilon, MinPts}(c)$ which is the occurrence time and aggregated cluster density. The size of each node is depending on the number of contained cases in the cluster respectively the number of cluster elements that are currently stored in the hash table.

In the basic variant, OTOSO connects cluster nodes if the distance between cluster centers is below the distance threshold Δ_{TDS} and the nodes occur in consecutive time slots. The extension connects cluster nodes of distant time slots. This allows to identify temporally limited clusters that reoccur after a period of inactivity. In Fig. 2, OTOSO is applied to an event stream producing OPTICS plots for various timestamps. At a tickrate of $10k$ events, further intermediate results are requested. For four of these intermediate queries, we show the OPTICS plots in the top row of the figure. For each OPTICS plot, a vertical slice

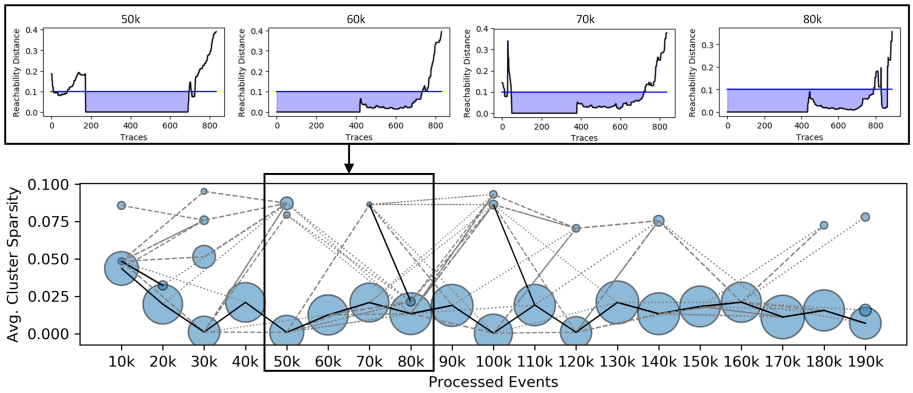


Fig. 2. OTOSO applied to an event stream. Each slice correcponds to a point in time and a hierarchy of clusterings at this timestamp.

in the OTOSO plot below is generated. Typically, a process produces one major cluster containing cases that behave ordinary. These are the large spheres in each slice. For the $50k$ mark, besides the major cluster, two variants of low density are active. Both are related to previous queries, but disappear for the next two queries. Solid lines indicate a strong similarity between clusters of consecutive clusters. Dashed lines indicate similarity between slices over a larger timeframe. Here, we only include lines connecting slices within a timeframe of $30k$ events. In slice $60k$, all variants disappear. In $70k$, a small variant emerges. It has some similarity with the major cluster in $50k$, but no connection to the major cluster in $60k$. Hence, the temporal deviation profile first covered this deviation, but the variant did not occur in the succeeding process window. Interestingly, the small cluster in slice $80k$ grows slightly in size, but drastically in density. Regarding the solid line, we recognize a close similarity between both clusters, so their behavior represented by the temporal deviation signature is also similar.

This visualization allows to detect different structural changes in an event stream. Lifecycles of emerging and vanishing variants can be followed as illustrated before. The connections of a cluster node indicates, whether this variant has disappeared or has been inactive for some time. If a node emerges without initial connection, the corresponding variant starts suddenly. Otherwise, a connected new node hints towards a gradually emerging variant. These mechanisms are related to types of concept drift, however it is difficult to clearly label the effects according to sudden, gradual and incremental drifts due to the complexity of an event stream. Many activities and therefore activity relations are included in the temporal deviation profile. Nevertheless, the OTOSO plot gives an overview over the whole structure. A sudden drift, for instance, will likely affect a small number of traces and will maybe only affect some activities. The abstraction level of the visualization is to high to register concept drifts with a high confidence, except they appear as large-scale effects.

5 Evaluation

In the following, we evaluate the correlation between the size of the hash table and the currency of the collected event data. Afterwards, we show the benefits of applying OTOSO in comparison to using density-based clustering on the data as a static data chunk. Finally, we build a stream of a sequence of event logs to show the capability to detect the transitions between dissimilar event stream sections. We uploaded OTOSO into a GitHub project[1], thereby the experiments can be reproduced.

5.1 Datasets

Working with pure synthetic datasets causes some issues concerning the detection simplicity of anomalies or clusters in the data. We need datasets that are realistic, because synthetic datasets allow too much freedom and often are unfairly beneficial to the method's evaluation. Therefore, we utilize the BPI challenge datasets from 2015[2] and 2017[3], in the following abbreviated as BPIC15 and BPIC17. BPIC15 contains data of building permit applications over four years in five Dutch municipalities. Five partitions show the process of each municipality individually. Each sublog contains about a thousand cases. The challenge of this dataset lays in its about 400 activities and its resulting complexity from the large number of potential relations. The publications regarding this challenge show, that there is a high similarity between sublog 1, 2 and 5 while sublog 3 and 4 represent a slightly different behavior. In BPIC17, a loan application process of a Dutch financial institute over one year is logged. The offer log contains only a subset of 24 offer related activities. 128985 events are recorded in 42995 cases. Due to its larger size, we are able to simulate an online observation of the whole fiscal year.

5.2 Hash Table Size

We use BPIC17 to investigate the influence of the hash table size on the currency of the data. Each event log is transformed into an event stream. Observing the stream event by event, each recent event is inserted into the hash table. Every 1000 events, we determine the average time difference to the current event timestamp. In Fig. 3, we show the results. Starting with a small hash table, which only contains 1000 cases, we compare three different dimensions for the table. In the first case, a table of height 10 with 100 buckets in each position is used. The second hash table has height 100 and width 10, while the third is a one-dimensional table of height 1000. The average recency is below 10 days. Towards the end, no new cases are starting, so no old cases are discarded and the table gets slightly outdated.

[1] https://github.com/Skarvir/OTOSO
[2] https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
[3] https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

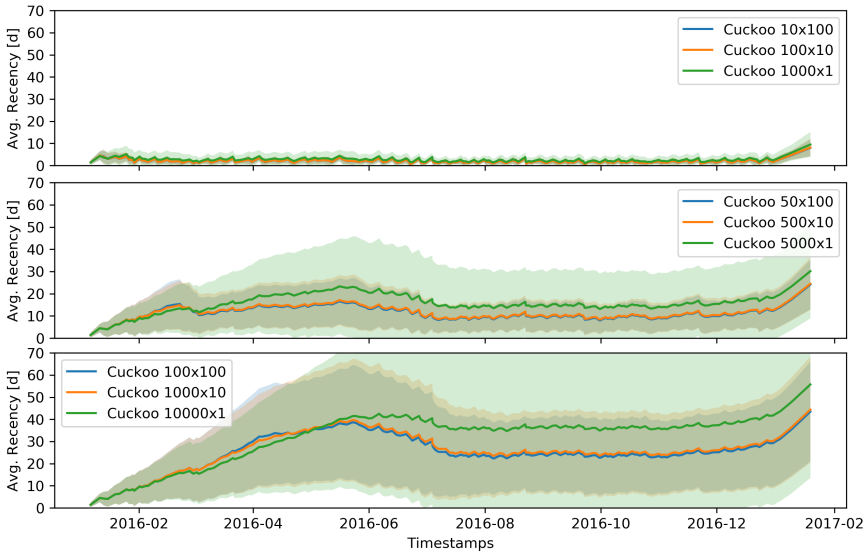
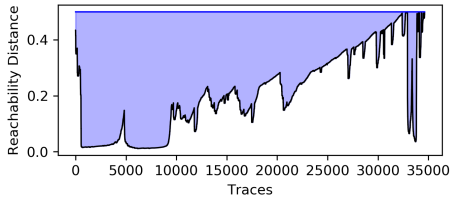


Fig. 3. Avg. recency and standard deviation is given for nine Cuckoo hash tables with different dimensions as $height \times width$.

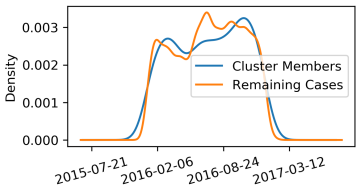
The second plot shows three hash tables of size 5000 having analogous changes regarding their dimensions. Due to the higher capacity, more cases can be stored and the table contains more obsolete items. Storing more items leads to a more stable clustering and following techniques are affected by noise or short-term outliers. There is no clear method to determine the best recency and the corresponding table size, since this is completely depending on the user-defined time window and the arrival frequency of events and cases. Finally, the application is also an important factor, since the detection of point-wise anomalies benefits from higher currency while the detection of long-term structures requires data with high stability. However, the important point we want to highlight is the advantage of using a two-dimensional hash table. The width allows shorter rehash cycles, which is already shown in [5, 7]. The new insight here is the greater recency for small numbers of buckets in each position. Already in the second plot, but much clearer in the third one with a hash table of size 10000, the one-dimensional hash table has a delay of about 40 days, while both variants with few buckets have smaller temporal shifts. The difference between using 10 or 100 buckets is rather marginal. Therefore, we recommend using small numbers of buckets, since the iteration over a large list of buckets is more time-consuming than rehashing at another position.

5.3 Static Clustering vs. Dynamic Clustering

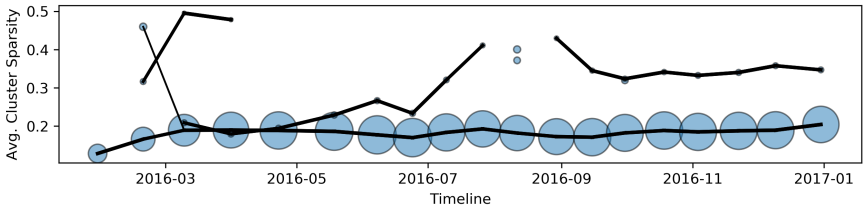
The BPIC17 dataset contains a significant cluster with deviating temporal behavior, that contains accepted offers with a delay in its execution. In Fig. 4a, we



(a) OPTICS plot for BPIC17.



(b) KDE for BPIC17.



(c) OTOSO plot for the event stream of BPIC17.

Fig. 4. OPTICS and OTOSO applied on the BPIC17 datalog. $MinPts = 100$ and results are yielded each $10k$ events.

show the result of OPTICS applied to the whole event log using the temporal deviation signatures as a representation. Using a neighborhood size of 0.5, two major clusters are yielded. The largest one contains the majority of cases and represents the baseline of this process. The second largest one is shown in OPTICS as a thinner and deep trough on the right side. Since this method yields a static overview over the temporal clustering structure, we would assume that the cluster is omnipresent during the complete event stream.

In Fig. 4c, the final OTOSO plot is given. After all events in the stream have been processed, the clusters are nodes with radii according to the number of contained cases. The height is determined by their density. Lines indicate a strong similarity between consecutive clusters. Thus, by following a line we observe the lifecycle of a specific cluster.

In the beginning, the results are not reliable. Many cases have been collected only partially yet. As a rule of thumb, we recommend to neglect insights from the first k cases if the hash table has size $k = h \cdot w$. Hence, starting with April, a baseline of large clusters has been emerged and retains an almost constant size for the remaining stream. More interesting is the other line above. It indicates a much smaller cluster, that still has a high density. During August the cluster vanishes but returns again in September. Instead, two new and dissimilar clusters emerge for this short period and vanish afterwards again. To show what OTOSO has highlighted there, we extract all cases contained in the previously mentioned deviating cluster. This set of cases corresponds to the thin and deep trough in Fig. 4a. For this cluster and also for the remaining cases, we plotted the starting times as a kernel density estimation in Fig. 4b. Here, we observe a peak in

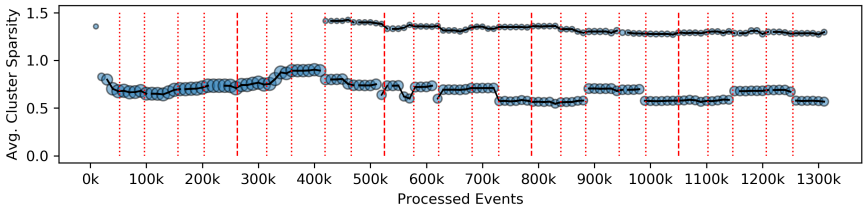


Fig. 5. OTOSO applied to a five-fold concatenation of all five BPIC15 sublogs. $MinPts = 100$ and an intermediate result is demanded every $10k$ events.

starting cases in August. The rising number of arriving cases, which do not belong to the variant cluster, shifts more weight towards the baseline cluster and the two new variants. The resulting loss in density for our previous variant cluster leads to its disappearance for one observation tick. While it is possible to detect such effects with static methods, this analysis is quite tedious. Besides, we already knew what we were looking for. OTOSO highlights this anomaly during the online observation of the event stream. In applications, that require short reaction times, observing the OTOSO visualization provides a very quick indication for an abnormal behavior.

5.4 OTOSO on Event Stream with Concept Drifts

Finally, we use the BPIC15 dataset to how concept drifts affect the structural overview. The dataset is quite small, so we concatenate all five sublogs into one larger event log. Further, we concatenated this event log 5 times with itself to create an even larger log with five segments or 25 sublogs. This event log is then transformed into an event stream.

In Fig. 5, the OTOSO plot is given after processing the event stream. As discussed before, we neglect the results from the first two segments of the stream. After $500k$ events have been processed, the hash table is filled sufficiently and the structure of the data starts to appear. The red lines indicate the border points when a sublog ends and a new one starts. Especially in the last two segments, there is a significant similarity in BPIC15 between sublog 1, 2 and 5 and also between 3 and 4. The black similarity line indicates this relation. There is a much sparser and small cluster above. We do not have expert knowledge to verify or explain its meaning. On the one hand, it is possible to neglect it due to its sparsity. On the other hand, this cluster exists in all sublogs and it shows a strong similarity. In reality, we would recommend a thorough examination, but due to the lack of expert knowledge, we have to dispense with further speculations.

6 Conclusion

In a world of continuously emerging digitalization, it is very important to get preliminary insights early and with a high level of abstraction. OTOSO provides

an online overview over structures in an event stream. Emerging or vanishing clusters are visually identified and lifecycles of those structures are tracked.

Although some structural dimensions are monitored like density, size and similarity of clusters, process data contains more information, which can be used to augment the structural overview plot. Also, the plot depends on suitable user-defined parameters. Estimating good parameters is a very difficult task. Thus, and because a data stream cannot be replayed, it is beneficial to enable on-demand parameter adaptations while results are visualized.

References

1. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: ordering points to identify the clustering structure. ACM Sigmod record **28**(2), 49–60 (1999)
2. Burattin, A., Carmona, J.: A framework for online conformance checking. In: International Conference on Business Process Management. pp. 165–177. Springer (2017)
3. Burattin, A., Sperduti, A., van der Aalst, W.M.: Control-flow discovery from event streams. In: 2014 IEEE Congress on Evolutionary Computation (CEC). pp. 2420–2427. IEEE (2014)
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. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo filter: Practically better than bloom. In: Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. pp. 75–88 (2014)
6. Hassani, M.: Concept drift detection of event streams using an adaptive window. In: ECMS. pp. 230–239 (2019)
7. Hassani, M., Siccha, S., Richter, F., Seidl, T.: Efficient process discovery from event streams using sequential pattern mining. In: 2015 IEEE symposium series on computational intelligence. pp. 1366–1373. IEEE (2015)
8. Maisenbacher, M., Weidlich, M.: Handling concept drift in predictive process monitoring. In: 2017 IEEE International Conference on Services Computing (SCC). pp. 1–8. IEEE (2017)
9. Navarin, N., Cambiaso, M., Burattin, A., Maggi, F.M., Oneto, L., Sperduti, A.: Towards online discovery of data-aware declarative process models from event streams. In: 2020 International Joint Conference on Neural Networks. IEEE (2020)
10. Richter, F., Lu, Y., Sontheim, J., Zellner, L., Seidl, T.: Toad: Trace ordering for anomaly detection. In: 2020 International Conference on Process Mining (ICPM). pp. 1–8. IEEE (2020)
11. Richter, F., Seidl, T.: Looking into the tesseract: Time-drifts in event streams using series of evolving rolling averages of completion times. Information Systems **84**, 265–282 (2019)
12. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. In: International Conference on Business Process Management. pp. 234–249. Springer (2014)
13. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.: Online conformance checking: relating event streams to process models using prefix-alignments. International Journal of Data Science and Analytics **8**(3), 269–284 (2019)

Performance Skyline: Inferring Process Performance Models from Interval Events*

Andrea Maldonado, Janina Sontheim, Florian Richter, and Thomas Seidl

Ludwig-Maximilians-Universität München, Munich, Germany
{maldonado, sontheim, richter, seidl}@dbs.ifi.lmu.de

Abstract. Performance mining from event logs is a central task in managing and optimizing business processes. Established analysis techniques work with a single timestamp per event only. However, when available, time interval information enables proper analysis of the duration of individual activities as well as the overall execution runtime. Our novel approach, performance skyline, considers extended events, including start and end timestamps in log files, aiming at the discovery of events that are crucial to the overall duration of real process executions. As first contribution, our method gains a geometrical process representation for traces with interval events by using interval-based methods from sequence pattern mining and performance analysis. Secondly, we introduce the performance skyline, which discovers dominating events considering a given heuristic in this case, event duration. As a third contribution, we propose three techniques for statistical analysis of performance skylines and process trace sets, enabling more accurate process discovery, conformance checking, and process enhancement. Experiments on real event logs demonstrate that our contributions are highly suitable for detecting and analyzing the dominant events of a process.

Keywords: interval events · performance analysis · process mining · dominant duration path · skyline operator

1 Introduction

To plan a process optimally, prevent mistakes as well as to answer questions about its performance, we need to know the process. The more substantial the knowledge, the better informed decisions users can take about their plan of action. Over a century ago Gantt charts [1] were introduced to schedule work according to resources in the manufacture industry. Since then data-centric process mining models aid to understand constantly changing processes by considering both, their prescription and the posterior description of run instances, in multiple fields. Often certain tasks in a process last long, without the user knowing whether that duration is expected or not. By taking performance indicators of the time dimension, such as the lead-, service- and waiting time into account, performance analysis examines event data over time [2].

* Supported by Munich Center for Machine Learning

This analysis enables businesses to discover performance patterns, optimize processes as well as to identify and prevent mistakes in them. In many cases, analyzing processes that contain big amounts of events often includes computational and visual overhead for the user, specially when only a subset of them might be interesting to asses the user's question. Furthermore including additional performance information for more substantial knowledge may worsen the visual charge. To alleviate this, we select a subset of events that may be specially interesting using the skyline operator [3], considering that events of dominant duration are crucial to process performance, and moreover optimization, resources usage and task prioritization.

Consider the following example: After a vacation visit to your favorite city, you write a review about the hotel you stayed in. This review is added to a platform's collection, from which hotel reputation companies forward customer feedback to hotels. Broadly speaking, this process ingests customer reviews from multiple sources, does multiple transformations, aggregates them and stores the result at a given location to provide hotels with an accurate rating. Since ratings may strongly influence guests booking decisions, an up-to-date result is essential to a host's reputation in the hospitality market. If all events of this process are executed sequentially, they all directly contribute to the overall duration of the process. For this reason it is often shorten by executing events parallely. Knowing events inter dependencies, service- and waiting times is advantageous to choose which activities should run in parallel. Independent events or events series of similar length can be run in parallel to make the process more efficient. Improving a process that already utilizes parallel event executions requires performance analysis of previously ran instances on a activity level. Focusing and speeding up activities that last considerably longer than others might have a higher impact on the overall duration than doing so on the ones that already perform relatively well on the same trace. Thus identifying these activities is key, specially for traces with a high amount of events and high service time deviation. Our performance skyline approach highlights events that dominate others in some given metric on one trace, in this occasion the metric is the service duration.

The next section, presents state-of-the art methods to analyze process performance as the performance spectrum miner and the critical path method. Subsequently, Section 3 presents interval events, the skyline operator and the geometric interval representation. These concepts form the basis of our methods. Section 4 and Section 5 introduces our contributions: The geometrical process representation, the performance skyline and three statistical analysis techniques. Following then, Section 6 demonstrates how our approaches work beyond theory by experimenting with them on real process logs from TrustYou GmbH, a German Guest Feedback and Hotel Reputation Software company. Lastly Section 7 closes listing achievements and further expansion possibilities as future work on this topic. Code and real log data for replicating our experiments are available on the open source project [https://github.com/andreamalhera/performanceskyline].

2 Related Work

Performance models regard temporal aspects of processes. The performance spectrum miner [2] maps all observed flows between activities together regarding their performance over time. Bringing a temporal perspective into process analysis, performance spectrum enables reliable pattern recognition for batching behavior. Nevertheless, using only one timestamp the performance spectrum forces models to overlook some aspects like waiting time, actual event duration and actual trace duration i.a. service time. Not extracting this knowledge restricts models and pattern detection methods derived from it. Other models, as for example PROM's dotted chart [4], use two-dimensional space projections with start time in the horizontal axis and case ids in the vertical axis to describe processes. Nevertheless it is also limited by using a single timestamp. Event intervals [5] have proven useful to extract insights about the idle periods of processes even from events of a single timestamp, but are limited by the assumption that all tasks occur sequentially.

When logs provide additional time interval information, performance insight may be mined using e.g. interval events. Heuristics miner for time intervals [6, 7] uses interval events to mine the dependency relations among activities in a process more precisely. Similarly to transactional events [8] with *transaction types* like *start* and *complete*, interval events have already been defined by others [2, 6, 7, 9, 10], but slightly differently than in this paper. In the first one instance of an activity, which starts and ends, is described as two transactional events of the same activity. Other definitions of interval events implicitly assume that an event ends exactly when the next one begins. Another kind of multi-timestamped events are queue events in a single station queue log [11], which are used to predict delays in service processes online and thus improve customer experience. Being highly adapted to queues, queue events are not suitable to answer other performance questions nor to handle more complex processes, containing non-sequential activities as well. Disregarding complexity and in other cases provided multiple timestamps, hinders to identify gaps between two events.

Flow analysis [12] is a family of control-flow model based techniques to estimate the overall performance of a process given some knowledge about the performance of its tasks. These promising techniques can be extended [13] to also mine performance relations between a set of events and the overall process. Moreover another possible extension could focus on finer granularity for the dominating tasks regarding a given heuristics. Flow analysis is also restricted by process complexity, working properly for control-flow based models and furthermore block-structured models [8] only. Even when one disposes of logs containing interval events, it is a challenge to find suitable ways to integrate additional timestamps for events in a model including as much information as necessary but without generating an visual overload. Comparably the non-control-flow based model, multi-channel performance spectrum for predictive monitoring, [10] classifies cases using multiple performance-related dimensions, yet intra-case features of individual cases remain undiscovered due to its relatively coarse granularity and inter-case design.

The critical path method (CPM) [12, 14–16] also filters interesting points from a process. The critical path comprehends the longest series of dependent events required from start to end, which add up to the overall trace duration [16]. Applying the CPM, identifies the critical path in petri-nets and precedence network process models [17] by noting *estimated/early* and *late*, start time and end time for each activity. Broadly this methods demonstrates how combining state-of-the-art process mining with other approaches results in suitable opportunities. Even though, real processes often present deviation in their duration. Since this approach only considers constant duration of activities across traces and focuses on precedence rather than process performance, analysis could be enriched by performance mining and statistical inspection of time deviating events.

3 Preliminaries

3.1 Interval Events

A process instance can be split into events. An **event** $e = (c, a, t) \in \mathbb{N} \times A \times \mathbb{N}$ is as an tuple consisting of case id c, an activity id a, and a timestamp t. If any two events contain the same case id, they belong to the same trace. A trace is an instance of a process, containing multiple events. An **interval event** $e = (c, a, t^+, t^-) \in \mathbb{N} \times A \times \mathbb{N} \times \mathbb{N}$ is as an tuple consisting of case id c, an activity id a, a start timestamp t^+ and an end timestamp t^-. The duration of an event e_i its duration can be computed as $(\pi_{t^-}(e_i) - \pi_{t^+}(e_i))$, where $\pi_k(e_i)$ is the value for key k in event e_i.

3.2 Skyline Operator

In the field of database queries, the Skyline Operator [3] filters out a set of interesting points from a potentially large set of data points. Whether or not a point is interesting depends on metrics given by the user and if a point is not dominated by any other. It can be used for example to find interesting hotel matches, meaning all hotels that are not worse than any other hotel in nearness to the beach and price. We call the line connecting the set of dominating points of interest: the dominant path. To the best of our knowledge skyline operators have not been used in the field of process mining before. Mostly because often control-flow models are used to analyze a process. For the purpose of performance analysis in processes we consider the dominant duration path of a trace, which comprehends the series of events, which last the longest in a trace or process from start to end and add up to the overall trace duration, similar to dominant points of interest presented in [3]. Consider the following example: A trace is composed of two events, A and B, both starting at t_1. Additionally, A ends at t_3, and B ends at t_2 with $t_3 > t_2$. Thus, the overall duration of the trace is $(t_3 - t_1)$ and only A is part of the dominant path. Consequently, to decrease the process duration based on this trace, event A needs to be sped up. Decreasing the duration of only event B would not improve the overall performance, it is not part of the dominant path and its duration $(t_2 - t_1)$ is lower than $(t_3 - t_1)$. For our approach the skyline operator was implemented using Allen's interval terms [18]: The performance skyline includes all events without *during* relationship to any other event in the same trace.

3.3 Geometric interval representation

In a geometric interval representation [20] temporal intervals are projected to points in a two-dimensional space, as in Fig. 1. At the top of both subfigures of Fig. 1, a schematic representation of a series containing 4 events A, B, C and D is depicted. Below it events are projected as points in a two-dimensional space. Using start and end time as axes. E.g. event A starts at t_1 and finishes at t_3; event B starts at t_2 and finishes at t_7; event C starts at t_4 and ends at t_5; and event D starts at t_6 and finishes at t_7, lasting for $(t_7 - t_6)$ time units.

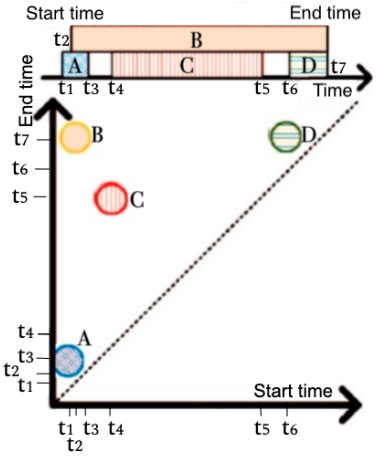


Fig. 1. Schematic representation of four example intervals depicted in the geometric interval representation with start time and end time as axes. Ad. [19].

4 Performance Models for Interval Events

4.1 Geometrical Process Representation

Process interval events as defined previously in Section 3.1 contain temporal intervals, thus can be visualized in the geometric interval representation. Figure 2 describes an example trace, which entails four events similar to Fig. 1 with their corresponding start; end timestamps and a different activity each. Events A, B, C, and D are connected through a line marking they correspond to the same trace. Event B lasts the longest in this trace, since it is furthest away from the zero-duration diagonal. In contrast, the event with activity A appears to have the lowest duration. Events on the same vertical, start at the same time; those sharing one horizontal position end at the same time and if a line passing through two events is parallel to the zero-duration diagonal, they last the same.

4.2 Performance Skyline

The **performance skyline** ρ_c of a trace σ_c is the largest sub sequence of events $\rho_c = (e_1, \cdots, e_i \ldots, e_j, \ldots, e_n)$, where $\rho_c \subseteq \sigma_c$, and $\pi_{t^-}(e_i) \leq \pi_{t^-}(e_j)$ for all $1 \leq i \leq j \leq n$. Additionally, because a trace of interval events is ordered based on the start timestamps $\pi_{t^+}(e_i) \leq \pi_{t^+}(e_j) \Leftrightarrow i \leq j$. For the reason that events in the performance skyline in the case of start time and end time as axis are those which directly contribute to the overall duration of a process at any given point, they are equivalent to the set of events on the dominant path, previously presented in Section 3. If $e_i \in \rho_c$, there is no other event, which starts before $\pi_{t^+}(e_i)$ and ends after $\pi_{t^-}(e_i)$. The performance skyline of the trace in Fig. 2 is depicted in Fig. 3. In this example $\rho_{T_1} = \{A, B, D\}$ compose the performance skyline. C does not belong to the performance skyline, because even though $\{C, B\} \in \sigma_{T_1}$ and $\pi_{t^-}(C) = t_5 \leq t_7 = \pi_{t^-}(B)$, also $\pi_{t^+}(C) = t_4 \not\leq t_2 = \pi_{t^+}(B)$.

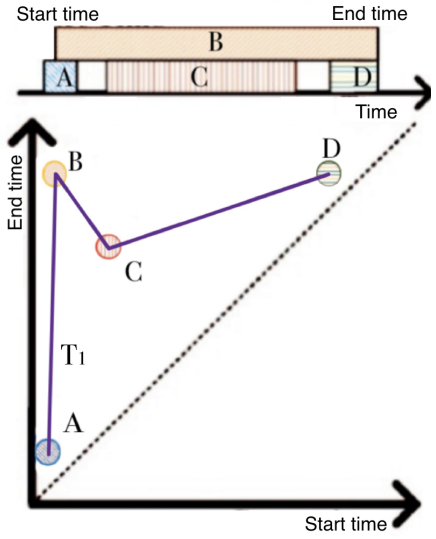


Fig. 2. Schematic representation of one example trace with four activities as intervals in [19]

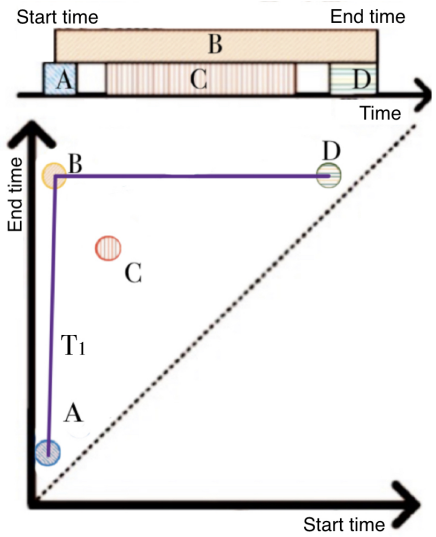


Fig. 3. Performance skyline for example trace T_1 in Fig. 2. Ad. [19].

To understand a process we analyze a representative set of traces. To include and compare events from several traces of the same process with each other, these are aligned to the left by subtracting the start time value $\pi_t^+(e_1)$ to all timestamps $\pi_t^+(e_i), \pi_t^-(e_i)$ for all depicted events in the same trace. This way for every trace ρ_c the first aligned event e'_1 starts at $\pi_t^+(e'_1) = \pi_t^+(e_1) - \pi_t^+(e_1) = 0$ and ends at $\pi_t^-(e'_1) = \pi_t^-(e_1) - \pi_t^+(e_1)$. Any other aligned event e'_i starts and ends relatively to it, computed $\pi_t^+(e'_i) = \pi_t^+(e_i) - \pi_t^+(e_1)$ and $\pi_t^-(e'_i) = \pi_t^-(e_i) - \pi_t^+(e_1)$ correspondingly. Thus events that usually start at a certain time after the whole process starts are easier to compare with each other. From here on in this paper, events in all presented traces are aligned.

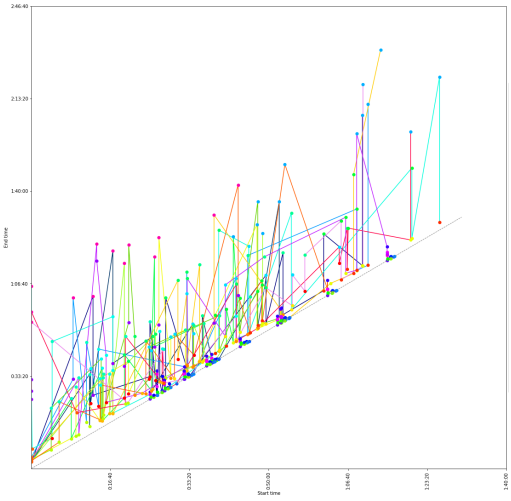


Fig. 4. Real log snippet from industry process with 390 event points of 30 activities, with corresponding point colors, from 13 traces represented by different line colors. Horizontal axis shows starting times within the first 01:06hrs and vertical axis shows ending times between 00:05hrs-02:00hrs.

Figure 4 shows a sample of a real log containing 390 events describing 30 activities on 13 traces of a process. Points in similar positions and same color represent similar activities. Detected patterns between traces identifies behavior

that could be expected from future traces of the same process. In this case similarities between traces can be observed in the peak often on the last event on most traces depicted as a blue point. Even so, contemplating all events of several traces simultaneously challenges recognizably in the visualization and burdens performance with computational overhead. Selecting only a subset of interesting events, e.g. those on the dominant path, to form a baseline of expected behavior for a trace set eases its comparison between traces as well as with future ones. For this purpose statistical analysis techniques will be introduced next.

5 Statistical Analysis Techniques

Methods in this section generalize the process analysis by considering multiple traces in the same performance skyline model and furthermore depicting stochastic summaries of these traces in the plot. Results visualized in this section originate from real logs.

5.1 Average Trace Skyline

The **average trace** $\bar{\sigma} = \{\bar{e}_{a_1}, \ldots, \bar{e}_{a_i}, \ldots, \bar{e}_{a_n}\}$ of a process trace set is the resulting trace of averaging all events start and end times for each activity on the trace set, i.e. for every activity the start time results in $\pi_t^+(\bar{e}_{a_i}) = \frac{1}{m}\sum_{j=1}^{m} \pi_t^+(e_j)$ and the end time in $\pi_t^-(\bar{e}_{a_i}) = \frac{1}{m}\sum_{j=1}^{m} \pi_t^-(e_j)$, where $\pi_a(e_j) = a_i$. An average trace is suitable as a comparable expectation for inquires that involve all activities of a trace set. It eases the view to gain representative knowledge about all activities start and end times as well as the relationships between consecutive activities. Figure 5 shows the average trace of the depicted trace set in Fig. 4. Comparing the form of the average trace to the trace set's, e.g. peaks in the 4th and last activities of the average trace with the ones of similar color in the trace set, a common behavior of the underlying process is revealed.

Moreover combining the average trace and the performance skyline in the *average trace skyline* results in a description of a representative dominant path for multiple traces of a process. For further details on the dominant path, see Section 3. The **average trace skyline** of a process is the performance skyline of the average trace. An average trace skyline is suitable to evaluate performance expectations for a trace set because it facilitates gaining knowledge about activities that are often part of the dominant path and their relations to each other. Figure 6 shows the average trace skyline of the trace set in Fig. 4, which is the performance skyline of the trace in Fig. 5. This average trace skyline highlights five out of thirty activities, which are part of the dominant path. With this information, the user can focus on those sparing them of unnecessary visual and computational overhead.

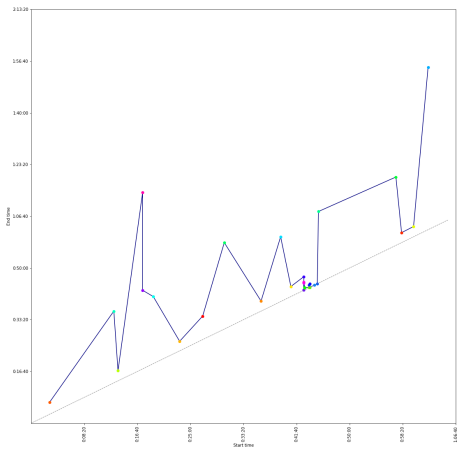


Fig. 5. Average trace from Fig. 4 with 30 activities. Horizontal axis shows starting times within the first 01:06hrs and vertical axis shows ending times between 00:05hrs-02:00hrs.

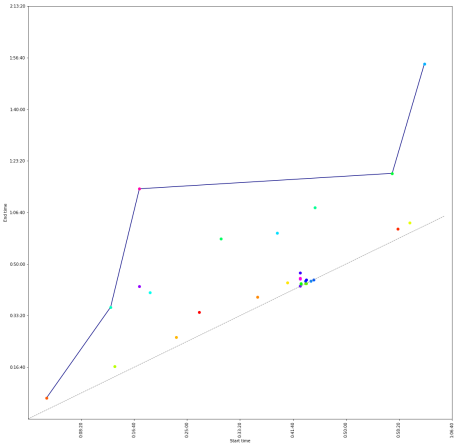


Fig. 6. Performance skyline with five activities on the dominant path marked by line. Horizontal axis shows starting times within the first 01:06hrs and vertical axis shows ending times between 00:05hrs-02:00hrs.

5.2 Average Skyline Trace

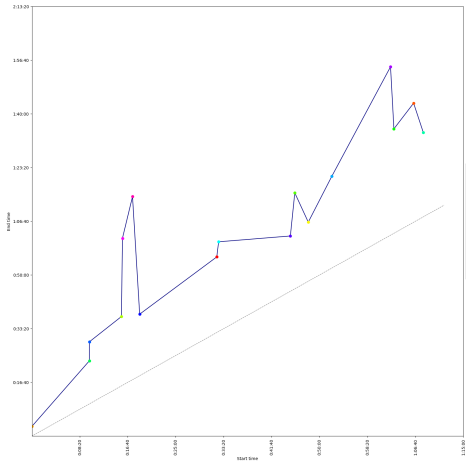


Fig. 7. Average skyline trace with 30 activities. Horizontal axis shows starting times within the first 01:06hrs and vertical axis shows ending times between 00:05hrs-02:00hrs.

The **average skyline trace** $\bar{\rho} = \{\bar{e}_1, \ldots, \bar{e}_i, \ldots, \bar{e}_n\}$ of a process is the resulting skyline from averaging activities in performance skylines of all traces. $\pi_t^+(\bar{e}_{a_i}) = \frac{1}{m}\sum_{j=1}^{m}\pi_t^+(e_j)$ and the end time in $\pi_t^-(\bar{e}_{a_i}) = \frac{1}{m}\sum_{j=1}^{m}\pi_t^-(e_j)$, where $\pi_a(e_j) = a_i$ and $e_j \in \rho_k$, with ρ_k being a performance skyline from a trace in the trace set. An average skyline trace is suitable to consider performance aspects for a given trace set, because it offers knowledge about activities that might be part of any trace's dominant path, here dominant duration path, and their relations to each other. Consequently it is more inclusive regarding what activities to include than the average trace skyline. Figure 7 shows the average of the performance skylines of traces in Fig. 4. This average trace skyline highlights seventeen out of thirty activities, more than the average trace skyline in Fig. 6. With this information, the user can focus on duration dominant activities for any of the traces. This is useful in case traces contain a diverse set of duration dominant activities between them.

5.3 Expected Skyline Activity Set

Having computed performance skylines of a process trace set $R_\Sigma = \{\rho_1, \ldots, \rho_j, \ldots, \rho_n\}$, the **expected skyline activity set** R_A of that process is the set of activities that have a probability of appearance on performance skylines that is equal or higher than a given threshold t_A. The appearance probability for an activity a_k is computed as follows:

$$P(\pi_a(e_i) = a_k) = \frac{|\pi_a(e_i) = a_k|}{|e_i \in \rho_j|}, where\ e_i \in \rho_i$$

Furthermore $t_{a_k} \leq P(\pi_a(e_i) = a_k) \Rightarrow a_k \in R_\Sigma$. After computing appearance probabilities, an expected threshold value shall be chosen to define the expected skyline activity set of the presented process. The higher the chosen threshold value, the fewer activities will be part of the expected skyline activity set. Analogously the lower the chosen threshold value, the more activities will be part of the expected skyline activity set. An expected skyline activity set is suitable to analyze performance of a process across events and traces; as well as provide estimation and knowledge about deviation of events in a process dominant path. Fig. 8 shows a bar chart, where each bar represents one of seventeen activities in the data set and their length corresponding appearance probabilities in R_Σ. Choosing e.g. $t_A = 60\%$ results in $|R_\Sigma| = 4$, containing following activities: `UpdateCrawlStartTask`, `SplitCrawlInputTask`, `DumpTask(target_filename=review_2yold)(chunk=prep)(sql_filename=review)` and `ConvertDumpTask(filename=review_2017_3)(chunk=prep)(sql_filename=review)`.

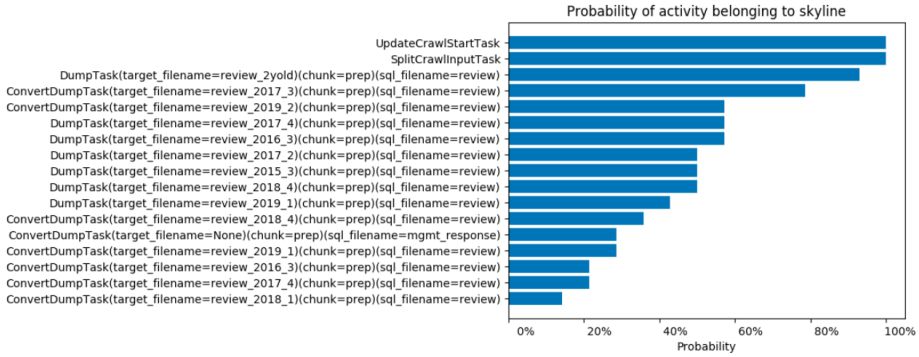


Fig. 8. Performance skyline activity set with corresponding probabilities of activity belonging to the performance skyline.

6 Discussion

Considering that performance skyline explicitly includes multiple traces, it offers a suitable method for performance analysis and thus provides the user to take more informed scheduling and planing decisions than the one offered by combining critical path method and process discovery models [17]. Furthermore introduced interval events bearing multiple timestamps extend performance spec-

trum [2] analysis techniques by including bi-dimensional information about relations between events within one trace or process while still offering a slim visualization. This way both time stamps can be used to take waiting times and events' duration into consideration.

The experimental dataset comprehends three months of logs for a process called `daily` at TrustYou GmbH, a German Guest Feedback and Hotel Reputation Software company. Broadly explained this process ingests customer reviews from multiple sources, does multiple transformations, aggregates them and stores the result at a given location. Being a data process, only computing resources are involved and thus control-flow deviations such as order of activities execution, do not vary without showing performance deviations on interdependent activities as well. The data collection compounds 62,074 interval events spread among 50 traces. It contains 261 different values for activity id. A trace has on average 1238 events. Most activities appear mostly once in a trace, except for six of them which correspond to a few hundred events.

Additionally taking only a subset of dominant activities of interest into account to describe a whole process reduces computational time an eases the search for potential improvement and answers to performance questions that might only concern a certain metric. For example finding events of dominant duration is useful when searching long lasting single activities that can be optimized while also regarding their order of execution and parallelization, which might be inflexible due to their inter-dependencies. For the experiment trace set an average of $5,22\%$ of its events are part of the performance skyline. In order to include the performance knowledge of multiple traces at once, statistical analysis techniques select duration dominant activities that concern any, most or some of the traces. Different techniques serve multiple purposes and data and show advantages to solve various matters: First, the average trace skyline includes only activities that belong to the dominant path. These are highly suited for comparing the average behavior of activities with each other. With this information independent non-dominant events can be paralleled to duration dominant events and thus performance of the whole process can be optimized. Nevertheless being very exclusive with average duration activities, which means that if there is an activity A, which due to high performance variance often appears on the performance skyline for some traces, but which is on average dominated by another activity B. Activity A will not be part of the average trace skyline. Furthermore, groups of traces that have different sets of activities on their skyline might be representative, and conforming, without appearing most often. For this reason the order of steps, averaging and computing a skyline, for a trace set leads to different expectation skylines. Second, the average skyline trace includes every activity on any skyline in the set to the average skyline trace. Even if this technique is advantageous to compare all duration dominant activities, it can produce an expectation skyline trace that is significantly sensitive to outliers because it includes activities that might only be part of a skyline computed from some traces or even the average trace. Lastly, as a trade off between only including most frequent dominant path's activities and all activities from any dominant path disre-

garding their relevance, an expected skyline activity set provides estimation and knowledge about performance deviation of events in a process dominant path, which can be used e.g. to identify trace anomalies. All of our presented statistical analysis techniques enable more informed decisions taking, e.g. how to best schedule events, without overflowing the visualization. Furthermore as a rather data-driven opposed to control-flow based approach, our performance skyline yields flow analysis [12] like results for mining non-block-structured models.

7 Conclusion and Future Work

With our new approach of *performance skyline* we introduce a novel approach for the performance mining of events containing multiple timestamps. Combining interval based sequence pattern mining and process mining techniques facilitates more accurate process discovery by integrating additional performance knowledge across traces and events. Applying statistical analysis techniques on the performance skyline enrich dominant path analysis with probabilistic performance knowledge enabling more complete conformance checking, detecting and discerning patterns, and thus adapting processes to be faster and more resourceful. Results from applying these methods to the real data set for a company exemplifies how combining performance mining and sequence pattern mining techniques is most suitable to identify and analyze the dominant path in a process model using a trace set containing interval events. Future work involves further experiments with variations on implementation of the skyline operator, alignment, and skylines on other dominant features besides duration, as waiting time, or even non-time related aspects as memory usage. Moreover investigation of inferring models from streaming interval events as well as general research for suitability solving tasks in process discovery, conformance checking, and process enhancement could be expanded:

- Broader **process discovery**: identify loops and choices in the performance skyline aggregations, extending the model by adding more information or enriching further the visualization of already present items, researching more variations of this model, as adding a third dimension plotting information about resources, dependencies, further timestamps and `case id`.
- **Conformance checking** for recognition and prediction tasks: Anomaly detection on trace - activity and event level, drift recognition, and more specifically predictive process monitoring for interval events, using e.g. skyline expectation maximization [21] for performance prediction on event level.
- **Process enhancement**: recognizing bottlenecks, or using detected probabilistic pattern knowledge for optimal networks queuing and resource allocation efficiently.

Acknowledgements

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibility for its content.

References

1. H. Gantt, “A graphical daily balance in manufacture,” vol. 24, pp. 1322–1336, 06 1920.

2. V. Denisov, E. Belkina, D. Fahland, and W. van der Aalst, "The performance spectrum miner: visual analytics for fine-grained performance analysis of processes," in *BPMTracks 2018*, ser. CEUR Workshop Proceedings, F. Casati, Ed. CEUR-WS.org, 9 2018, pp. 96–100.
3. S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," 12 2000.
4. P. Robledo, "Process mining plays an essential role in digital transformation," in *https://medium.com/@pedrorobledobpm/process-mining-plays-an-essential-role-in-digital-transformation-384839236bbe*.
5. S. Suriadi, C. Ouyang, W. M. Van Der Aalst, and A. H. Ter Hofstede, "Event interval analysis: Why do processes take time?" *Decision Support Systems*, vol. 79, pp. 77–98, 2015.
6. A. Burattin and A. Sperduti, "Heuristics miner for time intervals." in *ESANN*, 2010.
7. A. Burattin, "Heuristics miner for time interval," in *Process mining techniques in business environments*. Springer, 2015, pp. 85–95.
8. W. M. P. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Heidelberg: Springer, 2016.
9. F. Richter and T. Seidl, "Tesseract: Time-drifts in event streams using series of evolving rolling averages of completion times," in *Business Process Management*, J. Carmona, G. Engels, and A. Kumar, Eds. Cham: Springer International Publishing, 2017, pp. 289–305.
10. V. Denisov, D. Fahland, and W. M. van der Aalst, "Predictive performance monitoring of material handling systems using the performance spectrum," in *2019 International Conference on Process Mining (ICPM)*. IEEE, 2019, pp. 137–144.
11. A. Senderovich, M. Weidlich, A. Gal, and A. Mandelbaum, "Queue mining–predicting delays in service processes," in *International conference on advanced information systems engineering*. Springer, 2014, pp. 42–57.
12. M. Dumas, M. La Rosa, J. Mendling, and H. Reijers, "Fundamentals of business process management. 2013."
13. A. Senderovich, M. Weidlich, and A. Gal, "Context-aware temporal network representation of event logs: Model and methods for process performance analysis," *Information Systems*, vol. 84, pp. 240–254, 2019.
14. L. R. Shaffer, J. Ritter, and W. L. Meyer, *The critical-path method*. McGraw-Hill, 1965.
15. J. Fondahl, "A non-computer approach to the critical path method for the construction industry," 06 1961.
16. J. Santiago and D. Magallon, "Critical path method," *CEE 320 – VDC SEMINAR*, feb 2009.
17. M. M V, L. Thomas, and A. Basava, "Efficient process mining through critical path network analysis," 02 2014.
18. J. Allen, "Maintaining Knowledge About Temporal Intervals," vol. 26, no. 11. New York, NY, USA: ACM, 1983, pp. 832–843.
19. J. W. Marwan Hassani, Yifeng Lu and T. Seidl, "A geometric approach for mining sequential patterns in interval-based data streams," *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 2128–2135, 2016.
20. H. Samet, "Hierarchical representations of collections of small rectangles," *ACM Comput. Surv.*, vol. 20, no. 4, p. 271–309, Dec. 1988. [Online]. Available: https://doi.org/10.1145/50020.50021
21. T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.