

Detecting Surprising Situations in Event Data

Christian Kohlschmidt, Mahnaz Sadat Qafari, and Wil M. P. van der Aalst

Process and Data Science Chair (PADS)
RWTH Aachen University Aachen, Germany
christian.kohlschmidt@rwth-aachen.de,
{m.s.qafari,wvdaalst}@pads.rwth-aachen.de

Abstract. Process mining is a set of techniques that are used by organizations to understand and improve their operational processes. The first essential step in designing any process reengineering procedure is to find process improvement opportunities. In existing work, it is usually assumed that the set of problematic process instances in which an undesirable outcome occurs is known prior or is easily detectable. So the process enhancement procedure involves finding the root causes and the treatments for the problem in those process instances. For example, the set of problematic instances is considered as those with outlier values or with values smaller/bigger than a given threshold in one of the process features. However, on various occasions, using this approach, many process enhancement opportunities, not captured by these problematic process instances, are missed. To overcome this issue, we formulate finding the process enhancement areas as a context-sensitive anomaly/outlier detection problem. We define a process enhancement area as a set of situations (process instances or prefixes of process instances) where the process performance is surprising. We aim to characterize those situations where process performance is significantly different from what was expected considering its performance in similar situations. To evaluate the validity and relevance of the proposed approach, we have implemented and evaluated it on a real-life event log.

Keywords: Process mining · process enhancement · context-sensitive outlier detection · surprising instances.

1 Introduction

Considering the current highly competitive nature of the economy, it is vital for organizations to continuously enhance their processes in order to meet the best market standards and improve customer experience. Process enhancement involves many steps, including finding the process areas where improvements are possible, designing the process reengineering steps, and estimating the impact of changing each factor on the process performance. By conducting all these steps, organizations can benefit from applying process mining techniques. The first step of process enhancement is detecting those process areas where an improvement is possible. Process mining includes several techniques for process monitoring

and finding their friction points. However, these techniques have the hidden assumption that all the process instances (cases) are the same. So the set of problematic cases can be easily identified. For example, the problematic cases can be identified as the ones with an outlier value with respect to a process feature. Another common method is using a threshold for a specific process feature. However, considering the variety of the cases, it is possible that a solution solves the problem for one group of cases while aggravating the problem for another group. Moreover, using the current techniques, the performance of the process in some cases can be considered normal and acceptable compared to the overall behavior of the process, while it can be considered surprising (i.e. anomalous or undesirable) when just considering their similar cases. This phenomenon can lead to overlooking some of the process enhancement opportunities.

As another issue, there are several process instances where the process performs significantly better than other similar process instances. Analyzing the process behavior while performing these process instances can lead to invaluable clues on how to improve the process. Usually, this source of information is neglected by the current process mining techniques.

To overcome these issues, we formulate finding those areas where a process enhancement is possible as the problem of finding those groups of process situations where the process performance is significantly different from their similar situations. Here, we define a *process situation* (or simply a *situation*) as a process instance or a prefix of it. The proposed method includes four steps (1) enriching and extracting the data from the event log (2) finding a set of sets of similar situations (which we call a *vicinity cover* and each set of similar situations is a *vicinity*). Naturally, a measure is needed to measure the similarity between instances and identify vicinities. However, having access to such a measure is a strong assumption. Thus we use a machine learning technique to determine the vicinities in the absence of such a measure. (3) The next step involves finding the set of surprising situations in each vicinity (if any exist). (4) Finally, a list of detected sets of surprising situations is presented to the user ordered by their effect on the process and how surprising they are. These findings can be further analyzed to understand the reason for the different behavior of the process in these surprising situations and gain insights on how to improve the process. Figure 1 shows the general overview of the proposed method.

For example, consider that in a loan application process with 20 cases, we are interested in finding those cases where their throughput is surprising. In this example, each process instance (case) is a situation. Also, we consider two situations similar if the Levenshtein distance of their activity sequence is at most one. Figure 2 shows the graph for the cases of this loan application, where each

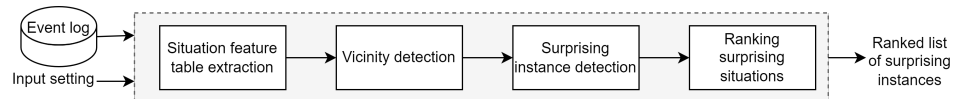


Fig. 1: The general overview of the proposed method.

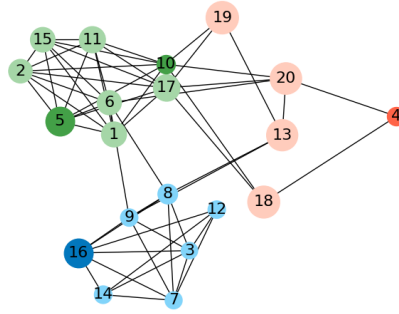


Fig. 2: A graph representing the similarity of situations in a loan application example. Each node represents a situation (a process instance). Two situations are similar if the Levenshtein distance of their activity sequences is at most one. The vicinity of a node is the set of process instances in the same community. Three vicinities have been detected in this example, which are colored red, blue, and green. Surprising situations are highlighted with a darker color. The throughput of each situation is proportional to the size of its corresponding node.

case corresponds to a node. Two cases are connected if they are similar. The size of each node is proportional to its throughput. The colors (blue, green, and red) indicate the vicinities found by the Louvain community detection algorithm [3]. The nodes highlighted with darker colors are the surprising cases where the throughput is significantly different from the other cases in the same vicinity. In this example, the throughput was worse than expected for cases 5 and 16 and better than expected for cases 4 and 10. The process owner can gain actionable insights by analyzing the behavior of the process in these cases, particularly in comparison with their vicinity, to enhance the performance of the process in other similar cases in the future. Note, if we just had considered the overall performance of this process, these four cases would not have been detected as their throughput are not far from the average throughput of all cases.

The rest of the paper is organized as follows. In Section 2, a brief overview of the related work is given. In Section 3, the proposed method is presented. The experimental results are discussed in Section 4. Finally, in Section 5, the conclusion is presented.

2 Related work

Existing research on context-aware anomaly detection in process mining is closest to our work. Here we provide an overview of anomaly detection techniques.

Most existing methods investigate anomalies considering the control-flow perspective (e.g., [1, 2, 7, 9, 10, 16]). These methods generate a reference model from the event log and apply conformance checking to detect anomalous behavior. A

subgroup of these methods known as *deviance mining approaches* investigate performance anomalies [9]. In [16], the authors identify deviations and bottlenecks by replaying the event log on an enrich process model with performance information. In [7], the authors analyze the deviations between a process model and an event log to identify which deviations enforce positive performance. In [8], the anomalous cases in event logs are detected using window-based and Markovian-based techniques. The drawback of control-flow approaches is that they ignore a wide range of non-control-flow data, which can be used for more sophisticated context-sensitive anomaly detection methods.

The authors of [4] propose an anomaly detection approach that incorporates perspectives beyond the control-flow perspective, such as time and resource-related information. This approach marks events as anomalies based on a certain likelihood of occurrence, however, case anomalies are not considered.

Other approaches in this category only focus on specific use cases. The authors of [13] analyze suspicious payment transactions to identify money laundering within a money transfer service. They propose an approach to match the transactions with the expected behavior given by a process model to identify many small transactions that end up on the same account. [14] identifies surprisingly short activity execution times in a process by automatically inferring a Bayesian model from the Petri net representation of the process model. The authors of [15] use fuzzy association rule learning to detect anomalies. As these approaches specialize in specific use cases, they do not apply to identify anomalies in a general process.

A third category is domain-based anomaly detection. For example, the authors of [11] propose an approach that supports the identification of unusual or unexpected transactions by encoding the cases and assigning an anomaly score to each case. They use the domain knowledge of domain experts to update the assigned anomaly scores. The approaches in this category require domain knowledge to label cases, which limits their applicability.

3 Method

Process mining techniques usually start by analyzing an event log. An event log is a collection of cases where each case is a sequence of events, in which each event refers to a case, an activity, and a point in time. More formally,

Definition 1 (Event, Case, Event log). *Let \mathcal{C} be the universe of case identifiers, \mathcal{A} be the universe of activities, \mathcal{T} be the universe of timestamps. Moreover, let $\mathcal{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_n\}$ be the universe of domain-dependent data attributes. We define the universe of events as $\mathcal{E} = \mathcal{C} \times \mathcal{A} \times \mathcal{T} \times \mathcal{D}_1 \times \dots \times \mathcal{D}_n$ and each element $e = (c, a, t, d_1, \dots, d_n) \in \mathcal{E}$ an event. Let \mathcal{E}^+ be the universe of (non-empty) finite and chronologically ordered sequences of events. We define a case as a sequence of events $\gamma \in \mathcal{E}^+$ in which all events have the same case identifier; i.e. $\forall e_i, e_j \in \gamma \pi_c(e_i) = \pi_c(e_j)$ where $\pi_c(e)$ returns the case identifier of event $e \in \mathcal{E}$. We define an event log, L , as a set of cases in which each case has a unique*

case identifier; i.e., $\forall \gamma, \gamma' \in L(\exists e \in \gamma \exists e' \in \gamma \pi_c(e) = \pi_c(e')) \implies \gamma = \gamma'$. We denote the universe of all event logs with \mathcal{L} .

We assume that we know the process feature that captures the property of the process that the process owner is interested in its optimization. We call this feature *target feature* and denote it with tf where $tf \in \mathcal{TF} = \mathcal{A} \times \mathcal{D}$. Note that the target is composed of an attribute name and an activity name, which indicate the attribute value should be extracted from the events with that activity name. The attribute name can be any of the attributes captured by the event log or a derived one. Moreover, we assume that we know *descriptive features*, which are the set of process features that are relevant in measuring the similarity of the situations. In the following, we explain the surprising situation detection steps.

3.1 Situation Feature Table Extraction

To find the surprising situations, we have to extract the data in the form of tabular data from the event log. As the detected surprising situations are meant to be used for root cause analysis, it is important to respect the temporal precedence of cause and effect, indicating that the cause must occur before the effect. Therefore, we extract the data from that prefix of the case that has been recorded before the target feature. We call such a prefix a *situation*. More formally:

Definition 2 (Situation). Let $L \in \mathcal{L}$, $\gamma = \langle e_1, \dots, e_n \rangle \in L$, $prfx(\langle e_1, \dots, e_n \rangle) = \{\langle e_1, \dots, e_i \rangle \mid 1 \leq i \leq n\}$, a function that returns the set of non-empty prefixes of a given case, and $tf \in \mathcal{TF} = \mathcal{A} \times \mathcal{D}$ a target feature. We define the universe of all situations as $\mathcal{S} = \bigcup_{L \in \mathcal{L}} S_L$ where $S_L = \{\sigma \mid \sigma \in prfx(\gamma) \wedge \gamma \in L\}$ is the set of situations of event log L . We call each element $\sigma \in \mathcal{S}$ a situation. Moreover, we define $sit \in (\mathcal{L} \times \mathcal{TF}) \times 2^{\mathcal{S}}$ to be the a function that returns $\{\sigma \in S_L \mid \pi_a(\sigma) = act\}$ for a given $L \in \mathcal{L}$ and $tf = (att, act)$, where $\pi_a(\sigma)$ returns the activity name of the last event of σ .

We call the data table created by extracting data from situations a *situation feature table*. Please note that each row of the situation feature table extracted from $sit(L, tf)$ corresponds to a situation in it and this correspondence forms a bijection. To enrich the event log and extract the situation feature table, we use the method presented in [12].

3.2 Vicinity detection

Informally, a vicinity is a set of similar situations and a vicinity cover of $S \subseteq \mathcal{S}$ is a set of vicinities of its situations such that their union covers S . Let $cov \in 2^{\mathcal{S}} \rightarrow 2^{2^{\mathcal{S}}}$ in which $\forall S \subseteq \mathcal{S} \forall S' \in cov(S) (S' \neq \emptyset \wedge (\forall \sigma, \sigma' \in S' sim(\sigma, \sigma') = 1))$ and $\forall S \subseteq \mathcal{S} \cup_{S' \in cov(S)} S' = S$. Here, $sim \in \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$ is an indicator function indicating if σ and σ' are similar, for $\sigma, \sigma' \in \mathcal{S}$.

Using a coverage function, we define a vicinity cover of a set of situations extracted from an event log with respect to a specific target feature as follows:

Definition 3 (Vicinity and Vicinity Cover). Let $S = \text{sit}(L, tf)$ be the set of situations extracted from $L \in \mathcal{L}$ with respect to the target feature $tf \in \mathcal{TF}$ and $\text{cov} \in 2^S \rightarrow 2^{2^S}$ be a coverage function. We simply define a vicinity cover of S as $\text{cov}(S)$ and we call each member of $V \in \text{cov}(S)$ a vicinity of S . We denote the universe of all vicinities by \mathcal{V} .

In the sequel, we explain the vicinity detection method separately for the case where we know the similarity measure and the case where such a similarity measure is not known.

Vicinity Detection With a Similarity Measure. Let $d \in \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ be a distance measure. Then we can say a situation is similar to another situation if their distance is less than α . Now, we can define the similarity function as $\text{sim}_{d,\alpha} \in \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$ such that $\text{sim}_{d,\alpha}(\sigma_1, \sigma_2)$ returns 1 if $d(\sigma, \sigma') \leq \alpha$ and 0 otherwise, for all $\sigma, \sigma' \in \mathcal{S}$. In this case, we can determine the vicinity cover of the set of situations through the coverage function (Definition 3) in which $\text{sim}_{d,\alpha}(\cdot, \cdot)$ is the similarity function. Another method is to create a graph $G = (S, E)$ in which each node corresponds to one of the situations extracted from the event log. There is an edge between two nodes if the distance of their corresponding situations is smaller than α . Using a *community detection* algorithm on this graph, we can determine the vicinities. Note that in this case two situations are similar if their corresponding nodes are in the same community and each detected community is a vicinity. A community detection function aims at finding (potentially overlapping) sets of nodes that optimize the modularity within the similarity graph. Modularity measures the relative density of edges inside the communities compared to edges outside the communities.

As another option we can use a clustering method to detect vicinities. We use k -means as the clustering model to explain the method; however, the general idea is similar to using other clustering models. To find the surprising situations using a clustering model, we first cluster the situations using k -means, with a predefined k , based on their descriptive features. In this method, two situations are similar if they belong to the same cluster and each cluster forms a vicinity. Please note that in this case the similarity measure is used to measure the distance between each situation and the centroids of clusters.

Vicinity Detection Without a Similarity Measure. The availability of a distance function is a strong assumption. Considering the complexity of the real-life event data, even for specialists, it is a challenging task to determine such a distance function. Hence, we use machine learning techniques to detect surprising situations in the data. In this case, the process expert needs to know the set of process features relevant to measuring the similarity of the situations and not the exact distance measure. Here we briefly mention the vicinity detection method using a classification model.

We mainly use a decision tree as the classification model. We train a decision tree on the data trying to predict the target feature tf using descriptive features.

In this method, we consider two situations similar if they belong to the same node of the tree. Moreover, we consider the set of situations corresponding to each node of the decision tree (or each node in a subset of nodes of the decision tree, such as leaves) as a vicinity.

3.3 Surprising Situation Detection

We define the surprising situations in each vicinity as those situations in that vicinity that significantly differ from the other situations (in that vicinity). Suppose that $D \in \mathcal{V} \rightarrow \bigcup_{V \in \mathcal{V}} 2^V$ where $\forall V \in \mathcal{V} : D(V) \subseteq V$ is a function that, given a set of similar situations (a vicinity), returns its subset of surprising ones. We call such a function a *detector*. For example, a detector function can be a function that returns the subset of situations that exceed a user-defined threshold value for the target feature. Using this function, we define the set of surprising situations of a vicinity as follows:

Definition 4 (Surprising Situation Set). *Let $V \in \mathcal{V}$ be a vicinity and $D \in \mathcal{V} \rightarrow \bigcup_{V \in \mathcal{V}} 2^V$ where $\forall V \in \mathcal{V} : D(V) \subseteq V$ be a detector function. We define $D(V)$ as the set of surprising situations in V .*

We can find the set of all sets of surprising situations of the set of situations by applying the detector function on all the vicinities of its vicinity cover.

Definition 5 (Surprising Situation Sets). *Let $S = \text{sit}(L, tf)$ be the set of situations extracted from $L \in \mathcal{L}$ with respect to target feature $tf \in \mathcal{TF}$, $\text{cov}(S)$ a vicinity cover of S , and detection function $D \in \mathcal{V} \rightarrow \bigcup_{V \in \mathcal{V}} 2^V$. We define the surprising situation sets of S as $\{D(V) \mid V \in \text{cov}(S)\}$.*

3.4 Ordering Surprising Situations

We define two criteria to order the detected surprising situations: *surprisingness* and *effectiveness*. Suppose U is the set of surprising situations in a vicinity V . Surprisingness of U measures how rare it is to see such a situation in its vicinity, whereas effectiveness measures how beneficial it is to enhance the process based on the findings of root cause analysis of U . More precisely:

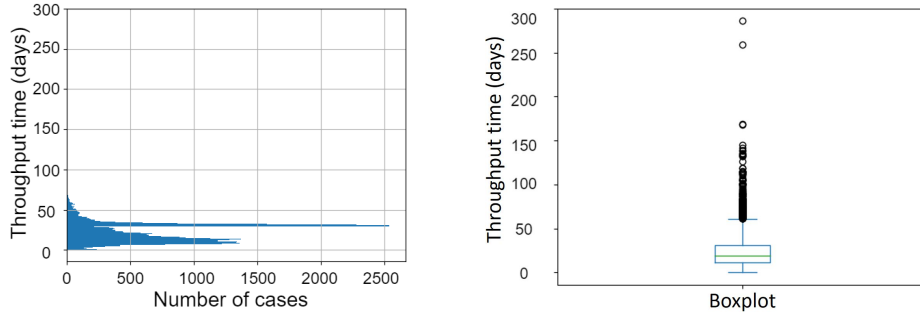
Definition 6. *Let $V \in \mathcal{V}$ be a vicinity and $U \subseteq V$ the set of surprising situations in V , and $\beta \in (0, 1]$ a threshold. We define the surprisingness of U as:*

$$\text{surp}(U) = \beta \mid \text{avg}(U) - \text{avg}(V \setminus U) \mid + (1 - \beta) \frac{\#(U)}{\#(V)}$$

and the effectiveness of U as:

$$\text{eff}(U) = \begin{cases} (\text{avg}(V \setminus U) - \text{avg}(U)) \times \#(V \setminus U) & \text{avg}(U) < \text{avg}(V \setminus U) \\ (\text{avg}(U) - \text{avg}(V \setminus U)) \times \#(U) & \text{avg}(U) > \text{avg}(V \setminus U) \end{cases}$$

where $\#(A)$ denotes the cardinality of A and $\text{avg}(A) = \frac{\sum_{s \in A} \pi_{tf}(s)}{\#(A)}$ for each $A \subseteq S$ is the average value of the target feature tf for the situations in A .



(a) Distribution of the throughput time for the BPI Challenge 2017 event log capturing the duration from the start to the end of each case.

(b) Detected outliers of throughput time of cases of BPI Challenge 2017 event log using boxplot. Cases durations above 61 days are considered anomalous.

Fig. 3: The throughput time for the BPI Challenge 2017 event log.

In the above definition, we assume that the lower values for tf are more desirable. If this assumption does not hold, the effectiveness can be similarly defined.

4 Experimental Results

To evaluate the proposed framework¹, we present the result of applying it on the event log for BPI Challenge 2017 [5]. This event log represents an application process for a personal loan or overdraft within a global financing organization taken from a Dutch financial institute. We consider throughput as the target feature. The majority of the cases in the process take between 5 and 40 days. The average duration for all cases in the event log is around 22 days. Figure 3a shows the distribution of the throughput time.

Boxplots are frequently used to identify performance anomalies [6]. Thus we use boxplots as the baseline and call this approach the *baseline*. The resulting boxplot is shown in Figure 3b. Using this method, 255 cases with a throughput of more than 61 days have been considered anomalous. These are the detected anomalies without any context-awareness of the process.

To apply our approach, we used the following case-level attributes as descriptive features: *application type*, *loan goal*, *applicant's requested loan amount*, and the *number of offers* which is a derivative attribute indicating how many times the loan application institute offered a loan to the customer. Note that in this experiment, each case is a situation.

We apply surprising situation detection using a similarity measure, a classification method (using a decision tree), and also a clustering method (using

¹ The implemented tool is available at <https://github.com/ckohlschm/detecting-surprising-instances>.

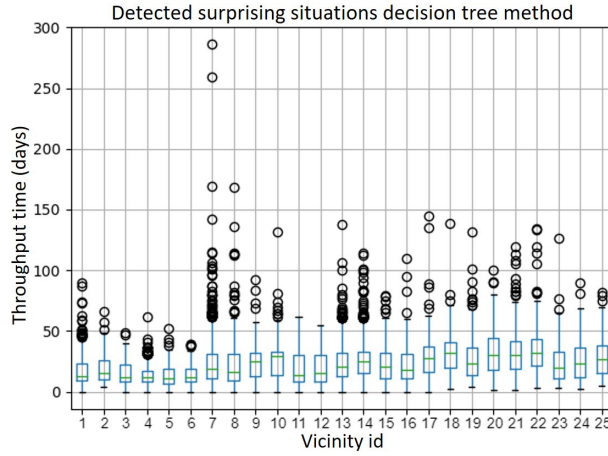


Fig. 4: Detected surprising situations in each vicinity defined by the decision tree method.

k-means clustering). We call these three approaches *similarity based method*, *decision tree method*, and *k-means clustering method* respectively. In all these methods, to maximize the applicability of the implemented tool and to minimize the required domain knowledge, we use the boxplot as the detector function (Definition 4) to find the surprising situations in each vicinity.

Decision tree method. For this experiment, we trained a decision (regression) tree with a maximum depth of 5 and a minimum number of instances per leaf of 100. We consider the vicinities described by the leaves of the tree. Figure 4 shows the detected surprising situations for the leaves in the decision tree where each leaf is labeled with a number. Some of the highlights of the comparison of the results of the decision tree method and the baseline are as follows:

- Application_1839367200 (Case duration 62 days) is barely considered an outlier in the total dataset, but in its vicinity (Vicinity 4: one offer, limit raise, loan goal car, requested amount > 11.150) it is far from the average which is 14 days.
- Vicinity 19, where the number of offers is more than 3 and the requested amount ≤ 13.162 includes seven surprising situations. These situations have not been considered outliers by the baseline method. One possible interpretation of this result is that high throughput is acceptable in such situations. The same applies to vicinity 20.
- Vicinity 5 (one offer, limit raise, Unknown loan goal, requested amount ≤ 3000) contains 3 surprising situations that are all overlooked by the baseline method. The vicinity contains 338 cases with an average throughput time of 13 days which makes cases with a duration of more than 40 days surprising. The same applies to vicinities 3 and 6.

Figure 5 shows the surprisingness (on the left) and effectiveness (on the right) of the sets of surprising situations detected by the decision tree method. The set of surprising situations in vicinity 17 has the highest surprisingness. This vicinity includes 126 situations, where 6 are surprising with an average throughput of 100 days, whereas the other situations in the vicinity have an average of 27 days. These are the cases with two offers that use their loan to pay their remaining home dept and the requested amount is at most 24,500. The set of surprising situations in vicinity 7 has the highest effectiveness. These situations correspond to the customers with one offer that apply for a new credit. Removing the problem that causes the delay in these surprising situations would reduce the average throughput time for similar cases by more than one day.

k-means clustering method. In this approach, we used *k*-means clustering to identify vicinities. For *k* we use the value 25, which is the number of the vicinities in the decision tree method and Euclidean distance as similarity measure. This method results in detecting a total of 280 surprising situations. The plot on the left side of Figure 6 shows the surprising situations detected in each vicinity.

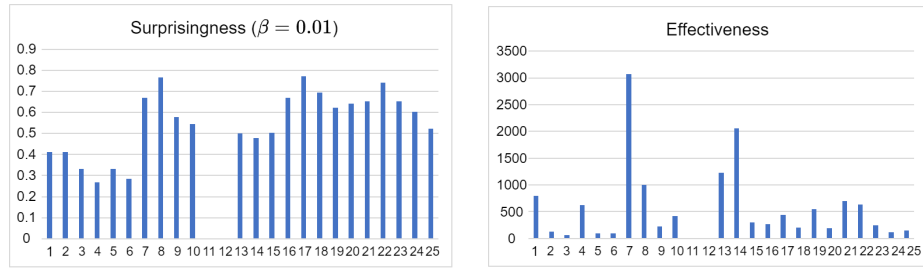


Fig. 5: Surprisingness and effectiveness of the surprising situations identified by the decision tree method.

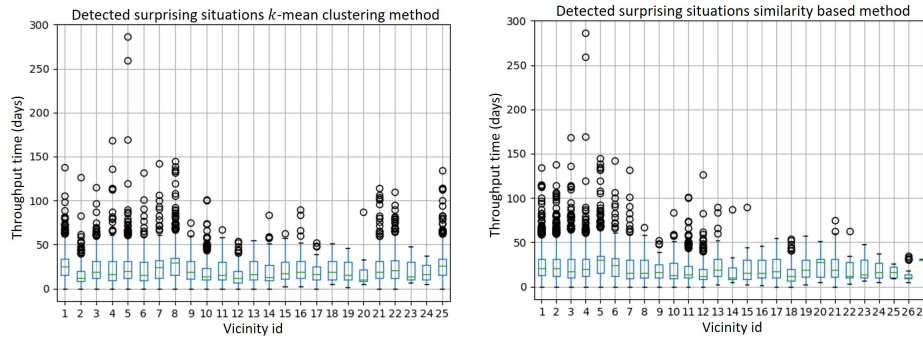


Fig. 6: Detected surprising situations by the *k*-means clustering and similarity based method.

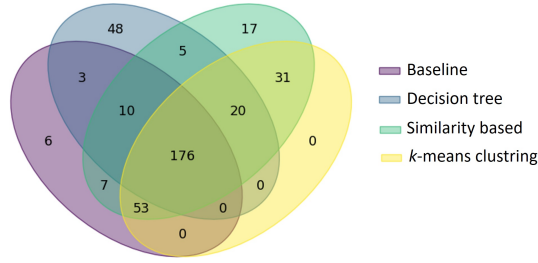


Fig. 7: Venn Diagram showing the intersection of detected surprising situations using the different methods.

Similarity based method. We run the similarity based approach where the distance measure is the Euclidean distance of normalized descriptive features (using min-max method). Then, we use 1.4, which results in 27 clusters (close to 25), as the threshold to generate a graph. To find the vicinities, we used the *Louvain community detection method* [3] on this graph. The plot on the right side of Figure 6 shows the surprising situations detected in each vicinity.

It is worth noting that the set of surprising situations detected by different methods was not exactly the same. Figure 7 shows that all the methods agree on 176 detected surprising situations and for all other situations at least one method does not select it.

5 Conclusion

Finding the process enhancement areas is a fundamental prerequisite for any process enhancement procedure that highly affects its outcome. It is usually assumed that these process areas are known in advance or can be detected easily. However, utilizing simple methods have the danger of overlooking some of the opportunities for process enhancement or targeting the wrong ones. In this paper, we formulate the process of finding process enhancement areas as a method for finding surprising situations; i.e., detecting those situations where the process behavior is significantly different from similar situations.

We have implemented the proposed framework with different methods and evaluated it using real event logs. The experiment shows that the detected surprising (anomalous) situations are overlapping but not identical to the ones of the baseline, which is currently a common method for finding anomalies. It shows that to find the best result, it is best to use our framework complementary to the existing methods; i.e., using both context-sensitive and non-context-sensitive methods for finding the process enhancement areas.

Acknowledgment

We thank Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. F. d. L. Bezerra and J. Wainer. Fraud detection in process aware systems. *International Journal of Business Process Integration and Management*, 5(2):121–129, 2011.
2. F. d. L. Bezerra and J. Wainer. A dynamic threshold algorithm for anomaly detection in logs of process aware systems. 2012.
3. V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.
4. 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.
5. J. J. Carmona, M. de Leoni, B. Depaire, and T. Jouck. Process Discovery Contest 2017. 5 2021.
6. R. Conforti, M. La Rosa, and A. H. M. ter Hofstede. Filtering out infrequent behavior from business process event logs. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):300–314, 2016.
7. M. Dees, M. d. Leoni, and F. Mannhardt. Enhancing process models to improve business performance: A methodology and case studies. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 232–251. Springer, 2017.
8. N. Gupta, K. Anand, and A. Sureka. Pariket: Mining business process logs for root cause analysis of anomalous incidents. In *International Workshop on Databases in Networked Information Systems*, pages 244–263. Springer, 2015.
9. H. Nguyen, M. Dumas, M. L. Rosa, F. M. Maggi, and S. Suriadi. Business process deviance mining: Review and evaluation. *CoRR*, abs/1608.08252, 2016.
10. S. Pauwels and T. Calders. An anomaly detection technique for business processes based on extended dynamic bayesian networks. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 494–501, 2019.
11. R. Post, I. Beerepoot, X. Lu, S. Kas, S. Wiewel, A. Koopman, and H. Reijers. Active anomaly detection for key item selection in process auditing. In *International Conference on Process Mining*, pages 167–179. Springer, Cham, 2021.
12. M. S. Qafari and W. M. P. van der Aalst. Feature recommendation for structural equation model discovery in process mining. *Progress in Artificial Intelligence*, pages 1–25, 2022.
13. R. Rieke, M. Zhdanova, J. Repp, R. Giot, and C. Gaber. Fraud detection in mobile payments utilizing process behavior analysis. In *2013 International Conference on Availability, Reliability and Security*, pages 662–669. IEEE, 2013.
14. A. Rogge-Solti and G. Kasneci. Temporal anomaly detection in business processes. In *International Conference on Business Process Management*, pages 234–249. Springer, 2014.
15. R. Sarno, F. Sinaga, and K. R. Sungkono. Anomaly detection in business processes using process mining and fuzzy association rule learning. *Journal of Big Data*, 7(1):1–19, 2020.
16. W. M. P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.