Information Systems 90 (2020) 101438

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Mining association rules for anomaly detection in dynamic process runtime behavior and explaining the root cause to users

Kristof Böhmer *, Stefanie Rinderle-Ma

Research Group Workflow Systems and Technology, Faculty of Computer Science, University of Vienna, Vienna, Austria

ARTICLE INFO

Article history:
Received 1 February 2019
Received in revised form 1 June 2019
Accepted 10 September 2019
Available online 18 September 2019
Recommended by Gottfried Vossen

Keywords:
Anomaly detection
Process runtime behavior
Root cause
Association rule mining
Process change

ABSTRACT

Detecting anomalies in process runtime behavior is crucial: they might reflect, on the one side, security breaches and fraudulent behavior and on the other side desired deviations due to, for example, exceptional conditions. Both scenarios yield valuable insights for process analysts and owners, but happen due to different reasons and require a different treatment. Hence a distinction into malign and benign anomalies is required. Existing anomaly detection approaches typically fall short in supporting experts when in need to take this decision. An additional problem are false positives which could result in selecting incorrect countermeasures. This paper proposes a novel anomaly detection approach based on association rule mining. It fosters the explanation of anomalies and the estimation of their severity. In addition, the approach is able to deal with process change and flexible executions which potentially lead to false positives. This facilitates to take the appropriate countermeasure for a malign anomaly and to avoid the possible termination of benign process executions. The feasibility and result quality of the approach are shown by a prototypical implementation and by analyzing real life logs with injected artificial anomalies. The explanatory power of the presented approach is evaluated through a controlled experiment with users.

1. Introduction

Information security, while being one of the top priorities for companies, is currently shifting from prevention-only approaches to detection and response — according to Gartner [1]. One analyst states: *"The shift to detection and response approaches spans people, process and technology elements and will drive a majority of security market growth over the next five years"* [1]; predicting a 113 billion worldwide spending on information security by 2020.

The vehicle to integrate processes, users, and technology and hence to drive digitalization efforts in companies are executable processes implemented through Process-Aware Information Systems (PAIS). Security problems in PAIS can be found based on anomaly detection approaches, i.e., by revealing anomalous process execution behavior which can indicate fraud, misuse, or unknown attacks, cf. [2,3]. Typically, whenever anomalous behavior is identified an *alarm* is sent to a security expert. Subsequently, the expert determines the alarm's root cause to choose an appropriate anomaly *countermeasure*, such as, terminating an anomalous process, ignoring a false alarm, or manually correcting process execution behavior, cf. [3].

The survey on security in PAIS [4], however, shows that the majority of existing approaches focuses on prevention and only few approaches tackle detection and response, particularly at runtime and change time of processes.

1.1. Problem statement and research questions

Analyzing anomaly detection alarms and choosing countermeasures is *challenging*, cf. [5]. This applies also to the process domain as many processes operate in *flexible open environments* [6]. Hence, thousands of alarms must be carefully analyzed as they could be false positives that report benign behavior as anomalous [3,5] (e.g., because of incorrectly interpreted noise or ad hoc changes).

Existing work on process anomaly detection, cf. [2,3,7], reports only if an execution is *anomalous or not*. However, we assume that additional information, e.g., which behavior motivated the (non-) anomalous decisions or the anomaly severity, are a necessity. Without such information, anomalies, likely, cannot be fully understood and it becomes hard to differentiate between harmful anomalies and false positives, but also to choose appropriate countermeasures as anomalies vary in effect and form.

Further on, existing process focused work frequently applies *monolithic* anomaly detection signatures, cf. [2,7], to identify

* Corresponding author.
E-mail addresses: kristof.boehmer@univie.ac.at (K. Böhmer), stefanie.rinderle-ma@univie.ac.at (S. Rinderle-Ma).

https://doi.org/10.1016/j.is.2019.101438

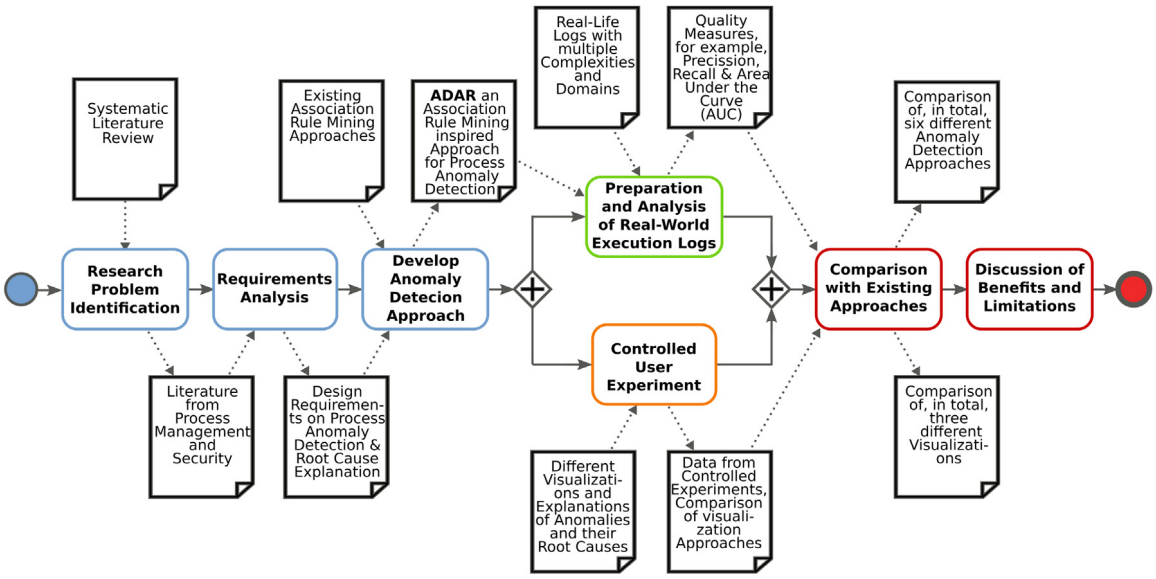

Fig. 1. Applied design science inspired research method.

anomalies. Such signatures, compress all expected execution behavior into a complex interconnected structure. As a consequence, the signatures must be recreated from scratch whenever a process changes, are hard to understand, and can be computationally intense to create. Monolithic signatures were also found to be overly detailed and specific so that benign noise or ad hoc changes (such as slightly varying resource assignments or activity execution orders) are typically reported as anomalies [3]. This could hinder an organization as benign process executions could be unnecessarily terminated.

Altogether, this work aims at a flexible approach to detect anomalies, distinguish their cause into benign and malign behavior, and explain their root cause to the users. This is reflected by the following research questions:

1. **RQ1:** How to distinguish anomalies caused by benign behavior, such as, changes from malign behavior?
2. **RQ2:** How to reduce the effort for anomaly detection, particularly in flexible process environments?
3. **RQ3:** How to explain the root cause of anomalies effectively and efficiently to users?

1.2. Contribution and research method

To address RQ1–RQ3 this work proposes a novel unsupervised anomaly detection heuristic. Instead of monolithic signatures it applies small sets of independent association rules ($\mapsto$ RQ1, RQ2) as follows: Let P be a process that should be monitored for anomalies and let L hold all execution traces t of P. The given behavior in L is represented as a set of association rules R (a signature, resp.). To analyze if a novel execution trace $t' \notin L$ is anomalous it is determined which rules in R are supported by t'. Supported means that a trace complies to the conditions specified by the rule. If it is found that t' has a lower rule support than the trace $t \in L$ that is most similar to t' then t' is identified as anomalous and an alarm is triggered. Doing so, individual rules can easily be replaced if a process changes ($\mapsto$ RQ2).

Another advantage of employing a rather simple formalisms such as association rules, when compared to more expressive formalisms such as LTL, is understandability ($\mapsto$ RQ3). Typically, the more expressive rules become the more likely it is that they are misunderstood and the more computational intense it is to mine them, cf. [8,9]. Nonetheless, in order to cover the most prominent process perspectives, the work at hand supports association rules with respect to control flow, resources, and temporal process execution behavior.

Furthermore, the proposed approach prevents false positives as it supports noise and ad hoc changes ($\mapsto$ RQ1). This is because process executions, differently to monolithic signatures, no longer must be completely represented by the signatures, but only by a fraction of the rules which a signature is composed of, cf. [3]. This also enables to provide more details about the individual anomalies, as it can be reported which rules a process execution (trace resp.) supports and which not, but also the anomaly severity. The latter is composed of the aggregated and automatically calculated rule significance of each non-supported rule.

Overall, this work follows the design science research methodology, cf. [10]. The specific method is depicted in Fig. 1. At first, existing relevant literature was thoroughly analyzed based on a systematic literature review on process runtime anomaly detection, cf. [3]. Hereby, a range of limitations could be identified, such as, the lack of root cause analysis capabilities in the business process anomaly detection area, forming a foundation to address and identify related research problems and design requirements in the following.

Subsequently, design requirements are derived from existing work on anomaly detection in the security domain, in general, and the process domain specifically. As a result an existing rule formalism, i.e., association rules, lays the foundations for a novel anomaly detection approach, cf. Sections 2 and 3. This artifact is evaluated in two ways. In Section 4, its feasibility and the quality of the results are shown based on a proof-of-concept implementation and by performing a cross validation with real life process executions, injected anomalies and multiple state-of-the-art comparison approaches. Secondly, the expressive power of the approach is evaluated based on a controlled experiment with users (cf. Section 5).

In this experiment, different visualizations and representations of anomalies and their root cause are compared by the users in finding answers to questions on the anomalies. The correctness of the answers is analyzed in order to gain insights on the effectiveness of the proposed approach. In addition, the time for giving such answers is also measured to evaluate its efficiency. All findings are then compared with related work (cf. Section 6) and the achieved root cause analysis capabilities are discussed

Table 1
Exemplary running example log L containing the exemplary traces t_1 and t_2.

Event e	Process P	Trace t	Activity ea	Resource er	Start timestamp es	End timestamp ec
e_1	P_1	t_1	A	Mike	1	5
e_2	P_1	t_1	B	Mike	6	9
e_3	P_1	t_1	C	Sue	12	16
e_4	P_1	t_2	B	Mike	18	21
e_5	P_1	t_2	A	Tom	22	29
e_6	P_1	t_2	C	Sue	32	38

in Section 7. Stakeholders of the proposed approach are process managers and security experts.[1]

Overall, this work contributes towards a flexible anomaly detection approach that is robust against process change and noise. Its visualization seems promising to foster a more effective and efficient understanding to anomaly root causes than existing approaches.

2. Requirements, fundamentals, and overall approach

This section starts off with a summary on design requirements for an anomaly detection approach in flexible process runtime behavior. Fundamental notions are introduced in the sequel. Finally, we sketch the overall approach which is presented formally in Section 3.

2.1. Design requirements

The design requirements for the approach are selected based on the systematic literature review on approaches for anomaly detection in process runtime behavior conducted in 2017 [3]. In the following, these requirements are augmented – where applicable – with more general research challenges, as identified in [4].

- **DR1: Detecting anomalies during runtime:** Both reviews [3,4] find that most of the existing approaches focus on ex post anomaly detection. Although this can also yield valuable insights the need for detecting anomalies when they are actually happening, i.e., during runtime when a process model is currently executed, is prevalent. This is, because ex post analysis can only reveal that an anomalous incident, such as, an attack was already completely executed, and, e.g., that attackers have already achieved their malicious goals. In comparison dynamic runtime analysis capabilities bear the chance to delay or stop an attack in its tracks before it unfolds potential negative effects, such as, data loss.
  Challenges: performance and process changes *Design choices:* avoid monolithic signatures, provide lightweight, simplistic and flexible signatures
  How to address: represent signature by short comprehensible association rules as "pieces" of a lightweight signature
- **DR2: Reduction of False Positives:** Change and flexibility are one of the open research challenges stated in [3], but will currently result in *"assessing correct behavior as anomalous"* and hence pump up the number of false positives.
  Challenges: distinction of false positives, high frequency of changes
  Design choices: keep signature complexity and interconnection to a minimum in order to be able to adapt them at any given point in time; support root cause analysis in order to distinguish between benign and malign anomalies
  How to address: represent signature by association rules which can easily be adapted in case of deviating behavior; this facilitates distinction into benign and malign anomalies
- **DR3: Explaining Root Causes:** [4] states that *"a holistic security approach also includes the involvement and awareness of humans"*. However, [3] finds that most approaches operate in *"black box mode"*, i.e., it is not sufficient to present only that a trace was detected as being anomalous. Instead details on anomaly severity, causes, and further explanatory information should be provided.
  Challenges: determine root cause, convey information to users
  Design choices: integrate root cause representation into anomaly detection, i.e., the results of the anomaly detection should already provide information on root causes; representation by visualization
  How to address: association rules enable tracing back root causes in a fine-granular way; use representation inspired by Ishikawa or "fishbone" diagrams as they have proven useful for root cause identification [12,13]

Design requirements DR1–DR3 address a subset of the challenges stated in [3,4]. Another challenge is, for example, to deal with instance spanning anomalies instead of limiting the approaches to find single instance anomalies. For the work at hand, however, DR1–DR3 lead to a balanced approach focusing on a flexible, runtime anomaly detection with integrated root cause explanation support.

2.2. Fundamental notions

This paper proposes an anomaly detection heuristic to classify process execution traces as anomalous or not. For this association rules are mined from a bag of recorded execution traces L (i.e., a log). This is beneficiary as L:

(a) represents real process execution behavior;
(b) incorporates manual adaptions, noise, and ad hoc changes;
(c) is automatically generated during process executions; and
(d) is independent from abstracted/outdated documentation, cf. [14].

The proposed approach is *unsupervised* as the traces in L are not labeled, formally:

Definition 1 (*Execution Log*)**.** Let L be a bag of `execution traces` $t \in L$; $t := \langle e_1, \ldots, e_n \rangle$ holds an ordered list of execution `events` $e_i := (ea, er, es, ec)$; e_i represents the execution of activity $e_i.ea$, by resource $e_i.er$, which started at timestamp $e_i.es \in \mathbb{R}_{>0}$ and completed at $e_i.ec \in \mathbb{R}_{>0}$; the order of t is determined by $e_i.es$.

This notion represents information provided by process execution log formats such as, the eXtensible Event Stream,[2] and enables the representation of the control, resource, and temporal perspective on a trace level. As this work is not yet aiming on the analysis of additional perspectives and event/case attributes, we opted not to represent such information, for the sake of simplicity, in Definition 1. Accordingly, the first event in the running example (cf. Table 1) would be denoted as $e_1 = (\texttt{A}, \texttt{Mike}, 1, 5)$.

The following *auxiliary* functions are used:

[1] Note that this work extends [11].

[2] http://xes-standard.org/ — IEEE 1849–2016 XES Standard.

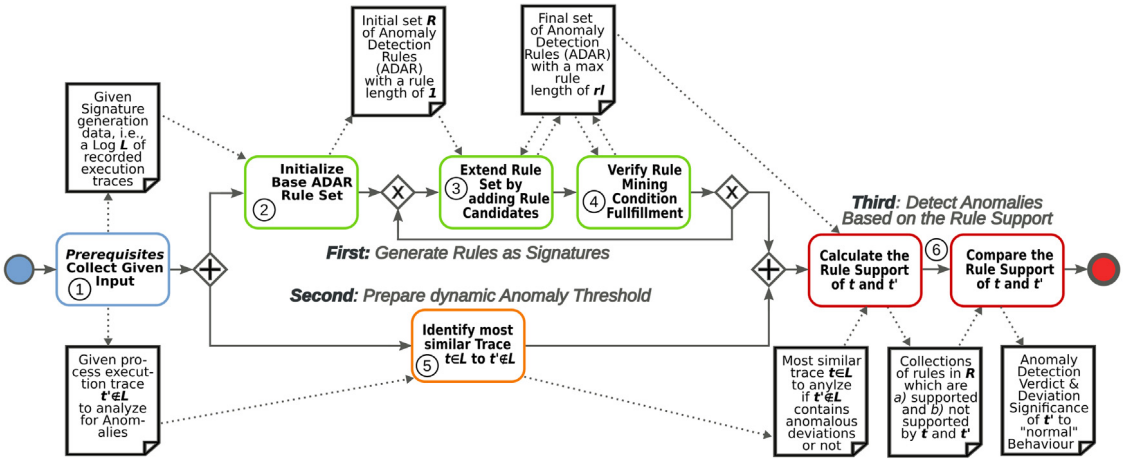

Fig. 2. Proposed rule based process anomaly detection approach — overview.

- $\{\cdots\}^0$ returns a random element from a set/bag.
- $c := a \oplus b$ appends b to a copy c of the collection given by a.
- $\langle\cdot\rangle^l$ retains the last element of a list
- $\langle\cdot\rangle_i$ retains the list item with index $i \in \mathbb{N}_{>0}$.
- $\langle\cdot\rangle_i^+$ retains all list items with an index $> i$.

Fig. 2 provides an overview of the proposed anomaly detection heuristic; algorithms are presented in Section 3. Firstly, a signature R is created for a process P based on the associated execution log L (rule mining), i.e., L is assumed as given input ①. Here, a signature R is a set of rules that represent behavior mined from L. A rule represents, e.g., that activity `C` should be a successor of `A`.

The applied rule mining approach is inspired from the Apriori algorithm [15] which mines valuable relationships in large unordered data sets as association rules. However, process execution traces are temporally ordered, e.g., based on the start timestamp of each event. To take this aspect into account (a) the Apriori algorithm is adapted accordingly, cf. Section 3; and (b) association rules, as defined in [15], are extended into Anomaly Detection Association Rules (**ADAR**) — (rule for short), cf. Definition 2.

The applied adaptions to the Apriori algorithm are inspired from related sequence mining approaches, i.e., GSP [16] and, especially, *PrefixSpan*; enabling to gradually reduce the computational effort (which is relative to the complexity and amount of traces analyzed throughout rule mining) during each rule mining step the longer each rule becomes, cf. [17]. Overall this enables to achieve a sufficient rule mining performance, cf. Section 4, while the implementation is still simplistic and can directly be applied on the given expandable Definition 1 of traces, execution events, and logs used throughout this work. Enabling ADARs to represent common process execution behavior, to detect control, resource, and temporal execution behavior deviations (i.e., potential anomalies) from expected behavior as unsupported rules, as those can be indicators for potential fraud, misuse, or novel attacks, cf. [2,7,18,19].

For the analysis of temporal execution behavior, (*fuzzy*) *temporal association rules* can be applied. However, related mining approaches were found to be designed with different objectives in mind than this work; such as, the identification of seasonality [20], the analysis of inter-event-intervals [21], or the analysis of unordered item lifespans [22]. In comparison, this work is interested into analyzing individual activity execution durations in combination with control flow sequences which requires to combine concepts mentioned beforehand. Accordingly, ADARs are formally defined as:

Definition 2 (*ADAR*)**.** Let L be a set of execution traces.

A `rule` r is defined as $r := \langle rp_1, \ldots, rp_m \rangle$ with `conditions` $rp_j := (ra, rd)$.

For $j = 1, \ldots, m$, $rp_j.ra$ represents an expected activity and $rp_j.rd \subseteq \{\texttt{low}, \texttt{avg}, \texttt{high}\}$ an optional execution duration represented as classes `low`, `avg`, and `high`. All conditions in r must be matched by $t \in L$ to conclude that t supports r, cf. Definition 4. The condition indices j represent their expected order in r, i.e., rp_j must be matched by a trace before rp_{j+1} can be.

The following projections were defined for rules (r) and conditions (rp): $rt(r) \mapsto \{\texttt{control}, \texttt{temporal}, \texttt{SoD}, \texttt{BoD}\}$ represents that a rule can specify control flow (control for short), temporal, or resource behavior. The latter focuses on the assignment of resources to activities which is analyzed in the form of Separation of Duty (SoD) or Binding of Duty (BoD), cf. [23]. Depending on the rule condition type (e.g., control or temporal) different elements (i.e., $rp.ra$ and $rp.rd$) of each rule condition $rp := (ra, rd)$ become relevant. In the following examples non-relevant elements will be denoted as $\cdot$. For example, a control flow focused rule condition for activity `A` will, in the following, be denoted as $rp := (\texttt{A}, \cdot)$ because the duration representation $rp.rd$ is solely relevant for temporal rule conditions, cf. Example 1.

Further, as rules (conditions) are mined based on given traces (events); we define projections for rules (conditions) on their trace (event), such that:

- $rtr(r) := t$ returns a trace $t \in L$ which has motivated the rule r during rule mining.
- $re(rp) := e$ is similar to $rtr(r)$ but returns an event $e \in t \in L$ which motivated the rule condition rp during rule mining.

It was chosen to focus on the described three process execution behavior perspectives (i.e., control, temporal, and resource behavior) as we found that these are most commonly covered by existing work, cf. [3]. Support for additional perspectives can be added by specifying additional conditions, see Section 3.1.

Example 1 (*ADAR, Cf. Table 1*)**.** Imagine rule $r_1 := \langle rp_1, rp_2 \rangle$ where $rt(r_1) = \texttt{control}$, $rp_1 = (\texttt{A}, \cdot)$, and $rp_2 = (\texttt{B}, \cdot)$ is matched with the running example. As r_1 is a control flow rule, execution duration classes are not relevant/defined. While trace t_1 supports r_1 the second does not. This is because an execution of activity `A` succeeded by an execution of activity `B` is only given in trace t_1. If $rp_1.ra = \texttt{A}$ and $rp_2.ra = \texttt{C}$ then the rule would be supported by both traces t_1 and t_2.

2.3. Overall approach

The applied rule mining consists of *three* stages (cf. Fig. 2). Initially, the basis for the mining is laid by converting each event e, given by L's traces, into an individual rule (see ②). Hence, at this stage each rule only holds a single condition, so that $\forall r \in R; |r| = 1$. In the following, these initial set of rules (the individual rules in R, resp.) is repeatedly extended and verified to create the final anomaly detection rule set.

Example 2 (*Single Rule Condition, Cf. Table 1*)**.** When assuming that the running example log only consists of t_1 then the initial rule set is $R := \{r_1, r_2, r_3\}$ where each rule consists of a single rule condition, e.g., $r_1 := \langle rp_1 \rangle$ where $rp_1.ra = \texttt{A}$ given that $rt(r_1) = \texttt{control}$.

Subsequently, rule *extension* and *verification* approaches are applied in an iterative way. The rule extension, see ③, extends each rule in R by one additional rule condition in each possible way to identify new potential rules.

Example 3 (*Rule Extension, Cf. Table 1*)**.** To extend r_1 all successors of activity A (i.e., the last rule condition, $\mapsto r_1^l$, in r_1, cf. Definition 3) are determined, i.e., activity B and C. Secondly, activity B and C are utilized to extend the ADAR r_1 into $r_1' := \langle rp_1, rp_2' \rangle$ and $r_1'' := \langle rp_1, rp_2'' \rangle$ where $rp_2'.ra = \texttt{B}$ and $rp_2''.ra = \texttt{C}$.

Formally, the rule extension is defined as:

Definition 3 (*Extending Individual ADARs*)**.** Let $\mathcal{E}$ be the bag of all events in a log L, $\mathcal{RP}$ be the set of all rule conditions and R be the set of all rules. Let $r := \langle rp_1, \ldots, rp_m \rangle$ be a rule and $t := \langle e_1, \ldots, e_n \rangle \in L$ be an execution trace with $rtr(r) = t$. Rule extension function $ext : R \times L \mapsto R$ extends r by:

$$ext(r, t) := \{r \oplus torp(e, rt(r)) | e \in E_{aux}\} \tag{1}$$

where

- $torp : \mathcal{E} \times \{\texttt{control}, \texttt{temporal}, \texttt{SoD}, \texttt{BoD}\} \mapsto \mathcal{RP}$ converts an event e into a rule condition rp of the given type, e.g., control.
- $E_{aux} := \{e' \in t | e'.es > e''.es; e'' = re(rp_m)\}$ determines events which will individually be attached to r to span a set of extended rules.

Section 3.1 defines how the extension described in Definition 3 is performed for each of the four rule types, i.e., $rt(r) \in \{\texttt{control}, \texttt{temporal}, \texttt{SoD}, \texttt{BoD}\}$.

Finally, rule verification is applied ④. Each rule is verified by analyzing its support, i.e.,

$$sup(r, L) := |\{t \in L | mp(r, t) = \texttt{true}\}| \,/\, |L| \tag{2}$$

where function mp is defined in Definition 4. The support of a rule represents the percentage of traces in L that a rule could be successfully mapped to (match the rule conditions, resp.). If the support (i.e., the percentage of traces $t \in L$ that a rule supports) of a rule is below a user configurable threshold $mins \in [0, 1]$, then the rule is removed from R. Subsequently, the rule extension and verification steps are applied repeatedly until the rules in R are extended to a user configurable maximum length of $rl \in \mathbb{N}_{\geq 1}$ rule conditions.

The verification variables *mins* and *rl* enable to fine tune rules for specific use cases and process behavior. For example, we found that the mining of longer rules resulted in stricter signatures than the mining of short rules. In comparison choosing a low *mins* value could result in overfitting the signatures and a high amount of rules. Further discussions on the variables are given in Section 4.

We also have investigated alternative measures, common to association rules, for the calculation of rule significance, e.g., *lift* or *confidence*. While those were found to result in mining a smaller set of rules (signature, resp.) it also increased the number of false positives reported by the proposed approach. We were left with the impression that this was caused because the smaller, e.g., more confident rule set resulted in a stricter representation of the behavior in L — providing less freedom for fluctuations and ad hoc chances (i.e., overfitting occurred). Unfortunately, this effect persisted even after relaxing the chosen thresholds in an effort to mitigate it.

Definition 4 (*ADAR Mapping*)**.** Let $r := \langle rp_1, \ldots, rp_m \rangle$ be a rule (cf. Definition 2) and $t := \langle e_1, \ldots, e_n \rangle$ an execution trace (cf. Definition 1). Mapping function $mp : R \times L \mapsto \{\texttt{true}, \texttt{false}\}$ determines if r is supported by (matching to, resp.) t. Rule type $rt(r)$ determines the matching strategy to be applied, see Section 3.1.

Example 4 illustrates the ADAR mapping.

Example 4 (*ADAR Mapping, Cf. Table 1*)**.** The rule r_1', as given in Example 3 achieves a support of 0.5. This is because it expects that activity A is succeeded by activity B. Accordingly, it can only be mapped onto trace t_1 but not on t_2. Imagine, that *mins* was defined as 0.9, then r_1' would be removed during the verification phase from R as $0.5 < 0.9$. In comparison rule r_1'' would not be removed as it is supported by both traces t_1 and t_2 (i.e., $sup(r_1'', L) = 1$ so that $1 \not< 0.9$). Rule r_1'' matches to (is supported by, resp.) traces where A is succeeded by activity C.

Finally, the mined rules R (the signature) are applied to classify a given process execution trace $t' \notin L$ as anomalous or not. For this a trace $t \in L$ is identified that is most similar to t', see ⑤. The similarity between traces is measured based on the occurrence of activities in the compared traces, cf. Definition 5. Then t' and t are mapped onto R's rules to determine the aggregated support of both traces. Finally, if the aggregated support of t' is below the aggregated support of t then the given trace t' is classified as being anomalous, see ⑥.

3. ADAR based Anomaly Detection

This section presents the algorithms for the approach set out in Fig. 2.

3.1. Association rule mining for Anomaly Detection

The applied ADAR mining approach, cf. Algorithm 1, combines the main mining steps described in Fig. 2. This is the rule set *initialization*, along with the iteratively applied rule *extension* (cf. Definition 3) and *verification* steps (cf. Definition 4). Depending on the *user chosen* rule type $ty \in \{\texttt{control}, \texttt{temporal}, \texttt{SoD}, \texttt{BoD}\}$ different algorithms are applied to mine either control, temporal, or resource behavior given in L into rules. While each rule type is mined individually, rules of all types can be combined in a single signature (i.e., the union of all individual rule sets).

The auxiliary function $torp(e, ty) : rp$ (cf. Definition 3) transforms an event e into a rule condition rp. Depending on the chosen rule type ty one out of the three rule condition mining approaches presented in Section 3.1.1 to Section 3.1.3 is applied. For example, if $ty = \texttt{control}$ then the control flow rule mining approach presented in Section 3.1.1 is used.

Algorithm ruleMining(*log L, min support mins, max rule length rl, rule type ty*)
  Result: set of mined rules R (i.e., a signature)
  $R := \varnothing$; // initially the rule set (signature, resp.) is empty
  foreach $t \in L$ **do** // initializing the rule set R with base rules
    foreach $e \in t$ **do**
      $R := R \cup \{\langle torp(e, ty)\rangle\}$ // initial base rule with one condition
  for $rcsize := 0$ **to** rl **do** // generate rules up to a size of rl conditions per rule
    $R := \{ext(r, rtr(r)) | r \in R\}$ // extend rules in R, cf. Definition 2 and Definition 3
    foreach $r \in R$ **do**
      if $sup(r, L) < mins$ // verification, calc. rule support, cf. Definition 4 **then**
        $R := R \setminus \{r\}$ // remove r from R because its support is too low
  return R // final set of mined rules R for behavior given by the log L

Algorithm 1: Mines rules for a given execution log L and rule type ty.

3.1.1. Mining control flow ADARs

Control flow rules represent expected activity orders, e.g., that activity A should be succeeded by activity C during a process execution. Hereby, control flow rules enable to identify process misuse, cf. [2], such as, the execution of a financially critical "bank transfer" activity without the previous execution of a usually mandatory "transfer conformation" activity.

Event to control condition Accordingly, during rule extension, an event e is transformed into a rule condition $rp = (e.ea, \cdot)$. Given the running example (cf. Table 1) e_1 would be transformed into rule condition $rp = (\texttt{A}, \cdot)$.

Control ADAR Support Trace t supports a control flow rule r if t holds all activity executions specified by the rule conditions in $rp \in r$, cf. Fig. 2, ①. Further the activity executions must occur in accordance to the order of rule conditions in r, ②. This represents that activity executions are mutual dependent on each other. ①and ②are verified by Algorithm 2 to determine if a trace t supports the control flow rule r.

Example 5 (*Control Flow ADAR, Cf. Table 1*)**.** Rule $r = \langle rp_1, rp_2\rangle$ where $rp_1.ra = \texttt{A}$ ($rp_1 = (\texttt{A}, \cdot)$, resp.) and $rp_2.ra = \texttt{B}$ (meaning that activity A must be succeeded by B) would only be supported by trace t_1 but not by t_2.

Algorithm ControlSupport(*trace t, control flow rule r*)
  Result: if r is supported by $t \mapsto$ true or not $\mapsto$ false
  for $j = 1$ **to** $|r|$ // $|\langle\cdot\rangle|$ retains the length of the list $\langle\cdot\rangle$ **do**
    for $i = 1$ **to** $|t|$ **do**
      if $t_i.ea = r_j.ra$ // verify control flow rule condition matching **then**
        $t := t_i^+$; $r := r_j^+$; **break** // remove successfully matched parts of t, r
  return $|r| = 0$?*true:false* // return true if r fully matches to t else false

Algorithm 2: Checks if a trace t supports the control flow rule r.

This work applies a relaxed rule matching. Hence, a rule is assumed as supported by a trace as long as this trace contains at least a single combination of events that match to the rule conditions. This enables to deal with loops and concurrency. Moreover, the approach is flexible enough to avoid struggle with noise and ad hoc changes. However, as found during the evaluation it is still specific enough to differentiate benign and anomalous process executions, cf. Section 4.

3.1.2. Mining temporal ADARs

Temporal rules focus on activity durations (i.e., the timespan between the start and completion of an activity execution). Those were found to be a significant indicator for fraud and misuse, cf. [18,19]. However, while control flow rules focus on representing distinct values (e.g., explicitly activity A is expected) this is not possible for temporal rules. This is because distinct durations, e.g., one second or one hour, are so specific that even a minor temporal variation, which we assume as being likely, would render a rule to be no longer supported by a trace. This can, potentially, result in false positives.

To tackle this challenge we apply *fuzzy temporal rules*. These rules are not representing durations with explicit values but with duration classes. These classes represent, for example, that the expected duration of activity A is roughly comparable or below/above the average execution duration of activity A — given by the traces in L. In this work three duration classes are in use, i.e., $PDC := \{\texttt{low}, \texttt{avg}, \texttt{high}\}$. Increasing the number of classes would be possible but it was found that this can result in overfitting the generated temporal rules (signature).

Event to temporal condition A temporal rule condition consists of an expected activity execution along with its expected duration classes. For this the activity execution duration is represented as a subset of the possible duration classes PDC. So, based on an event e a temporal condition rp is constructed by defining the expected activity, i.e., $rp.ra := e.ea$ and selecting one or more duration class which are expected to be observed, e.g. $rp.rd := \{\texttt{low}\} \subseteq PDC$.

Algorithm TempClass(*event e, log L, duration classes PDC, widen* $w \in [0; 1]$)
  Result: set of representative duration classes $DC \subseteq PDC$ for e
  // calculate durations D for L, min and max duration, duration class timespan *part*, duration d of event e, relative class timespan widening $wspan$
  $D := \{e'.ec - e'.es | e' \in t, t \in L : e'.ea = e.ea\}$
  $min := \{d | d \in D, \forall d' \in D; d \leq d'\}^0$; $max := \{d | d \in D, \forall d' \in D; d \geq d'\}^0$
  $part := (max - min)/|PDC|$; $d := e.ec - e.es$; $wspan = part \cdot w$; $i := 0$
  foreach $pdc \in PDC$ // check for each class in PDC if it is representative **do**
    $start := min - wspan + part \cdot i$; $end := start + wspan \cdot 2 + part$; $i := i + 1$
    if $d \geq start \wedge d \leq end$ **then**
      $DC := DC \cup \{pdc\}$
  return DC // set of representative duration classes for event e

Algorithm 3: Determines for a log L the duration class for an event e.

Algorithm 3 determines the representative duration classes for an event e based on L. Hereby, variable $w \in [0; 1]$ "widens" the covered timespan of each duration class so that the rule support calculation becomes less strict to prevent overfitting. Compare Fig. 3. It depicts the three duration classes of PDC and how widening affects them. For example, while the activity duration ①can clearly be represented by class low this is not the case for the duration ②. As this duration is between the avg and the high class the "widening" (w) comes into effect, so that ②is represented by both classes. Accordingly, the two exemplary constraints $rp_1.rd = \{\texttt{avg}\}$ and $rp_2.rd = \{\texttt{high}\}$ would both match to ②.

Example 6 (*Temporal ADAR, Cf. Table 1*)**.** Converting e_1 into a temporal rule condition rp_1 results in $rp_1.ra = \texttt{A}$ while $rp_1.rd = \{\texttt{low}\}$ when using a widening factor of $w := 0.1$. Given this widening factor, the average class for activity A would match durations between 3.9 and 4.1. In comparison event e_5 would convert into a condition rp_2 so that $rp_2.ra = \texttt{A}$ while $rp_2.rd = \{\texttt{high}\}$.

Example 7 (*Temporal ADAR, Cf. Table 1*)**.** Converting e_4 into a temporal rule condition rp_1 results in $rp_1.ra = \texttt{B}$ while $rp_1.rd = \{\texttt{avg}\}$ when using a widening factor of $w := 0.2$. Given this widening factor, the average class for activity B would match durations between 2.8 and 3.2. In comparison event e_2 would convert into a condition rp_2 so that $rp_2.ra = \texttt{B}$ while $rp_2.rd = \{\texttt{avg}\}$ as both e_4 and e_2 match to the average execution duration class of B (i.e., both events show an execution duration of 3).

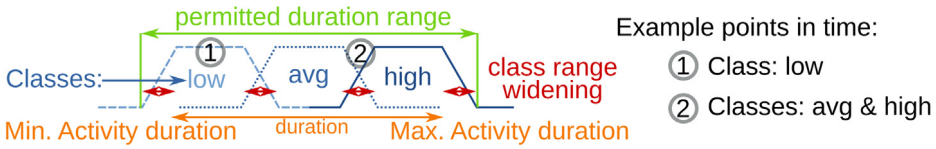

Fig. 3. Duration class representation for temporal ADARs.

Temporal ADAR Support A trace t supports the temporal rule r (i.e., $rt(r) = \texttt{temporal}$) if t holds all activity executions specified by the rule conditions in r with the expected durations. In addition, these activity and duration pairs must occur in the expected order given by r's conditions. To verify this Algorithm 2, is extended by calculating and comparing duration classes, cf. Algorithm 4.

Algorithm `TempRuleSupport(`*trace* t, *temporal rule* r, *duration classes* PDC, $w \in [0; 1]$`)`
- **Result**: `true` if r is supported by t; `false` otherwise
- **for** $j = 1$ **to** $|r|$ `//` $|\langle\cdot\rangle|$ `retains the length of the list` $\langle\cdot\rangle$ **do**
  - **for** $i = 1$ **to** $|t|$ **do**
    - **if** $t_i.ea = r_j.ra \wedge ((TempClass(t_i, L, PDC, w) \cap t_i.rd) \neq \emptyset)$ **then**
      - $t := t_i^+$; $r := r_j^+$; **break** `//` `cf. Algorithm 2`
- **return** $|r| = 0$?*true*:*false* `//` `return true if` r `fully matches` t `else false`

Algorithm 4: Checks if trace t supports the temporal rule r

3.1.3. Mining SoD and BoD ADARs

Separation and Binding of Duty rules represent expected relative pairs of activities and resource assignments, cf. [23], i.e., all activities covered by a SoD rule must be executed by different resources while all activities covered by a BoD rule must be executed by the same resource. Failing to support resource rules can be an indicator for fraudulent behavior, cf. [2,7].

Event to resource condition Converting an event e into a SoD or BoD rule condition rp is performed by extracting the activity related to e, i.e., $rp.ra := e.ea$. Accordingly, for e_1, in Table 1, $rp.ra = \texttt{A}$ holds.

Resource ADAR Support To verify if a trace t supports a resource (res. in short) rule r, a set is generated that holds all resources that have executed activities which are specified in r's conditions (cf., $rp.ra$), i.e., $RS := \{e.r | e \in t \wedge e.ea \in \{rp.ra | rp \in r\}\}$

For a BoD rule it is expected that all executions utilize the same resource, i.e., $|RS| = 1$. For a *SoD* rule the amount of resources taking part in the activity executions should be equal to the amount of conditions, i.e., $|RS| = |r|$.

Example 8 (*Res. ADAR SoD, Cf. Table 1*)**.** Rule $r = \langle rp_1, rp_2 \rangle$ where $rp_1.ra = \texttt{A}$, $rp_2.ra = \texttt{B}$, and $rt(r) = \texttt{SoD}$ would only be supported by trace t_2 but not by t_1. This is because, here, r specifies that activity `A` and `B` must be executed by `different` resources. Hence for trace t_1 the set $RS = \{\texttt{Mike}\}$ (i.e., $|RS| = 1$) while $|r| = 2$ so that $|RS| \neq |r|$.

Example 9 (*Res. ADAR BoD, Cf. Table 1*)**.** Rule $r = \langle rp_1, rp_2 \rangle$ where $rp_1.ra = \texttt{A}$, $rp_2.ra = \texttt{B}$, and $rt(r) = \texttt{BoD}$ would only be supported by trace t_1 but not by t_2. This is because, here, r specifies that activity `A` and `B` must be executed by the `same` resource. Accordingly, for trace t_1 the set $RS = \{\texttt{Mike}\}$, such that, $|RS| = 1$ while for t_2 the set RS would become $RS = \{\texttt{Mike}, \texttt{Tom}\}$ such that $|RS| \neq 1$. In the latter case BoD would be violated.

3.2. ADAR based anomaly detection

The mined ADARs (i.e., a signature) are applied to classify a given trace $t' \notin L$ as anomalous or not. For this the artificial *likelihood* of t' is calculated and compared with the likelihood of the trace $t \in L$ that is most similar to t'. If t' is identified as *less* likely it is assumed as being anomalous, cf. Definition 5. Hereby, the presented approach follows and exploits the common assumption that anomalous behavior is less likely than benign behavior, cf. [3,7,18], while also enabling dynamic anomaly threshold identification. In comparison to existing work, cf. [3], the latter increases the flexibility of the proposed approach. This is because this work, other than, e.g. [18], does not rely on a fixed predefined detection threshold which is fixated once and from there on needs to be suitable for "all" potentially upcoming process executions. To further extend on this concept one could create an average likelihood threshold based on a range of most similar traces to reduce the risk of overfitting — which will be explored in future work. A comparable idea was implemented by us in [7].

The artificial likelihood of a trace is determined by aggregating the overall support (based on L) of the rules which the trace is supporting, cf. Definition 4. This implies: the less rules a trace supports the less likely it and its occurrence is assumed to be and the more likely it is anomalous.

Definition 5 (*Anomaly Detection*)**.** Let L be a bag of all traces t (i.e., an execution log) and t' be an execution trace with $t' \notin L$. Let further R be a set of rules, i.e., a signature that was mined for L. Whereas $\mathcal{R}$ be the set of all signatures and $\mathcal{L}$ be the set of all Logs, such that, $L \in \mathcal{L}$. Anomaly detection function $adec : \mathcal{R} \times \mathcal{L} \mapsto \{\text{true, false}\}$ is defined as:

Case 1: $adec(R, t') := \texttt{true}$ `if`

$$\sum_{\substack{r \in R \\ mp(r,t')=\texttt{true}}} sup(r, L) < \sum_{\substack{r \in R \\ mp(r,tsim(t',L))=\texttt{true}}} sup(r, L)$$

Case 2: $adec(R, t') := \texttt{false}$ `otherwise.`

where $tsim(t', L)$ returns the trace $t \in L$ that is most similar to $t' \notin L$.

The proposed anomaly detection approach requires to identify, for a given trace $t' \notin L$, the most similar trace $t \in L$. For this function $tsim(t, L) : t$ is applied. In detail: both traces are first converted into bags of activities (each one holds activities executed by the respective trace). Subsequently, the Jaccard similarity $J(\{\cdots\}, \{\cdots\})$, cf. [24], between both activity bags is calculated. This means: the more equal activities[3] the traces contain (have executed) the more similar they are. For example: $J(\{\texttt{A}, \texttt{C}\}, \{\texttt{B}, \texttt{C}\}) = |\{\texttt{A}, \texttt{C}\} \cap \{\texttt{B}, \texttt{C}\}| \,/\, |\{\texttt{A}, \texttt{C}\} \cup \{\texttt{B}, \texttt{C}\}| = 0.\overline{3}$.

More calculation intensive approaches, such as, edit distances (e.g., the Damerau–Levenshtein distance) could be applied to get a more thorough representation of activity orders, as demonstrated by us in [25]. Such can especially be beneficial if activity orders in the analyzed traces are fluctuating while roughly the same activities are executed overall. Such a situation did not emerge throughout the evaluation. Hence, while the underlying similarity measure can be user chosen the described approach was found to be fast and sufficient during the evaluation, cf. Section 4.

3.3. Fostering root Cause analysis and understandability

The detection of anomalies constitutes only the first step in a large anomaly management process [12]. In addition it must also

[3] Activity equivalence is considered as label equivalence here.

be analyzed which context, behavior, policies or errors motivated the classification of an execution as being anomalous, cf. [26]. The latter is typically subsumed as *root cause analysis* which paves the ground for the selection and application of suitable anomaly countermeasures (e.g., the termination or adaption of anomalous instances). In general, "*root cause analysis is a structured investigation that aims to identify the true cause of a problem and the actions necessary to eliminate it*" [12].

So far, *root cause analysis* capabilities have been mostly neglected in the process anomaly detection domain, cf. [3]. Related work focuses on the technical perspective of detecting anomalies in the first place. However, this can significantly lower the positive impact of such detection approaches as it is our assumption that an anomaly can only be truly resolved by understanding and addressing its cause. Accordingly, this work proposes the novel visualization technique **A_Viz** to foster anomaly focused root cause analysis to address following limitations:

Severity: existing process anomaly detection approaches frequently generate *binary* results, i.e., they mark traces either as anomalous or not, cf. [3]. However, this does hardly support experts when in need to (a) assess the severity of an reported anomaly and (b) deduce the reasons why an execution was deemed anomalous. Thus, as stated in [9], providing only binary results is generally not sufficient. Hence, we propose to utilize the aggregated rule support of a signature/traces (t vs. t') as an indicator for the deviation severity between a trace and a signature.

Granularity: binary detection results are also insufficient to perform a thorough anomaly analysis as they do not indicate which specific parts of a trace did not comply to the utilized signatures. In comparison, the proposed approach comprises the signature from multiple fine granular rules which can be individually reported as supported or not; enabling to report which parts of a given trace were affected by an identified anomaly in a fine granular way.

Simplicity and clarity: during the evaluation the mined signatures were found to contain a relative low amount of rules (e.g., below 100 temporal and control rules) while an even lower amount of rules was typically violated by an anomalous trace. Given the low amount of (violated) short and simple rules those can, likely, easily be grasped and taken into account by experts when analyzing process executions (traces resp.) which were found to be anomalous for anomaly root causes.

Transparency: is required to foster trust in the anomaly detection results to achieve a wide spread use of related detection approaches. Hence, we propose that it must not only be obvious that an execution trace is anomalous and why but also which parts of traces were *not* identified as being anomalous. This enables experts to assess strengths and weaknesses of a detection approach but also to identify and report overlooked anomalous behavior to form a training set which can be used to further improve related anomaly detection approaches.

Relation: Existing approaches were found to utilize separate screen locations to display (a) the execution trace, and (b) information about related anomaly detection results. We assume that this hardens it for an expert to draw conclusions about relations between anomalous execution events and the related anomaly detection results (e.g., which rules were violated by a given event). Hence, such related details should, likely, be located in close proximity.

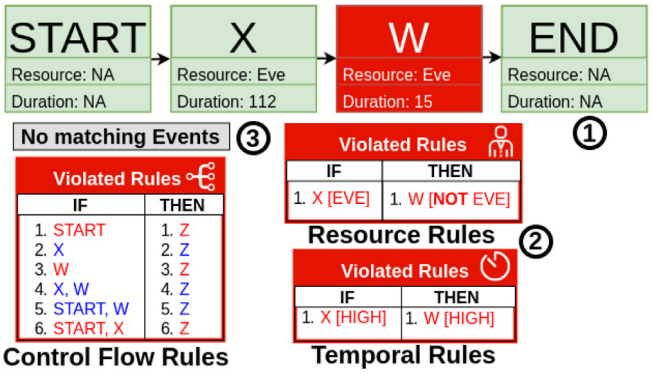

Fig. 4. Visualizing a rule violation with **A_Viz** (example).

A_Viz builds and expands on visualization approaches utilized in related areas, such as, business process compliance research. There, business process models and executions are, for example, colored to indicate that a given execution part, such as an activity, is compliant (green) or not (red), cf. [27]. Further, Ishikawa (fishbone) diagrams are applied to organize the relations between problem causes, cf. [28]. The latter, are typically created manually while **A_Viz** can be created automatically based on the analyzed traces and mined rules.

Fig. 4 gives an example of the proposed visualization approach. It consists of two main parts: The process execution trace ①which was deemed as being anomalous, and ②/③, i.e., the rules which were found to be violated (not supported, resp.) by that trace.

Here, color is applied to visualize differences between process execution events which have not violated any rules (green) and events which have violated rules (red) (i.e., which do not comply to expected behavior). Note, that also a third color is utilized (blue) indicating that a less significant rule was violated (i.e., a rule which is partly also violated by process executions which were, overall, classified as being non-anomalous).

The latter can occur in the case of loops and parallel executions, were, e.g., activity executions overlap each other in different ways and orders or were, e.g., the same activity can be observed four or fife times in a row — depending on the number of loop iterations. The chosen colors/shades were combined in a way that they are still distinguishable by applicants which suffer from red–green deficiency. The latter was verified by checking out multiple potential color combinations with affected test subjects.

The proposed visualization takes over ideas from Ishikawa diagrams while positioning each graphical element. Hence, the rules which are violated by an execution event are positioned just below the event which triggered the rule violation decision. Compare with ②, it depicts a violated resource rule which aims at checking whether or not `X` and `W` are executed by different resources (SoD), cf. Fig. 4. As this is not the case the SoD rule is violated, `W` is identified as the culprit and the violated rule is displayed just below the potentially malicious execution event for `W`. This rule positioning approach was motivated by our assumption that preceding execution behavior (the IF part of each rule) sets the expected behavior for succeeding behavior (the THEN part of each rule). Hence, we assume the event related to the THEN part of the rule as the wrongdoer and position the violated rule below it — similar to the concepts used in Ishikawa diagrams.

The proposed positioning approach is not applicable at all times. So, if an activity should be executed but is missing completed from an visualized trace then the executing event matching the THEN part of the rule is missing too. To stay consistent with the general idea to positing the violated rules below the

THEN part a workaround is applied by summarizing such events below a "No matching Events" caption, see ③in Fig. 4.

Overall, A_Viz supports *all three* rule types outlined throughout Section 3. Hereby, small icons, positioned in the top right corner of each rule box, in the form of tridents (*control flow*), clocks (*temporal behavior*), and stick figures (*resource behavior*, i.e., SoD or BoD) visualize the associated rule type. The icons are selected according to suggestions from literature on process compliance, e.g. [29]. The explanatory power of the proposed root cause visualization and analysis is evaluated in Section 5.

4. Evaluating the Detection quality of the proposed Anomaly Detection approach

The evaluation utilizes *real life* process execution logs from multiple domains and artificially injected anomalies in order to assess the anomaly detection performance and feasibility of the proposed approach. It was necessary to inject artificial anomalies as information about real anomalies are not provided by today's process execution log sources [3].

The utilized logs were taken from the BPI Challenge 2015[4] (BPIC5) and Higher Eduction Processes (HEP), cf. [30]. These logs were chosen because they are: (a) widely applied throughout existing process anomaly detection work, such as, [7,31,32], fostering comparability; (b) realistic and cover real world business process execution behavior; (c) enabling to evaluate the described approach based on logs with two different complexity levels (high $\mapsto$ BPIC5, medium $\mapsto$ HEP) and domains; and (d) containing all the relevant information in sufficient granularity, as, for example, we found that alternative logs frequently do not contain details on the related resources.

The BPIC5 logs hold 262,628 execution events, 5649 instances, and 398 activities. The logs cover the processing of building permit applications at five (BIPC5_1 to BIPC5_5) Dutch building authorities between 2010 and 2015. The HEP logs contain 28,129 events, 354 instances, and 147 activities — recorded from 2008 to 2011 (i.e., three academic years $\mapsto$ HEP_1 to HEP_3). Each trace holds the interactions of a student with an e-learning platform (e.g., exercise uploads). All logs contain sufficient details to apply the proposed approach (e.g., execution events, activities, timestamps, resource assignments, etc.).

The logs were randomly separated into training (for signature generation) and test data (for the anomaly detection performance evaluation). Subsequently, randomly chosen test data entries were randomly mutated to inject artificial anomalies. By randomly choosing which, how many, and how frequently mutators are applied on a single chosen test data entry (trace, resp.) this work mimics that real life anomalies are diverse and occur in different strengths and forms. Further the application of mutators enables to generate labeled non-anomalous (i.e., non-mutated) and anomalous (i.e., mutated) test data entries. Hereby, it becomes possible to determine if both behavior "types" are correctly differentiated by the proposed approach (cross validation). The applied mutators inject random control flow, temporal, and resource anomalies:

(a) *Control Flow* — mutators which randomly mutate the order and occurrence of activity execution events; and (b) *Temporal* — randomly chosen activity executions get assigned new artificial execution durations; and (c) *Resource* — activity/resource assignments are mutated to mimic, for example, BoD anomalies.

The applied mutators were adapted from our work in [7,19]. Combining multiple mutators enables to represent the diversity of real life anomalies. In addition, the applied random training and test data separation also evaluates if the proposed approach is capable of dealing with *benign* noise and ad hoc changes by not identifying them as anomalous. This is, because the test data contains benign behavior which is not given by the training data (e.g., bening ad hoc changes). The given evaluation results are an average of multiple evaluation runs, cf. Table 4. This enables to even out random aspects, such as, the random data separation and trace mutation.

Metrics and Evaluation Here, the feasibility of the presented approach is analyzed. For this, a cross validation is performed to determine if known anomalous (mutated) execution traces are correctly differentiated from known non-anomalous (non-mutated) ones. Through this four performance indicators are collected: True Positive (TP) and True Negative (TN), i.e., that anomalous (TP) and non-anomalous (TN) traces are correctly identified. False Positive (FP) and False Negative (FN), i.e., that traces were incorrectly identified as anomalous (FP) or non-anomalous (FN). Finally, these indicators are aggregated into:

(a) *Precision* $P = TP/(TP+FP)$ — if identified anomalous traces were in fact anomalous; and (b) *Recall* $R = TP/(TP+FN)$ — if all anomalous traces were identified (e.g., overly generic signatures could result in overlooking anomalies); and (c) *Accuracy* $A = (TP+TN)/(TP+TN+FP+FN)$ — a general anomaly detection performance impression; $TP, TN, FP, FN \in \mathbb{N}$; $P, R, A \in [0; 1]$.

An optimal result would require that TP and TN are high while FP and FN are low so that the accuracy becomes close to one. Further, the F_β-measure, Eq. (3), provides a configurable harmonic mean between P and R, cf. [33]. Hence, $\beta < 1$ results in a precision oriented result while $\beta = 1$ generates a balanced result.

$$F_\beta = \frac{(\beta^2+1)\cdot P \cdot R}{\beta^2 \cdot P + R} \tag{3}$$

Results The results were generated based on the BPIC 2015 and HEP process execution logs and following proof of concept implementation: https://github.com/KristofGit/ADAR The implementation was found to be capable of creating a signature within minutes and requiring only seconds to classify a trace as anomalous or not. Once generated the signatures can be reused and easily adapted by adding new rules or removing old ones, e.g., to address concept drift.

Primary tests were applied to identify appropriate configuration values, e.g., the maximum rule length $rl := 3$ (control and temporal) and $rl := 2$ (resource). Typically utilizing longer rules results in stricter signatures which are prone to overfitting, cf. [34]. Here, this seems not to be the case as the given dynamic threshold calculation utilized in this work mitigates this effect as (a) both, the trace analyzed for anomalies and its most similar counterpart in L commonly support similar rules; and (b) shorter rules (i.e., rules with a length below rl) are also generated, based on the outlined iterative extension and validation based rule mining approach, cf. Definition 3, and become part of the rule set R. Hence, a single long unsupported rule does not have a dramatic effect on the detection results (anomaly assessment, resp.). However, given that lower values for rl do not only result in shorter easier to grasp rules but also reduced calculation times low values were used for rl in the following. A comparison of the effect of different rl rule length configuration values on the anomaly detection performance (represented as average F1-measures) is given in Table 2. The best three results are printed in bold letters.

In comparison, the minimum support a rule has to achieve *mins* during the mining phase, to be accepted as a part of the signature, was set to 0.9 for control & resource rules and 0.8 for temporal rules. For this variable it was found that higher values could potentially result in a very small rule set or in finding no rules at all. In comparison, using a lower value could result in

[4] http://www.win.tue.nl/bpi/2015/challenge — DOI: 10.4121/uuid:31a308ef-c844- 48da-948c-305d167a0ec1.

Table 2
Effect of different *rl* values on F_1, cf. Eq. (3).

rl Control/Temporal	6	5	4	3	2
2	**0.9**	0.89	0.85	0.84	0.87
3	0.87	0.86	0.84	0.86	**0.9**
4	0.88	0.84	0.83	0.85	0.86
5	0.88	0.87	0.83	0.8	0.88
6	0.86	0.89	**0.9**	0.86	0.89

rl Resources Rules

Table 3
Effect of different *mins* values on F_1, cf. Eq. (3).

mins Control/Resource	1	0.9	0.8	0.7	0.6	0.5
0.5	0.78	0.75	0.74	0.72	0.6	0.7
0.6	0.76	0.78	0.76	0.76	0.74	0.76
0.7	0.76	0.78	0.78	0.77	0.75	0.77
0.8	0.77	0.76	0.79	0.78	0.77	0.77
0.9	0.77	0.76	**0.91**	**0.87**	**0.81**	0.76
1	0.77	0.77	0.77	0.76	0.75	0.78

mins Temporal Rules

finding a very high amount of rules. This does not necessarily result in better anomaly detection results as it increases, as we found, the risk of generating overfitting signatures. A comparison of the effect of different minimum support configuration values on the anomaly detection performance (represented as average F_1-measures) is given in Table 3. The best three results are printed in bold letters.

Finally, the fuzzy temporal rule generation can be configured based on the chosen temporal class widening variable w which was set to 0.2. Lowering this value will result in stricter signatures (temporal rules, resp.) that could potentially struggle when dealing with noise and ad hoc changes while a higher value would result in potentially overlooking anomalies as the signatures become less strict. However, when experimenting with different values for w (between, 0.01 and 0.6) the F_1 results, cf. Eq. (3), only fluctuated by 1%–3%. Given the low amount of configuration variables we assume that existing optimization algorithms should, likely, be able automatically find optimal settings for the proposed approach based on given training data.

The average evaluation results are shown in Table 4. Overall, an average accuracy of 81% was achieved along with an average precision of 77% and an average recall of 89%. Given these results we conclude that the proposed approach is feasible to identify the injected anomalies in the analyzed process execution data. Moreover, it becomes visible that the detection of diverting anomalous behavior becomes harder the more diverse and complex the benign behavior is (e.g., because of noise or ad hoc changes). Accordingly the anomaly detection performance of the BPIC 2015 logs are lower than the results for the HEP logs. Nevertheless, an average anomaly detection accuracy of 75% was achieved for the more challenging BPIC 2015 process execution log data.

Comparison with existing anomaly detection approaches

We have roughly compared the proposed approach against five alternative anomaly detection approaches, cf. Table 5. From those, [7,31] were specifically tailored for detecting anomalies in business process executions. In comparison, [35–37] apply generic anomaly detection techniques, such as, clustering. The tests were executed based on the BPIC 2015 logs as such are more challenging than the HEP logs and thus enable a more conclusive evaluation. Here, the Area Under the Curve (AUC) metric is applied to compare the listed anomaly detection techniques as this metric was found to be commonly used by comparison approaches, such as, [31,38].

In comparison to alternative approaches, such as, [7] it was found that the proposed approach achieves a lower anomaly detection performance, cf. Table 5. However, we were left with the impression that, with regards to quality attributes, such as, understandability, computational performance requirements or visualization capabilities, a significant improvement was achieved. For example, [7] utilizes large monolithic signatures which need to be completely recreated from scratch if, e.g., the process model which instantiates the to be analyzed traces is changed. In comparison, the proposed approach was found to generate the signatures faster and can also reuse some of the generated rules if the underlying process model changes. Overall, this gives the experts the flexibility to choose the correct tool for their needs as, for now, there seems to be a trade off between detection performance and simplicity/understandable of the results and signatures.

5. Evaluating the proposed root Cause analysis visualization approach (A_Viz)

This part of the evaluation has the objective to investigate the effect of **A_Viz** on the analysis of anomalous process executions detected by the presented anomaly detection approach. For this an expected process execution anomaly analysis workflow is replicated. Within the conducted experiment, the participants are, hence, provided with (a) multiple anomalous execution traces, (b) a related process documentation, while (c) given the task to identify and report as many distinct anomalies as possible. The focus of this evaluation is on correctness (how many anomalies were correctly identified and reported) and response time (how long did it take the participants to complete the given tasks). Both variables are frequently utilized to construct understandability, i.e., enabling to deduce if the proposed visualization supports the identification of anomalies, cf. [39].

Context The experiments and data collection were conducted throughout calendar week three in 2019 with 25 participants. From these 25, six participants were classified as experts as they can show of more than five years of practical (or research) experience in the field of process management and analysis — either as a researcher or process analyst. The remaining participants were students who enrolled in the course "Workflow Technology" (WT) or "Business Intelligence" (BUS) (optional part of the Master in the Computer Science curricula) at the University of Vienna throughout the winter term 2018/2019; both courses put a distinct focus on business process design and analysis. It will, in the following, be differentiated between both participant groups of "non-experts", i.e., students, and "experts" in order to determine if the use of **A_Viz** affects both groups differently. Note, according to [40], conducting experiments with students "*is not a major issue as long as you are interested in evaluating the use of*

Table 4
Anomaly detection performance of the presented approach.

Log	HEP_1	HEP_2	HEP_3	BPIC5_1	BPIC5_2	BPIC5_3	BPIC5_4	BPIC5_5
Precision	0.86	0.87	0.85	0.73	0.70	0.78	0.69	0.69
Recall	0.98	0.98	0.97	0.90	0.85	0.75	0.87	0.84
Accuracy	0.91	0.91	0.90	0.78	0.74	0.77	0.73	0.73
$F_{0.5}$-measure	0.88	0.88	0.87	0.76	0.73	0.77	0.72	0.72
F_1-measure	0.92	0.92	0.91	0.80	0.77	0.77	0.77	0.76

Table 5
Anomaly detection performance comparison.

Detection technique	ADAR	Dynamic BN	Likelihood	Feature bagging	LOF	SOD
Reference	This paper	[31]	[7]	[37]	[35]	[36]
AUC	0.76	0.84	0.77	0.56	0.55	0.6

a technique by novice or non-expert ... engineers. Students are the next generation ... professionals and, so, are relatively close to the population of interest." Further studies even argue that students can serve as sufficient representatives for experts, cf. [41–43].

Material, Tasks, and Design Each task of the experiment consists of two steps (*a*) reading and understanding a given process documentation, and (*b*) analyzing three related process execution traces to spot and report anomalies by comparing (*a*) and (*b*). For (*a*) two types of documentation are utilized:

Optimal Documentation is up to date, contains a BPMN based visualization of the process model along with a textual documentation of that model which, inter alia, describes temporal and resource relations, such as, BoD and SoD constraints. This documentation type could represent, for example, the early days of a business processes when its design/implementation was just completed and all documentation is still up to date and available.

Typical Documentation: Here it is assumed that the original documentation was lost or is outdated. Hence, the last resort for an anomaly analyst is to utilize historic process executions (in the form of execution traces), which are deemed as benign, to deduce expected benign behavior from them. This documentation type represents a common scenario where business processes have been in use for some time and have been affected by undocumented changes such that related documentation is missing, no longer applicable or even misleading. This documentation type is, based on our experience, common and representative for aged business processes.

The group of participants was split into subgroups of three. The members of the same group were always uniformly provided with the same documentation type (i.e., optimal or typical). It was *randomly and evenly* determined which group utilizes which kind of documentation throughout the experiment. The latter enabled to achieve an almost perfectly balanced ratio between the optimal (13 participants) and typical (12 participants) documentation.

In addition to the documentation the experiment requires that given process execution traces must be analyzed for anomalies, i.e., (*b*). Traces were visualized in three different ways:

A_Viz is a visualization using colors and rules. See Section 3.3 for an outline and an example depicted in Fig. 4.

Colors represents a simplification of **A_Viz** which mimics common visualization styles applied by existing conformance checking approaches, cf. [27]. Colors are utilized to visualize that an execution event is either most likely related to an anomaly (red) or not (i.e., benign behavior ↦ green), see Fig. 5 for an example application of this visualization. In comparison to the **A_Viz** visualization details on the violated rules are *not* provided.

No support represents the status quo in existing work, cf. [3]. Here, a process analyst only receives the information that a given execution trace contains one or more anomalous execution events, while additional details about which events are likely affected/related by/to anomalies are *not* provided. Fig. 6 gives an example for this. Note, this trace visualization uses a gray background color to enable the differentiation between this visualization style and the other two alternatives presented here.

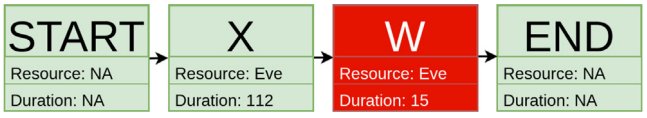

Fig. 5. Anomaly visualization using **colors** (example). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

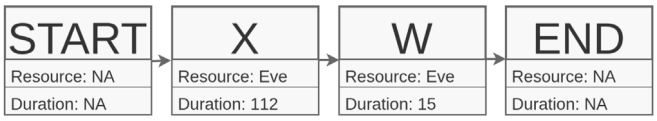

Fig. 6. Example without visualization support.

Task Generation Procedure For each of the two documentation types three matching execution traces were created (i.e., six in total) and mutated to contain randomly chosen anomalies, on randomly chosen events, with random strengths (at least/most 2/4 events were mutated for each trace) — comparably to the mutation approach applied in Section 4. Each of the mutated traces was visualized in all three style (i.e., 18 different visualizations were generated in total). For each participant the order of traces was randomized along with the visualization styles. While doing so it was ensured that the traces match the documentation handed out to the group the respective participant belongs to. Note, each of the three traces was visualized in a randomly chosen style while ensuring that each participant deals with each visualization style exactly once (i.e., each participant had to analyze three traces, each trace was visualized in a unique visualization style).

By following this computer aided and randomized design we aimed at *avoiding learning effects* and *selection bias*. Further, the tasks, traces, processes, and visualizations given throughout the experiment are not taken from the material of the two related courses. All materials handed out to the students are publicly available at https://github.com/KristofGit/A_Viz to support a replication of the study. This includes a *preparatory document* which provides basic information on BPMN and the way the execution traces and its events are visualized throughout the experiment. In addition to the two documentation types, all anomalous traces in the three different visualizations, and the utilized master data sheet are included. The latter is utilized to collect data about previous experience in related areas (such as process research) from the participants (at the start of the experiment). After the experiment that sheet was utilized again to document and obtain a general impression concerning the three different visualization styles. For this questions were asked, such as, which of the three visualization styles resulted in the highest confidence that no anomaly was overlooked or that the reported anomalies were, in fact, true positives.

Pre Test Four weeks before the experiments, pre-tests were conducted with four experts to verify if the created documentation was clear and understandable, the technical infrastructure was capable, the tasks were doable in time, and all necessary information was provided before and during the experiment. With regards to the latter aspects it was planned that a single group does not require more than 60 min to complete the given tasks

Table 6
Anomaly report correctness. Optimal documentation (strict).

	A_Viz	Colors	No support
Temporal	1.00	0.91	0.83
SoD	1.00	1.00	0.83
BoD	1.00	0.83	1.00
Missing Act.	1.00	0.80	0.83
Swapped Act.	1.00	1.00	0.83
New Act.	1.00	0.83	1.00

Table 7
Anomaly report correctness, typical documentation (strict).

	A_Viz	Colors	No support
Temporal	1.00	0.75	0.60
SoD	1.00	0.50	0.50
BoD	0.88	0.60	0.50
Missing Act.	1.00	0.72	0.81
Swapped Act.	NA	NA	NA
New Act.	1.00	0.83	1.00

Table 8
Anomaly report correctness, optimal documentation (relaxed).

	A_Viz	Colors	No support
Temporal	1.00	1.00	0.83
SoD	1.00	1.00	1.00
BoD	1.00	1.00	1.00
Missing Act.	1.00	0.80	1.00
Swapped Act.	1.00	1.00	0.83
New Act.	1.00	1.00	1.00

Table 9
Anomaly report correctness, optimal documentation (relaxed).

	A_Viz	Colors	No support
Temporal	1.00	0.75	0.60
SoD	1.00	1.00	1.00
BoD	0.88	0.60	1.00
Missing Act.	1.00	0.72	0.81
Swapped Act.	NA	NA	NA
New Act.	1.00	0.83	1.00

(i.e., to read the provided documentation and spot anomalies in three different execution traces). Those 60 min comprise of a 15 min preparation session which reiterates the objective of the experiment, answers questions with regards to the different visualization types and the provided process documentation. All participants were able to complete the experiment within the 60 min period (during the pre test and the real experiment).

Based on the feedback gathered throughout the pre tests some minor changes were performed, ranging from fixing typos to slight wording adaptions to prevent potential misunderstandings. Overall, the feedback was positive and no critical issues were observed.

Experiment Execution Two weeks before the experiment five minute talks about its goals and motivations were given throughout one session of both courses (i.e., WT and BUS). Those talks motivated the relation of the experiment to the respective course and outlined the respective benefits each student could gain when participating (one/two bonus points could be earned by participating in WT/BUS).

Before these talks preparatory material was made available together with a registration form at CEWebS[5] to enable potential participants to sign up for their preferred group/timeslot. CEWebS is a custom e-learning platform which is utilized by a number of computer science courses at the University of Vienna (including WT and BUS). The preparatory material and all other materials contain informal natural language descriptions of approaches and tasks along with practical examples. This enabled us to present the material equally to each participant in an approachable manner, as suggested in [44–46]. Note, while performing the experiment all participants were always able to access all the material at once.

Before the beginning of each group timeslot each attending participant was seated randomly and the experiment materials were handed out as printed documents. The latter used a combination of A4 and A3 pages whenever appropriate to increase content readability and clarity (e.g., the process model documentation was always printed on A3 pages so that no page flips are necessary by the participants). Next, the participants were brought up to speed about the tasks, visualization styles used, and the general procedures to follow. This included the website which was utilized by each participant to report identified anomalies.

Data Set Collection Throughout the experiment each participant described/reported identified anomalies by using a custom made website. On this website it was possible to select (a) the event which was the source of an anomaly (or that an expected event is missing), and (b) the anomaly type (i.e., missing activity, swapped activity, novel activity, SoD or BoD violation or temporal violation). Each anomaly could be reported individually until the participant decided that all anomalies for the current trace/task were identified and switched over to the next task/trace. Utilizing an electronic data collection (i.e., anomaly reporting) system enabled us to automatically track each participants' actions (e.g., when he or she started with reporting anomalies and when this process was completed for a given task/trace). Further, it prevents the occurrence of equivocal markings, timestamp documentations or that participants forget to annotate all necessary details — as it was observed, e.g., in [47].

During their work each participant utilized a laptop computer provided by the researchers which had all the necessary tools (such as, the website utilized to report anomalies) already preinstalled. From these machines no network connection was possible and the participants were arranged in a way that limit opportunities for cheating — which was further hampered by the random trace/task/visualization style order each participant was assigned with. Note, the publicly available material collection also contains a copy of the beforehand mentioned electronic data collection (i.e., anomaly reporting) solution (website, resp.). Further, the raw data collected throughout the experiment is available at https://github.com/KristofGit/A_Viz.

Analysis: 25 participants took part in the experiment. Each participant was assigned with a unique combination of documentation type, trace order, and trace visualization style. Tables 6 and 7 give the results when applying a strict analysis of the anomaly reports. Here *strict* means that an anomaly is only assumed as correctly reported if the anomaly type *and* the culprit event were correctly identified. In comparison, Tables 8 and 9 give the results when applying a relaxed analysis, i.e., an anomaly is also counted as detected if the correct anomaly type was chosen, but the culprit event was wrong. Note that the results collected for the typical documentation do not contain any details on anomalies caused by swapped activity as this mutation operator was never applied on the related traces due to the random nature of the mutation selection and application.

In all scenarios, **A_Viz** (average correctness 0.98) outperforms visualization approaches **Colors** (average correctness 0.82) and **No support** (average correctness 0.79). While the longest experiment run took 55 min from start to completion the shortest took 35 min. Overall, it was found that experts were, on average, able to complete the given tasks faster than non-experts, mainly, because they extracted the necessary details from the given documentations faster. An overview on the anomaly reporting times

5 https://cewebs.cs.univie.ac.at/

Table 10
Average anomaly report duration, optimal documentation.

	A_Viz	Colors	No support
Temporal [sec]	55	150	63
SoD [sec]	132	106	35
BoD [sec]	49	144	50
Missing Act. [sec]	145	50	97
Swapped Act. [sec]	235	107	65
New Act. [sec]	105	44	50

Table 11
Average anomaly report duration, typical documentation.

	A_Viz	Colors	No support
Temporal [sec]	69	131	43
SoD [sec]	82	57	39
BoD [sec]	119	99	138
Missing Act. [sec]	106	190	138
Swapped Act. [sec]	NA	NA	NA
New Act. [sec]	38	311	91

is given in Tables 10 and 11. At first, it might seem that the participants were significantly faster when reporting anomalies when not being supported by the visualization. However, this was likely not the case as it was observed that the "No support" visualization was frequently handled in a different way than, e.g., **A_Viz**, i.e., the participants first extracted all the anomalies in advance and subsequently entered them into the report website in bulk which distorts the collected durations/timestamps. Hence, the related data on temporal behavior must be taken with caution.

Another difference between experts and non-experts could be spotted in the result quality: experts made almost no errors throughout the experiment. The few errors made by experts occurred at tasks which were not supported by **A_Viz**. Unfortunately, only six experts participated in the experiment so that a clear differentiation between the compared scenarios is hardly possible.

When comparing relaxed and strict results it seems that almost all anomalies were identified by the participants when applying a relaxed analysis, cf. Tables 8 and 9. However, it was found that the participants frequently started to spot anomalies "everywhere" when being unsure if some behavior is anomalous or not. This effect was especially visible in the data collected for the **No support** visualization and seems to be less significant the more support the visualization provides — which could be an indicator that the proposed visualizations increase understandability. One interpretation is that providing no root cause analysis support could motivate analysts to become overly suspicious which could potentially increase the likelihood to choose incorrect anomaly countermeasures.

Discussion & Impact When preparing the visualization we assumed that the more support and information is provided by a visualization the easier, precise, and faster the identification of individual anomalies, hidden in a given execution trace, becomes. However, this assumption was only partly confirmed by the results and the oral feedback provided by the participants.

In detail: for "simple" control flow related tasks such as reporting novel or missing activities the **No support** visualization outperformed the **Color** visualization. We refer to such tasks as being simple as, e.g, comparing the trace with a given process model is sufficient to solve them. In comparison, resource or temporal behavior violations require to correlate multiple events at once. Based on our observations a likely reason for this is that without any support the participants (a) invested more effort and checked for anomalies more thoroughly, (b) marked observations and anomalies on the provided handouts, and (c) checked and verified more potential cases of anomalous behavior before starting to enter any information into the anomaly report website.

These observations were also confirmed by oral feedback provided by the participants after completing the experiment.

Nevertheless, it was found that **A_Viz** outperformed both comparison approaches. This is, most likely, as it explicitly points out anomalies which, as we assume, are relatively hard to spot (i.e., anomalies related to resources and temporal behavior). The personal preferences expressed by each participant on the master data sheet support this impression, i.e., **A_Viz** dominates the categories "highest confidence" and "personal preference". However, given the limited amount of participants and scenarios further studies must be performed to verify and expand these observations.

When expanding the outlined idea of simple and more challenging tasks it could be observed that the application of supportive visualizations, i.e., **Colors** and **A_Viz**, seems the more beneficial the more complex a task becomes. For example, we assume that the identification of anomalies is simpler when analyzing traces related to the "optimal documentation" than when analyzing traces related to the "typical documentation". This assumption is motivated by the fact that the optimal documentation already describes, e.g., resource constraints in an informal manner while the typical documentation requires to deduce them from a set of execution traces. The results in Tables 6 and 7 seem to support this assumption. Hence, we found that **A_Viz** seems to be especially helpful if the process documentation is outdated or missing — which we assume to be a realistic and common situation for typical anomaly detection scenarios. Nevertheless, additional studies are necessary to verify if this observation holds when given, e.g., more complex process executions and process models as the models utilized here only contained 7 activities each and a small amount of basic gateways (parallel and XOR).

Limitations and Extensions The conducted **A_Viz** evaluation followed the assumption that organizations have to cope with a wide range of domains, process models, and anomaly reports simultaneously. Accordingly, the respective expert for each domain, process model, and execution – which was identified as being anomalous – is potentially, not available all the time. Hence, based on our observations, generic Business Process Management (BPM) experts are frequently forced into the role of "*limited*" domain experts and business analysts. For example, by reading up on the process model documentation before deciding on and conforming reported anomalies (as simulated throughout this experiment).

This also motivated the applied visualization styles (e.g., languages and symbols), which we found to be well known to the targeted BPM user group. For this a minimalistic visualization was implemented that represents the *cause* (the IF part of each rule) and *effect* (the THEN part of each rule) of each reported anomaly in a condensed form. This enables to represent all the relevant information in close proximity to each other to support deductions on an execution's anomaly state (e.g., to confirm an anomaly and terminate the related execution).

Accordingly, while the proposed visualization is well suited for BPM experts, less specialized stakeholders, such as, managers will likely struggle with the chosen representation without additional training. Further, preparing fundamental security related changes based on identified anomalies and root causes requires to incorporate analysts which have a *deep* understanding of the related domains (i.e., domain experts). This scenario was not evaluated here, but is seen as a promising extension for future work. Hence, we see **A_Viz** mainly as a first step towards creating awareness for supporting root cause analysis in the business process anomaly detection domain.

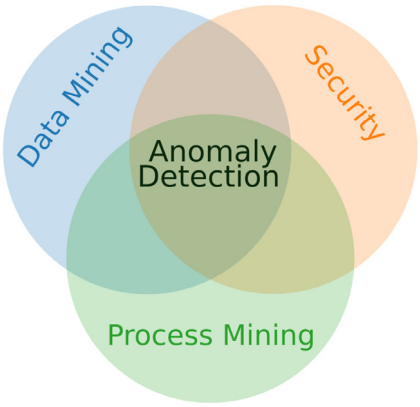

Fig. 7. Related areas.

6. Related work

(Business Process) Anomaly detection employs techniques from areas, such as, data mining and machine learning and is relevant for a number of application domains, e.g., intrusion and fraud detection [18]. In particular, anomaly detection in the business process domain combines two viewpoints: Firstly, a method side which applies process-oriented techniques, such as, conformance checking along with techniques from mining cross-sectional and temporal data and, secondly, the application side, i.e., the detection of deviations in process execution behavior. Hereby, it is related and influenced by a range domains and concepts stemming from data and process mining but also security in general, see Fig. 7. This is because detecting anomalies is often connected with security issues [4], for example, the detection of fraud and misuse. In the following, we discuss related approaches from the security domain and from the process domain as well as approaches on root cause analysis. The discussions also address the method point of view.

6.1. Anomaly detection in the security domain

In the security domain, anomaly detection and root cause analysis are major research areas. However, existing approaches are too specialized to be applied to process data, cf. [19,48], because they focus on single unique use cases and data formats, such as, specific network protocols, e.g., the Session Initiation Protocol, cf. [49]. These approaches can hardly be generalized and applied to process execution logs which hold different data, formats, and contextual attributes. In comparison, generic and more flexible anomaly detection approaches from the data analysis domain frequently show a sub par performance when being applied on process executions, cf. Table 5 and [31].

One could argue that instead of applying anomaly detection the process definition could be secured by applying security focused modeling notations, cf. [23]. In real world scenarios, this would require to be aware of all potential sources for security incidents during the design phase and to constantly update the processes to meet novel security challenges. In comparison the proposed anomaly detection approach is self learning and can also deal with process changes automatically.

This flexibility also differentiates the proposed approach from existing work, e.g., [2], which was found to frequently apply either overly strict signatures [3] (potentially resulting in false positives) or *soft matchers*, cf. [19]. Soft matchers widen the area of behavior which a signature matches to. Hereby, the risk of false positives can be reduced. However, as soft matchers require domain and expert knowledge and are manually defined we assume that they are hardly applicable given the huge amount of processes which are currently in use. We assume that the proposed automatic approaches, e.g., to dynamically choose a comparison trace based on its similarity or to factor in rule significance, are necessary.

6.2. Detecting anomalies in process behavior

This section provides an overview on anomaly detection approaches in process behavior based on the systematic literature study in [3]. One option to categorize the approaches is based on the applied method, e.g., by distinguishing supervised learning and unsupervised learning methods.

For supervised learning classification is applied as follows: the process execution data, i.e., the logs, are considered as labeled training data that does not contain any anomalies. From this, a classifier is learned and new process instance data can be classified into non-anomalous or anomalous behavior. Based on the technique, the classifier varies.

For *process mining* techniques, the classifier is typically a reference process model that is discovered from the training data, cf. [50–54]. By contrast, [55] mines an automaton and is specifically tailored to deal with infrequent behavior (which would, elsewise, be interpreted as anomalous). [56] advocates an anomaly score for sliding windows on the log data.

In comparison, *conformance checking* and *filtering* techniques, cf. [57–59], show some resemblance, e.g., to [50], as they are also capable of detecting deviations between process models and process executions, cf. [60,61]. However, we were left with the impression that, while conformance checking and filtering show potential, related techniques are not perfectly up to the task. This is, because related conformance checking and filtering work, for example, was found to be “overly” strict which can increase false positive rates — a drawback which *similar* process mining based approaches, which were mentioned beforehand, such as [55], mitigate. For example, by precisely relaxing mined process models and related anomaly detection thresholds.

For *rule mining* techniques – such as, the work at hand – rules are derived in order to serve as a classifier. An approach based on Support Vector Machines (SVM) is presented in [62]. The most comparable work to the approach at hand is [63]. It applies association rules for anomaly detection in processes. However, rules are largely manually generated (e.g., a user define the expected maximum activity duration) and order dependencies between activities are not verified.

In [19], we present an approach to detect temporal anomalies for multiple instances that is based on *time series* mining.

Unsupervised approaches are *clustering-based* and aim at determining clusters of non-anomalous and anomalous behavior where the latter is most likely represented by small and scattered clusters. The clusters are based on similarity between process executions, e.g., based on the resources [64], the temporal perspective [65], and the control flow [66]. Other control flow oriented approaches include [67,68]. In [7], clustering is combined with likelihood graphs.

Further approaches employ *neural networks*, cf. [69], signatures based on regular expressions [70], and statistical anomaly detection [71].

Currently, several shortcomings in existing work hinder the application of anomaly detection in real world applications. At first, a significant amount of process anomaly detection approaches support only single process perspectives. Secondly, existing work does hardly support the analysis of identified anomalies and mostly applies monolithic signatures which are hard to grasp, struggle with noise and ad hoc changes, but also cannot be partially updated whenever the underlying process changes.

6.3. Compliance checking and root cause analysis

Compliance checking approaches such as [9,27] also utilize rule-based definitions of expected process behavior. The goal is to analyze the process definition and execution for compliance violations and their root causes, cf. [9,72]. However, such work typically does not take noise and ad hoc changes into account, possibly resulting in false positives. Moreover, rule formalisms such as LTL enable expressive rules, but at the price of increased complexity. This work, by contrast, aims at simplicity, balanced with flexibility. Moreover, the definition of the compliance rules often requires in depth domain and process knowledge and hence results in manual effort. The work at hand, by contrast, offers automatic mining of association rules.

As said before a crucial step in compliance management is the analysis and explanation of root causes. The process of root cause analysis comprises the following steps according to [12]: at first, the problem should be understood. Here tools such as flow charts are proposed. In the approach at hand, accordingly, process models can mined from the log. Secondly, a problem cause brainstorming should take place. This step can be added to the approach if desired. Thirdly, the problem cause data is collected and analyzed. These steps are achieved by the association rule mining approach. Subsequently, the root cause is to be identified. Here it is suggested to use cause-and-effect charts as they provide *"an easily applied tool used to analyze possible causes of a problem"* [12]. Accordingly, this work uses cause-and-effect charts based on Ishikawa ("fishbone") diagrams. The final steps of problem elimination and solution implementation are out of scope of this work, but constitute promising research directions for future work.

Alternatives for anomaly and root cause visualization are provided in literature. In [73], a visualization for anomalies in business processes is provided. For this, impact factors can be specified and related to a fraud amount. This approach can be used to further investigate factors behind the deviations, but does not analyze the process deviations directly. An approach to discover deviations between process graphs and their instance traffic is offered in [74]. Deviations can be spotted, but the root cause is neither visualized nor explained. A survey on visual analytics options based on current process mining frameworks such as ProM.[6] is provided in [75] For spotting anomalies in process behavior the described visualizations using dotted charts seems particularly useful, however, it does not shed light on their root causes.

Further related work can be found in the *predictive monitoring* area, cf. [76,77]. The latter, strives to predict the outcome and execution behavior of business processes and business process changes. While such predictions provide valuable insights for business process management, they are only of limited used for anomaly root cause analysis. This is because such work typically focuses, e.g., on predicting execution costs or durations but not on describing the cause and effects of anomalous behavior.

7. Discussion and outlook

The paper set out three research questions, i.e., how to detect anomalies by reducing false positives (**RQ1**), how to balance effort and flexibility for anomaly detection (**RQ2**), and how to explain the root cause for anomalies to users (**RQ3**). We conclude that the proposed approach is able to detect anomalies (↦**RQ1**) as the conducted evaluation showed an average anomaly detection recall of 89%. This goes hand in hand with a substantial simplification of the generated signatures compared to previous work in [7] (complex likelihood graphs vs. short rules) which in principle fosters the understandability of the signatures and identified anomalies (↦**RQ2**).

Both research questions **RQ1** and **RQ2** require an adequate treatment of flexibility by ad hoc process changes and noise as both might lead to false positives or in other words an insufficient distinction between benign and malign behavior.

The proposed approach applies the common assumption that benign behavior is *more likely* than anomalous behavior, cf. [3]. Nevertheless benign *noise* and *ad hoc changes* still occur in the signature mining data (i.e., L) but also in the traces that are analyzed for anomalies, cf. [3]. These kinds of behavior can, if the applied signature is too strict or overfitting, be misinterpreted as being anomalous and so result in *false positives*. Hence, the proposed approach applies three strategies to mitigate this risk:

Similarity: given traces are not compared with strict fixed signatures or thresholds. Instead the signature and the expected behavior is individually and automatically adapted for the trace that is analyzed by dynamically selecting a similar trace in L which is utilized as a source for comparable behavior.

Rule significance: the significance and impact of each rule is dynamically calculated during the anomaly detection phase. For this the rule support given by L (i.e., the percentage of traces in L that the rule supports) is utilized. Hence, each rule gets automatically assigned an individual significance.

Signature strictness: it is not necessary that a trace matches all rules a signature R is composed of. Instead the applied approach aggregates the support of each rule such that a trace can "compensate" an unsupported rule by supporting other rules. Such a relaxed approach, in comparison to stricter existing work, cf., [50], provides a basis to deal with noise and ad hoc changes, cf. [19].

We conclude that the approach takes a next step towards the reduction of false positives by being aware of process change and noise.

As the identified anomalies (and related traces) can be complex and hard to understand we argue that anomaly detection approaches should support experts when analyzing anomalies along with the related alarms. We assume that it is necessary to identify but also to understand anomalies to choose appropriate anomaly countermeasures. For this, inter alia, the following information is required: (a) which part of an execution trace is affected by an anomaly; and (b) the anomaly severity, cf. [5]. It is shown how this information is provided by the proposed rule based anomaly detection approach (↦ **RQ3**). Moreover, the proposed root cause visualization can foster the understanding and identification of root causes of anomalies. To our knowledge this is the first *process anomaly detection* approach that does so.

In future work, the performance of the proposed approach will be further optimized. This is because currently each of the three rule types is mined independently of each other. However, we found that for example the control flow rules and the temporal rules partly have connected conditions (i.e., the respective activities they apply to). We assume that such similarities can be exploited. Moreover, we will analyze if the proposed approach can be applied to clean up execution logs. Hereby, if logs that are riddled with anomalous traces they could be "cleaned" by the proposed unsupervised approach so that they can be successfully utilized by existing semi- or even supervised anomaly detection approaches, cf. [3]. Finally, we will extend the conducted **A_Viz** evaluation by incorporating domain experts and more complex execution scenarios.

[6] http://www.processmining.org/prom/start

Acknowledgement

This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Gartner, Gartner says detection and response is top security priority for organizations in 2017, 2017, Newsroom Press Release. https://gtnr.it/2HsXdOG.
[2] F. Bezerra, et al., Anomaly detection using process mining, in: Enterprise, Business-Process and Information Systems Modeling, Vol. 29, Springer, 2009, pp. 149–161.
[3] K. Böhmer, S. Rinderle-Ma, Anomaly detection in business process runtime behavior–challenges and limitations, 2017, arXiv arXiv:1705.06659.
[4] M. Leitner, S. Rinderle-Ma, A systematic review on security in Process-Aware Information Systems - Constitution, challenges, and future directions, Inf. Softw. Technol. 56 (3) (2014) 273–293.
[5] K. Julisch, Clustering intrusion detection alarms to support root cause analysis, Inf. Syst. Secur. 6 (4) (2003) 443–471.
[6] M. Reichert, B. Weber, Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies, Springer, 2012.
[7] K. Böhmer, et al., Multi-perspective anomaly detection in business process execution events, in: Cooperative Information Systems, Springer, 2016, pp. 80–98.
[8] C. Czepa, H. Tran, U. Zdun, T.T.T. Kim, E. Weiss, C. Ruhsam, Plausibility checking of formal business process specifications in linear temporal logic, in: CAiSE Forum, 2016, pp. 1–8.
[9] L.T. Ly, F.M. Maggi, M. Montali, S. Rinderle-Ma, W.M. van der Aalst, Compliance monitoring in business processes: Functionalities, application, and tool-support, Inf. Syst. 54 (2015) 209–234.
[10] R. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer, 2014.
[11] K. Böhmer, S. Rinderle-Ma, Association rules for anomaly detection and root cause analysis in process executions, in: Advanced Information Systems Engineering, 2018, pp. 3–18.
[12] B. Andersen, T. Fagerhaug, Root Cause Analysis: Simplified Tools and Techniques, ASQ Quality Press, 2006.
[13] M. Ben-Daya, Failure mode and effect analysis, in: Handbook of Maintenance Management and Engineering, Springer, 2009, pp. 75–90.
[14] G. Greco, A. Guzzo, L. Pontieri, Mining taxonomies of process models, Data Knowl. Eng. 67 (1) (2008) 74–102.
[15] R. Agrawal, R. Srikant, et al., Fast algorithms for mining association rules, in: Very Large Data Bases, Vol. 1215, 1994, pp. 487–499.
[16] R. Srikant, R. Agrawal, Mining sequential patterns: Generalizations and performance improvements, in: International Conference on Extending Database Technology, Springer, 1996, pp. 1–17.
[17] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: The prefixspan approach, IEEE Trans. Knowl. Data Eng. 16 (11) (2004) 1424–1440.
[18] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, Comput. Surv. 41 (3) (2009) 15.
[19] K. Böhmer, S. Rinderle-Ma, Multi instance anomaly detection in business process executions, in: Business Process Management, Springer, 2017, pp. 77–93.
[20] G. Zimbrão, J.M. de Souza, V.T. de Almeida, W.A. da Silva, An algorithm to discover calendar-based temporal association rules with item's lifespan restriction, in: The Second Workshop on Temporal Data Mining, 2002.
[21] M.H. Namaki, Y. Wu, Q. Song, P. Lin, T. Ge, Discovering graph temporal association rules, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, ACM, 2017, pp. 1697–1706.
[22] C.-H. Chen, G.-C. Lan, T.-P. Hong, S.-B. Lin, Mining fuzzy temporal association rules by item lifespans, Appl. Soft Comput. 41 (2016) 265–274.
[23] A.D. Brucker, I. Hang, G. Lückemeyer, R. Ruparel, SecureBPMN: Modeling and enforcing access control requirements in business processes, in: Access Control Models and Technologies, ACM, 2012, pp. 123–126.
[24] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining. 1st, Pearson Addison Wesley, 2005.
[25] K. Böhmer, S. Rinderle-Ma, Probability based heuristic for predictive business process monitoring, in: OTM Confederated International Conferences" on the Move to Meaningful Internet Systems", Springer, 2018, pp. 78–96.
[26] M. Heravizadeh, J. Mendling, M. Rosemann, Dimensions of business processes quality (QoBP), in: Business Process Management, Springer, 2008, pp. 80–91.
[27] L.T. Ly, S. Rinderle-Ma, D. Knuplesch, P. Dadam, Monitoring business process compliance using compliance rule graphs, in: On the Move to Meaningful Internet Systems, Springer, 2011, pp. 82–99.
[28] I. Kaoru, What is Total Quality Control: The Japanese Way, Prentice Hall, 1985.
[29] D. Knuplesch, M. Reichert, A. Kumar, A framework for visually monitoring business process compliance, Inf. Syst. 64 (2017) 381–409.
[30] T. Vogelgesang, G. Kaes, S. Rinderle-Ma, H. Appelrath, Multidimensional process mining: Questions, requirements, and limitations, in: CAISE Forum, Springer, 2016, pp. 169–176.
[31] S. Pauwels, T. Calders, An anomaly detection technique for business processes based on extended dynamic Bayesian networks, in: ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 494–501.
[32] I. Teinemaa, A. Leontjeva, K.-O. Masing, BPIC 2015: Diagnostics of Building Permit Application Process in Dutch Municipalities, BPI Challenge Report 72, 2015.
[33] N. Chinchor, B. Sundheim, MUC-5 evaluation metrics, in: Message Understanding, in: Computational Linguistics, 1993, pp. 69–78.
[34] W.-K. Wong, A. Moore, G. Cooper, M. Wagner, Rule-based anomaly pattern detection for detecting disease outbreaks, in: AAAI/IAAI, 2002, pp. 217–223.
[35] M.M. Breunig, H.-P. Kriegel, R.T. Ng, J. Sander, LOF: identifying density-based local outliers, in: ACM SIGMOD Record, Vol. 29, No. 2, ACM, 2000, pp. 93–104.
[36] H.-P. Kriegel, P. Kröger, E. Schubert, A. Zimek, Outlier detection in axis-parallel subspaces of high dimensional data, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2009, pp. 831–838.
[37] A. Lazarevic, V. Kumar, Feature bagging for outlier detection, in: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, ACM, 2005, pp. 157–166.
[38] A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, Pattern Recognit. 30 (7) (1997) 1145–1159.
[39] B. Hoisl, S. Sobernig, M. Strembeck, Comparing three notations for defining scenario-based model tests: A controlled experiment, in: Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the, IEEE, 2014, pp. 180–189.
[40] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Trans. Softw. Eng. 28 (8) (2002) 721–734.
[41] P. Runeson, Using students as experiment subjects–an analysis on graduate and freshmen student data, in: Empirical Assessment in Software Engineering, 2003, pp. 95–102.
[42] M. Svahnberg, A. Aurum, C. Wohlin, Using students as subjects-an empirical evaluation, in: Empirical Software Engineering and Measurement, ACM, 2008, pp. 288–290.
[43] M. Höst, B. Regnell, C. Wohlin, Using students as subjects – a comparative study of students and professionals in lead-time impact assessment, Empir. Softw. Eng. 5 (3) (2000) 201–214.
[44] M. Knobelsdorf, C. Frede, Analyzing student practices in theory of computation in light of distributed cognition theory, in: Proceedings of the 2016 ACM Conference on International Computing Education Research, ACM, 2016, pp. 73–81.
[45] H. Habiballa, T. Kmet', Theoretical branches in teaching computer science, Internat. J. Math. Ed. Sci. Tech. 35 (6) (2004) 829–841.
[46] F.C. Richardson, R.M. Suinn, The mathematics anxiety rating scale: psychometric data, J. Couns. Psychol. 19 (6) (1972) 551.
[47] C. Czepa, U. Zdun, On the understandability of temporal properties formalized in linear temporal logic, property specification patterns and event processing language, IEEE Trans. Softw. Eng. (2018).
[48] M. Gupta, J. Gao, C.C. Aggarwal, J. Han, Outlier detection for temporal data: A survey, Knowl. Data Eng. 26 (9) (2014) 2250–2267.
[49] K. Rieck, S. Wahl, P. Laskov, P. Domschitz, K.-R. Müller, A self-learning system for detection of anomalous sip messages, in: Services and Security for Next Generation Networks, Springer, 2008, pp. 90–106.
[50] W.M. Van der Aalst, A.K.A. de Medeiros, Process mining and security: Detecting anomalous process executions and checking process conformance, Theoret. Comput. Sci. 121 (2005) 3–21.
[51] R. Accorsi, T. Stocker, On the exploitation of process mining for security audits: the conformance checking case, in: ACM Symposium on Applied Computing, 2012, pp. 1709–1716.
[52] R. Rieke, M. Zhdanova, J. Repp, R. Giot, C. Gaber, Fraud detection in mobile payments utilizing process behavior analysis, in: Availability, Reliability and Security, 2013, pp. 662–669.
[53] F. de Lima Bezerra, J. Wainer, Algorithms for anomaly detection of traces in logs of process aware information systems, Inf. Syst. 38 (1) (2013) 33–44.
[54] S. Mardani, M.K. Akbari, S. Sharifian, Fraud detection in process aware information systems using MapReduce, in: 2014 6th Conference on Information and Knowledge Technology (IKT), IEEE, 2014, pp. 88–91.

[55] R. Conforti, M.L. Rosa, A.H.M. ter Hofstede, Filtering out infrequent behavior from business process event logs, IEEE Trans. Knowl. Data Eng. 29 (2) (2017) 300–314.
[56] N. Gupta, K. Anand, A. Sureka, Pariket: Mining business process logs for root cause analysis of anomalous incidents, in: Databases in Networked Information Systems, 2015, pp. 244–263.
[57] S.J. van Zelst, M.F. Sani, A. Ostovar, R. Conforti, M.L. Rosa, Filtering spurious events from event streams of business processes, in: Advanced Information Systems Engineering - 30th International Conference, CAiSE 2018, Tallinn, Estonia, June 11–15, 2018, Proceedings, 2018, pp. 35–52. http://dx.doi.org/10.1007/978-3-319-91563-0_3.
[58] M.F. Sani, S.J. van Zelst, W.M.P. van der Aalst, Applying sequence mining for outlier detection in process mining, in: On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part II, 2018, pp. 98–116. http://dx.doi.org/10.1007/978-3-030-02671-4_6.
[59] N. Tax, N. Sidorova, W.M.P. van der Aalst, Discovering more precise process models from event logs by filtering out chaotic activities, J. Intell. Inf. Syst. 52 (1) (2019) 107–139, http://dx.doi.org/10.1007/s10844-018-0507-6.
[60] M. Alizadeh, X. Lu, D. Fahland, N. Zannone, W.M.P. van der Aalst, Linking data and process perspectives for conformance analysis, Comput. Secur. 73 (2018) 172–193, http://dx.doi.org/10.1016/j.cose.2017.10.010.
[61] G. Li, W.M.P. van der Aalst, A framework for detecting deviations in complex event logs, Intell. Data Anal. 21 (4) (2017) 759–779, http://dx.doi.org/10.3233/IDA-160044.
[62] F.M. Maggi, A.J. Mooij, W.M.P. van der Aalst, Analyzing vessel behavior using process mining, in: Situation Awareness with Systems of Systems, 2013, pp. 133–148.
[63] R. Sarno, et al., Hybrid association rule learning and process mining for fraud detection, Comput. Sci. 42 (2) (2015).
[64] T. Zhu, Y. Guo, J. Ma, A. Ju, Business process mining based insider threat detection system, in: Advances on P2P, Parallel, Grid, Cloud and Internet Computing, 2016, pp. 467–478.
[65] P. Hsu, Y. Chuang, Y. Lo, S. He, Using contextualized activity-level duration to discover irregular process instances in business operations, Inform. Sci. 391 (2017) 80–98.
[66] F. Folino, G. Greco, A. Guzzo, L. Pontieri, Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction, Data Knowl. Eng. 70 (12) (2011) 1005–1029.
[67] B. Depaire, J. Swinnen, M. Jans, K. Vanhoof, A process deviation analysis framework, in: Business Process Management Workshops, 2012, pp. 701–706.
[68] C. Linn, D. Werth, Sequential anomaly detection techniques in business processes, in: Business Information Systems Workshops, 2016, pp. 196–208.
[69] T. Nolle, S. Luettgen, A. Seeliger, M. Mühlhäuser, Analyzing business process anomalies using autoencoders, Mach. Learn. 107 (11) (2018) 1875–1893.
[70] K. Böhmer, S. Rinderle-Ma, Automatic signature generation for anomaly detection in business process instance data, in: Enterprise, Business-Process and Information Systems Modeling, 2016, pp. 196–211.
[71] A. Rogge-Solti, G. Kasneci, Temporal anomaly detection in business processes, in: Business Process Management, 2014, pp. 234–249.
[72] E. Ramezani, D. Fahland, W.M.P. van der Aalst, Where did I misbehave? Diagnostic information in compliance checking, in: Business Process Management, 2012, pp. 262–278.
[73] M.C. Hao, D.A. Keim, U. Dayal, J. Schneidewind, Business process impact visualization and anomaly detection, Inf. Vis. 5 (1) (2006) 15–27.
[74] S. Kriglstein, G. Wallner, S. Rinderle-Ma, A visualization approach for difference analysis of process models and instance traffic, in: Business Process Management, 2013, pp. 219–226.
[75] S. Kriglstein, M. Pohl, S. Rinderle-Ma, M. Stallinger, Visual analytics in process mining: Classification of process mining techniques, in: EuroVis Workshop on Visual Analytics, 2016, pp. 43–47.
[76] A. Cuzzocrea, F. Folino, M. Guarascio, L. Pontieri, Predictive monitoring of temporally-aggregated performance indicators of business processes against low-level streaming events, Inf. Syst. 81 (2019) 236–266, http://dx.doi.org/10.1016/j.is.2018.02.001.
[77] C. Witt, M. Bux, W. Gusew, U. Leser, Predictive performance modeling for distributed batch processing using black box monitoring and machine learning, Inf. Syst. 82 (2019) 33–52, http://dx.doi.org/10.1016/j.is.2019.01.006.