"The final authenticated version is available online at https://link.springer.com/chapter/10.1007%2F978-3-319-91563-0_1."

Association Rules for Anomaly Detection and Root Cause Analysis in Process Executions

Kristof Böhmer and Stefanie Rinderle-Ma

University of Vienna, Faculty of Computer Science
{kristof.boehmer,stefanie.rinderle-ma}@univie.ac.at

Abstract. Existing business process anomaly detection approaches typically fall short in supporting experts when analyzing identified anomalies. Hereby, false positives and insufficient anomaly countermeasures might impact an organization in a severely negative way. This work tackles this limitation by basing anomaly detection on association rule mining. It will be shown that doing so enables to explain anomalies, support process change and flexible executions, and to facilitate the estimation of anomaly severity. As a consequence, the risk of choosing an inappropriate countermeasure is likely reduced which, for example, helps to avoid the termination of benign process executions due to mistaken anomalies and false positives. The feasibility of the proposed approach is shown based on a publicly available prototypical implementation as well as by analyzing real life logs with injected artificial anomalies.

Keywords: Anomaly detection, process, root cause, rule mining

1 Introduction

Process *anomaly detection* enables to reveal anomalous process execution behavior which can indicate fraud, misuse, or unknown attacks, cf. [3, 4]. 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. [4].

Analyzing anomaly detection alarms and choosing countermeasures is *challenging*, cf. [13]. This applies also to the process domain as processes operate in *flexible open environments*, cf. [4]. Hence, thousands of alarms must be carefully analyzed as they could be false positives that report benign behavior as anomalous [13, 4] (e.g., because of incorrectly interpreted noise or ad hoc changes).

Existing work, cf. [4, 3, 6], reports only if an execution is *anomalous or not*. However, we assume that additional information, e.g., which behaviour 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. [3, 6], to identify anomalies. Such signatures, compress all the expected execution behavior into a complex interconnected structure. Thus, they 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 [4]. This could hinder an organization as benign process executions could be unnecessarily terminated.

To address these limitations this work proposes a novel unsupervised anomaly detection heuristic. Instead of monolithic signatures it applies small sets of independent association rules. Hereby, individual rules can easily be replaced if a process changes. As association rules only represent direct relations between variables (e.g., activity `A` and `C` occur during the same execution) they are easy to understand but lack in expressiveness compared to other formalisms, such as, Liner Temporal Logic (LTL) – which we see as an *advantage*. We assume that 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. [10, 14], hence, simple formalisms (e.g., association rules) foster mining and understandability.

Further, the proposed approach prevents false positives as it supports noise and ad hoc changes. 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. [4]. 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 automatically calculated rule significance of each non-supported rule.

Let P be a process that should be monitored for anomalies and let L hold all execution traces t of P. The key idea is to represent the given behavior in L as a set of association rules R. To analyze if a novel execution trace $t' \notin L$ is anomalous it is determined which rules in R are supported by t' (ex post). 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.

This work follows the design science research methodology, cf. [21]. For this, design requirements were derived from existing work on anomaly detection in the security domain, in general, and the process domain. 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, cf. Section 4. First, its feasibility is shown by performing a cross validation with real life process executions and injected anomalies. Secondly, the findings are compared with related work (cf. Section 5) and the achieved root cause analysis capabilities are discussed, cf. Sections 3 and 6. Stakeholders of the proposed approach are process managers and security experts.

2 Prerequisites and General Approach

This paper proposes an anomaly detection heuristic to classify process execution traces as anomalous or not. For this association rules are mined from a set 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. [11]. The proposed approach is *unsupervised* as the traces in L are not labeled, formally:

Definition 1 (Execution Log). *Let L be a set of* `execution traces` *$t \in L$; $t := \langle e_1, \cdots, e_n \rangle$ 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 $e_i.ec \in \mathbb{R}_{>0}$; t's order is given by $e_i.es$.*

This notion represents information provided by process execution log formats such as, the eXtensible Event Stream, and enables the analysis of the control, resource, and temporal perspective. Accordingly, the first event in the running example, cf. Table 1, is $e_1 = (\texttt{A}, \texttt{Mike}, \texttt{1}, \texttt{5})$. *Auxiliary* functions: $\{\cdots\}^0$ 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 $i \in \mathbb{N}_{>0}$ while $\langle \cdot \rangle_i^+$ retains all list items with an index $> i$.

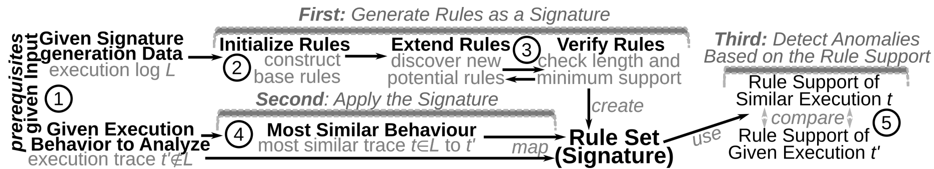

Fig. 1. Proposed rule based process anomaly detection approach – overview

Figure 1 provides an overview of the proposed anomaly detection heuristic; algorithms are presented in Sect. 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`.

Here, rule mining is inspired from the Apriori algorithm [2] which mines value relationships in large unordered datasets 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 association rules are extended into Anomaly Detection Association Rules (ADAR) – (rule for short) – to detect control, resource, and temporal anomalies in process execution traces, formally:

Definition 2 (ADAR). *$r := \langle rp_1, \cdots, rp_m \rangle$ is a* `rule` *with* `conditions` *$rp_j := (ra, rd)$. Let $t \in L$ be a trace. All conditions in r must be matched by t to conclude*

that t supports r, cf. Def. 4. The condition indices j represents their expected order in r, i.e., rp_j must be matched by a trace before rp_{j+1} can be. Each rule condition rp_j represents an expected activity $rp_j.ra$ and an optional execution duration represented as classes, i.e., $rp_j.rd \subseteq \{$low, avg, high$\}$ based on L.

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, 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. [7]. Further, as rules (conditions) are mined based on given traces (events); we define projections for rules (conditions) on their trace (event) so that $rtr(r) := t$ ($re(rp) := e$).

Imagine the 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 in Table 1. 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.

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

The applied rule mining consists of *three* stages. Initially, see ②, the basis for the mining is laid by converting each event e, given by L's traces, into an individual rule. 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. 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 iteratively applied. The rule extension, see ③, extends each rule in R by one additional rule condition in each possible way to identify new potential rules. For example, to extend r_1 all successors of activity A (i.e., the last rule condition, $\mapsto r_1^l$, in r_1, cf. Def. 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, this is defined as:

Definition 3 (Extending individual ADARs). *Let $\mathcal{E}$ be the set of all events in a log L and $\mathcal{RP}$ be the set of all rule conditions. Let $r := \langle rp_1, \cdots, rp_m \rangle$ be*

a rule and $t := \langle e_1, \cdots, e_n \rangle \in L$ *be an execution trace with* $rtr(r) = t$*. Rule extension function* $ext : R \times L \mapsto R$ *extends* r *by:*

$$ext(r,t) := \{r \oplus torp(e, rt(r)) | e \in \{e' \in t | e'.es > e''.es; e'' = re(r^l)\}\} \qquad (1)$$

where $torp : \mathcal{E} \times$ $\{$`control`*,* `temporal`*,* `SOD`*,* `BOD`$\}$ $\mapsto \mathcal{RP}$ *converts an event* e *into a rule condition* rp*, cf. Def. 5; Sect. 3.1 defines how this is performed for each of the four rule types, i.e.,* $rt(r) \in \{$`control`*,* `temporal`*,* `SOD`*,* `BOD`$\}$*.*

Finally, rule verification is applied ④. Each rule is verified by analyzing its support, i.e., $sup(r, L) := |\{t \in L | mp(r, t) = \texttt{true}\}| \, / \, |L|$ (function mp is defined in Def. 4). Here the support of a rule represents the percentage of traces in L a rule could be successfully mapped on (match the rule conditions, resp.). If the support (i.e., the percentage of traces $t \in L$ a rule supports) of a rule is below user configurable $mins \in [0, 1]$ then the rule is removed from R. Subsequently, the rule extension and verification steps are applied repeatedly till 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.

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

The rule r_1', as given previously, achieves a support of 0.5 for the running example, cf. Table 1. This is because it expects 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. Def. 6. 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. 1.

3.1 Anomaly Detection Association Rule Mining

The applied ADAR mining approach, cf. Alg. 1, combines the main mining steps described in Fig. 1. This is the rule set *initialization*, along with the iteratively applied rule *extension* (cf. Def. 3) and *verification* steps (cf. Def. 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).

```
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 := ∅; // initially the rule set (signature, resp.) is empty
  foreach t ∈ L do // initializing the rule set R with base rules
    foreach e ∈ t do
      R := R ∪ {⟨torp(e, ty)⟩} // initial base rule with one condition, Def. 5
  for rcsize := 0 to rl do // generate rules up to a size of rl conditions per rule
    R := {ext(r, rtr(r))|r ∈ R} // extend rules in R, cf. Def. 3 and Def. 2
    foreach r ∈ R do
      if sup(r, L) < mins // verification, calc. rule support, cf. Def. 4 then
        R := R \ {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.

Definition 5 (Transforming events to ADAR conditions). *The auxiliary function $torp(e, ty) : rp$ transforms an* ***event*** *e into a rule* ***condition*** *rp. Depending on the chosen rule type (given by variable ty) one out of the* ***three*** *following rule condition mining approaches is applied. For example, if $ty =$* `control` *then the following control flow rule mining approach is used.*

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. [3], 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, transforming an event e, during rule extension, into a rule condition: $rp = (e.ea, \cdot)$. Hence, when transforming e_1, as given by the running example in Table 1, into a rule condition rp then $rp = (\texttt{A}, \cdot)$.
Control ADAR Support Trace t supports a control flow rule r if t holds all activity executions specified by the rule conditions in $rp \in r$, 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. ① and ② are verified by Alg. 2 to determine if a trace t supports the control flow rule r. Accordingly, a rule $r = \langle rp_1, rp_2 \rangle$ where $rp_1.ra = \texttt{A}$ ($rp_1 = (\texttt{A}, \cdot)$, resp.) and $rp_1.ra = \texttt{B}$ (meaning that activity `A` must be succeeded by `B`) would only be supported by trace t_1 but not by t_2 in the running example, cf. Table 1.

```
Algorithm ControlSupport(trace t, control flow rule r)
  Result: if r is supported by t ↦ true or not ↦ false
  for j = 1 to |r| // |⟨·⟩| retains the length of the list ⟨·⟩ 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 but also it enables to be flexible enough to not 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.

Mining Temporal ADARs Temporal rules focus on activity durations. Those were found to be a significant indicator for fraud and misuse, cf. [8, 5]. 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 propose *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 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. 2. It depicts the three duration classes of PDC and how widening affects them. For example, while the activity duration ① can 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 ② is 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.

```
Algorithm TempClass(event e, log L, duration classes PDC, widen w ∈ [0;1])
    Result: set of representative duration classes DC ⊆ 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' ∈ t, t ∈ L : e'.ea = e.ea}
    min := {d | d ∈ D, ∀d' ∈ D; d ≤ d'}^0; max := {d | d ∈ D, ∀d' ∈ D; d ≥ d'}^0
    part := (max − min)/|PDC|; d := e.ec − e.es; wspan = part · w; i := 0
    foreach pdc ∈ PDC // check for each class in PDC if it is representative do
        start := min − wspan + part · i; end := start + wspan · 2 + part; i := i + 1
        if d ≥ start ∧ d ≤ end then
            DC := DC ∪ {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.

Hence, when converting e_1, see the example in Table 1, into a temporal rule condition rp_1 it would be defined as $rp_1.ra = \texttt{A}$ 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 $rp_2.rd = \{\texttt{high}\}$.

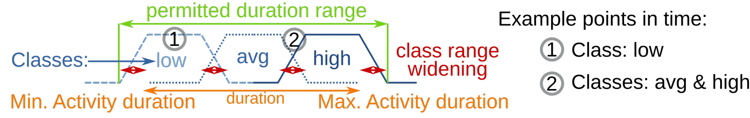

Fig. 2. Duration class representation, motivating example

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 Alg. 2, is extended by calculating and comparing duration classes, cf. Alg. 4.

```
Algorithm TempRuleSupport(trace t, temporal rule r, duration classes PDC, w ∈ [0;1])
    Result: true if r is supported by t; false otherwise
    for j = 1 to |r| // |⟨·⟩| retains the length of the list ⟨·⟩ do
        for i = 1 to |t| do
            if t_i.ea = r_j.ra ∧ ((TempClass(t_i, L, PDC, w) ∩ t_i.rd) ≠ ∅) then
                t := t_i^+; r := r_j^+; break // cf. Alg. 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

Mining SoD and BoD ADARs Separation and Binding of Duty rules represent expected relative pairs of activities and resource assignments, cf. [7], 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 behaviour, cf. [3, 6].
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 is supporting a resource rule r, a set is generated that holds all resources $RS := \{e.r|e \in t \wedge e.ea \in \{rp.ra|rp \in r\}\}$ that have executed activities which are specified in r's conditions (cf., $rp.ra$). 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., $|RS| = |r|$. Accordingly, a 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 (cf. Table 1). This is because for trace t_1 the set $RS = \{\texttt{Mike}\}$ (i.e., $|RS| = 1$) while $|r| = 2$ so that $|RS| \neq |r|$.

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. Def 6. Hereby, the presented approach follows and exploits the common assumption that anomalous behavior is less likely than benign behavior, cf. [8, 4, 6]. 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. Def. 4. This implies: the more rules a trace supports the more likely it and its occurrence is assumed to be.

Definition 6 (Anomaly detection). *Let L be an execution log and t' be an execution trace with $t' \notin L$. Let further R be a set of rules, i.e., signature, that was mined for L and let $\mathcal{R}$ be the set of all signatures. Anomaly detection function $adec : \mathcal{R} \times \mathcal{L} \mapsto \{true, false\}$ with*

$$adec(R,t) := \begin{cases} \texttt{true} & \text{if} \sum\limits_{\substack{r \in R \\ mp(r,t')=true}} sup(r,L) < \sum\limits_{\substack{r \in R \\ mp(r,tsim(t',L))=true}} sup(r,L) \\ \texttt{false} & \text{otherwise.} \end{cases}$$

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. [19], between both activity bags is calculated. This means: the more equal activities[1] the traces contain (have executed) the more

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

similar they are, i.e., $J(\{\mathtt{A},\mathtt{C}\},\{\mathtt{B},\mathtt{C}\}) = |\{\mathtt{A},\mathtt{C}\} \cap \{\mathtt{B},\mathtt{C}\}| \,/\, |\{\mathtt{A},\mathtt{C}\} \cup \{\mathtt{B},\mathtt{C}\}| = 0.\overline{3}$. Overall the underlying similarity measure can be user chosen but this approach was found to be simple, fast, and sufficient during the evaluation, cf. Section 4.

Dealing with Change and Flexibility The proposed approach applies the common assumption that benign behavior is *more likely* than anomalous behavior, cf. [4]. 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. [4]. 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: a trace must not match to all rules a signature R is composed of. Instead the applied approach aggregates the support of each rule so that a trace can "compensate" an unsupported rule by supporting other rules. Such a relaxed approach, in comparison to stricter existing work, cf., [1], provides a basis to deal with noise and ad hoc changes, cf. [5].

Fostering Root Cause Analysis and Understandability One of the main driver for this work was the need for and lack of *root course analysis* capabilities in the process anomaly detection domain, cf. [4]. An insufficient support for root cause analysis can, for example, harden it to choose and apply *appropriate* countermeasures for identified anomalies. This aspect is tackled by:

Severity: existing process anomaly detection approaches frequently generate *binary* results, i.e., they mark traces either as anomalous or not, cf. [4]. However, this does not allow to assess the severity of an anomaly. 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 fine granularly report which parts of a given trace were affected by an identified anomaly.

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). Given the low amount of 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 root cause analysis, first of all, the proposed approach estimates the severity of each anomaly, which enables to quickly rank anomalies as less or more crucial. Secondly, it reports which rules are supported or not. This enables to quickly grasp the differences between expected and observed behaviour. Hereby, it can, for example, become evident that a combined resource and control rule violation origins from an inexperienced employee which can be contacted to explain and sort out the situation (countermeasure selection). Alternatively, it could become evident that a large amount of rules are not supported and that the execution must be terminated, as a countermeasure, to prevent further harm.

4 Evaluation

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. The utilized logs were taken from the BPI Challenge 2015[2] (BPIC) and Higher Eduction Processes (HEP), cf. [20].

The BPIC logs hold 262,628 execution events, 5,649 instances, and 398 activities. The logs cover the processing of building permit applications at five (BIPC_1 to BIPC_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 evenly randomly separated into training (for signature generation) and test data (for the anomaly detection performance evaluation). Subsequently, 50% randomly chosen test data entries were randomly mutated to inject artifical 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* – two mutators which randomly mutate the order and occurrence of activity execution events; and *b*) *Temporal* – randomly chosen activity

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

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 [5, 6]. 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 following results are an average of 100 evaluation runs. 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. 2, provides a configurable harmonic mean between P and R, cf. [9]. Hence, $\beta < 1$ results in a precision oriented result while $\beta = 1$ generates a balanced result.

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

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 calculating a signature within minutes and required 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. To ensure that for both data sources roughly the same amount of traces is analyzed only traces which did take place during 2015 were used from the BPIC 2015 logs.

Primary tests were applied to identify appropriate configuration values, e.g., the maximum rule length $rl := 3$ (control and temporal) and $rl := 2$ (resource). Longer rules can result in stricter potentially overfitting signatures. This is because longer rules contain more details and thus provide less flexibility than shorter ones. 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 0.9/0.8 for control/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 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.

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 would 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. Given the low amount of configuration variables we assume that existing automatic 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 Tab. 2. 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.

	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 2. Anomaly detection performance of the presented approach

5 Related Work

The most comparable work [18] also applies association rules for anomaly detection in processes. However, rules are largely manually generated (e.g., a user defines the expected maximum activity duration) and order dependencies between activities are not verified. Our survey on anomaly detection in business processes [4] reveals several shortcomings in most existing work: *a*) only single process perspectives are supported; and *b*) the analysis of anomalies is not fostered/supported; and *c*) monolithic signatures are frequently utilized – which are hard to grasp, struggle with noise and ad hoc changes, but also cannot be par-

tially updated whenever the underlying process changes. Likely, these limitations hinder the application of process anomaly detection in the real world.

Further, compliance checking approaches are related, cf. [15, 14]. These approaches utilize, e.g., rule based, definitions of expected process behavior to analyze its definition and execution for deviations and, partially, deviation root causes, cf. [16, 14]. However, such work typically does not take noise and ad hoc changes into account, possibly resulting in false positives. Further the expected behavior definitions are typically assumed as given as their creation can result in extensive (manual) efforts that require in depth domain and process knowledge.

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. [12, 5], because they focus on single unique use cases and data formats, such as, specific network protocols, e.g., the Session Initiation Protocol, cf. [17]. These approaches can hardly be generalized and applied to process execution logs which hold different data, formats, and contextual attributes.

One could argue that instead of applying anomaly detection the process definition could be secured by applying security focused modeling notations, cf. [7]. 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.

6 Discussion and Outlook

This paper focuses on two main challenges *a*) the detection of anomalies in process executions; and *b*) taking a first step towards supporting the analysis of detected anomalies. We conclude that this paper was able to meet the first challenge as the conducted evaluation showed an average anomaly detection recall of 89%. This goes with a substantial simplification of the generated signatures compared to previous work in [6] (complex likelihood graphs vs. short rules) which fosters the understandability of the signatures and identified anomalies.

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. For this experts would, as we assume, require, inter alia, information about *a*) which part of an execution trace is affected by an anomaly; and *b*) the anomaly severity, cf. [13]. It is shown how such information can be provided by the proposed rule based anomaly detection approach. To our knowledge this is the first *process anomaly detection* approach that does so. We assume that it is necessary to identify but also to understand anomalies to choose appropriate anomaly countermeasures.

Future work will concentrate on expanding and evaluating the proposed approaches' root cause analysis capabilities. For this, visualization and management tools will be created that enable to handle the provided information (e.g., which rules are supported or not) in an interactive manner. Further, user studies will be performed to assess the benefits of the provided information in detail.

References

1. Van der Aalst, W.M., de Medeiros, A.K.A.: Process mining and security: Detecting anomalous process executions and checking process conformance. Theoretical Computer Science 121, 3–21 (2005)
2. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Very Large Data Bases. vol. 1215, pp. 487–499 (1994)
3. Bezerra, F., et al.: Anomaly detection using process mining. Enterprise, business-process and information systems modeling 29, 149–161 (2009)
4. Böhmer, K., Rinderle-Ma, S.: Anomaly detection in business process runtime behavior–challenges and limitations. arXiv arXiv:1705.06659 (2017)
5. Böhmer, K., Rinderle-Ma, S.: Multi instance anomaly detection in business process executions. In: Business Process Management. pp. 77–93. Springer (2017)
6. Böhmer, K., et al.: Multi-perspective anomaly detection in business process execution events. In: Cooperative Information Systems. pp. 80–98. Springer (2016)
7. Brucker, A.D., Hang, I., Lückemeyer, G., Ruparel, R.: Securebpmn: Modeling and enforcing access control requirements in business processes. In: Access Control Models and Technologies. pp. 123–126. ACM (2012)
8. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. Computing Surveys 41(3), 15 (2009)
9. Chinchor, N., Sundheim, B.: Muc-5 evaluation metrics. In: Message understanding. pp. 69–78. Computational Linguistics (1993)
10. Czepa, C., et al.: Plausibility checking of formal business process specifications in linear temporal logic pp. 1–8 (2016)
11. Greco, G., Guzzo, A., Pontieri, L.: Mining taxonomies of process models. Data & Knowledge Engineering 67(1), 74–102 (2008)
12. Gupta, M., Gao, J., Aggarwal, C.C., Han, J.: Outlier detection for temporal data: A survey. Knowledge and Data Engineering 26(9), 2250–2267 (2014)
13. Julisch, K.: Clustering intrusion detection alarms to support root cause analysis. Information and System Security 6(4), 443–471 (2003)
14. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Information systems 54, 209–234 (2015)
15. Ly, L.T., Rinderle-Ma, S., Knuplesch, D., Dadam, P.: Monitoring business process compliance using compliance rule graphs. In: On the Move to Meaningful Internet Systems. pp. 82–99. Springer (2011)
16. Ramezani, E., Fahland, D., van der Aalst, W.M.P.: Where did I misbehave? diagnostic information in compliance checking. In: Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings. pp. 262–278 (2012)
17. Rieck, K., Wahl, S., Laskov, P., Domschitz, P., Müller, K.R.: A self-learning system for detection of anomalous sip messages. In: Services and Security for Next Generation Networks, pp. 90–106. Springer (2008)
18. Sarno, R., et al.: Hybrid association rule learning and process mining for fraud detection. Computer Science 42(2) (2015)
19. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to data mining. 1st (2005)
20. Vogelgesang, T., et al.: Multidimensional process mining: Questions, requirements, and limitations. In: CAISE Forum. pp. 169–176. Springer (2016)
21. Wieringa, R.: Design science methodology for information systems and software engineering. Springer (2014)