Noname manuscript No.
(will be inserted by the editor)

Discovering Anomalous Frequent Patterns From Partially Ordered Event Logs

Laura Genga · Mahdi Alizadeh · Domenico Potena · Claudia Diamantini · Nicola Zannone

the date of receipt and acceptance should be inserted later

Abstract Conformance checking allows organizations to compare process executions recorded by the IT system against a process model representing the normative behavior. Most of the existing techniques, however, are only able to pinpoint where individual process executions deviate from the normative behavior, without considering neither possible correlations among occurred deviations nor their frequency. Moreover, the actual control-flow of the process is not taken into account in the analysis. Neglecting possible parallelisms among process activities can lead to inaccurate diagnostics; it also poses some challenges in interpreting the results, since deviations occurring in parallel behaviors are often instantiated in different sequential behaviors in different traces. In this work, we present an approach to extract anomalous frequent patterns from historical logging data. The extracted patterns can exhibit parallel behaviors and correlate recurrent deviations that have occurred in possibly different portions of the process, thus providing analysts with a valuable aid for investigating nonconforming behaviors. Our approach has been implemented as a plug-in of the ESub tool and evaluated using both synthetic and real-life logs.

Keywords Subgraph Mining · Association Mining · Conformance Checking · Partially Ordered Logs

1 Introduction

Modern organizations are usually driven by the processes needed to create and deliver their products and services. Many organizations document their processes using *process models*. A process model provides a graphical (and often formal) representation of how processes have to be performed. However, in most organizations the behavior prescribed by process models is not enforced by the underlying IT systems. As a result, actual process executions may deviate from the prescribed behavior.

Laura Genga, Mahdi Alizadeh, Nicola Zannone
Eindhoven Univesity of Technology,
E-mail: {l.genga,m.alizadeh,n.zannone}@tue.nl

Claudia Diamantini, Domenico Potena
Università Politecnica delle Marche
E-mail: {d.potena,c.diamantini}@univpm.it

Deviations may be due to several reasons. For example, due to rapid changes in the market or in the organization structure, a process model might become outdated. In this case, deviations indicate that the model should be changed to capture the actual process behavior. Another reason for deviations is to deal with exceptional situations that arise during process executions. For example, an employee may skip some activities to handle an emergency more quickly. These exceptions should be examined to ensure that they conform to guidelines defined for handling these situations. Based on the analysis, the identified exceptions might be integrated into the process model or preconditions for their execution might be defined. Deviations can also indicate that the opportunity for mistakes or even frauds exists. By analyzing these deviations, certain control measures may be defined or the process (and, possibly, the IT system) may be redesigned to prevent the occurrence of undesired behaviors in the future.

To assess the conformance of their business processes, organizations usually employ logging mechanisms to record process executions in *event logs* and auditing mechanisms to analyze those logs. Event logs consist of *traces*, each recording data related to the activities performed during a process execution. In last decades, several research efforts have been devoted to the development of automatic and semi-automatic techniques for assisting organizations in the analysis of their event logs to assess the compliance of their processes. Among them, *conformance checking* [2] has been gaining an increasing attention from both practitioners and researchers. Conformance checking techniques usually replay each trace of an event log against a process model to detect possible deviations. State-of-the-art conformance checking techniques are based on the notion of *alignment* [55]. Given a trace and a process model, an alignment maps the events of the trace to the ones of a complete run of the model (see [55] for a formal definition of alignment). Through this mapping, an analyst can detect possible discrepancies between the observed and the prescribed behavior. Although a large body of research has addressed issues related to the building of alignments (e.g., [2,6]), supporting the *analysis* of the detected deviations has received little attention so far. Alignments enable a punctual and detailed analysis of single process executions or, at most, of groups of identical executions. However, manual analysis of single executions can likely become challenging, or even unfeasible, when an analyst has to explore a bunch of process executions (e.g., executions of the last six months, the last year...), involving hundreds or thousands of executions. This is especially true when the process under examination involves many possible variants and/or concurrent activities, since single executions will likely differ significantly among each other. We argue that to effectively support process diagnostics further elaboration has to be performed on the detected deviations to provide the analyst with a sort of "deviations dashboard", reporting analytics and interesting trends regarding the occurred deviations. Indeed, this type of output offers a more high-level perspective on deviant behaviors, from which it is possible to grasp meaningful insights on the deviations that would be otherwise scattered among a multitude of alignments. However, most of the well-known conformance checking techniques typically only provide some basic statistics, like the number of different discrepancies, or the average degree of compliance of process traces to the model.

To address these issues, in a previous work [22] we have proposed a framework designed to infer anomalous patterns representing *recurrent* deviations, together with their *correlations*. Given an event log and a process model, we applied a frequent subgraph mining technique to extract relevant subgraphs and introduced a conformance checking algorithm to identify the anomalous ones. Then, we proposed to correlate anomalous subgraphs by exploiting frequent itemset algorithms to detect the subgraphs that frequently occurred together and inferring ordering relations among them. Our framework enables an analysis of the deviations that can be considered as orthogonal to the output provided by classic alignment-based techniques. First, instead of determining the discrepancies for each process execution, our approach determines which deviations occurred most frequently. This is motivated by the fact that we could reasonably

expect that, in most cases, an analyst would like to ignore very rare deviations and rather focus on those that occur repeatedly and, hence, are more likely envisaged to occur in the future. Moreover, our framework infers portions of process executions involving one or more deviant behaviors, instead of single discrepancies, to provide the most common context(s) of execution within the process in which a given deviations occurred. Finally, the framework allows exploring possible correlations among deviations. It is often the case that a set of low-level deviations are a manifestation of a single high-level deviation [3], like the swapping of (sequences of) activities. These higher-level deviant behaviors typically turn out to be much more meaningful for diagnosis than individual low-level deviations. However, in order to diagnose them from a set of alignments, an analyst has to manually investigate the detected (low-level) deviations to reconstruct what happened, which usually turns out to be an error-prone and time-consuming task.

However, the approach in [22] is devised to derive anomalous patterns from *totally ordered* traces, with the result that the actual control-flow remains hidden. Extracting anomalous behaviors from sequential traces presents a number of drawbacks. When process executions are represented as totally ordered traces, the same concurrent behavior can be recorded differently in different traces, depending on the activities' execution time. For instance, given two concurrent activities A and B, we might find totally ordered traces where A was executed before B and other traces where B was executed before A. It is straight to see that recognizing a concurrent behavior from a set of sequential behaviors is far from trivial. As a consequence, given a set of totally ordered traces, one might infer different sequential deviant behaviors that actually correspond to the same deviation. This situation might lead to imprecise diagnoses and even to miss relevant deviant behaviors. Furthermore, it can have a negative impact in terms of both information loss and comprehensibility of the outcome. These issues are as more significant as higher is the degree of concurrency in the model (an example is provided in Section 3.1).

This work extends our previous work to deal with concurrency. We propose an approach for discovering anomalous frequent patterns showing recurrent and correlated deviations modeling both sequential and concurrent behaviors. In particular, we propose to extract frequent anomalous patterns from *partially ordered traces*, i.e. traces that explicitly represent concurrent behavior among process activities. The main contributions of this work are:

- We propose a conformance checking algorithm tailored to check the conformance of partially ordered subtraces exhibiting concurrent behavior.
- We investigate and formalize ordering relations between subgraphs exhibiting concurrent behavior with respect to a process model. Based on the identified relations, we devise an approach to infer partially ordered subgraphs that show correlations among recurrent anomalous behaviors.
- We propose transformation rules to convert partially ordered subgraphs into anomalous patterns expressed as Petri nets.
- We present an implementation of the proposed approach as a software plug-in of the *Esub* tool [16].
- We evaluate the proposed approach using one synthetic log and two real-life logs and compare the obtained results with the ones obtained using totally ordered traces (i.e., the approach in [22]).

The remainder of the paper is organized as follows. Section 2 introduces the concepts and notation used through the work. Section 3 provides an overview of the proposed approach along with a motivating example. Section 4 describes our approach for mining relevant subgraphs. Section 5 presents an algorithm for checking the compliance of subgraphs. Section 6 presents an approach to identify partially ordered subgraph and provides guidelines to build anomalous

patterns from them. Section 7 presents experiment results. Finally, Section 8 discusses related work and Section 9 draws conclusions and delineates future work.

2 Preliminaries

This section recalls the main concepts used throughout the paper. We start with the concept of *process model*, which describes the prescribed behaviors of a process. In this work, we use *labeled Petri nets* to represent process models.

Definition 1 (Labeled Petri Net) A *labeled Petri net* is a tuple $(P, T, F, A, \ell, m_i, m_f)$ where P is a set of places, T is a set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation connecting places and transitions, A is a set of labels for transitions representing process *activities*, $\ell : T \to A$ is a function that associates a label with every transition in T, m_i is the initial marking and m_f is the final marking.

In a Petri net, transitions represent activities and places represent states. We distinguish two types of transitions, namely invisible and visible transitions. Visible transitions are labeled with activity names. Invisible transitions are used for routing purposes or represent activities that are not observed by the IT system. Given a Petri net N, the set of activity labels associated with invisible transitions is denoted with $\mathsf{Inv}_N \subseteq A$. Multiple transitions can have the same label (i.e., they represent the same activity). Such transitions are called duplicate transitions.

The state of a process model is represented by a marking $m \in \mathbb{B}(P)$[1], i.e. a multiset of *tokens* on the places of the net. A process model has an initial marking m_i and a final marking m_f. A transition is *enabled* if each of its input places contains at least a token. When an enabled transition is fired (i.e., executed), a token is taken from each of its input places and a token is added to each of its output places. Given a labeled Petri net N, $m \xrightarrow{t}_N m'$ denotes the firing of an enabled transition t from marking m that leads to a new marking m'.

Process executions, so-called *process instances*, are typically recorded in logs. More precisely, the execution of an activity generates an *event*. The events related to a process instance are usually ordered in the log. In this work, we adopt *partially ordered traces*, also referred to as *instance graphs* in the literature [36]. Partially ordered traces explicitly model causal relations and parallelisms among process activities, thus providing a more accurate representation of process behaviors.

Definition 2 (Event, P-Trace, Log) Let $N = (P, T, F, A, \ell, m_i, m_f)$ be a labeled Petri net and $\mathsf{Inv}_N \subseteq A$ the set of invisible transitions in N. An *event* e represents a unique recorded execution of an activity $a \in A \setminus \mathsf{Inv}_N$. For an event e, $case(e)$ denotes the process instance in which e occurred and $act(e)$ the activity associated to e. The set of all possible events is denoted by $\mathcal{E}$. A *partially ordered trace* (or *p-trace*) σ is a directed acyclic graph (E, W) where $E \subseteq \mathcal{E}$ is the set of events related to a single process instance, i.e. for all $e_i, e_j \in E, case(e_i) = case(e_j)$, and W is a set of edges such that $(e_i, e_j) \in W$ indicates that event e_i has led to the execution of e_j. The edges in W define a partial order[2] over E. A *log* $\mathcal{L}$ is a set of p-traces.

It is worth noting that although many logging systems record sequential log traces, this does not affect the applicability of our approach. In fact, a number of approaches have been proposed

[1] $\mathbb{B}(X)$ represents the set of all multisets over X.

[2] Given a set Γ, a partial order over Γ is a binary relation $\preceq \subseteq \Gamma \times \Gamma$ that is *i)* irreflexive, i.e. $\gamma \not\preceq \gamma$, *ii)* antisymmetric, i.e. if $\gamma \preceq \gamma'$, then $\gamma' \not\preceq \gamma$ and *iii)* transitive, i.e. if $\gamma \preceq \gamma'$ and $\gamma' \preceq \gamma''$, then $\gamma \preceq \gamma''$.

in literature to derive p-traces from sequential log traces (see, e.g., [17,23,30,40,51]). Note that different techniques apply different strategies to infer concurrency among process activities. As a result, the structure of the obtained p-traces and, thus, the structure of the inferred deviant behaviors may vary on the basis of the selected technique. The choice of the employed technique depends on different factors, like the information available on the process, the data stored in the event log, the purpose of the analysis and so on. Nevertheless, the applicability of our approach is not constrained to the technique used to derive the p-traces.

In this work, we use the reachability graph of a Petri net to assess the conformance of p-traces with a process model. We first introduce the notion of marking reachability and then we formally define the notion of reachability graph.

Definition 3 (Marking Reachability) Let $N = (P, T, F, A, \ell, m_i, m_f)$ be a labeled Petri net. A marking $m \in \mathbb{B}(P)$ is *one step reachable* from a marking $m' \in \mathbb{B}(P)$ if there exists a transition $t \in T$ such that $m \xrightarrow{t}_N m'$. By extension, marking m' is *reachable* from marking m if there exists a sequence $\rho = \langle t_1, t_2, \ldots, t_n \rangle \in T^*$ such that $m \xrightarrow{t_1}_N m_1 \xrightarrow{t_2}_N \ldots \xrightarrow{t_n}_N m'$. We call ρ a *firing sequence* and, by abuse of notation, we write $m \xrightarrow{\rho}_N m'$ to indicate that firing sequence ρ leads from marking m to marking m'. The set of reachable markings of N, denoted $\mathcal{R}(N)$, is the set of all markings reachable from the initial marking m_i, i.e. $\mathcal{R}(N) = \{m \mid \exists \rho \in T^* : m_i \xrightarrow{\rho}_N m\}$.

Note that a marking is reachable by itself with a firing sequence of length 0.

Definition 4 (Labeled Transition System) A labeled transition system is a tuple (Q, Λ, R, q_0) where Q is a set of states, Λ is a set of labels, $R \subseteq Q \times \Lambda \times Q$ is a set of labeled transitions and q_0 is the initial state.

Definition 5 (Reachability Graph) Let $N = (P, T, F, A, \ell, m_i, m_f)$ be a labeled Petri net. The *reachability graph* of N, denoted $RG(N)$, is a labeled transition system (Q, Λ, R, q_0) where $Q = \mathcal{R}(N)$ is the set of markings in N reachable from the initial marking m_i, $\Lambda = A$, $R = \{(m, \ell(t), m') \in \mathcal{R}(N) \times A \times \mathcal{R}(N) | \exists_{t \in T}\ m \xrightarrow{t}_N m'\}$ is the set of labeled transitions and $q_0 = m_i$.

Intuitively, the reachability graph of a Petri net exhibits all behaviors allowed by the net. In particular, its paths represent the possible firing sequences allowed by the net.

3 Approach

In this section, we discuss the motivations for this work through a running example within the banking domain and provide an overview of our approach.

3.1 Motivating Example

Banks typically have internal processes that establish how activities and controls should be performed. Fig. 1 shows a process model representing the checking of the profile of a receiver of a money transfer[3], represented as a Petri net. The process starts with the activity *Start Receiver Processing* (SRP). Then, a pre-profiling phase is executed, starting with activity *Start Receiver*

[3] The complete money transfer process can be found in [52].

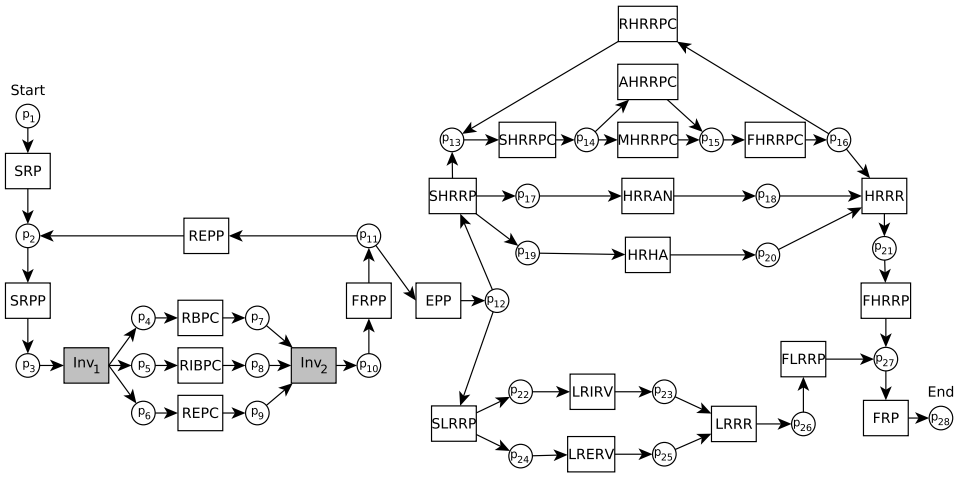
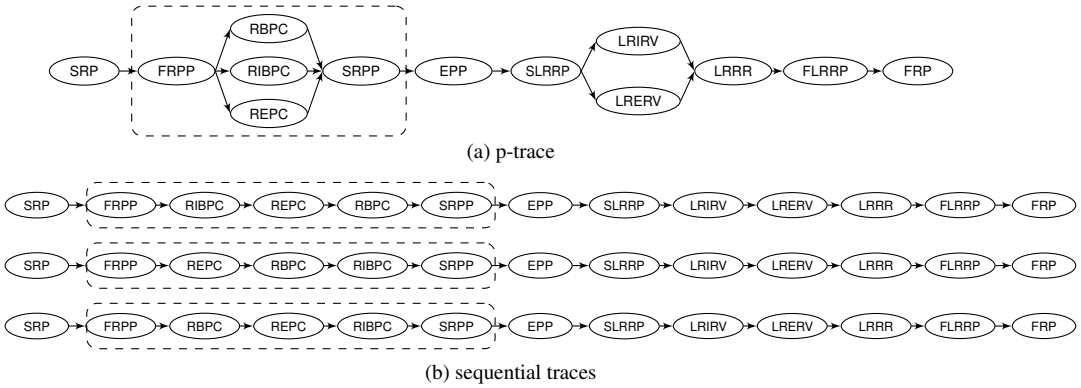


Fig. 1: Money transfer process. The boxes represent the transitions that are associated with process activities; gray boxes represent invisible transitions. The text in the boxes represents activity labels.

Pre Profiling ($SRPP$). This phase involves a set of controls on the receiver's bank, inter-bank and external profiling ($RBPC$, $RIBPC$ and $REPC$ respectively). The pre-profiling phase ends with activity *Finish Receiver Pre Processing* ($FRPP$). If further checking is required, activity *Request Extra Pre Profiling* ($REPP$) is performed. The obtained profile is then evaluated (EPP); on the basis of this evaluation, the process continues either with the management of "high risk" profiles, or with the management of "low risk" profiles. The management of high risk profiles starts with activity *Start High Risk Receiver Processing* ($SHRRP$) and involves the execution of three parallel subprocesses. The first subprocess, starting with activity *Start High Risk Receiver Profile Check* ($SHRRPC$), involves either an automatic or manual profile check ($AHRRPC$ and $MHRRPC$) and ends with activity *Finish High Risk Receiver Profile Check* ($FHRRPC$). After that, an additional check can be required, represented by activity *Request High Risk Receiver Profile Check* ($RHRRPC$). The other two subprocesses involve an analysis of the receiver's history ($HRHA$) and a notification to an external authority about the profile that has being processed ($HRRAN$). At the end of these three subprocesses, the receiver's request is registered ($HRRR$) and the high risk profile checking is marked as completed ($FHRRP$). The processing of low risk profiles starts with activity *Start Low Risk Receiver Processing* ($SLRRP$) and involves just an internal and external profile verification ($LRIV$ and $LREV$, respectively) before registering the receiver ($LRRR$) and marking the checking phase as completed ($FLRRP$). The process ends with the execution of activity *Finish Receiver Processing* (FRP).

Although a process model defines the activities and workflows to be performed, i.e. how processes should be executed, reality may deviate from such a model. Consider, for instance, the case where pre-profiling checking is not performed properly. More precisely, activity $FRPP$ is performed immediately after SRP (i.e., the execution of the activities $FRPP$ and $SRPP$ is swapped), which means that the pre-profiling step has been marked as completed before executing the prescribed controls. This might occur for many reasons. For instance, there might be an implicit work practice of skipping pre-profiling checking when the receiver is an important and trusted customer of the bank; or, it might represent malpractices, leading to a wrong evaluation of the risks associated to the receiver's profile.



(a) p-trace

(b) sequential traces

Fig. 2: Instance of the process model in Fig. 1 where activities $FRPP$, $SRPP$ have been swapped, represented by means of a p-trace (a) and by means of some sequential traces (b).

A number of conformance checking techniques have been proposed to assist organizations in the analysis of the observed user behavior against the normative behavior and in the detection of deviations from the specifications. However, these techniques typically focus on the analysis of single process instances. Therefore, these approaches often provide limited insights about deviations and their occurrence, which makes it difficult for an organization to understand and manage them. For instance, if a certain deviation occurs very rarely, it likely corresponds to exceptional circumstances that should be investigated and addressed individually. On the other hand, if it occurs repeatedly, investigating every occurrence of the deviation individually can be time consuming and costly. In these cases, it would be desirable to identify *anomalous patterns* representing recurrent anomalous behaviors (i.e., deviations from the specification). Anomalous patterns, thus, can provide a baseline for the investigation of deviations requiring systematic and persistent corrective actions that address the source of the problem and, thus, are able to cope with all the occurrences of the identified deviations.

Moreover, to assess the compliance of the observed behavior with the normative behavior, existing conformance checking techniques compare process instances against the process model. However, most of these techniques are only able to deal with *sequential* traces. A sequential trace is a particular case of p-traces (Definition 2) in which the events in the trace are totally ordered.

Extracting anomalous behaviors from sequential traces, however, presents a number of drawbacks. In particular, the flat representation of sequential traces does not make it possible to represent the control flow of the process and, thus, possible parallelisms between process activities remain hidden. This can have a negative impact both in terms of information loss and in terms of comprehensibility of the outcome by a human analyst. These issues are as more significant as higher is the degree of concurrency in the model.

As an example, consider the swapping of activities $FRPP$ and $SRPP$ discussed above. Fig. 2a shows the corresponding p-trace and Fig. 2b shows (a subset of) the sequential traces obtained by flattening the p-trace. If we match each sequential trace against the process model, we obtain different possible instantiations of the anomalous behavior, which are highlighted by a dashed rectangle in the traces of Fig. 2b. Intuitively, these traces differ for the order in which activities $RBPC$, $RIBPC$ and $REPC$ have been executed. It is up to the analyst to recognize that all of them are related to the same parallel behavior, which can be time-consuming and challenging, especially when the level of concurrency is high. Moreover, sequential instances have an overall *support* (intended as the percentage of process instances involving the behavior

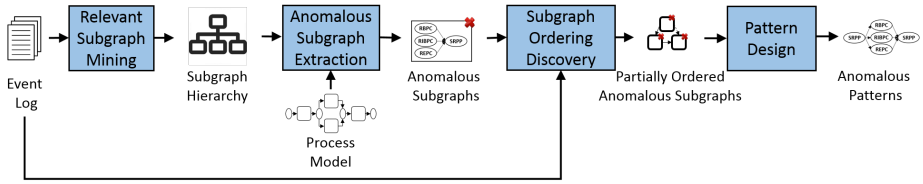


Fig. 3: Overview of the approach

at least once) much lower than the support of the real behavior. As a consequence, when we look for recurrent behaviors, we might miss some (or, in the worst case, all) sequential instances of a parallel behavior.

To address these issues, concurrency has to be taken into account when deriving anomalous behaviors. In our example, considering concurrency (i.e., comparing a p-trace, instead of its corresponding sequential traces, against the model) would make it possible to derive the pattern wrapped by the dashed rectangle in Fig. 2a. It is straightforward to observe that this pattern provides a more accurate and compact description of the anomalous behaviors, thus providing diagnostics that are more meaningful and easy to understand for a human analyst. Moreover, adopting this representation allows us to compute the support of a certain behavior without any information loss, thus providing a more accurate detection of recurrent behaviors.

3.2 Framework

In this work, we propose an approach to mine recurrent anomalous behavioral patterns from historical logging data, explicitly taking into account the process control-flow structure. The approach extends our previous work [22] in order to deal with process concurrency. An overview of the approach is given in Fig. 3. It consists of four main phases: *(i) relevant subgraph mining*, *(ii) anomalous subgraph extraction*, *(iii) subgraph ordering discovery* and *(iv) pattern design*.

Given an event log comprising p-traces, we first apply a frequent subgraph mining technique to extract *relevant subgraphs* (Section 4). The mined subgraphs can exhibit concurrent executions of activities and are structured in a hierarchy with respect to the inclusion relation occurring among subgraphs. In the anomalous subgraph extraction phase (Section 5), a novel conformance checking algorithm is exploited to extract *anomalous subgraphs* (i.e., subgraphs that do not fit the prescribed behavior defined in the process model) from the subgraph hierarchy by comparing the mined subgraphs against the process model. In the last two phases (Section 6), we discover ordering relations between anomalous subgraphs with respect to their occurrence in the p-traces and exploit these relations to design anomalous patterns representing frequent anomalous behaviors. In particular, *partially ordered anomalous subgraphs* are derived by detecting ordering relations among correlated anomalous subgraphs. We finally provide an approach to construct *anomalous patterns* from partially ordered anomalous subgraphs through transformation rules.

The patterns obtained using the proposed approach can be exploited in various ways. First, they highlight frequent and correlated anomalous behaviors in historical logging data, thus providing a valuable support to diagnostics. They can be used to enhance classic conformance checking techniques to detect high-level deviations, as proposed in [3]. Depending on the context and activities forming the patterns, they can be exploited for supporting the redesign of the process and/or for implementing additional control mechanisms. For instance, a pattern might point out that the process model is outdated. In this case, one may repair the process model to better represent the real processes, as done for example in the work of Fahland et al. [20]. In

other cases, a pattern might represent exceptional behaviors, which should be properly examined. Based on the analysis, these exceptions may also be explicitly modeled in the process model and preconditions can be defined to regulate their execution. When these exceptions represent undesired behaviors, an analysis of the captured patterns can also lead to the definition of control measures or a revision of the process model to prevent the occurrence of errors or frauds in the future. Beside their exploitation at design time, the patterns inferred using our approach can be exploited at run-time by providing a baseline for the definition of monitoring solutions tailored to run-time detection of anomalous behavior in process executions, relieving an analyst from the burden of reevaluating situations already analyzed.

4 Relevant Subgraphs Mining

The first phase of the approach aims to identify relevant *subgraphs* occurring in the p-traces recorded in the log. A subgraph γ of a p-trace is a directed acyclic graph formed by a subset of the nodes and a subset of the edges of the p-trace. To determine the set of relevant subgraphs, we apply a *Frequent Subgraph Mining* (FSM) algorithm to the set of p-traces in the log. The choice of the FSM algorithm depends on the notion of "subgraph relevance" to be used. Existing FSM algorithms often adopt the subgraph *support* as the underlying notion of relevance. The support of a subgraph is related to its occurrence frequency, which can refer either to all occurrences of the subgraph in a graphs set or to the percentage of graphs involving the subgraph at least once (thus ignoring multiple occurrences of the subgraph in the same graph). In this work, we relate the relevance of a subgraph both to its frequency and size. More specifically, we evaluate the relevance of a subgraph in terms of its Description Length (DL), i.e. the number of bits needed to encode its representation, computed as the sum of the number of bits needed to encode its nodes and the number of bits needed to encode its edges (further details on DL are provided in [27]). Subgraphs with the highest score in terms of DL represent the largest subgraphs that can be obtained at a given frequency. Intuitively, these subgraphs represent core pieces of executions of the process. Extending these subgraphs results in larger, but less frequent, subgraphs representing different possible extensions of the process execution path described by the core subgraphs. To the best of our knowledge, the only FSM algorithm that explicitly considers DL is SUBDUE [33].

Given a graph set G and a subgraph γ, SUBDUE uses an index based on DL, hereafter denoted by $\nu(\gamma, G)$, which is computed as $\nu(\gamma, G) = \frac{DL(G)}{DL(\gamma)+DL(G|\gamma)}$ where $DL(G)$ is the DL of G, $DL(\gamma)$ is the DL of γ and $DL(G|\gamma)$ is the DL of G *compressed* using γ, i.e. the graph set in which each occurrence of γ is replaced with a single node. Intuitively, the relevance of a subgraph is associated with its compression capability: the higher the value of ν is, the lower the DL value of the graph dataset compressed by γ is.

SUBDUE works iteratively. At each step, it extracts the subgraph with the highest compression capability, i.e. the subgraph corresponding to the maximum value of the ν index. This subgraph is then used to compress the graph set. The compressed graphs are presented to SUBDUE again. These steps are repeated until no more compression is possible. The outcome of SUBDUE is a hierarchical structure, where mined subgraphs are ordered according to their relevance, showing the existing inclusion relationships among them. Top-level subgraphs are defined only through elements belonging to input graphs (i.e., nodes and arcs), while lower level subgraphs contains also upper level subgraphs as nodes. Descending the hierarchy, we pass from subgraphs that are very common in the input graph set to subgraphs occurring in a few input graphs.

An example of SUBDUE hierarchy is shown in Fig. 4. Here, we have two top-level subgraphs, γ_{20} and γ_{28}, and two lower-level subgraphs, γ_{24} and γ_{29}. The hierarchy shows inclusion relationships between subgraphs. Specifically, γ_{24} is a child of γ_{20} because it involves γ_{20} in

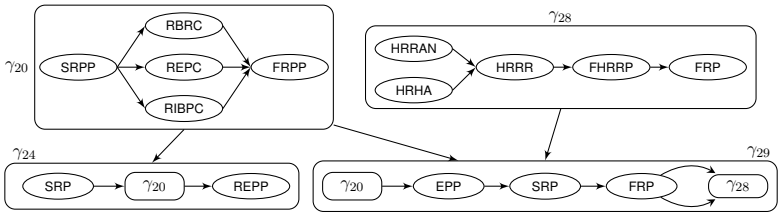


Fig. 4: Example of subgraph hierarchy obtained using SUBDUE

its definition. Similarly, γ_{29} is a child of both γ_{20} and γ_{28}. In other words, γ_{24} and γ_{29} are possible "specializations" of γ_{20} and of γ_{20} and γ_{28} respectively, since they extend the higher level subgraphs with other elements. The hierarchical structure of subgraphs discovered by SUBDUE becomes the input of the next phase of the approach.

5 Anomalous Subgraph Extraction

The second step of our approach aims to determine among the subgraphs identified in the previous step those that do not fit the given process model. In contrast to most of the existing conformance checking algorithms, the algorithm proposed in this section is tailored to check the conformance of subgraphs corresponding to portions of process executions. The core idea of this algorithm is to replay a subgraph against the given process model represented by its reachability graph.

Given a subgraph γ and the reachability graph $RG(N)$ of a Petri net N, Algorithm 1 determines whether γ is "compliant" or "noncompliant" (i.e., anomalous) with $RG(N)$. First, the algorithm generates all sequences that can be obtained by flattening the subgraph (line 1). Then, each of these sequences is analyzed against $RG(N)$ (lines 2-17). The algorithm uses a stack to store the portion of the reachability graph to be visited. In particular, it stores triples of the form (q_i, Q_i, seq_i) where q_i is a visited state in $RG(N)$, Q_i is the set of states from which q_i can be reached through a sequence of invisible transitions and seq_i is the sequence of activities that still need to be compared with $RG(N)$. Note that the set of states Q_i is needed to deal with loops formed by invisible transitions, as discussed below.

Given a sequence b, the stack is initialized by inserting a triple $(q, \{\}, \langle b_2, \ldots, b_n \rangle)$ for each state in $RG(N)$ with an incoming edge labeled with the first activity of b (lines 4-5). Then, at each iteration, Algorithm 1 retrieves a triple from the stack. If no activity remains to be checked, it means that there exists a firing sequence in $RG(N)$ whose labels match with the sequence of activities in b. Thus, the algorithm sets the value of $seqFound$ to $true$ (line 9) and starts comparing the next sequence with $RG(N)$. Otherwise, the algorithm compares the label of outgoing arcs from the current state with the next activity in the sequence (lines 11-15). If they match, the algorithm adds a new triple to the stack for the tail of such an outgoing edge, and the search continues.

Note that the algorithm is robust with respect to the presence of invisible transitions. Edges labeled with invisible transitions are taken into account while exploring the search space, but are not used while matching the paths with the obtained sequences from a subgraph (lines 14-15). To detect loops containing only invisible transitions and avoid exploring them again, the algorithm keeps track of the states from which the current state can be reached through a sequence of invisible transitions. If the next state belongs to the set of states already visited, the algorithm does not explore it again; otherwise, it is explored. If for all the sequences obtained by flattening a

Algorithm 1: subgraphConformanceChecking

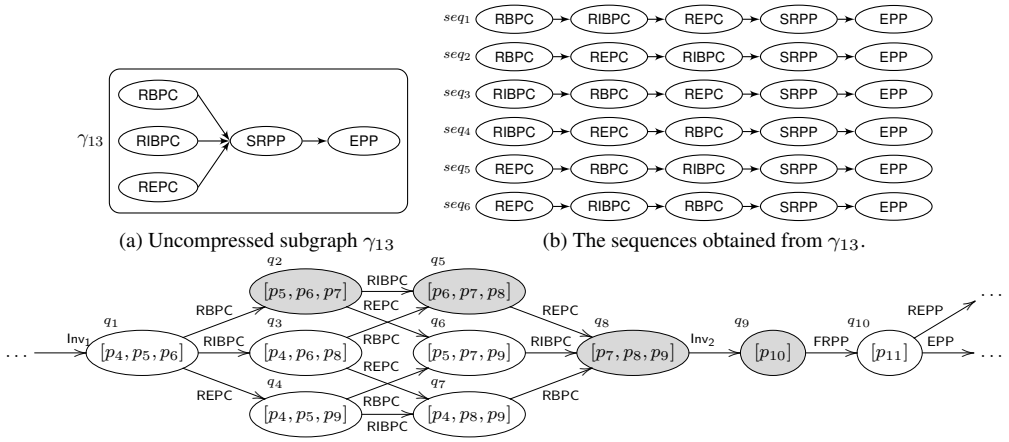
Input : Subgraph γ, Reachability graph $RG(N) = (Q, \Lambda, R, q_0)$
Output : Result of comparison between the subgraph γ and reachability graph $RG(N)$

1 Let B be the set of all sequences obtained by flattening subgraph γ;
2 **foreach** $b = \langle b_1, \ldots, b_n \rangle \in B$ **do** // Check compliance of sequences with $RG(N)$
3 $seqFound \leftarrow false$;
4 **foreach** $q \in Q$ *with an incoming arc labeled with* b_1 **do** // Initialize the stack
5 add $(q, \{\}, \langle b_2, \ldots, b_n \rangle)$ to $stack$;
6 **while** $stack \neq empty \wedge seqFound = false$ **do**
7 extract $(q_k, \{q_j, \ldots, q_l\}, \langle b_i, \ldots, b_n \rangle)$ from $stack$;
8 **if** $\langle b_i, \ldots, b_n \rangle = \langle\rangle$ **then**
9 $seqFound \leftarrow true$;
10 **else**
11 **foreach** $(q_k, a_t, q_t) \in R$ **do**
12 **if** $a_t = b_i$ **then**
13 add $(q_t, \{\}, \langle b_{i+1}, \ldots, b_n \rangle)$ to $stack$;
14 **else if** $a_t \in Inv_N \wedge q_t \notin \langle q_j, \ldots, q_l \rangle$ **then**
15 add $(q_t, \{q_j, \ldots, q_l, q_k\}, \langle b_i, \ldots b_n \rangle)$ to $stack$;
16 **if** $seqFound = false$ **then**
17 **return** noncompliant; // Subgraph γ is noncompliant with $RG(N)$
18 **return** compliant; // Subgraph γ is compliant with $RG(N)$

subgraph, the algorithm finds a firing sequence in $RG(N)$ whose labels match with the sequence, the subgraph is marked as compliant (line 18); otherwise, it is marked as noncompliant (line 17).

Fig. 5 shows an example of application of Algorithm 1. Consider subgraph γ_{13} in Fig. 5a and (a portion of) the reachability graph corresponding to the Petri net of Fig. 1 in Fig. 5c. The sequences obtained by flattening the subgraph are shown in Fig. 5b. Starting from the first sequence, i.e. seq_1, the algorithm looks for the states with an incoming edge whose label matches with the label of the first node in the sequence, i.e. $RBPC$. There are four states with an incoming edge labeled $RBPC$, i.e. q_2, q_5, q_6 and q_8. Starting from these states, Algorithm 1 checks if there exists a sequence of edges in the reachability graph whose labels matches with the sequence obtained from the subgraph. Consider, for instance, that Algorithm 1 selects q_2. The states reached by the algorithm from this state are denoted in gray in Fig. 5c. We can observe that there is a sequence of edges whose labels match the labels of the second and third nodes in seq_1, i.e. $\langle (q_2, RIBPC, q_5), (q_5, REPC, q_8) \rangle$. Then, q_9 can be reached from q_8 because (q_8, Inv_2, q_9) is a transition of the reachability graph and Inv_2 is a label representing an invisible transition. However, q_9 does not have any outgoing edge labeled with $SRPP$, i.e. the label of the fourth node in seq_1. Therefore, the algorithm explores the other initial states (i.e., q_5, q_6 and q_8) to find such a sequence in the reachability graph. As such a sequence does not exist, the subgraph is marked as noncompliant.

It is worth noting that usually *inclusion relations* exist among the subgraphs returned by subgraph mining techniques. Namely, it is likely to have one or more subgraphs involved in larger subgraphs. These inclusion relations affect the extraction of correlations among deviations (Section 6) and, consequently, the definition of anomalous patterns. Specifically, subgraphs related by an inclusion relation are highly correlated, thus yielding redundant information. A possible way to avoid the presence of inclusion relations among mined subgraphs consists in exploiting special constraints when mining the subgraphs, returning, for instance, only the maximal ones (as done by, e.g. SPIN [28] and MARGIN [49]). However, setting constraints based on the subgraphs structure might lead to miss subgraphs describing interesting deviations. A more accurate approach to deal with this issue consists in analyzing the inclusion relations



(a) Uncompressed subgraph γ_{13}

(b) The sequences obtained from γ_{13}.

(c) Portion of the reachability graph of the Petri net in Fig. 1. The states denoted in gray are the explored states in the reachability graph for seq_1 starting from $(q_2, \langle RIBPC, \ldots, EPP \rangle)$.

Fig. 5: Example of application of Algorithm 1.

among the mined subgraphs to determine which of them have to be taken into account and which have to be excluded when deriving partially ordered subgraphs to avoid the generation of redundant information.

In this work, we are interested in deriving noncompliant subgraphs that are *representative* of relevant anomalous behaviors, namely all those subgraphs γ_i such that *i)* γ_i captures a noncompliant behavior, and *ii)* there does not exists any other subgraph γ_j such that $relevance(\gamma_j) > relevance(\gamma_i)$[4] and $\gamma_j \subset \gamma_i$. Representative subgraphs represent the smallest and most relevant noncompliant subgraphs. They allow us to define anomalous patterns representing most relevant deviations, without incurring in the redundancy issue previously discussed. Hereafter, we call those subgraphs *anomalous subgraphs*.

To identify anomalous subgraphs, inclusion relations among noncompliant subgraphs obtained using Algorithm 1 have to be analyzed. To this end, we exploit the hierarchical structure returned by SUBDUE, since it makes inclusion relations explicit. Indeed, by definition every node in the hierarchy is included in all its descendants.

In [22] we presented an approach to assess the compliance of sequential subgraphs, i.e. subgraphs formed by a sequence of activities, and extract anomalous subgraphs. In this setting, anomalous subgraphs correspond to the *minimal* noncompliant subgraphs. Indeed, if a subgraph does not comply with the model, its descendants also do not comply with the model, since they also contain the noncompliant sequence of activities represented in the subgraph. Therefore, when a noncompliant subgraph is identified, the algorithm presented in [22] marks it as anomalous and prunes all the sub-hierarchies rooted in the subgraph.

When considering concurrency, however, this property does not hold. In particular, a subgraph may be marked as noncompliant because it does not exhibit concurrent behavior as defined in the model. However, these subgraphs may be extended by descendants with the required activities. Thus, in presence of concurrency, the descendants of a noncompliant subgraphs can comply with the model. As a consequence, anomalous subgraphs might not correspond to minimal noncompliant subgraphs.

[4] The definition of $relevance$ depends on the FSM algorithm exploited to derive the subgraphs. In this work, we have adopted DL (see Section 4).

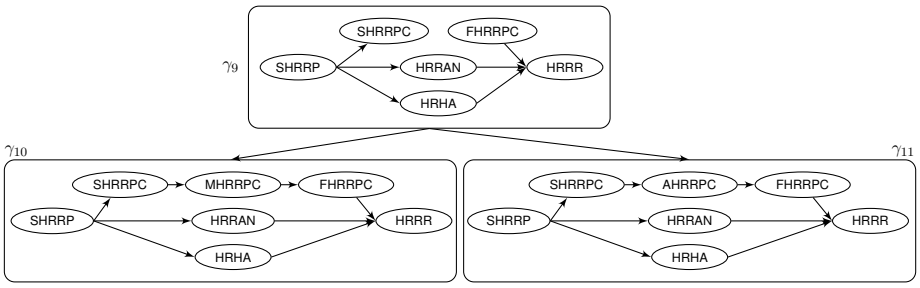


Fig. 6: Uncompressed subgraph γ_9, γ_{10}, and γ_{11} and their relations in the hierarchy.

Algorithm 2: findAnomalousSubgraphs

Input : Subgraph hierarchy H, Reachability graph $RG(N)$
Output : Anomalous subgraphs AS

1 Let TS be top-level subgraphs in H;
2 $AS \leftarrow \emptyset$;
3 **foreach** $ts \in TS$ **do**
4 $\quad AS \leftarrow AS \cup checkDescendants(H, ts, RG(N))$;
5 **return** AS;

For example, consider the process model in Fig. 1. According to this model, either $MHRRPC$ or $AHRRPC$ should be executed before the execution of $FHRRPC$. Suppose that, due to the choice construct that involves place p_{14}, events recording the execution of these activities occur less frequently than others and, thus, subgraph γ_9 in Fig. 6 is obtained. This subgraph is marked as noncompliant by Algorithm 1 as it it cannot be replayed by the process model. However, the direct descendants of this subgraph, γ_{10} and γ_{11}, contain an extra node with label $MHRRPC$ and $AHRRPC$ respectively, which makes them compliant with the model.

To deal with these situations, we have extended and revised the approach presented in [22] to also analyze the descendants of noncompliant subgraphs. If a noncompliant subgraph has some compliant descendants, it means that the subgraph does not capture concurrent behavior as specified in the model. This subgraph is not marked as anomalous but its direct descendants are iteratively analyzed to determine whether they should be marked as anomalous. For example, subgraph γ_9 in Fig. 6 is not considered anomalous because all its descendants are compliant. On the other hand, if all the descendants of the subgraph are noncompliant, only the upper-level subgraph is marked as anomalous, since it represents the most representative anomalous behavior compared to its descendants.

This idea is captured by algorithm $findAnomalousSubgraphs$ (Algorithm 2). Given a subgraph hierarchy H and the reachability graph $RG(N)$ of a Petri net N, Algorithm 2 returns the anomalous subgraphs to be considered for the design of anomalous patterns. The algorithm starts by identifying the top-level subgraphs in H (line 1). For each top-level subgraph ts, $checkDescendants$ (Algorithm 3) is used to explore the subgraph hierarchy rooted in ts and identify anomalous subgraphs in the hierarchy (line 4). The union of identified anomalous subgraph is stored in AS and is returned as anomalous subgraphs.

Algorithm 3 takes a subgraph hierarchy H, a subgraph γ and the reachability graph $RG(N)$ of Petri net N as input and returns the set of anomalous subgraphs with respect to the sub-hierarchy of H rooted in γ and $RG(N)$. First, the algorithm checks whether γ is compliant with $RG(N)$ using Algorithm 1 (line 1). Then, the direct descendants of the subgraph are identifies (line 2). If the subgraph has no descendants, depending on whether it is compliant or

Algorithm 3: checkDescendants

Input : Subgraph hierarchy H, Subgraph γ, Reachability graph $RG(N)$
Output : Identified anomalous subgraphs after exploring γ and its descendants in H

```
1  result ← subgraphConformanceChecking(γ, RG(N));
2  Let D be the set of direct descendants of γ in H;
3  if D = ∅ then
4      if result = compliant then
5          return ∅;                  // Subgraph γ is compliant and has no descendant
6      else
7          return {γ};                // Subgraph γ is noncompliant and has no descendant
8  else
9      AS_d ← ∅;
10     foreach d ∈ D do
11         AS_d ← AS_d ∪ checkDescendants(H, d, RG(N));
12     if result = compliant then
13         return AS_d;                                   // Subgraph γ is compliant
14     else
15         if D ⊆ AS_d then
16             return {γ};            // All descendants of subgraph γ are noncompliant
17         else
18             return AS_d; // Only some descendants of subgraph γ are noncompliant
```

noncompliant, an empty set (line 5) or the set containing the subgraph is returned (line 7). If the subgraph has some descendants, the algorithm explores the sub-hierarchy rooted in each descendant recursively (line 11) and identifies anomalous subgraphs in each sub-hierarchy. The set of the identified anomalous subgraphs is stored in AS_d. If the subgraph is compliant, all descendants marked as anomalous are returned (line 13). Otherwise, the algorithm checks whether all the direct descendants of the subgraph are in AS_d (line 15). If this is the case, it means that the subgraph and all of its descendants are noncompliant. Thus, only the subgraph γ, which involves the most representative deviant behavior, is marked as anomalous and, thus, returned to the parent subgraph (line 16). If some descendants of the subgraph are compliant and some are noncompliant, the algorithm returns the noncompliant subgraphs identified by exploring the descendants of the subgraph (line 18).

Fig. 7 shows an example of subgraph hierarchy that can be obtained using SUBDUE. In the figure, compliant subgraphs are denoted by a green dash-dotted line rectangle, anomalous subgraphs are denoted by a red full line rectangle, noncompliant subgraphs with some compliant descendants are denoted by a blue dotted line rectangle, and noncompliant subgraphs for which all siblings (with respect to one parent subgraph) are noncompliant, are denoted by an orange dashed line rectangle. The numbers on the top left of rectangles represent the order in which the algorithm explores subgraphs, the set on the top right of rectangles shows the anomalous subgraphs identified after exploring the subgraph and its descendants, the label on the bottom left of rectangles represents whether the subgraph is compliant (C) or noncompliant (NC) with the reachability graph of the Petri net. Subgraphs γ_1, γ_3 and γ_9 are not marked as anomalous subgraphs because some of their descendants are compliant. In this case, their noncompliant descendants, namely γ_2, γ_5 and γ_6, are marked as anomalous subgraphs. As subgraph γ_{12} and all of its descendants are noncompliant, Algorithm 3 only marks γ_{12} as an anomalous subgraph in this sub-hierarchy. For the same reason, subgraph γ_8 is also marked as an anomalous subgraph. Note that some subgraphs may be reachable from multiple root subgraphs in the subgraph hierarchy. The compliance of these subgraphs is determined with respect to each root subgraph individually. Thus, they are marked as anomalous if they are anomalous with respect to at least one of these root subgraphs. For example, subgraph γ_2 is reachable from both subgraphs

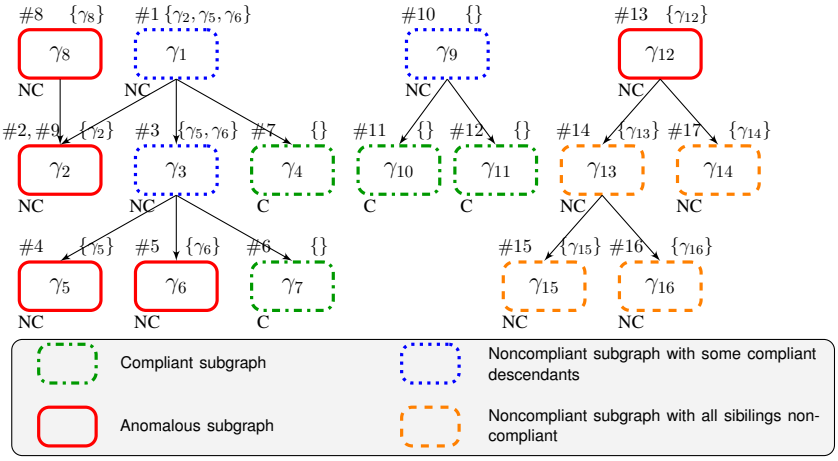


Fig. 7: Example of application of Algorithm 2. The numbers on the top left of rectangles represent the order in which the algorithm explores them, the set on the top right of rectangles shows subgraphs marked as anomalous after exploring the subgraph and its descendants, the label on the bottom left of rectangles represents whether the subgraph is compliant (C) or noncompliant (NC) with the reachability graph of the Petri net.

γ_1 and γ_8. Algorithm 3 marks γ_2 as anomalous because it is anomalous with respect to the hierarchy rooted in subgraph γ_1.

6 Subgraph Ordering Discovery and Pattern Design

Anomalous behaviors may comprise deviations that manifest in different (and possibly not connected) portions of a process execution. Consider, for instance, the case where an activity a_1 that should be executed at the beginning of the process has been swapped with an activity a_2 that should be executed at the end of the process. This anomalous behavior will be likely captured as two distinct anomalous subgraphs, one involving a_1 and one involving a_2. The last two steps of the approach address those situations. In particular, these steps aim to derive ordering relations among anomalous subgraphs describing how apparently different anomalous behaviors are usually correlated, and show how to build anomalous patterns from the discovered partially ordered subgraphs by means of classical process control-flow constructs. In the remainder of this section, we discuss these steps in detail.

6.1 Deriving ordering relations among subgraphs

To infer ordering relations among anomalous subgraphs, we first need to determine which ones tend to occur together. To this end, we generate an occurrence matrix C of the anomalous subgraphs obtained in the previous step of the approach. The occurrence matrix C is an $n \times m$ matrix where n is the number of extracted anomalous subgraphs, m is the number of p-traces in the log and each cell $c_{ij} \in C$ represents the number of occurrence of the $j\text{-}th$ subgraph in the $i\text{-}th$ p-trace.

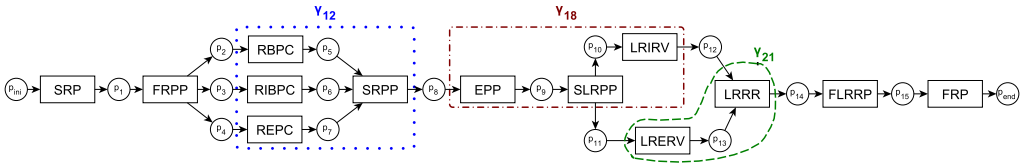


Fig. 8: P-trace of Fig. 2a converted in a Petri net.

Determining whether a subgraph occurs in a p-trace corresponds to resolving a *subgraph isomorphism problem* [32]. Formally, given two graphs $g = (V, W)$ and $g' = (V', W')$, the goal of subgraph isomorphism is to check if there exists a graph $g'' = (V'', W'')$ such that $V'' \subseteq V$, $W'' \subseteq W$ and g'', g' are *isomorphic*, namely if it is possible to map all the nodes and edges of g' on g'' (involving the corresponding labels). Subgraph g'' is usually referred as an *embedding* of g' in g. Several algorithms have been proposed in the literature to efficiently compute subgraph isomorphism (see, e.g. [42,46,50]). In this work, we exploit the subgraph isomorphism technique implemented within the SUBDUE algorithm [33]. Given a subgraph γ and a p-trace σ, such an algorithm returns all the embeddings of γ in σ, thus allowing us to build the occurrence matrix of anomalous subgraphs.

We apply well-known *frequent itemset algorithms* [26] to the obtained occurrence matrix in order to derive the sets of subgraphs that co-occur with a support above a given threshold. These sets (hereafter referred to as *frequent itemsets*) represent the sets of subgraphs that frequently occur together in the p-traces forming the log. Note that frequent itemsets can consist of a single subgraph. These itemsets represent deviant behaviors that occur frequently but for which it is not possible to infer any relation with other deviant behaviors. In particular, they might represent atomic deviant behaviors (i.e., involving a single deviant behavior) or they might actually involve a set of deviant behaviors that tend to occur close to each other during process executions (i.e., few other activities occur in between), with the result that the whole deviant behavior is captured by a single subgraph.

To determine how the subgraphs in a frequent itemset are related to each other, we infer ordering relations between the subgraphs of the itemset in a pairwise fashion. Before defining these relations, we introduce some notation. We first observe that a p-trace can be transformed into a Petri net through graph transformation rules (see [19,43] for examples of transformations from various graph notations into Petri nets). The transformation of a p-trace into the corresponding Petri net is straightforward. Intuitively, the transitions in the Petri net correspond to the nodes (i.e., events) of the p-trace and places correspond to edges. An algorithm for transforming a p-trace into the corresponding Petri net is given in Appendix A. Hereafter, $\overline{\sigma}$ denotes the representation of a p-trace σ as a Petri net. Fig. 8 shows the Petri net obtained by converting the p-trace of Fig. 2a. For the sake of readability, we sometimes abuse the notation and denote transitions using the activity label of the corresponding events (as done in Fig. 8).

Transforming a p-trace into a Petri net allows us to exploit the reachability graph encoding the behavior of the p-trace to reason on the states (represented as markings) of the subgraphs within the p-trace and, thus, correctly determine ordering relations between the subgraphs occurring in the p-trace. Let $\sigma = (E, W)$ be a p-trace and $\gamma = (E', W')$ a subgraph of σ (with $E' \subseteq E$ and $W' \subset W$). We denote $M^{\sigma}_{ini}(\gamma)$ the set of markings in $\mathcal{R}(\overline{\sigma})$ that enable the firing of an event in γ. Intuitively, $M^{\sigma}_{ini}(\gamma)$ is the set of markings reachable through a firing sequence from the initial marking m_i of $\overline{\sigma}$ such that it does not includes any event of γ and the next event that can be fired belongs to γ. Formally, $M^{\sigma}_{ini}(\gamma) = \{m \mid \exists \rho \in E^* : m_i \xrightarrow{\rho}_{\overline{\sigma}} m \wedge \nexists e \in E' : e \in \rho \wedge \exists m' \in \mathcal{R}(\overline{\sigma}) : m \xrightarrow{e_i}_{\overline{\sigma}} m' \wedge e_i \in E'\}$. Similarly, $M^{\sigma}_{end}(\gamma)$ denotes the set of markings

in $\mathcal{R}(\overline{\sigma})$ reached by firing all events in γ. Formally, $M^{\sigma}_{end}(\gamma) = \{m \mid \exists \rho = \langle e_1, \ldots, e_n \rangle \in E^* : m_i \xrightarrow{\rho}_{\overline{\sigma}} m \wedge \forall e \in E' \; e \in \rho \wedge e_n \in E'\}$, i.e. $M^{\sigma}_{end}(\gamma)$ is the set of markings reachable through a firing sequence ρ from the initial marking m_i of $\overline{\sigma}$ that includes all the events of γ with the additional constraint that the last event of ρ belongs to γ.

We now have the machinery necessary to define ordering relations between subgraphs with respect to a p-trace. Let (γ_i, γ_j) be an ordered pair of subgraphs occurring in a p-trace σ.[5] We can derive one of the following ordering relations for (γ_i, γ_j):

Strictly Sequential: γ_j is strictly sequential to γ_i, denoted as $\gamma_i \rightarrow_{sseq} \gamma_j$, iff *(i)* $M^{\sigma}_{end}(\gamma_i) = M^{\sigma}_{ini}(\gamma_j)$ and *(ii)* $|M^{\sigma}_{end}(\gamma_i)| = |M^{\sigma}_{ini}(\gamma_j)| = 1$, where $|X|$ denotes the cardinality of a set X. Intuitively, the strictly sequential relation states that events corresponding to the nodes of subgraph γ_j will occur immediately after all the events corresponding to the nodes of subgraph γ_i have occurred. Specifically, *(i)* imposes that events in γ_j are enabled by the firing of all events in γ_i, thus guaranteeing that the events in γ_j can occur only after all the events in γ_i have occurred; *(ii)* guarantees that the next event that can be observed after the firing of all events in γ_i belongs to γ_j.

Sequential: γ_j is sequential to γ_i, denoted as $\gamma_i \rightarrow_{seq} \gamma_j$, iff *(i)* for every marking $m' \in M^{\sigma}_{ini}(\gamma_j)$ there exists a marking $m \in M^{\sigma}_{end}(\gamma_i)$ such that m' is reachable from m,[6] *(ii)* for every marking $m \in M^{\sigma}_{end}(\gamma_i)$ there exists a marking $m' \in M^{\sigma}_{ini}(\gamma_j)$ such that m' is reachable from m, *(iii)* $M^{\sigma}_{end}(\gamma_i) \cap M^{\sigma}_{ini}(\gamma_j) \neq \emptyset$ and *(iv)* $|M^{\sigma}_{ini}(\gamma_j)| > 1$ or $|M^{\sigma}_{end}(\gamma_i)| > 1$. Intuitively, *(i)* and *(ii)* guarantee that events in γ_j can only occur after all events in γ_i have occurred; *(iii)* guarantees that there exists some state reached by firing all events in γ_i in which events in γ_j are enabled. It is worth noting that *(i)* of the strictly sequential relation implies these three conditions. However, differently from the strictly sequential relation, the sequential relation allows some events (not belonging to any of the two subgraphs) to occur between the events in γ_i and the events in γ_j. This is captured by *(iv)*.

Eventually: γ_j eventually occurs after γ_i, denoted as $\gamma_i \rightarrow_{ev} \gamma_j$, iff *(i)* $M^{\sigma}_{end}(\gamma_i) \cap M^{\sigma}_{ini}(\gamma_j) = \emptyset$ and *(ii)* there exists a marking $m' \in M^{\sigma}_{ini}(\gamma_j)$ that is reachable from a marking $m \in M^{\sigma}_{end}(\gamma_i)$. Intuitively, the eventually relation states that an event in γ_j can only occur after all events in γ_i have occurred and an arbitrary number of events (at least one) must occur between the two subgraphs. In particular, we can observe that *(i)* and *(ii)* guarantee that an event in γ_j can only occur after all events in γ_i have occurred, i.e. for every marking $m' \in M^{\sigma}_{ini}(\gamma_j)$ there exists a marking $m \in M^{\sigma}_{end}(\gamma_i)$ such that m' is reachable from m and for every marking $m \in M^{\sigma}_{end}(\gamma_i)$ there exists a marking $m' \in M^{\sigma}_{ini}(\gamma_j)$ such that m' is reachable from m. (Note that these conditions corresponds to *(i)* and *(ii)* of the sequential relation.) Moreover, the disjointness of $M^{\sigma}_{end}(\gamma_i)$ and $M^{\sigma}_{ini}(\gamma_j)$ guarantees that every firing sequence from a marking $m \in M^{\sigma}_{end}(\gamma_i)$ to a marking $m' \in M^{\sigma}_{ini}(\gamma_j)$ has at least length 1, i.e. at least one event has to occur after all events in γ_i have occurred in order to enable any event in γ_j.

Interleaving: γ_i and γ_j are interleaving, denoted as $\gamma_i \rightarrow_{int} \gamma_j$, iff there exist a marking $m \in M^{\sigma}_{ini}(\gamma_j) \setminus M^{\sigma}_{end}(\gamma_i)$ and a marking $m' \in M^{\sigma}_{end}(\gamma_i)$ such that m' is reachable from m. Intuitively, the interleaving relation captures all cases where an event in γ_j can occur before all the events corresponding to the nodes of the subgraph γ_i have occurred. For instance, interleaving

[5] The term "ordered pair" should not be confused with ordering relation. Here, ordered pair indicates that we assess the ordering relation between the first element and the second element of the ordered pair.

[6] Recall from Section 2 that a marking is reachable by itself with a firing sequence of length 0.

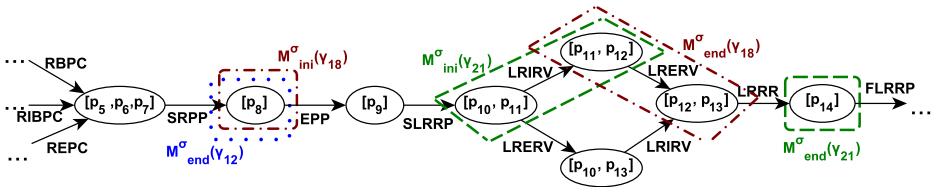


Fig. 9: Portion of the reachability graph for the Petri net in Fig. 8.

relations capture cases where the two subgraphs contain events that can occur concurrently and cases where the subgraphs share some nodes (i.e., the two subgraphs are overlapping [22]).

To derive the relations between the subgraphs in a frequent itemset, we assess the ordering relations between their embeddings pairwise. Specifically, for each ordered pair of embeddings (γ_i, γ_j) of the subgraphs occurring in an itemset, we analyze the reachability from the set of markings resulting from the execution of γ_i (i.e., $M_{end}(\gamma_i)$) to the set of markings enabling the execution of γ_j (i.e., $M_{ini}(\gamma_j)$) in the reachability graph corresponding to (the Petri net representation of) the p-traces in which the itemset occurs. Note that a different type of relation can hold for (γ_i, γ_j) and (γ_j, γ_i). In particular, it is easy to observe that, if the relation for pair (γ_i, γ_j) is *strictly sequential*, *sequential* or *eventually*, the relation for (γ_j, γ_i) is *interleaving*. In such cases, we discard pair (γ_j, γ_i) and, thus, the *interleaving* relation from the analysis as it is not representative of the ordering relations between the two subgraphs. Moreover, it is worth noting that, in the presence of noisy logs (and of multiple embeddings of the same subgraph), we can detect unreliable relations. To deal with this issue, we evaluate the occurrence frequency of ordering relations within the p-traces in which the itemset occurs and only consider those ordering relations whose occurrence frequency is above a given threshold.

As an example, consider the p-trace represented as a Petri net in Fig. 8 and frequent itemset $\{\gamma_{12}, \gamma_{18}, \gamma_{21}\}$. The embeddings of each subgraph are delimited by a dotted (γ_{12}), a dot-dashed (γ_{18}) and a dashed (γ_{21}) line in the figure (as there is only one embedding for each subgraph in the itemset, hereafter we do not distinguish a subgraph from its embeddings.) To define the ordering relations among these subgraphs in the p-trace, we need to analyze the reachability graph of the p-trace. Fig. 9 shows a portion of the reachability graph corresponding to the Petri net in Fig. 8. For the sake of space, we only consider the relevant cases. Let us start with the ordered pair of subgraphs $(\gamma_{12}, \gamma_{18})$. To determine the ordering relation between these two subgraphs, we have to compare the set of marking reached after γ_{12} is executed (i.e., $M^{\sigma}_{end}(\gamma_{12})$) with the set of markings enabling the execution of activities in γ_{18} (i.e., $M^{\sigma}_{ini}(\gamma_{18})$). From Fig. 9 we can observe that $M^{\sigma}_{end}(\gamma_{12}) = \{[p_8]\}$ and $M^{\sigma}_{ini}(\gamma_{18}) = \{[p_8]\}$. Hence, we have $\gamma_{12} \rightarrow_{sseq} \gamma_{18}$. Intuitively, this relation states that the events in γ_{18} are enabled when all events in γ_{12} have occurred. This is also immediate by observing the Petri net in Fig. 8. Note that, since we have identified a strictly sequential relation for the ordered pair $(\gamma_{12}, \gamma_{18})$, we do not need to check the ordered pair $(\gamma_{18}, \gamma_{12})$.

Consider now the ordered pair of subgraphs $(\gamma_{12}, \gamma_{21})$. To determine the ordering relation between these two subgraphs, we have to compare the set of marking reached after γ_{12} is executed (i.e., $M^{\sigma}_{end}(\gamma_{12})$) with the set of markings enabling the execution of the activities in γ_{21} (i.e., $M^{\sigma}_{ini}(\gamma_{21})$). From Fig. 9, we can observe that $M^{\sigma}_{ini}(\gamma_{21}) = \{[p_{10}, p_{11}], [p_{11}, p_{12}]\}$. Since there exists a firing sequence from marking $[p_8]$ to every marking in $M^{\sigma}_{ini}(\gamma_{21})$ and the length of such paths is greater than 1, we derive $\gamma_{12} \rightarrow_{ev} \gamma_{21}$. Note that $M^{\sigma}_{ini}(\gamma_{21})$ includes marking $[p_{10}, p_{11}]$ besides marking $[p_{11}, p_{12}]$. Although this may appear counterintuitive by looking at the Petri net in Fig. 8, in presence of concurrency, it should be taken into account

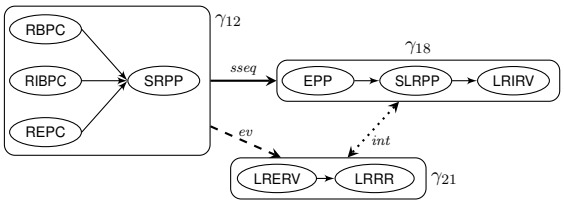


Fig. 10: Partially ordered anomalous subgraph

that activities outside a subgraph can be executed in parallel to the activities of the subgraph. Specifically, we can observe in Fig. 9 that marking $[p_{10}, p_{11}]$ enables the firing of an activity in γ_{21}, i.e. $[p_{10}, p_{11}] \xrightarrow{LRERV} [p_{10}, p_{13}]$ is allowed by the model. We stress that ignoring marking $[p_{10}, p_{11}]$ results in discarding behaviors involving subgraph γ_{21}, which can lead to identify the wrong relation between subgraphs.

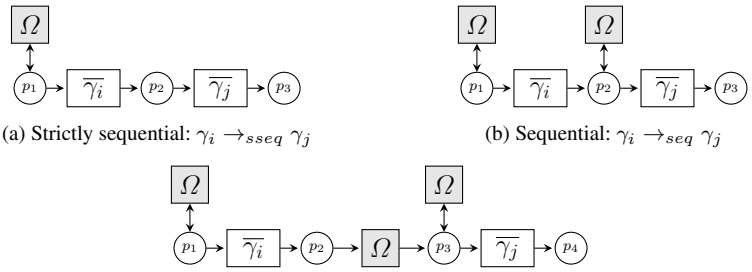
Finally, let us consider the ordered pair $(\gamma_{18}, \gamma_{21})$. To determine the relation between these two subgraphs, we have to consider the relation between the markings in $M^{\sigma}_{end}(\gamma_{18})$ and $M^{\sigma}_{ini}(\gamma_{21})$. As discussed above, $M^{\sigma}_{ini}(\gamma_{21}) = \{[p_{10}, p_{11}], [p_{11}, p_{12}]\}$; moreover, we can observe in Fig. 9 that $M^{\sigma}_{end}(\gamma_{18}) = \{[p_{11}, p_{12}], [p_{12}, p_{13}]\}$. From Fig. 9, it is easy to observe that there exists a firing sequence from a marking in $M^{\sigma}_{ini}(\gamma_{21}) \setminus M^{\sigma}_{end}(\gamma_{18})$ to a marking in $M^{\sigma}_{end}(\gamma_{18})$. Specifically, firing sequence $\rho = \langle LRERV, LRIRV \rangle$ leads from marking $[p_{10}, p_{11}]$ to marking $[p_{12}, p_{13}]$, i.e. $[p_{10}, p_{11}] \xrightarrow{\rho} [p_{12}, p_{13}]$. Therefore, $\gamma_{18} \rightarrow_{int} \gamma_{21}$. Once again, we remark that ordering relations between subgraphs in an itemset should be inferred based on their markings within the p-trace (Fig. 9) rather than on the basis of the configuration of places in the Petri net representing the p-trace (Fig. 8). In fact, by looking at Fig. 8, one may infer a sequential relation for the ordered pair $(\gamma_{18}, \gamma_{21})$. This, however, captures only a particular behavior of the subgraphs within the p-trace. On the other hand, it is evident from the reachability graph in Fig. 9 (which explicitly represents the behavior of the p-trace) that, in general, some activities in γ_{18} and γ_{21} can be executed concurrently and, therefore, the two subgraphs are interleaving.

The identified ordering relations between subgraphs of a frequent itemset can be combined to define the *partially ordered subgraph* for the itemset. Fig. 10 represents the partially ordered anomalous subgraph obtained by combining the ordering relations found for frequent itemset $\{\gamma_{12}, \gamma_{18}, \gamma_{21}\}$. Note that the support of a partially ordered subgraph is always smaller than or equal to the support of the frequent itemset from which it is derived. Moreover, it depends on the support of the ordering relations among these subgraphs. In particular, the support of a set of partially ordered subgraphs is at most equal to the percentage of traces in which the less frequent of its ordering relations holds.

6.2 Pattern Design

Partially ordered subgraphs represent how anomalous subgraphs that frequently occur together are correlated. However, in order to exploit them for improving the process model and/or for conformance checking, partially ordered subgraphs have to be transformed into a more suitable representation. In this section, we present a number of transformation rules to construct *anomalous patterns* from partially ordered subgraphs.

To provide a formal representation of anomalous patterns, we model them as Petri nets. Moreover, we use the notion of Ω-transition introduced in [44] to ensure the applicability and



(a) Strictly sequential: $\gamma_i \rightarrow_{sseq} \gamma_j$

(b) Sequential: $\gamma_i \rightarrow_{seq} \gamma_j$

(c) Eventually: $\gamma_i \rightarrow_{ev} \gamma_j$

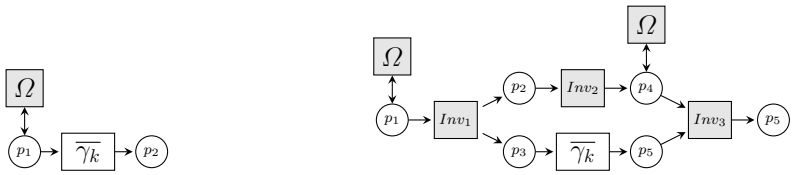
Fig. 11: Anomalous patterns encoding strictly sequential, sequential and eventually relations

reuse of anomalous patterns. Intuitively, Ω-transitions are used to mimic the occurrence of any other transition than the ones expected by the pattern in a given state. This makes it possible to abstract the patterns from events whose occurrence is not representative of the corresponding anomalous behavior.

Fig. 11 represents the anomalous patterns obtained by two anomalous subgraphs, γ_i and γ_j, connected through a strictly sequential, a sequential and an eventually relation where $\overline{\gamma_i}$ and $\overline{\gamma_j}$ denote the Petri net representation of γ_i and γ_j respectively.[7] The pattern encoding the strictly sequential relation (Fig. 11a) shows that the activities in γ_j should occur immediately after the activities in γ_i have been executed in order for the pattern to be successfully executed. Similarly, the pattern encoding the sequential relation (Fig. 11b) is successfully executed if the activities in γ_j occur after the activities in γ_i; however, in this case, other activities may be executed between the activities in γ_i and the activities in γ_j as represented by the Ω-transition connected to place p_2. On the other hand, in order for the pattern encoding the eventually relation (Fig. 11c) to be successfully executed it is required that at least one activity is executed between the activities in γ_i and the activities in γ_j. This is modeled in Fig. 11c by the Ω-transition between places p_2 and p_3 and the Ω-transition connected to place p_3. Note that the initial place p_1 of these patterns is connected to a Ω-transition. This structure is used to model the case where a pattern can occur in an arbitrary position within the p-trace.

The construction of anomalous patterns involving interleaving relations requires more attention. The interleaving relation can represent different situations, e.g. it can indicate that two subgraphs are (partially) concurrent or that they overlap on one or more activities. We abstract from the specific situation at hand by merging the two subgraphs involved in the relation into a new subgraph that encompasses the behaviors represented by both subgraphs and their interconnections. Let $\gamma_i = (E_i, W_i)$ and $\gamma_j = (E_j, W_j)$ be two subgraphs in an interleaving relation and $S = \{\sigma_1 \dots \sigma_n\}$ a set of partially ordered traces, each involving at least one embedding for each subgraph. The subgraph obtained by merging γ_i and γ_j is $\gamma_k = (E_i \cup E_j, W_i \cup W_j \cup \{(e, e') \mid ((e \in W_1 \wedge e' \in W_2) \vee (e' \in W_1 \wedge e \in W_2)) \wedge \forall_{\sigma_m=(E_m,W_m)\in S}((e, e') \in E_m)\})$. Intuitively, the set of nodes of γ_k is obtained by the union of the sets of nodes of γ_i and γ_j. Similarly, the set of edges of γ_k is obtained by the union of the sets of edges of γ_i and γ_j. In addition, the set of edges of γ_k includes the edges that connect nodes of γ_i to nodes of γ_j and vice versa (if any). The existence of these edges can be determined by observing how the embeddings of γ_i and γ_j are related in the p-traces in S.

[7] The Petri net representation of a subgraph can be obtained similarly to the one of a p-trace using Algorithm 4 in Appendix. For the sake of convenience, in Fig. 11 we represent the initial and final place of (the Petri net representation of) a subgraph externally to the rectangle representing the subgraph. For instance, in Fig. 11a, p_1 is the initial place of $\overline{\gamma_i}$, p_2 is the final place of $\overline{\gamma_i}$ and the initial place of $\overline{\gamma_j}$, and p_3 is the final place of $\overline{\gamma_j}$.



(a) Interleaving with no other activities in parallel (b) Interleaving with other activities in parallel

Fig. 12: Anomalous patterns encoding interleaving relations

An interleaving relation, however, does not indicate whether other activities/subgraphs can be executed between the subgraphs involved in the relation or not. To distinguish these cases, we propose two anomalous patterns for interleaving relations. The first pattern (Fig. 12a) aims to model situations in which no other activities occur in parallel to γ_k; the second pattern (Fig. 12b) captures the cases where other activities/subgraphs can be executed concurrently. In the figures, $\overline{\gamma_k}$ denotes the Petri net representation of γ_k. As for the patterns in Fig. 11, both patterns are preceded by a Ω-transition, representing that an arbitrary number of other activities can be executed before the activities in γ_k. However, the pattern in Fig. 12a represents that no other activities/subgraphs can occur concurrently; once an activity in γ_k has been fired, nothing else can be executed until the pattern is completely executed. On the other hand, the pattern in Fig. 12b allows for the concurrent execution of an arbitrary number of other activities before, during and after the execution of γ_k. This is modeled by the Ω-transition in parallel to γ_k. One might argue that the pattern in Fig. 12b is included in the pattern of Fig. 12a. Although this is indeed the case, the pattern in Fig. 12a is more precise and should be preferred when it is known that nothing can occur in parallel with γ_k, for instance, to prevent false positives when the pattern is used for conformance checking. Note that it is possible to automatically select the most suitable choice for the pattern at hand by checking the presence of activities executed in parallel to the activities of the partially ordered subgraphs in the traces in which these subgraphs occur.

The transformation rules presented in Figs. 11 and 12 show how two subgraphs connected by an ordering relation can be combined. However, partially ordered subgraphs can encompass more than two anomalous subgraphs. In this case, the ordering relations between the subgraphs forming a partially ordered subgraph can "interfere" with each other and, thus, user intervention may be required to design the corresponding anomalous pattern. In particular, user intervention may be required to determine in which order the subgraphs should be combined in order to construct the pattern that better fits the process. This is especially true in the presence of interleaving relations, where subgraphs are merged. As an example, consider the partially ordered subgraph in Fig. 10. We can observe in the figure that subgraphs γ_{12} and γ_{21} are connected by the eventually relation, suggesting that the pattern in Fig. 11c could be used. However, subgraph γ_{12} is connected to subgraph γ_{18} through a strictly sequential relation, and subgraphs γ_{18} and γ_{21} are interleaving. This may indicate that the Ω-transition between places p_2 and p_3 in Fig. 11c can be replaced by some events in γ_{18}, depending on the process model. Fig. 13 shows the pattern that can be obtained from the partially ordered subgraph in Fig. 10.[8]

The proposed approach supports an analyst in the exploration of the different patterns obtained by selecting and merging different pairs of subgraphs. In particular, by automatically generating possible alternatives to combine pairs of subgraphs, it is possible to provide a valuable aid to the analyst in selecting the best way of combining the entire set of subgraphs. The

[8] In the anomalous pattern of Fig. 13, invisible transition Inv_1 is introduced by Algorithm 4 (Appendix) when transforming subgraph γ_{12} into a Petri net to capture the concurrency between activities $RBPC$, $RIBPC$ and $REPC$.

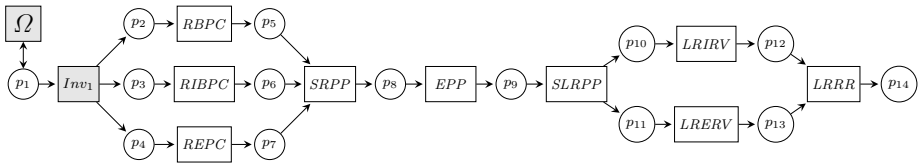


Fig. 13: Anomalous pattern corresponding to the partially ordered subgraph in Fig. 10

definition of a systematic approach for the (semi)automated generation of anomalous patterns from partially ordered subgraphs involving more than two subgraphs, however, requires more investigation. The design of such an approach is left for future work.

7 Experiments

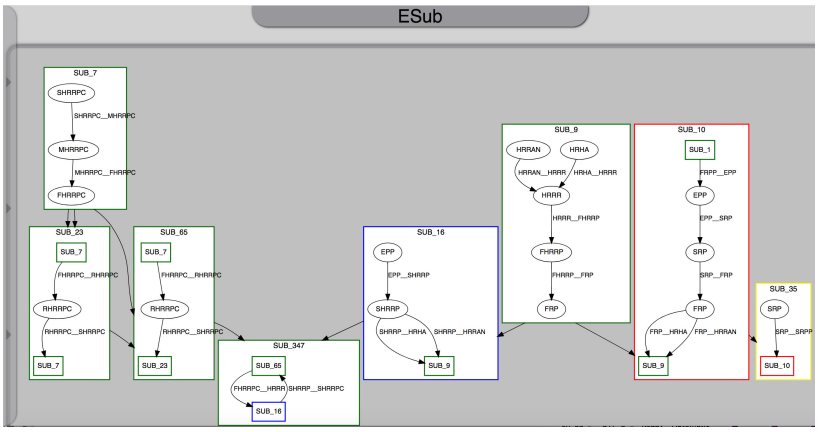
We have implemented our approach as two modules of the Esub tool [16], which extend the *Anomalous Subgraphs Checking* and *Partial Order Discovery* modules presented in [22] to deal with concurrency. The tool can be found at `http://kdmg.dii.univpm.it/?q=content/esub` and a demonstration of the tool in [21]. The first module implements steps 1 and 2 of the approach. It takes as input *(i)* a set of p-traces and *(ii)* the reachability graph of a Petri net, and uses SUBDUE to generate the subgraph hierarchy. Then, the module applies Algorithm 2 to extract anomalous subgraphs. Fig. 14a shows a screenshot of the module displaying a portion of the hierarchical structure derived by SUBDUE, indicating which subgraphs are anomalous (see Section 5 for an explanation of the meaning of the colors). The second module implements the third step of the approach; it takes as input the set of frequent itemsets and derives the partially ordered anomalous subgraphs. A screenshot of this module is shown in Fig. 14b. Each edge is labeled with the type of relation it represents. Solid lines are used for sequential and strictly sequential relations, dashed lines for eventually relations and dotted double-arrowed lines for interleaving relations.

To evaluate the approach, we performed a number of experiments on synthetic datasets and two real-world event logs. The aim of the experiments on synthetic data is to perform controlled experiments and assess the accuracy of the approach. We used real-life case studies to show that the approach provides useful insights and is robust to logs and models with real-life complexity. Both studies are reported in the following subsections.

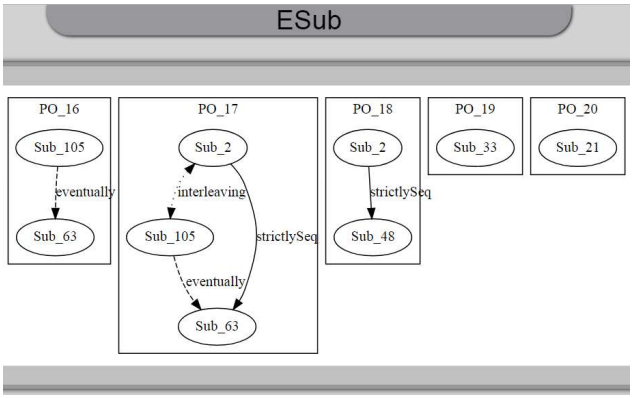
7.1 Synthetic Datasets

Settings For the experiments with synthetic data, we designed the Petri net modeling the bank transaction process in Fig. 1 using CPN tools (`http://cpntools.org/`) and generated two event logs, used in two set of experiments. Each log consists of 2000 p-traces but involves a different number of events, i.e. 39014 events and 39035 events. This difference is due to the different kind of deviations we inserted in each event log to assess the capability of the approach in detecting recurrent anomalous behaviors, as explained below.

In a first set of experiments, we have manipulated the generated p-traces by inserting a number of deviations, namely *swaps*, *repetitions* and *replacements*. A swap occurs when two or more activities are executed in the opposite order compared to the order defined by the model; we swapped the execution of activity $\langle SRPP \rangle$ with the one of $\langle FRPP \rangle$, and the execution of $\langle LRRR \rangle$ with the execution of $\langle FLRRP \rangle$. A repetition means that a given



(a) Anomalous Subgraphs Checking



(b) Partial Order Discovery

Fig. 14: ESub Modules for Anomalous Pattern Extraction.

(sequence of) activity(ies) is repeated multiple times (without belonging to a loop); we added two repetitions, namely the repetition of $REPP$ and of $LRERV$. Finally, a replacement indicates that a given (sequence of) activity(ies) is executed instead of another one; in our experiments, activity $SHRRP$ was replaced with sequence $\langle SRP, FRP \rangle$, and $\langle HRRR, FHRRP, FRP \rangle$ with $\langle EPP, SLRRP \rangle$. For each deviation, we set a probability of occurrence of 30%. By doing so, we obtain an event log involving several varied anomalous behaviors, where it is however possible to recognize some regularities.

To test the robustness of the approach in the presence of noise, in a second experiment we randomly added/removed process activities in portions of the process not involved by the occurrence of the deviations introduced for the previous experiments. The amount of noise added to a trace is equal to the 10% of the length of the trace. By doing so, we obtain an event log significantly affected by random noise. This can be considered an extreme situation as this amount of noise is much higher than what is typically expected in a real-world context.

For the experiments, we filtered out fitting traces as we are interested in the detection of recurrent anomalous behavior. Table 1 summarizes the event logs used for the experiments. The table reports the number of non-fitting traces, the total number of events and the average/minimum/maximum number of events per trace.

Experiment	Non-fitting traces	Events	Avg. events per trace	Min events per trace	Max events per trace
No random noise	1500	30238	20	13	41
With random noise	1995	38963	20	12	41

Table 1: Event logs used for the synthetic experiments

To extract the relevant subgraphs from the logs we used the traditional SUBDUE implementation available at `http://ailab.wsu.edu/subdue/`, in its default configuration.[9] To derive partially ordered subgraphs, we first used FP-Growth [26] to mine all co-occurring subgraphs with a minimum support threshold of 5%. We point out that this can be considered a good support value in our experiments. In fact, the frequency assigned to the deviations, together with the complex structure of the process model involving choice constructs, implies that we cannot expect to detect patterns with a very high support value. This reflects what we reasonable expect to find in a real-world context, where it is unlikely to detect very frequent anomalies (unless the process model is outdated). Ordering relations were inferred using a threshold of 50%.

It is worth noting that partially ordered subgraphs may have a support smaller than the 5% threshold set for the frequent itemsets. The support of a partially ordered subgraph depends both on the support of its itemset (that represents its upper bound) and on the support of the ordering relations connecting the subgraphs in the itemset. For instance, consider an event log consisting of 100 traces and an itemset involving two anomalous subgraphs γ_i, γ_j, whose support is 10%. Let us assume that an analysis of traces in which the itemset occurs shows a sequentially relation between the two subs in 90% of the cases and a strictly sequentially relation in the remaining ones. By setting a threshold for ordering relations equals to 50%, only the sequentially relation is considered in the partially ordered subgraph. It is easy to observe that the support of the resulting partially ordered subgraph is 9%.

To show the advantages of considering concurrency in the discovery of recurrent anomalous behavior, we compared the results obtained by the proposed approach with the results obtained using the approach in [22], that relies on sequential traces for the discovery of recurrent anomalous behavior. For the comparison, we considered the following criteria:

- *Number of mined subgraphs* (**Subgraphs**);
- *Number of anomalous subgraphs* (**Anomalous Subgraphs**);
- *Number and average support of partially ordered subgraphs* (**P.O.**, **Average support P.O.** respectively);
- *Average number of anomalous subgraphs per partially ordered subgraphs* (**Average subgraphs per P.O.**);
- *Average number of activities per partially ordered subgraphs* (**Average activities per P.O.**).

These criteria provide an indication of the quality of diagnostic information that can be obtained using both the approaches. For instance, the average number of anomalous subgraphs forming a partially ordered subgraphs indicates the ability of the approach to correlate anomalous behaviors occurring in (possibly) different portions of a process instance. On the other hand, the average number of activities in a partially ordered subgraph provides an indication of the complexity of the anomalous behaviors that can be captured by the approach.

Results Table 2 shows the results of our experiments with synthetic data. For the first set of experiments (no random noise), we obtained 329 anomalous subgraphs and 25 partially

[9] Note that the traditional SUBDUE implementation takes a single graph as input. Therefore, we generated a graph consisting of several disjoint (sub)graphs, each corresponding to a p-trace of the event log. The resulting graph was given as input to SUBDUE.

Experiment	Trace type	Subgraphs	Anomalous subgraphs	P.O.	Average support P.O.	Average subgraphs per P.O.	Average activities per P. O.
No random noise	p-trace	628	329	25	13.22%	1.48	8.24
	sequential	1945	22	13	11.87%	1	4.76
With random noise	p-trace	1117	485	657	4.85%	3.49	14.02
	sequential	5130	618	30	10.93%	1.46	3.43

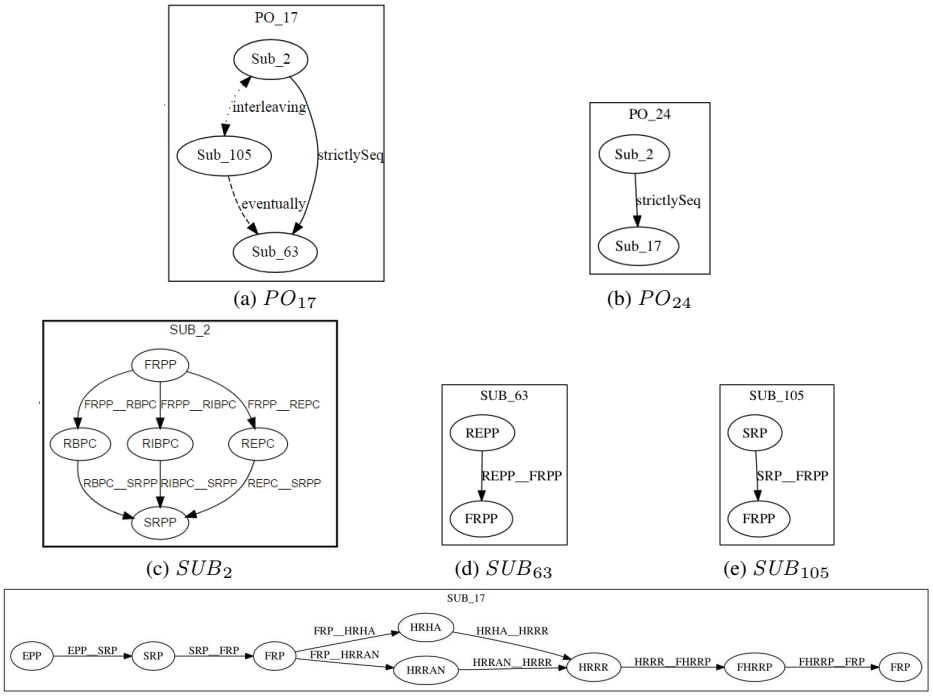
Table 2: Results for synthetic datasets

ordered subgraphs (P.O.) using p-traces. Partially ordered subgraphs have an average support of 13.22%. The analysis of the obtained partially ordered subgraphs shows that all the inserted deviations have been detected by the approach. It is worth noting that although the total number of subgraphs mined from p-traces was significantly lower than the number of subgraphs mined from sequential traces (628 versus 1945), we found much more anomalous subgraphs in the first case (i.e., 329 versus 22). This can be explained by recalling that when considering sequential traces, children of anomalous subgraphs are always anomalous; thus, they are pruned during the conformance checking phase. On the other hand, when concurrency is involved, children of anomalous subgraphs should be analyzed as well, as explained in Section 3.

It is worth noting that we found significantly more complex (and, hence, potentially more interesting) patterns when using p-traces, as shown by the average number of subgraphs and activities per partially ordered subgraphs. In particular, in the noise-free setting we found partially ordered subgraphs involving on average 1.46 and 8.24 activities when using p-traces; while all partially ordered subgraphs mined from sequential traces consist of only one subgraph and involve, on average, 4.76 activities. This is due to the presence of parallel behaviors in the process. In the experiment with the sequential traces, we have multiple sequential instantiations of parallel behaviors, each of them with a support corresponding to a fraction of the support of the behavior they belong to. It is straightforward to see that this affects the detection of correlations between different deviations.

The results obtained from the experiments with random noise confirm the insights we gained from the noise-free ones. We can observe that the insertion of random noise led to the generation of a much larger number of subgraphs. Since noise can be seen as a form of nonconformance, a significantly larger amount of anomalous subgraphs and of partially ordered subgraphs were obtained in these experiments, as shown in Table 2. In particular, the presence of noise led to generate subgraphs that are smaller on average than the ones obtained in the previous experiments. This can be easily observed in Table 2 by computing the ratio of the average number of activities in partially ordered subgraphs over the average number of subgraphs in partially ordered subgraphs. Obtaining smaller subgraphs may indicate that, while in the previous experiments we could mine subgraphs representing entire deviations, in the experiments with random noise the inserted deviations can be scattered among different subgraphs. Therefore, the ability of correlating anomalous behaviors is crucial to reconstruct deviations.

An analysis of the results of the experiments with random noise shows that all the inserted deviations can be detected using both p-traces and sequential traces. However, we obtained more complex and meaningful patterns when using p-traces, both as regards the average number of subgraphs (3.49 versus 1.46) and as regards the average number of activities (14.02 versus 3.43). This difference, as well as the gap between the number of partially ordered subgraphs mined from p-traces and from sequential traces (657 versus 30), is mainly due to the presence of concurrent behaviors, as for the previous experiments. Indeed, 17 of the partially ordered subgraphs mined from sequential traces involve just one subgraph, while the remaining involve at most two subgraphs, with the exception of a single partially ordered subgraph that involves three



(a) PO_{17} (b) PO_{24}

(c) SUB_2 (d) SUB_{63} (e) SUB_{105}

(f) SUB_{17}

Fig. 15: Partially ordered subgraphs PO_{17} and PO_{24} obtained from the experiment on synthetic data along with their subgraphs

subgraphs. This provides a clear evidence of the difficulties encountered in deriving correlations among deviations when process control-flow is not taken into account. It is worth noting that the higher complexity of patterns mined from p-trace also explains the gap arisen between the average support of partially ordered subgraphs mined from p-traces and sequential traces (4.85% versus 10.93%). Indeed, larger a partially ordered subgraph is and less likely is that a high number of its embeddings occur, especially in the noisy setting of our experiments. Indeed, it is reasonable to expect large subgraphs to have lower support than small subgraphs. Recall that in our experiments we introduced random noise in the log by randomly adding/removing process activities in some portions of the process. Therefore, larger a subgraph is, more likely is that the subgraph involves those activities. This decreases the possibilities of finding a high number of embeddings of these subgraphs and, thus, their support is generally low.

A qualitative analysis of the partially ordered subgraphs obtained in the experiments shows that all inserted deviations have been captured using both p-traces and sequential traces (with and without noise). To provide some insights on the output returned by the approach, we discuss some of the partially ordered subgraphs obtained in the experiments without random noise along with the corresponding subgraphs. Note that edges in the subgraphs are associated with a label of the form $headActivity\_tailActivity$, where $headActvity$ represents the source node of the edge and $tailActivity$ represents its target node.

Fig. 15 shows two partially ordered subgraphs obtained using p-traces, namely PO_{17} and PO_{24}. Partially ordered subgraph PO_{17} (Fig. 15a) represents one of the deviations that were inserted in the manipulated log, namely the swap of activities $SRPP$ and $FRPP$. It consists of

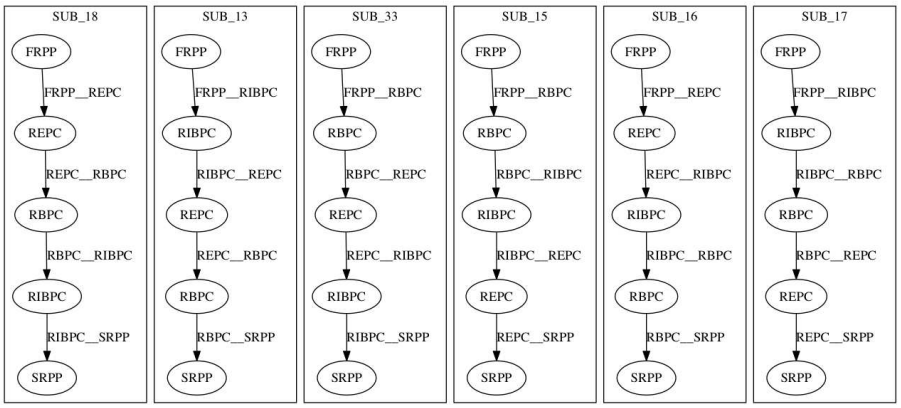


Fig. 16: Subgraphs SUB_{13}, SUB_{15}, SUB_{16}, SUB_{17}, SUB_{18}, SUB_{33}, corresponding to partially ordered subgraphs PO_4, PO_6, PO_5, PO_7, PO_8, PO_9 obtained from the experiments with sequential traces along with its subgraphs

three anomalous subgraphs, i.e. SUB_2 (Fig. 15c), SUB_{63} (Fig. 15d) and SUB_{105} (Fig. 15e). The ordering relations shown in the partially ordered subgraph can be easily justified taking into account the process model and the inserted deviations. In particular, SUB_2 and SUB_{105} share one node, i.e. $FRPP$, which motivates the interleaving relation; SUB_{63} starts immediately after SUB_2, therefore they are in a strictly sequential relation; finally, between SUB_{105} and SUB_{63} there is an eventually relation, since SUB_{63} occurs after SUB_{105} but not immediately after. Partially ordered subgraph PO_{24} (Fig. 15b) captures the replacement of activity $SHRPP$ with the sequence of activities $\langle SRP, FRP \rangle$. It consists of two anomalous subgraphs, i.e. SUB_2 (discussed above) and SUB_{17} (Fig. 15f), connected by a strictly sequential relation. These partially ordered subgraphs provide two significant examples of the capability of the approach in deriving anomalous subgraphs and in correlating them, thus providing a valuable support for the investigation of recurrent anomalous behaviors.

Although we were able to detect all the inserted deviations also using sequential traces, analysis of the partially ordered subgraphs shows that, as expected, several partially ordered subgraphs correspond to different sequential instantiations of the same parallel behavior. This significantly affects the usefulness of the obtained diagnostic information. First, it poses some challenges for the human analyst in interpreting the results. Consider, for instance, the partially ordered subgraphs in Fig. 16. These partially ordered subgraphs consist of only one anomalous subgraph representing a possible sequential instantiation of the swapping of activities $SRPP$ and $FRPP$. It is straightforward to observe that this outcome is much less expressive and intuitive than the one obtained using p-traces and does not provide any insight regarding the actual control flow. Indeed, the analyst has to manually explore and compare the subgraphs to derive that they actually represent the same behavior. This is far from trivial even in this simple setting and when addressing complex, real-life processes can easily become a time-consuming, error prone task.

Another relevant drawback of neglecting concurrency is that it can easily lead to loose relevant information. The support of each sequential instantiation corresponds to a fraction of the support of the concurrent behavior. For instance, the support of the behavior showing the swapping of activities $SRPP$ and $FRPP$ is 56.4%; while the support of the subgraphs in Fig. 16 ranges from 9.5% to 12%. This can lead to miss some or all the instantiations of a parallel behavior, especially when its support is close to the threshold set for mining the itemsets.

Furthermore, it makes it more challenging to detect correlated anomalous behaviors. A clear example of this issue is provided by the experiments without random noise on sequential traces; as shown in Table 2, all the partially ordered subgraphs mined in these experiments involve just one subgraph. Therefore, no correlation among deviations was detected; while in the experiment without random noise using p-traces, we were able to find several interesting correlations.

Finally, we would like to point out that, although the outcome obtained using sequential traces in presence of random noise looks simpler (and, hence, preferable for a human analyst) than the outcome obtained using p-traces, since it involves much less patterns, a deeper look to the obtained results highlighted the same drawbacks we discussed for the previous set of experiments, which are made even worse by the presence of random noise. In fact, although we were able to obtain some correlations, actually these correlations involve subgraphs representing portions of the same deviation, which explains why they frequently occur together. It is straightforward to see that these correlations are trivial and, thus, do not provide useful insights. Moreover, it is worth noting that in this experiment we also miss some of the sequential instantiation of parallel behaviors, which prevents the correct reconstruction of these behaviors. Although the presence of noise also affected partially ordered subgraphs mined from p-traces, leading to deviations spread within several subgraphs, we were anyway able to detect correlations involving different deviations; moreover, the partially ordered subgraphs obtained from p-traces properly reconstructed concurrent behaviors.

7.2 Real-life Dataset: BPI 2012 Challenge

Settings To evaluate the applicability of our approach to real-life settings, we used the event log recording the loan management process of a Dutch Financial Institute, which was made available for the 2012 BPI challenge [12]. The event log contains the events recorded for three intertwined subprocesses: subprocess A specifies how loan applications should be handled, subprocess O describes how loan offers should be handled, and subprocess W specifies how work items are processed.

To reduce the overall complexity of the event log and improve the results of process discovery, we preprocessed the event log based on the findings of other researchers [1, 11]. In particular, artificial start and end events were added to process instances, only events representing the completion of an activity were considered in the analysis, and potential redundant events were removed from the event log [11]. To ensure that process loans are completed, we filtered out all process instances staring in 2012 [1]. After preprocessing, the event log contains 85,426 events in 7,455 event traces. We used inductive miner [38] to discover a process model for this event log, shown in Fig. 17.

The log provided for the BPI challenge 2012 consists of sequential traces. We used Building Instance Graphs (BIG) algorithm [17] to construct p-traces from the sequential traces recorded in the event log. BIG algorithm exploits information encoded in a process model (i.e., causal relations between process activities) to construct p-traces. We applied our approach to the obtained process model and p-traces. Two different support thresholds were used to find frequent itemsets (i.e., 5% and 3%) and a threshold of 50% was used to derive ordering relations between subgraphs in a frequent itemset.

Results Table 3 shows the results of our experiments. In total, 292 subgraphs and 96 anomalous subgraphs were obtained. Using a threshold of 5% for frequent itemsets, 36 partially ordered subgraphs with average support of 6.15% were obtained. By decreasing this threshold to 3%, 81 partially ordered subgraphs with average support of 3.49% were obtained. To provide a concrete

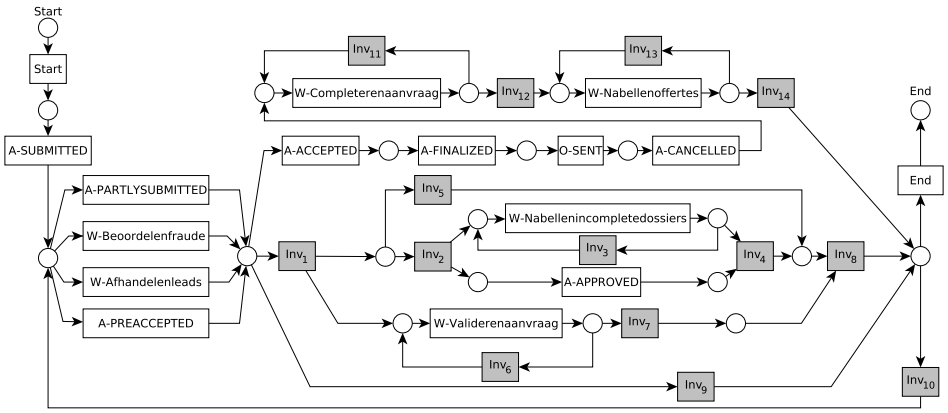


Fig. 17: The process model mined from BPI challenge 2012 event log using inductive miner.

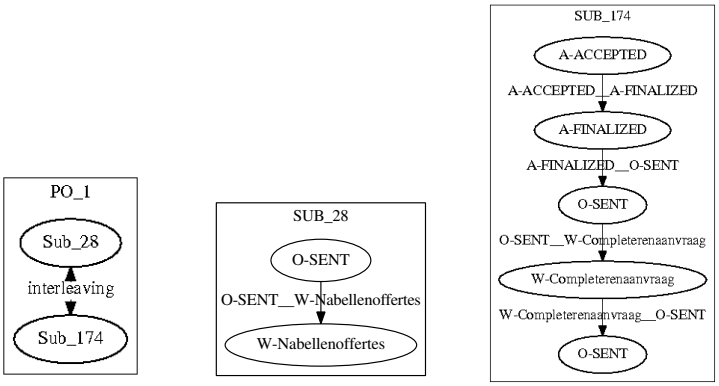
Experiment	Experiment setting (Ordering relation threshold, Frequent itemset threshold)	Subgraphs	Anomalous subgraphs	P.O.	Average support P.O.	Average subgraphs per P.O.	Average activities per P. O.
BPI2012-Exp_1	(50%, 5%)	292	96	36	6.15%	1.13	5.52
BPI2012-Exp_2	(50%, 3%)			81	3.49%	1.56	7.5

Table 3: Results of experiments on BPI Challenge 2012 event log

example of the outcome of the approach and illustrate its capability, next we discuss in detail one of the discovered partially ordered subgraphs of each experiment, namely PO_1 obtained from Exp_1 and PO_2 obtained from Exp_2.

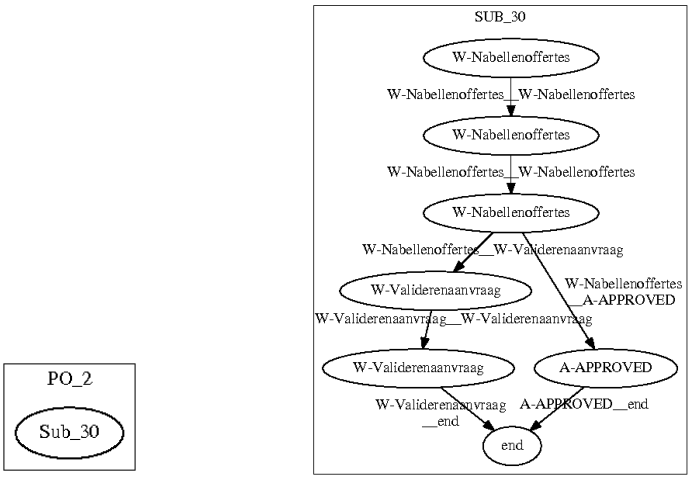
Fig. 18 shows PO_1 along with its subgraphs. As shown in the figure, PO_1 consists of two anomalous subgraphs, SUB_{28} and SUB_{174}, connected through an interleaving relation. This relation is due to the fact that these two subgraphs share the same instance of activity *O-Sent* and, thus, they overlap. Subgraph SUB_{174} shows that, after the offer was sent (*O-Sent*), activity *A-Canceled* was skipped. In addition, subgraph SUB_{174} shows that, after filling in information for the application (*W-Completerenaanvraag*), another offer was sent to the client, which is not allowed by the process model in Fig. 17. Subgraph SUB_{28} shows that, after sending the offer, the client was contacted for obtaining more information. However, according to the process model in Fig. 17, activity *W-Completerenaanvraag* should be executed before contacting the client. Looking at the behavior represented by PO_1 as a whole, we can observed that the offer (or a new offer) has been resent to the client after the information has been filled in for the application (*W-Completerenaanvraag*) and, then the client is contacted as prescribed by the model. We argue that further investigation should be performed to understand why multiple offers were sent to clients as well as why activity *A-Canceled* was skipped. It worth noting that some of the obtained partially ordered subgraphs (not reported here) also show that activity *A-Canceled* was often swapped with activity *W-Completerenaanvraag*.

It is worth mentioning that we did not discovered any partially ordered subgraph exhibiting parallel behavior with the setting of Exp_1 (i.e., a threshold of 5% for frequent itemset and a threshold of 50% for ordering relations). This is due to the fact that concurrent behaviors have a very low support in the BPI 2012 challenge event log. By decreasing the threshold used for frequent itemset to 3%, we were able to obtain partially ordered subgraphs containing these subgraphs. Fig. 19a shows one example of such partially ordered subgraphs. In particular, PO_2



(a) PO_1 (b) SUB_{28} (c) SUB_{174}

Fig. 18: Partially ordered subgraph PO_1 obtained from the experiment on real-life data.



(a) PO_2 (b) SUB_{30}

Fig. 19: Partially ordered subgraph PO_2 obtained from the experiment on real-life data.

consists of one anomalous subgraph, namely SUB_{30} (Fig. 19b). This subgraph shows that the activities for assessing (*W-Validerenaanvraag*) and approving (*A-Approved*) the application were executed immediately after calling the client (*W-Nabellenoffertes*). In addition, it shows that, during the assessment phase (*W-Validerenaanvraag*) for approving the request (*A-Approved*), the client was not contacted for obtaining missing information (*W-Nabellenincompletedossiers*). These cases should be investigated and, if they are recognized as normal behavior, the process model should be repaired to reflect this behavior.

The results confirm that the patterns obtained from the proposed approach can highlight frequent and correlated anomalous behaviors. By analyzing the obtained patterns, the process model might be repaired to represent the actual expected behavior. For example, the process model can be extended in order to allow skipping of activity *A-Canceled* or swapping it with activity *W-Completerenaanvraag*. The analyst may also decide to investigate anomalous

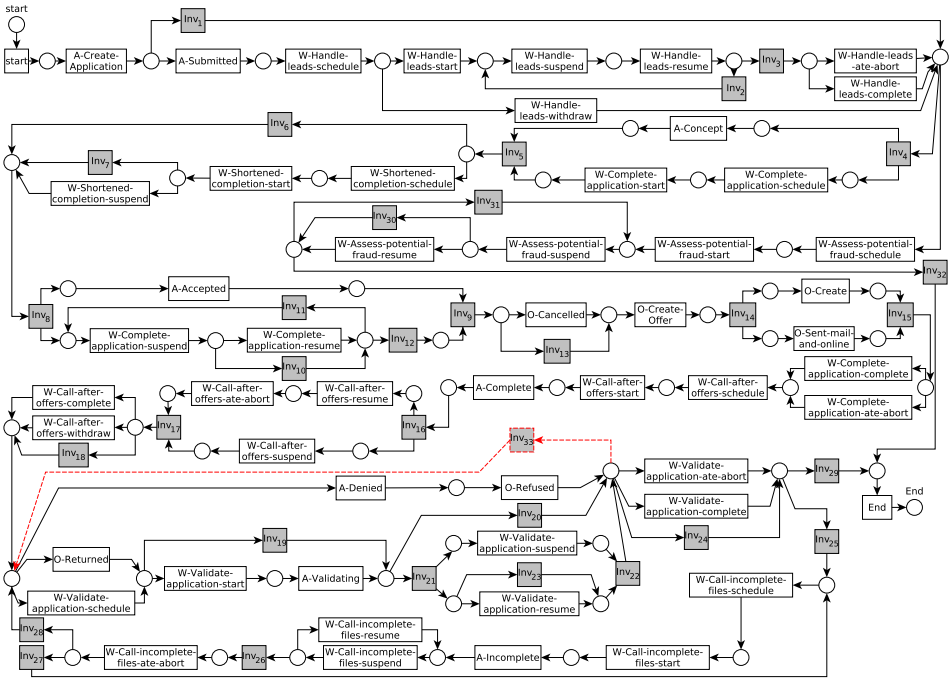


Fig. 20: The process model mined from BPI challenge 2017 event log using inductive miner.

behaviors to ensure that they conform to the defined guidelines. For example, sending multiple offers (*O-Sent*) to a client should be investigated to ensure that it is a justified exception. If this behavior corresponds to errors, actions should be taken to prevent future occurrence of them.

7.3 Real-life Dataset: BPI 2017 Challenge

Settings We also evaluated our approach using the event log that was made available for the 2017 BPI challenge [13]. This log records the executions of a process for handling credit requests within a financial company. The event log contains detailed information about applications submitted by clients, loan offers sent by the company, and work items processed by employees or by the system. At the end of the process, the submitted applications can be approved or declined by the company, or be canceled by the client.

To obtain a reliable process model, we preprocessed the event log. In particular, artificial start and end events were added to process instances. As it appears that process instances starting after October 2016 are not likely to be completed, we filtered them out. To avoid obtaining a spaghetti-like process model, we partition the event log into homogeneous subsets of process instances. In the experiments, we only focus on the analysis of applications that were denied. After preprocessing, the event log contains 124,866 events in 3,093 event traces. We used inductive miner [38] to discover a process model for this event log; the discovered process is reported in Fig. 20.

The log provided for the BPI challenge 2017 consists of sequential traces. As we did for the 2012 BPI challenge log, we used the BIG algorithm to construct p-traces from the sequential

Experiment	Experiment setting (Ordering relation threshold, Frequent itemset threshold)	Subgraphs	Anomalous subgraphs	P.O.	Average support P.O.	Average subgraphs per P.O.	Average activities per P. O.
BPI2017-Exp_1	(50%, 5%)	670	39	324	7.98%	3.40	21.48
BPI2017-Exp_2	(50%, 3%)			651	6.23%	3.47	22.70

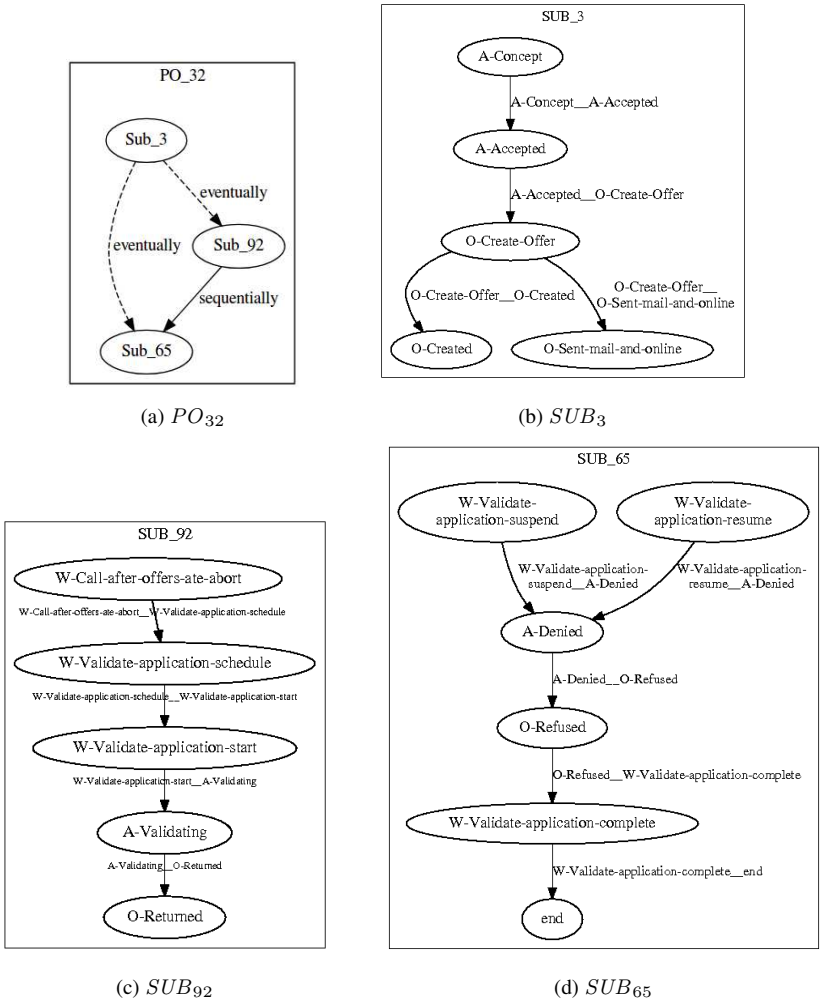
Table 4: Results of experiments on BPI Challenge 2017 event log.

traces recorded in the event log. We applied our approach to the obtained process model and p-traces using the same settings as described in Section 7.2.

Results Table 4 shows the results of our experiments. In total, 670 subgraphs and 39 anomalous subgraphs were obtained. Using a threshold of 5% for frequent itemsets, 324 partially ordered subgraphs with average support of 7.98% were obtained. We also performed an experiment in which the threshold for frequent itemsets was set to 3%. In this case, we obtained 651 partially ordered subgraphs with average support of 6.23%. In these experiments, the constructed anomalous subgraphs are highly correlated. Thus, in comparison to the results presented in Section 7.2, the obtained partially ordered subgraphs on average contain a larger number of anomalous subgraphs.

Differently from the experiments on the BPI 2012 challenge log, we discovered a number of partially ordered subgraphs exhibiting concurrency when the threshold for frequent itemsets was set to 5%. Next, we discuss in detail two of these partially ordered subgraphs, namely PO_{32} and PO_{140}, to provide a concrete example of the outcome of the approach and illustrate its capabilities.

Fig. 21 shows PO_{32} along with its subgraphs, namely SUB_3, SUB_{65} and SUB_{92}. SUB_3 is connected to SUB_{65} and SUB_{92} through eventually relations and SUB_{92} is connected to SUB_{65} through a sequential relation. SUB_3 shows that *W-Complete-application-suspend* was skipped, which is not allowed by the process model. In general, when an activity is started, it might be suspended (temporarily halted) and resumed again but these steps in the life-cycle of activities are not always necessary. SUB_{92} shows that *W-Validate-application-schedule* and *O-Returned* can be executed together and offers might be returned by clients while applications are being validated. However, according to the model, either *W-Validate-application-schedule* or *O-Returned* should be executed and the application should be validated only after an offer is returned. According to the transnational life-cycle model [54], an activity should be scheduled before it can start. Thus, *W-Validate-application-schedule* should always be executed before the execution of *W-Validate-application-start*. Moreover, we observed that in 2789 cases *O-Returned* was executed immediately after the execution of activities *W-Validate-application-start* and *A-Validating*. This means that the preliminary validation of an application (e.g., checking the submitted documents) started before an offer is returned, indicating a swapping between these two activities. SUB_{65} shows that some applications were denied (*A-Denied*) and offers were refused (*O-Refused*) immediately after the validation of applications is resumed (*W-Validate-application-resume*) or suspended (*W-validate-application-suspended*), which is not allowed by the process model. An application can be denied at different states of a process execution if it does not fit acceptance criteria. For example, along with an offer other documents such as payslips and bank statements might be required to be provided by a client. If the income is not sufficient, the application is denied even though the client returned the offer. This indicates that applications could be potentially denied also during the validation phase. Partially ordered subgraph PO_{32} shows that these behaviors can frequently occur together. This larger view of the deviant behavior shows that after an offer is returned the validation process can be suspended and resumed, and the application denied and the corresponding offer refused. As shown in the



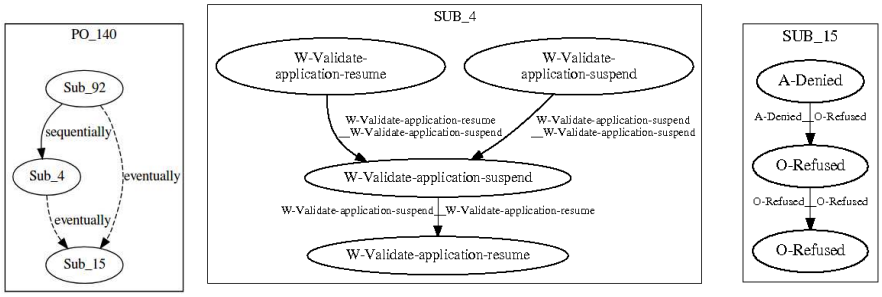
(a) PO_{32} (b) SUB_3

(c) SUB_{92} (d) SUB_{65}

Fig. 21: Partially ordered subgraph PO_{32} obtained from the experiment on BPI Challenge 2017 event log.

process model of Fig. 20, these activities belong to different branches of a choice construct. This suggests that the validation process and refusal of offers can be reiterated as represented by the (red) loop in Fig. 20.

Fig. 22 shows PO_{140} along with two of its subgraphs, namely SUB_4 and SUB_{15}. SUB_{92} is connected to SUB_4 through a sequentially relation and SUB_{92} and SUB_4 are connected to SUB_{15} through eventually relations. As discussed before, SUB_{92} (see Fig. 21c) indicates that offers might be returned by clients while applications are being validated. SUB_4 shows that the validation phase can be suspended and resumed again, while the process model does not allow it. According to the standard transnational life-cycle model [54], this behavior should be allowed. Thus, the process model should be revised to reflect this finding. SUB_{15} shows that *O-Refused* was performed multiple times, while according to the process model it must be performed once.



(a) PO_{140} (b) SUB_4 (c) SUB_{15}

Fig. 22: Partially ordered subgraph PO_{140} obtained from the experiment on BPI Challenge 2017 event log.

During the handling of a credit request, multiple offers might be sent to the client. In case the application is denied, all active offers should be refused.

The results confirm that the patterns obtained from the proposed approach can highlight frequent and correlated anomalous behaviors. In particular, patterns provide more accurate diagnostics and a better understanding of deviant behaviors. Based on the findings obtained from the analysis of the mined patterns, the process model might be repaired to better reflect the reality. For example, the process model can be extended in order to allow repetition of *O-Refused*, swapping of *O-Returned* with *W-Validate-application-start* and *A-Validating*, and the execution of *A-Denied* and *O-Refused* during the validation phase.

8 Related Work

This work embraces two main research areas, namely subgraph extraction and conformance checking. In this section, we discuss recent developments in these areas.

Subgraph Extraction Several approaches for subgraph extraction have been proposed in the area of business processes. Bose and van der Aalst [10] detect subprocesses by identifying sequences of events that fit a-priori defined templates; Hwang et al. [29] exploit a sequence pattern mining algorithm to derive frequent sequences of clinical activities from clinical logs; Leemans et al. [37] introduce an approach to derive "episodes", i.e. directed graphs where nodes correspond to activities and edges to *eventually-follow* precedence relations, which, given a pair of activities, state which one occurs later. Compared to these approaches, our work does not require defining any predefined template and extracts the subprocesses that are the most relevant according to their description length, thus taking into account both frequency and size in determining the relevance of subprocesses.

Other approaches aim to convert traces into directed graphs representing execution flows and, then, apply frequent subgraph mining techniques to derive the most relevant subgraphs. For instance, Hwang et al. [30] generate "temporal graphs", where two nodes are linked only if the corresponding activities have to be executed sequentially. The applicability of this approach, however, is limited to event logs storing starting and completion time of events. Greco et al. [23] propose an FSM algorithm that exploits knowledge about relationships among activities (e.g., AND/OR split) to drive subgraphs mining. Graphs are generated by replaying traces over

the process model; however, this algorithm requires a model properly representing the event log, which may not be available for many real-world processes. In contrast, our approach for subprocess extraction does not require neither the presence of special attributes in the event log nor a-priori models of the process or other domain knowledge. Moreover, Greco and colleagues exploit the relations shown in the process model to speed up the candidates generation step in the subgraph mining algorithm. However, this strategy is not suited for our purposes and might actually hinder the extraction of subgraphs involving deviations, since these subgraphs are not shown in the model. Greco and colleagues have extended their approach in [24] to mine both connected and unconnected subgraphs from workflow executions. They implement an a-priori algorithm to determine, among the set of frequent connected subgraphs mined with the approach proposed in [23], the ones that co-occur with a frequency above a given threshold. However, the authors do not explore ordering relations among the set of co-occurring subgraphs. Moreover, also in this case, they exploit knowledge from the model to speed up the search, which, as mentioned before, is not an effective strategy when dealing with anomalous subgraphs.

It is worth noting that once traces have been converted into directed graphs, one can apply any frequent subgraph mining technique to derive the subgraphs of interest. Subgraph mining is an active research area; a plethora of FSM approaches have been proposed in literature during the last years (see, e.g., [32] for a survey). Most of well-known techniques are *complete* approaches (e.g., AGM [31] and FSG [34] algorithms), namely they aim at deriving the entire set of subgraphs that fit user-defined requirements (typically, minimum support). Nevertheless, some heuristics techniques have been developed as well, like the SUBDUE algorithm, used in this work, or the GREW algorithm [35]. Heuristic techniques sacrifice completeness for efficiency. Namely, they do not aim to return the complete set of subgraphs, but only of those considered the most relevant according to a given metric. It is worth noting, however, that to the best of our knowledge, only SUBDUE exploits the description length metric to infer the most relevant subgraphs. Moreover, SUBDUE also supports the exploration of inclusion relationships between subgraphs by returning a hierarchy in which subgraphs are arranged according to these relationships. The use of taxonomies for the analysis of process behaviors has also been proposed by other process discovery techniques that aim to extract from a set of process executions a process model able to represent the underlying process [56]. In this context, taxonomies are used to enable an exploration of the process on different levels of abstractions. For instance, Bose and van der Aalst [9] propose to replace the subprocesses inferred with the technique proposed in [10] with single activities and, then, to mine a process model in which most frequent subprocesses are represented by a single node. This procedure can be repeated until the desired level of abstraction is reached. At the end, a taxonomy of process models is obtained, where the model at each level represents the process with a degree of abstraction higher than the models at lower levels. Mannhardt and Tax [41] adopt a similar idea, using local process models discovered by the technique proposed in [48] to determine the set of activities to replace. A different approach is proposed in [25], which iteratively refines the models at the current level of the taxonomy by properly clustering traces that fit such models and, then, derives from the clusters new models representing a portion of the behavior of the parent model with a higher level of detail than the latter. The taxonomies obtained using the aforementioned approaches differ from the one obtained using SUBDUE, which have been exploited in our work, both by construction and purpose. Indeed, our approach exploits SUBDUE hierarchy simply as a means to derive subgraphs inclusion relations, rather than to provide users with a multi-level process exploration tool.

Conformance Checking Conformance checking aims to verify whether the observed behavior recorded in an event log matches the intended behavior represented as a process model. Several efforts have been devoted in the last years to checking conformance and a number of techniques

have been proposed to verify the conformity of event logs with process specifications. Some approaches [8, 14, 47] propose to check whether event traces satisfy a set of compliance rules. Rozinat and van der Aalst [45] propose a token-based technique to replay event traces over a process model and use the information obtained from remaining and missing tokens to detect deviations. However, it has been shown that token-based techniques can provide misleading diagnostics. Recently, alignments have been proposed as a robust approach to conformance checking [55]. Alignments are able to pinpoint deviations causing nonconformity based on a given cost function. These cost functions, however, are usually based on human judgment and, hence, prone to imperfections, which can ultimately lead to incorrect diagnostics. To obtain probable explanations of nonconformity, Alizadeh et al. [5] propose an approach to compute the cost function by analyzing historical logging data, which is extended in [6] to consider multiple process perspectives. However, these techniques are only able to deal with sequential event traces (i.e., total ordered traces); thus, diagnostics can be unreliable when timestamps of events are coarse or incorrect.

A number of graph-based approaches for anomaly detection have been proposed to deal with concurrency in process executions (see [4] for a survey). For instance, Eberle et al. [18] present graph-based anomaly detection (GBAD). GBAD comprises three anomaly detection algorithms, each of them tailored to discover a certain type of anomaly, namely insertions, modifications and deletions. Similarly to our work, GBAD relies on SUBDUE to discover frequent subgraphs in a given graph; however, in GBAD the identified subgraphs are treated as normative patterns and are used as the baseline for the detection of anomalies. In contrast, our approach uses the subgraphs mined using SUBDUE as a representation of the frequent behaviors observed by the system whose compliance is assessed against a process model. Lu et al. [39] propose an approach to detect deviations in an event log by identifying frequent common behavior and uncommon behavior. In particular, this work computes mappings between events, thus highlighting similar and dissimilar behavior between process instances. Similarly to graph-based approaches, the work in [39] assumes that deviations are uncommon behavior; however, it is limited to deviations corresponding to insertion, i.e. activities that have been executed but are not allowed in the normative process. Process instances are fused in a representative execution graph based on the similarity of events. This graph, however, can provide misleading diagnostics about deviations as it is not able to properly represent correlated anomalous behaviors. In contrast, our approach builds anomalous patterns that account for behaviors that frequently happen together.

Most of the existing techniques for conformance checking, as the ones discussed above, are only able to detect simple anomalous behaviors like insertions, modifications and deletions of activities. Only a few conformance checking techniques attempt to discover more complex anomalies. For instance, Banescu et al. [7] use a token-based approach for checking conformance of an event log and a process model and classify anomalies by analyzing the configuration of missing and added tokens using deviation patterns. Adriansyah et al. [3] define anomalous patterns representing swappings and replacements of activities. These patterns are combined with the process model, and alignment-based techniques are used to discover their occurrence in the log. However, these approaches are only able to detect complex anomalies based on predefined anomalous patterns. In contrast, the approach proposed in this work does not rely on predefined anomalous patterns. Rather, our work aims to extract anomalous patterns representing frequent anomalous behavior from historical logging data.

In general, conformance checking techniques, including graph-based approaches for anomaly detection, differ from our approach in that they usually aim to detect particular instances of anomalous behavior whereas our approach aim to discover and analyze recurrent anomalous behaviors of arbitrary complexity. To the best of our knowledge, this is the first work that provides a systematic solution for the discovery and analysis of recurrent anomalous behaviors in the context of business process compliance.

It is worth noting that recently decomposition approaches have been introduced to reduce the computational complexity of conformance checking techniques [15,53]. These techniques propose to split a process model (typically represented as a Petri net) in a set of subprocesses from which it is possible to reconstruct the original model. The event log is then split in a set of sublogs, each of them obtained by projecting the original log on the set of activities related to one of the subprocesses. Conformance checking is applied to pairs of sublog and subprocesses, which allows a significant reduction of the computational complexity. Although in principle this approach might be applied in combination of our approach to speed up the extraction of subgraphs and the conformance checking step, it likely leads to some information loss. In fact, if subgraphs are extracted from sublogs, subgraphs involving activities belonging to different subprocesses cannot be captured.

9 Conclusions and Future Work

In this work, we have presented an extension to our previous approach for the discovery of anomalous patterns from historical logging data. In particular, the obtained patterns represent recurrent high-level deviations in process executions, which can be spanned across different (and not connected) portions of the process. The main novelty of the extended approach consists in the capability of dealing with concurrency, thus providing patterns that better reflects the control flow of processes. In particular, taking into account possible parallelisms enables a more accurate diagnosis of anomalous behaviors and provides analysts with a more comprehensive and compact representation of deviations.

The extended approach extracts relevant subgraphs from partially ordered traces, explicitly modeling possible parallelisms, rather than from totally ordered traces that are typically used by classic conformance checking techniques. This allows us to obtain relevant subgraphs that better reflect the control flow of processes. To identify anomalous subgraphs, we have proposed a novel conformance checking algorithm tailored to check the conformance of partially ordered subtraces exhibiting concurrent behavior. Moreover, we have investigated and formalized ordering relations between subgraphs exhibiting concurrent behavior with respect to a process model. Based on the identified relations, we have shown how to infer partially ordered subgraphs exhibiting correlations among recurrent anomalous behaviors. Finally, we have developed a set of guidelines to transform partially ordered subgraphs into anomalous patterns expressed as Petri nets.

The approach has been implemented as a plug-in of the Esub tool and has been validated using both synthetic and real-life logs. The experiments demonstrated the capability of the approach to return meaningful patterns capturing high-level deviations that, on the other hand, would have been difficult to identify without accounting for concurrency.

The anomalous patterns obtained using our approach can be exploited for several purposes. Among possible applications, we are investigating their application for online monitoring. This would allow for early detection of occurrences of recurring anomalous behaviors, for which accurate diagnostics are already available. Moreover, we are investigating how anomalous patterns can be exploited to guide the definition of measures for preventing and/or responding to anomalous behaviors, especially in the security context. In this work, we have provided an approach for the design of anomalous patterns from partially ordered subgraphs. However, additional support should be provided to analysts for the design of anomalous patterns. To this end, we are investigating approaches for the (semi)automated generation of anomalous patterns from partially ordered subgraphs. We also plan to investigate the use of decomposing techniques [15,53] in combination to our approach. In particular, investigation is required to understand to what extent these techniques affect the mining of anomalous patterns and the

trade-off between information loss and efficiency gain. Finally, in our study we observed that logs often record process executions as sequential traces of events. Although there exist several techniques for the generation of p-traces from sequential traces, the adopted technique can influence the (anomalous) behaviors that can be extracted using our approach and, thus, the patterns that can be obtained. An interesting direction for future work is the study of these techniques and, in particular, their impact on the quality of the patterns extracted using the proposed approach.

Acknowledgment This work has been partially funded by the NWO CyberSecurity programme under the PriCE project and by the ITEA2 project M2MGrid (No. 13011).

References

1. A. Adriansyah and J. M. Buijs. Mining Process Performance from Event Logs: The BPI Challenge 2012 Case Study. BPM Center Report BPM-12-15, BPMcenter.org, 2012.
2. A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Proceedings of IEEE International Enterprise Distributed Object Computing Conference*, pages 55–64. IEEE, 2011.
3. A. Adriansyah, B. F. van Dongen, and N. Zannone. Controlling break-the-glass through alignment. In *Proceedings of International Conference on Social Computing*, pages 606–611. IEEE, 2013.
4. L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: A survey. *Data Min. Knowl. Discov.*, 29(3):626–688, 2015.
5. M. Alizadeh, M. de Leoni, and N. Zannone. History-based construction of alignments for conformance checking: Formalization and implementation. In *Data-Driven Process Discovery and Analysis*, LNBIP 237, pages 58–78. Springer, 2014.
6. M. Alizadeh, M. de Leoni, and N. Zannone. Constructing probable explanations of nonconformity: A data-aware and history-based approach. In *Proceedings of Symposium Series on Computational Intelligence*, pages 1358–1365. IEEE, 2015.
7. S. Banescu, M. Petkovic, and N. Zannone. Measuring privacy compliance using fitness metrics. In *Business Process Management*, LNCS 7481, pages 114–119. Springer, 2012.
8. D. Borrego and I. Barba. Conformance checking and diagnosis for declarative business process models in data-aware scenarios. *Expert Syst. Appl.*, 41(11):5340–5352, 2014.
9. R. P. J. C. Bose, V. HMW, and W. M. P. van der Aalst. Discovering Hierarchical Process Models Using ProM. In *CAiSE Forum*, volume 107, pages 33–48. Springer, 2011.
10. R. P. J. C. Bose and W. M. P. van der Aalst. Abstractions in process mining: A taxonomy of patterns. In *Business Process Management*, LNCS 5701, pages 159–175. Springer, 2009.
11. R. P. J. C. Bose and W. M. P. van der Aalst. Process mining applied to the BPI Challenge 2012: divide and conquer while discerning resources. In *Business Process Management*, pages 221–222. Springer, 2012.
12. BPI Challenge 2012. Event log of a loan application process. `http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`, 2012.
13. BPI Challenge 2017. Event log of a loan application process. `https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b`, 2017.
14. F. Caron, J. Vanthienen, and B. Baesens. Comprehensive rule-based compliance checking and risk management with process mining. *Decision Support Systems*, 54(3):1357–1369, 2013.
15. M. de Leoni, J. Munoz-Gama, J. Carmona, and W. M. P. van der Aalst. Decomposing conformance checking on Petri nets with data. BPM Report BPM-14-06, BPMcenter.org, 2014.
16. C. Diamantini, L. Genga, and D. Potena. ESub: Exploration of Subgraphs. In *Proceedings of the BPM Demo Session*, pages 70–74. CEUR-WS.org, 2015.
17. C. Diamantini, L. Genga, D. Potena, and W. M. P. van der Aalst. Building instance graphs for highly variable processes. *Expert Systems with Applications*, 59:101–118, 2016.
18. W. Eberle, J. Graves, and L. Holder. Insider threat detection using a graph-based approach. *Journal of Applied Security Research*, 6(1):32–81, 2010.
19. D. Fahland. Translating UML2 Activity Diagrams Petri nets for analyzing IBM WebSphere Business Modeler process models. Informatik-Berichte 226, Humboldt-Universität zu Berlin, 2008.
20. D. Fahland and W. M. P. van der Aalst. Model repair—aligning process models to reality. *Information Systems*, 47:220–243, 2015.

21. L. Genga, M. Alizadeh, D. Potena, C. Diamantini, and N. Zannone. APD tool: Mining anomalous patterns from event logs. In *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling*, volume 1920 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.
22. L. Genga, D. Potena, O. Martino, M. Alizadeh, C. Diamantini, and N. Zannone. Subgraph Mining for Anomalous Pattern Discovery in Event Logs. In *Proceedings of International Workshop on New Frontiers in Mining Complex Patterns*. Springer, 2016.
23. G. Greco, A. Guzzo, G. Manco, and D. Saccà. Mining and reasoning on workflows. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):519–534, 2005.
24. G. Greco, A. Guzzo, G. Manco, and D. Saccà. Mining unconnected patterns in workflows. *Information Systems*, 32(5):685–712, 2007.
25. G. Greco, A. Guzzo, and L. Pontieri. Mining taxonomies of process models. *Data & Knowledge Engineering*, 67(1):74–102, 2008.
26. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, 2000.
27. L. Holder, D. Cook, and S. Djoko. Substructure Discovery in the SUBDUE System. In *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.
28. J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *Proceedings of SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–586. ACM, 2004.
29. Z. Huang, X. Lu, and H. Duan. On mining clinical pathway patterns from medical behaviors. *Artificial intelligence in medicine*, 56(1):35–50, 2012.
30. S. Hwang, C. Wei, and W. Yang. Discovery of temporal patterns from process instances. *Computers in industry*, 53(3):345–364, 2004.
31. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of European Conference on Principles of Data Mining and Knowledge Discovery*, pages 13–23. Springer, 2000.
32. C. Jiang, F. Coenen, and M. Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(01):75–105, 2013.
33. I. Jonyer, D. Cook, and L. Holder. Graph-based Hierarchical Conceptual Clustering. *Journal of Machine Learning Research*, 2:19–43, 2002.
34. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings IEEE International Conference on Data Mining*, pages 313–320. IEEE, 2001.
35. M. Kuramochi and G. Karypis. GREW – A Scalable Frequent Subgraph Discovery Algorithm. In *Proceedings of IEEE International Conference on Data Mining*, pages 439–442. IEEE, 2004.
36. K. B. Lassen and B. F. van Dongen. Translating message sequence charts to other process languages using process mining. In *Transactions on Petri Nets and Other Models of Concurrency I*, pages 71–85. Springer, 2008.
37. M. Leemans and W. M. P. van der Aalst. Discovery of frequent episodes in event logs. In *Proceedings of International Symposium on Data-driven Process Discovery and Analysis*, pages 1–31. CEUR-ws.org, 2014.
38. S. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Discovering Block-Structured Process Models From Event Logs - A Constructive Approach. In *Applications and Theory of Petri Nets and Concurrency*, pages 311–329. Springer, 2013.
39. X. Lu, D. Fahland, F. J. H. M. van den Biggelaar, and W. M. P. van der Aalst. Detecting deviating behaviors without models. In *Business Process Management Workshops*, pages 126–139. Springer, 2016.
40. X. Lu, D. Fahland, and W. M. P. van der Aalst. Conformance checking based on partially ordered event data. In *Business Process Management*, pages 75–88. Springer, 2014.
41. F. Mannhardt and N. Tax. Unsupervised event abstraction using pattern abstraction and local process models. CoRR abs/1704.03520, arxiv.org, 2017.
42. B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, 1998.
43. C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. Hofstede, and J. Mendling. From business process models to process-oriented software systems. *ACM Trans. Softw. Eng. Methodol.*, 19(1):2:1–2:37, 2009.
44. E. Ramezani, D. Fahland, and W. M. P. van der Aalst. Where Did I Misbehave? Diagnostic Information in Compliance Checking. In *Business Process Management*, pages 262–278. Springer, 2012.
45. A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.
46. D. C. Schmidt and L. E. Druffel. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *Journal of the ACM*, 23(3):433–445, 1976.
47. E. R. Taghiabadi, V. Gromov, D. Fahland, and W. M. P. van der Aalst. Compliance checking of data-aware and resource-aware compliance requirements. In *On the Move to Meaningful Internet Systems*, LNCS 8841, pages 237–257. Springer, 2014.

48. N. Tax, N. Sidorova, W. M. P. van der Aalst, and R. Haakma. Heuristic approaches for generating local process models through log projections. In *Proceedings of IEEE Symposium Series on Computational Intelligence*, pages 1–8. IEEE, 2016.
49. L. T. Thomas, S. R. Valluri, and K. Karlapalem. Margin: Maximal frequent subgraph mining. *ACM Transactions on Knowledge Discovery from Data*, 4(3):10, 2010.
50. J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.
51. N. van Beest, M. Dumas, L. García-Bañuelos, and M. La Rosa. Log delta analysis: Interpretable differencing of business process event logs. In *Business Process Management*, LNCS 9253, pages 386–405. Springer, 2015.
52. S. K. van den Broucke, J. Munoz-Gama, J. Carmona, B. Baesens, and J. Vanthienen. Event-based real-time decomposed conformance analysis. In *On the Move to Meaningful Internet Systems*, pages 345–363. Springer, 2014.
53. W. M. P. van der Aalst. Decomposing petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.
54. W. M. P. van der Aalst. *Process mining: data science in action*. Springer, 2016.
55. W. M. P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Int. Rev. Data Min. and Knowl. Disc.*, 2(2):182–192, 2012.
56. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. Weijters. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.

A Converting a p-trace into a Petri net

Algorithm 4 describes the conversion from a p-trace to a Petri net. The set of places is initialized by creating an initial place p_{ini} and an ending place p_{end} (line 1). The set of transitions in the Petri net consists of the nodes in the p-trace (line 2). We also create two invisible transitions, inv_{ini} and inv_{end}, which are connected to p_{ini} and p_{end} respectively (line 3). Invisible transition inv_{ini} is needed to capture the fact that events in E_{ini}, i.e. the set of transitions for which the corresponding node in the p-trace does not have an incoming edge, can occur concurrently. Similarly, inv_{end} is needed to capture the fact that events in E_{end}, i.e. the set of transitions for which the corresponding node in the p-trace does not have an outgoing edge, can occur concurrently. Note that inv_{ini} (inv_{end} respectively) can be omitted if E_{ini} (E_{end} respectively) consists of only one transition.

Then, a place p is added for each edge in the p-trace. Transitions and places are connected in such a way that for every $(e, e') \in W$ we have $(e, p), (p, e') \in F$. Each transition e_i for which the corresponding node in the p-trace does not have an incoming edge (i.e., $e_i \in E_{ini}$) is connected to invisible transition inv_{ini} through a newly created place p_i (lines 9-11). Similarly, each transition e_j for which the corresponding node in the p-trace does not have an outgoing edge (i.e., $e_j \in E_{end}$) is connected to invisible transition inv_{end} through a newly created place p_j (lines 12-14). The initial marking m_i and final marking m_f of the Petri net consist of p_{ini} and p_{end} respectively, i.e. $m_i = [p_{ini}]$ and $m_f = [p_{end}]$.

Algorithm 4: p-trace to Petri net conversion

Input : p-trace (E, W), set of activities A, labeling function $act : E \to A$
Output : Petri net $(P, T, F, A, act, m_i, m_f)$

$P \leftarrow \{p_{ini}, p_{end}\}$;
$T \leftarrow E \cup \{inv_{ini}, inv_{end}\}$ // with $inv_{ini}, inv_{end} \in \mathsf{Inv}$;
$F \leftarrow \{(p_{ini}, inv_{ini}), (inv_{end}, p_{end})\}$;
foreach $(e, e') \in W$ **do**
    $P \leftarrow P \cup \{p\}$;
    $F \leftarrow F \cup \{(e, p), (p, e')\}$;
Let $E_{ini} = \{e \in E \mid \nexists e' \in E : (e', e) \in W\}$;
Let $E_{end} = \{e \in E \mid \nexists e' \in E : (e, e') \in W\}$;
foreach $e_i \in E_{ini}$ **do**
    $P \leftarrow P \cup \{p_i\}$;
    $F \leftarrow F \cup \{(inv_{ini}, p_i), (p_i, e_i)\}$;
foreach $e_j \in E_{end}$ **do**
    $P \leftarrow P \cup \{p_j\}$;
    $F \leftarrow F \cup \{(e_j, p_j), (p_j, inv_{end})\}$;
$m_i \leftarrow [p_{ini}]$;
$m_f \leftarrow [p_{end}]$;
return $(P, T, F, A, act, m_i, m_f)$