Process Comparison Using Petri Net Decomposition

Tobias Brockhoff[1][0000−0002−6593−9444], Moritz Nicolas Gose[2], Merih Seran Uysal[1][0000−0003−1115−6601], and Wil M.P. van der Aalst[1][0000−0002−0955−6940]

[1] Chair of Process and Data Science, RWTH Aachen University, Germany
{brockhoff,uysal,wvdaalst}@pads.rwth-aachen.de
[2] RWTH Aachen University, Germany
moritz.gose@rwth-aachen.de

Abstract. Business processes drive the value creation at companies requiring them to constantly monitor and improve the former. The field of *Process Comparison* (PC) offers promising approaches to gain insight into differences between variants of a process that one can leverage to improve the latter. For example, one might consider the same process at different points in time or at different sites. Recent PC methods consider event logs containing data on real-life process executions the single source of truth. However, there often exist additional specifications that can be represented as Petri nets. In this paper, we propose an approach that leverages a given Petri net to compare two event logs in a hierarchical manner. To this end, we decompose the provided net into subprocesses and extract data on their executions from the event logs. Based on these executions, we exemplify how one can flexibly assess different aspects of a process (e.g., control flow, performance, or conformance). Using statistical tests, we eventually detect differences between subprocesses with respect to a selected aspect. Despite the approach is mostly agnostic to the decomposition applied, we present a decomposition strategy that we deem particularly suitable for PC. For this purpose, we consider the Refined Process Structure Tree of a Petri net and propose a novel preprocessing approach to improve the final decomposition. We implemented the approach in ProM and evaluate it in a real-life case study.

Keywords: Process Mining · Process Comparison · Process Variant Analysis · Business Process Intelligence

1 Introduction

Modern information systems record increasing amounts of data on business process executions. *Event data* are a special type of these data where each data point is an *event* comprising a *timestamp*, an *activity* name, and a *case id* related to a business case. *Process mining* methods are concerned with the analysis of event data and the implementation of event-data-to-knowledge pipelines. Ultimately,

process mining aims to provide insight to improve the process. *Process Comparison* (PC) approaches take this idea one step further revealing differences between two (or more) process variants. Differences can, for example, concern the process' control flow (e.g., decision likelihoods) or performance (e.g., cycle times). Thereby, PC provides insight into the effect of changes, comparing process executions before and after a change, or into particularities of a process' environment, comparing implementations at different sites.

Recent PC approaches often consider event data as the single source of truth and, therefore, only take event logs as inputs [10,8,25]. Yet, this neglects process models (e.g., BPMN models) as another valuable source of information. In practice, BPMN models are fundamental for the design and re-design of enterprise information systems [3] making them an essential part of a process' documentation. For analytical purposes, these models can be seamlessly converted into Petri nets. The advantages of incorporating process models into PC are manifold: (i) process models help to *manage the complexity of processes*. PC approaches solely based on event data often rely on follows relations between activities. However, example traces cannot properly represent these relations for large and concurrent processes (e.g., production processes). Besides, process models allow to explicitly consider duplicated transitions and to model and analyze interesting relations. For example, in [4], the authors add user-defined places to measure times for larger subprocesses within a single process variant. (ii) Submodels often define *logical units* enabling us to analyze differences at different levels of granularity. (iii) Finally, models provide a *natural context* to present the results of PC. They are the common ground in a projected view of the differences. Despite the advantages of model-based analysis, merely considering models usually results in a loss of information. For instance, model notations such as Petri nets do not support modeling frequencies. Besides, the actual process executions can deviate from the model or exhibit dependencies not represented by the model.

In this paper, we propose a hybrid approach, which takes two event logs and a shared process model as an input. Figure 1 shows the main concepts of the approach. In contrast to existing approaches, we compare the event logs in a hierarchical manner enabling process analysts to conduct an analysis at different levels of granularity. For example, in Figure 1, we detect a cycle time difference for a large subprocess on the first level of the hierarchy. By drilling down into the subprocess, one can refine this knowledge: (i) there is no significant difference regarding the time it takes to execute the redo part and (ii) one concurrent branch is even executed faster in the other process variant.

In this work, we use process models, transformed in Petri nets, to enable a hierarchical analysis, but we discover differences based on the event data. To this end, we decompose the Petri net into a hierarchy of subprocesses and compute subprocess executions relating subprocesses and event data. Based on these executions, we define measurements that assess different aspects of a process. In doing so, we not only consider the control-flow and performance perspective, but also propose to compare how process variants deviate from the model.

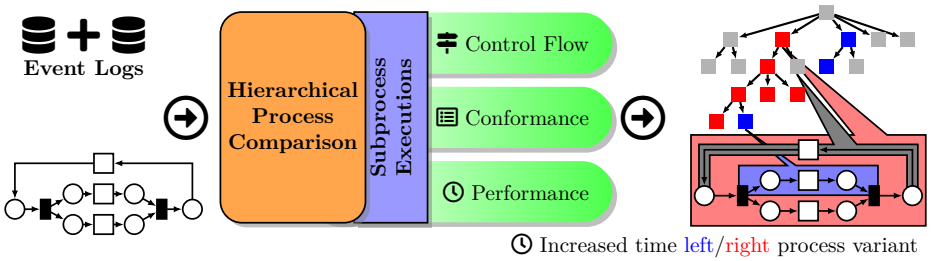


Fig. 1: Hierarchical Process Comparison on two event logs using a shared process model. First, we decompose the Petri net and relate the resulting subprocesses with the event data. Based on these subprocess executions, we compare the two process variants with respect to various perspectives. Ultimately, each vertex in the decomposition shows whether a difference was discovered for the respective subprocess. The color is chosen based on the effect size of the difference. Selecting a subprocess, it is highlighted in the original model.

While the approach is mostly agnostic to the decomposition applied, it is usually preferable that the nodes in the decomposition represent logically coherent subprocesses. We therefore propose a decomposition strategy that leverages the *Refined Process Structure Tree* (RPST) of a model [27]. In the RPST, each node corresponds to a subprocess that is entered and exited via a single node, respectively. However, as illustrated in Section 4.3, relaxing this requirement can improve the decomposition (e.g., to handle long-term dependencies). To this end, we propose an additional preprocessing step preceding the RPST computation, where we use the event log to remove places to improve the block-structuredness of the Petri net and, consecutively, the decomposition.

The remainder of this paper is structured as follows: we discuss related works and preliminary concepts in Sections 2 and 3, respectively. In Sections 4.1 and 4.3, we discuss the decomposition approach and illustrate its application to PC in Section 4.2. In Section 5, we evaluate the method in a real-world case study. Finally, we give our conclusion and directions for future work in Section 6.

2 Related Work

Our approach is directly related to works on PC and Petri net decomposition as well as to approaches for decomposed conformance checking.

For a comprehensive survey on PC, we refer the reader to [26]. Despite not being limited to the control flow or performance perspective [22], most PC approaches consider these. One can distinguish PC approaches that consider event logs as input [8,10,22,25,7,28], compare process models [14,6,18], or require a process model and two (or more) event logs as input [23,12,30,17,1]. Log-based approaches often represent follows relation between activities by means of a

graph [8,22,25,7,28]. Then, a statistical test is used to detect frequency differences [8,22,25]. For example, in their seminal approach, Bolt et al. represent the event logs by a shared transition system [8]. Like our approach, they introduce measurement functions to annotate the states and edges of the transition system and apply Welch's t-test to detect differences. In contrast to our work, the proposed measurements neither assess differences on a subprocess level nor consider relations between subprocesses. Besides, in contrast to model-based approaches, event log-based methods cannot represent duplicate activities.

Process model comparison methods analyze differences on the process specification level [14,6,18]. If event data is available, only considering models would discard the event data as a valuable source of information providing insight into decision likelihoods and time. Therefore, approaches for data-driven PC using process models generally enrich the latter with information obtained from the event data. Thereby, one can investigate aspects like execution times and de facto path frequencies in the model. For example, in [17], the authors merge two process models into a difference model which is further enriched with instance traffic information. For each edge, they then detect frequency differences. In [30], Wynn et al. explicitly consider Petri nets. Like our approach, they use alignments [5] to relate the model with the (potentially slightly deviating) data. Compared to our method, their approach provides more detailed results on waiting times between transitions. However, they only consider pairs of transitions rather than larger subprocesses and their relations. Moreover, they do not validate the statistical significance of the differences returned. In general, to the best of our knowledge, there currently exists no *hierarchical*, model-based PC approach.

Petri net decomposition techniques simplify models to (heuristically) solve problems on subnets. For different applications, various approaches have been proposed [2,9,13,27,32,16]. In this work, we compare process variants with respect to coherent sub-workflows (i.e., subprocesses) of the original model. To this end, we build on the notion of *single-entry, single-exit* (SESE)-fragments, which have originally been proposed in [27] as a unique, hierarchical decomposition of a process model into self-contained subnets. This notion expands upon an earlier definition by Johnson et al. [15]. The resulting, more granular decomposition coined the term *Refined Process Structure Tree* (RPST). Further improvements and a more efficient way to compute the RPST have been proposed in [24]. In this work, we propose an additional pre-processing step that can improve the decomposition for the sake of PC but weakens the structural guarantees.

An application of SESE fragments related to our approach is decomposed conformance checking [20]. While we also consider conformance in the context of RPSTs, we use fragments to aggregate the diagnostics, similar to [21], rather than improving the computational efficiency.

3 Petri nets and Process Mining Concepts

We denote sets by capital letters. Given a set S, its powerset is denoted by $\mathcal{P}(S)$, and the set of all multisets is denoted by $\mathcal{B}(S)$. For a multiset $m \in \mathcal{B}(S)$, the

multiplicity and an element $s \in S$ is $m(s) \in \mathbb{N}$. Given two multisets $m_1, m_2 \in \mathcal{B}(S)$, we write $m_1 + m_2$ ($m_1 - m_2$) to denote their sum (difference). Besides, we write $m_1 \leq m_2$ if $m_1(s) \leq m_2(s)$ holds for all $s \in S$. In an abuse of notation, we also apply these operators to pairs of sets and multisets.

The Kleene star (S^*) represents the set of all finite sequences over a (countable infinite) alphabet S. Given a sequence $\boldsymbol{\sigma} = \langle \sigma_1, \ldots, \sigma_n \rangle \in S^*$, we refer to its i$^{\text{th}}$ element as $\boldsymbol{\sigma}_i$. The length of $\boldsymbol{\sigma}$ is denoted by $|\boldsymbol{\sigma}|$. Let $I = \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, |\boldsymbol{\sigma}|\}$ be a set of indices. Assuming the order $i_1 < \cdots < i_m$, the index set I induces the subsequence $\boldsymbol{\sigma}_{[I]} = \langle \boldsymbol{\sigma}_{i_1}, \ldots, \boldsymbol{\sigma}_{i_m} \rangle$.

Petri Nets Let $\mathcal{A}$ denote the universe of activity labels.

Definition 1 (Labeled Petri Net). *Let $\tau \notin \mathcal{A}$ be a special silent label. A labeled Petri net is a tuple $N = (P, T, F, l)$, where (i) P is a finite set of places, (ii) T is a finite set of transitions, (iii) $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, and (iv) $l \colon T \to \mathcal{A} \cup \{\tau\}$ is a labeling function.*

Let $N = (P, T, F, l)$ be a Petri net. For an element $x \in P \cup T$, its preset (postset) are defined as ${}^\bullet x \coloneqq \{y \,|\, (y, x) \in F\}$ ($x^\bullet \coloneqq \{y \,|\, (x, y) \in F\}$). Given a set of edges $F' \subseteq F$, we denote the set of adjacent places by $P_{F'} \coloneqq \{p \in P \exists t \in T \,((p, t) \in F' \vee (t, p) \in F')\}$. Likewise, for the set $T_{F'}$ of adjacent transitions. The edge-induced subnet $N_{F'}$ is the Petri net $\left(P_{F'}, T_{F'}, F', l \restriction_{T_{F'}}\right)$, where $l \restriction_{T_{F'}}$ denotes the restriction of l on the adjacent transitions. The semantics of Petri nets are determined by marking places, firing (sequences of) transitions.

Definition 2 (Marking, Firing). *Let $N = (P, T, F, l)$ be a labeled Petri net. A marking $m \in \mathcal{B}(P)$ of N is a finite multiset of places. A transition $t \in T$ is enabled in m if ${}^\bullet t \leq m$. If t is* enabled*, firing t in m results in the marking $m' = (m - {}^\bullet t) + t^\bullet$, written as $m[t\rangle_N m'$.*

Definition 3 (Firing Sequence). *Let $N = (P, T, F, l)$ be a labeled Petri net and $m_I, m_F \in \mathcal{B}(P)$ be two markings. A sequence $\boldsymbol{\sigma} = \langle t_1, \ldots, t_n \rangle \in T^*$ is a* valid firing sequence *from m_I to m_F of N, written $m_I \xrightarrow{\boldsymbol{\sigma}}_N m_F$, if there exist markings $m_1, \ldots, m_{n+1}$ such that (i) $m_1 = m_I$, (ii) $m_{n+1} = m_F$, and (iii) for $1 \leq i \leq n$ we have $m_i[t_i\rangle_N m_{i+1}$.*

In process mining, a commonly considered class of Petri nets are *workflow nets* (WF-nets). A WF-net has a clear start and end, and all elements are on a directed path from the start to the end.

Definition 4 (Workflow Net (WF-net)). *A labeled Petri net $N = (P, T, F, l)$ is a labeled* workflow net *(WF-net) if (i) there is a unique source place p_I (i.e., $\{p_I\} = \{p \in P \,|\, {}^\bullet p = \emptyset\}$), (ii) there is a unique sink place p_F (i.e., $\{p_F\} = \{p \in P \,|\, p^\bullet = \emptyset\}$), and (iii) every node is on a path from p_I to p_F.*

The *workflow system net* (WF-sytem net) explicitly establishes a connection between a WF-net and its semantics (i.e., its initial and final marking).

Definition 5 (Workflow System Net (WF-sytem net)). *A* workflow system net *(WF-sytem net)* $\mathrm{SN} = (N, m_I, m_F)$ *comprises a labeled WF-net* $N = (P, T, F, l)$*, the initial marking* $m_I = [p_I]$*, and the final marking* $m_F = [p_F]$.

The *language* of a WF-sytem net comprises all sequences of visible activity labels of its valid firing sequences. Finally, a WF-sytem net is safe if we cannot reach a marking where a place contains multiple tokens.

Definition 6 (Safeness). *A WF-sytem net* $\mathrm{SN} = ((P, T, F, l), m_I, m_F)$ *is safe if there is no valid firing sequence* $\boldsymbol{\sigma} \in T^*$ *reaching a marking* $m' \in \mathcal{B}(P)$ *from* m_I *(i.e.,* $m_I \xrightarrow{\boldsymbol{\sigma}}_N m'$*) with* $m'(p) > 1$ *for some* $p \in P$.

Event Data We leverage Petri nets to compare two process variants based on data of their real-life executions. Each process execution corresponds to a business case and comprises information on the activities executed for this particular case. For each activity execution, we record the activity's name and a timestamp. While additional attributes are possible, they are not considered in this work. An *event log* collects multiple process executions.

Definition 7 (Event Log). *Let* $\mathcal{A}$ *and* $\mathcal{T}$ *denote (countable infinite) universes of activity names and timestamps. The set of all activity executions is defined as* $\mathcal{E} \coloneqq \mathcal{A} \times \mathcal{T}$*. A* trace $\boldsymbol{\sigma} = \langle (a_1, t_1), \ldots, (a_n, t_n) \rangle \in \mathcal{E}^*$ *is a finite sequence of activity executions respecting time—that is* $t_i \leq t_j$ *for all* $1 \leq i < j \leq n$*. An* event log $L \in \mathcal{B}(\mathcal{E}^*)$ *is a finite multiset of traces.*

Dealing with event data, we frequently use two types of projection. The projection $\pi_{S'} \colon S^* \to (S')^*$ of a sequence over a set S on a subset $S' \subseteq S$ only keeps the elements contained in S'. Consider the trace $\boldsymbol{\sigma} = \langle (a, 1), (b, 2), (c, 3) \rangle$. Projecting $\boldsymbol{\sigma}$ onto all executions of a and b yields the trace $\pi_{\{a,b\} \times \mathcal{T}}(\boldsymbol{\sigma}) = \langle (a, 1), (b, 2) \rangle$. Moreover, we write $\pi_{time} \colon \mathcal{E}^* \to \mathcal{T}^*$ ($\pi_{act} \colon \mathcal{E}^* \to \mathcal{A}^*$) to denote the projection of traces onto the associated timestamps (activities). For example, for $\boldsymbol{\sigma}$, we obtain the sequence $\pi_{time}(\boldsymbol{\sigma}) = \langle 1, 2, 3 \rangle$. In a slight abuse of notation, we also apply π^{time} and π^{act} to individual activity executions.

Alignments In real life processes, the traces recorded might not perfectly match with the prescribed Petri net model. Introducing a dedicated skip symbol, we additionally require that columns either contain a single skip symbol or that the trace's activity matches the transition's label. Thereby, an alignment represents a joined, synchronized execution of the trace and the model.

Definition 8 (Alignment [2]). *Let* $\boldsymbol{\sigma} \in \mathcal{E}^*$ *be a trace and* $\mathrm{SN} = (N, m_I, m_F)$*,* $N = (P, T, F, l)$*,* $l \colon T \to \mathcal{A}$ *be a WF-sytem net. Let* $\gg \notin \mathcal{A} \cup T$ *denote a special no-move symbol. We define the sets of synchronous, log, model, and all moves as* $M_{\mathrm{SYNC}} \coloneqq \{(e, t) \mid t \in T, l(t) \neq \tau, e \in \mathcal{E}, \pi^{act}(e) = l(t)\}$*,* $M_{\mathrm{LM}} \coloneqq \mathcal{E} \times \{\gg\}$*,* $M_{\mathrm{MM}} \coloneqq \{\gg\} \times T$*, and* $M_{\mathrm{ALL}} \coloneqq M_{\mathrm{SYNC}} \cup M_{\mathrm{LM}} \cup M_{\mathrm{MM}}$*, respectively. An alignment of* $\boldsymbol{\sigma}$ *and* N *is a sequence of moves* $\boldsymbol{\gamma} \in M_{\mathrm{ALL}}^*$ *such that*

(i) projection $\pi^1(\boldsymbol{\gamma})$ *on the first element, ignoring* $\gg$*, yields* $\boldsymbol{\sigma}$*—that is,* $\pi_{\gg}\left(\pi^1(\boldsymbol{\gamma})\right) = \boldsymbol{\sigma}$*—and*

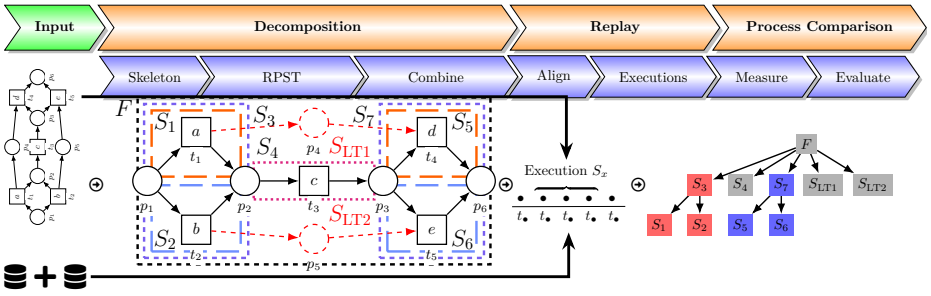


Fig. 2: Overview of our three-stage approach for comparing two process variants. First, we hierarchically decompose the model into subprocesses. Second, we replay the event data to relate the subprocesses with the data. Third, we compare the variants with respect to the subprocesses from various perspectives. Eventually, we project the differences onto the decomposition.

(ii) projection $\pi^2(\boldsymbol{\gamma})$ *on the second element, ignoring* $\gg$*, yields a valid firing sequence from* m_I *to* m_F *of* N*—that is,* $m_I \xrightarrow{\pi_{\gg}(\pi^2(\boldsymbol{\gamma}))}_N m_F$*,*

where $\pi_{\gg} := \pi_{(\mathcal{E}\cup T)\setminus\{\gg\}}$ *denotes the projection that removes skips.*

The set of best alignments is typically determined by a cost function minimizing the number of log and visible model moves. In this work, we assume that a single best alignment is given, which can make our approach non-deterministic. Finally, given an alignment $\boldsymbol{\gamma}$ of a trace $\boldsymbol{\sigma}$ and WF-sytem net $\mathrm{SN} = ((P, T, F, l), m_I, m_F)$, the marking reached after executing the first $1 \leq k \leq |\boldsymbol{\gamma}|$ steps is $m^{al}_{\boldsymbol{\gamma}}(k)$ with

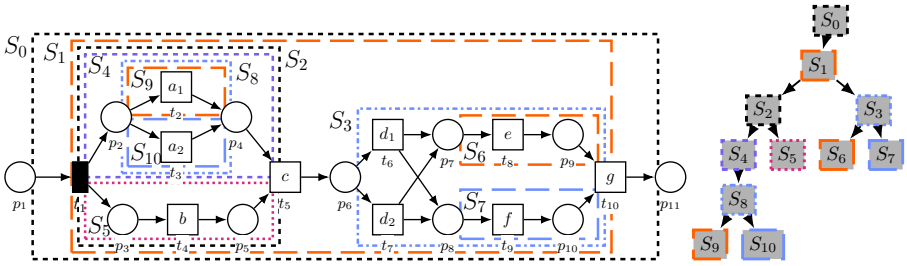
$$m_I \xrightarrow{\pi_{\gg}(\pi^2(\boldsymbol{\gamma}_{[\{1,\ldots,k\}]}))}_N m^{al}_{\boldsymbol{\gamma}}(k). \tag{1}$$

4 Hierarchical Process Comparison

This section presents our approach for comparing the executions of two process variants based on a shared process model. To avoid boundary cases (e.g., empty alignments), we assume that event logs do not contain empty traces and that WF-sytem nets have at least one transition. Figure 2 shows an overview of our approach that has three stages: (i) the *model decomposition* stage, where we hierarchically decompose the model into subprocesses; (ii) the *replay stage*, where we relate the subprocesses with the event data; and (iii) the *comparison stage*, where we compare the process variants with respect to different perspectives.

4.1 Hierarchical Decomposition

Existing Petri net decomposition approaches [27,2,9,13,16] focus on the semantic relation between the original Petri net and the sub-nets. For example, the



(a) Petri net N_1 (b) Decomposition

Fig. 3: Illustration of a hierarchical WF-net decomposition. Subfigure (a) shows a WF-net and ten $(S_0, \ldots, S_9)$ (*single-entry, single-exit* (SESE)) subprocesses. Each subprocess contains the edges inside the illustrated rectangle. A hierarchy of these subprocesses is depicted in Subfigure (b).

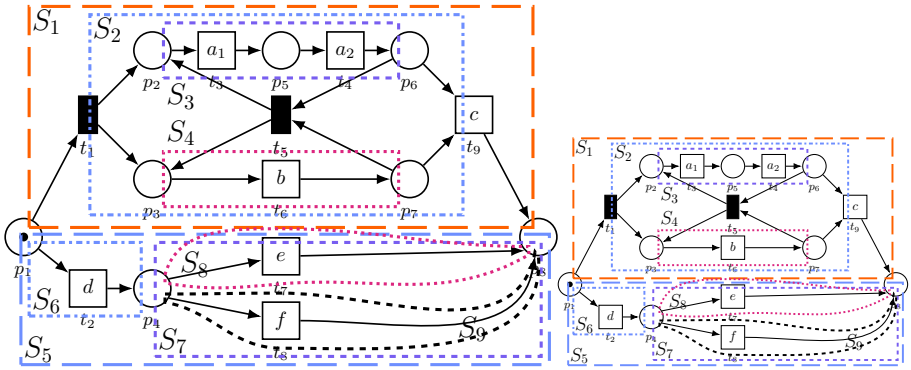
decomposition proposed in [2] is motivated by the idea of stitching together *valid firing sequences* of the sub-nets to an (over-optimistic) execution of the original net. In contrast, as illustrated in Figure 2, we align the log with the original net. Thereby, we gain more freedom to decompose the net in a hierarchical manner without facing problems when relating process executions to sub-nets. Yet, computing alignments can be computationally costly. In this work, we employ a generic hierarchical decomposition based on the edges of a WF-net.

Definition 9 (Hierarchical Decomposition). *Let $N = (P, T, F, l)$ be a WF-net. A hierarchical decomposition of N is a tree $H = (\mathbb{S}, E), \mathbb{S} \subseteq \mathcal{P}(F), E \subseteq \mathbb{S} \times \mathbb{S}$ rooted at a vertex $v_0 \in \mathbb{S}$ and downward-pointing edges such that $v_0 = F$; for $(v_1, v_2) \in E$, we have $v_1 \supset v_2$; and for $S \in \mathbb{S}$, the induced subnet N_S is connected.*

Intuitively, each vertex in the hierarchy corresponds to an edge induced subnet of the original net. Under this interpretation, the original net is at the root, and each non-root vertex's net is a subnet of its parent's net—that is, it comprises a subset of its parent's net's places, transitions, and edges. We require connectedness of subprocesses to later define the semantics of a *subprocess execution*.

Figure 3b shows a decomposition of the WF-net in Figure 3a. Note that, given the ten subprocesses in Figure 3a, Definition 9 does not enforce a unique hierarchy. A decomposition where S_0 is the root and all remaining subprocesses are S_0's children would also be valid. While a deeper hierarchy is usually preferable, there can be exceptions. For example, an analysis that investigates parent-child relations can benefit from this additional freedom (cf. Equation (10)).

In general, Definition 9 does not impose strong restrictions on the decomposition applied. To relate event data to transitions of a Petri net, we use alignments where we consider the complete trace and net. In contrast to relating individual subprocesses independently of each other, this guarantees a globally consistent assignment without additional considerations (e.g., an event cannot be assigned to different transitions having the same label). Therefore, the measurements proposed in Section 4.2 are mostly independent of the decomposition.



(a) Petri net N_3 (b) Decomposition of N_3

Fig. 4: WF-net $N_3 = (P_3, T_3, F_3, l)$ illustrating hierarchical PC. The edges in the colored box in (a) depict the canonical fragments of the net. Subfigure (b) depicts a decomposition of N_3.

Despite our approach being mostly agnostic to the decomposition applied, structural subprocess properties can play a role when interpreting the results. In particular, we distinguish subprocesses—so-called *fragments* [27]—where control flow enters and exits through a single node, respectively.

Definition 10 (*Single-entry, single-exit* (SESE) Subprocess). *Let $N = (P, T, F, l)$ be a WF-net and $S \subseteq F$ be a set of edges such that N_S is connected. A vertex $v \in P_S \cup T_S$ is a* boundary *vertex of N_S if it is the source or the sink of N or if v is incident to edges $e_1 \in S$ and $e_2 \notin S$; otherwise v is an* internal *vertex. The subprocess S is a SESE subprocess if there exists entry and exit vertices $v_i, v_e \in P_S \cup T_S$ such that (i) v_i (v_e) are boundary vertices; (ii) no incoming edge of v_i is in S or all outgoing edges of v_i are in S; (iii) no outgoing edge of v_e is in S or all incoming edges of v_e are in S; (iv) there is no other boundary vertex $v \in (P_S \cup T_S) \setminus \{v_i, v_e\}$.*

All subprocesses illustrated in Figure 3a are SESE subprocesses.

4.2 Measuring Differences

To compare event logs using the decomposition, we first relate the subprocesses to the event data. We then define various measurements that, for example, assess differences in the control flow or performance of subprocesses. Finally, we use hypothesis tests to detect differences with respect to the measurements. We illustrate the following concepts and a few interesting measurements on the WF-net shown in Figure 4a. Figure 4b shows its decomposition.

Subprocess Executions In contrast to works that project the data onto the subprocesses [2,20], we extract subprocess executions from alignments. Replaying the alignment, we consider a subprocess being under execution as long as an

associated place contains a token. Therefore, we require *safe* WF-nets to avoid intermingled executions—that is, multiple, simultaneous executions of a subprocess. If a place contains two tokens "created" by two events, it becomes unclear to which event we should to relate the "consuming" event. To distinguish the consumption and production of tokens, we introduce two helper functions. Given an alignment $\boldsymbol{\gamma}$ and a subprocess S, we count the number of tokens contained in S after the first k steps (inclusive) and the number of tokens contained before the k^{th} step produced tokens:

$$\#^{al}_{S,\boldsymbol{\gamma}}(k) = \left|[p \in m^{al}_{\boldsymbol{\gamma}}(k) \,|\, p \in P_S]\right|, \tag{2}$$

$$\#^{al,-}_{S,\boldsymbol{\gamma}}(k) = \left|\left[p \in \left(m^{al}_{\boldsymbol{\gamma}}(k) - \begin{cases} [] & \text{if } \pi^2(\boldsymbol{\gamma}_k) = \gg \\ \pi^2(\boldsymbol{\gamma}_k)^\bullet & \text{else} \end{cases}\right) \middle| \, p \in P_S\right]\right|. \tag{3}$$

Next, we define the intervals during which a subprocess is under executions.

Definition 11 (Subprocess Execution). *Let* $\mathrm{SN} = ((P,T,F,l), m_I, m_F)$ *be a WF-sytem net,* $S \subseteq F$ *be a set of edges,* $\boldsymbol{\sigma} \in \mathcal{E}^*$ *be a trace, and* $\boldsymbol{\gamma} \in M^*_{\mathrm{ALL}}$ *be an alignment of* $\boldsymbol{\sigma}$ *and* SN. *The* partial execution intervals *of* S *given* $\boldsymbol{\gamma}$ *are*

$$\begin{aligned} E^{part}_{\mathrm{SN},\boldsymbol{\gamma},S} = \{\{i,\ldots,j\} | 1 \le i \le j \le |\boldsymbol{\gamma}| \quad & \textit{(Intervals)} \\ \wedge \forall i \le k < j \left(\#^{al}_{S,\boldsymbol{\gamma}}(k) \ge 1\right) \quad & \textit{(Token contained)} \\ \wedge \exists i \le k \le j \left(\pi^2(\boldsymbol{\gamma}_k) \in T_S\right) \quad & \textit{(Transition fired)} \\ \wedge \forall i < k < j \left(\#^{al,-}_{S,\boldsymbol{\gamma}}(k) = 0 \rightarrow \pi^2(\boldsymbol{\gamma}_k) \in T_S\right)\Big\}. \quad & \textit{(No short-circuiting loops)} \end{aligned} \tag{4}$$

The complete execution intervals *of* S *given* $\boldsymbol{\gamma}$ *are the maximal partial execution intervals*

$$E^{exec}_{\mathrm{SN},\boldsymbol{\gamma},S} = \{\mathcal{I}_1 \in E^{part}_{\mathrm{SN},\boldsymbol{\gamma},S} \,|\, \forall \mathcal{I}_2 \in E^{part}_{\mathrm{SN},\boldsymbol{\gamma},S} (\mathcal{I}_1 \subseteq \mathcal{I}_2 \rightarrow \mathcal{I}_1 = \mathcal{I}_2)\}. \tag{5}$$

Equation (4) gives the conditions for a subprocess to be considered under executions. First, the subprocess must contain a token. Second, at least one transition of the subprocess must fire as for place-bounded subprocesses a token can pass without entering the subprocess. For example, consider p_1 in Figure 4a and the subprocesses S_1 and S_5. Despite both subprocesses contain p_1, a trace can only enter one. Third, there is no outside short-circuiting loop (i.e., a loop containing a single transition that is not adjacent to the subprocess) that consumes the last token and produces a new token in it. For example, consider the subprocess S_3 in Figure 4a and the following alignment $\boldsymbol{\gamma}_2$:

$$\begin{array}{rl} \#^{al,-}_{S_3,\boldsymbol{\gamma}_2}(k) & 0\ 0\ 1\ 0\ 0\ \ 0\ \ 0\ 0\ 1\ 0 \\ \#^{al}_{S_3,\boldsymbol{\gamma}_2}(k) & (1\ 1\ 1\ 1\ 1\ \{1)\ 1\ 1\ 1\ 0\} \\ \boldsymbol{\gamma}_2 = & \underline{\gg\ a_1\ \ b\ \ a_2\ \ a_2\ \ \gg\ \ a_1\ \ a_2\ \ b\ \ c} \\ & \underline{t_1\ \ t_3\ \ t_6\ \ t_4\ \gg\ \ t_5\ \ t_3\ \ t_4\ \ t_6\ \ t_9} \end{array}. \tag{6}$$

The token counts of S_3, displayed on top of the alignment, show that t_5 consumes the last token before it produces a new token in S_3. Yet, t_5 is not contained in

S_3. Finally, Equation (5) defines a subprocess' executions as the longest intervals during which the process is considered being under execution. In Equation (6), the parentheses and braces indicate the resulting executions.

Subprocess Measurements Based on the subprocess executions, we can compare processes with respect to various aspects on different levels of granularity.

Definition 12 (Trace Measurement). *Let* $\mathrm{SN} = ((P, T, F, l), m_I, m_F)$ *be a WF-sytem net,* $S \subseteq F$ *be a set of edges,* $\boldsymbol{\sigma} \in \mathcal{E}^*$ *be a trace, and* $\boldsymbol{\gamma} \in M^*_{\mathrm{ALL}}$ *be an alignment of* $\boldsymbol{\sigma}$ *and* SN. *The universe of perspectives is* $\mathcal{F}$. *The subprocess trace measurement with respect to a perspective* $p \in \mathcal{F}$ *is a function* $\mu^p_{\mathrm{SN},S} \colon M^*_{\mathrm{ALL}} \to \mathbb{R} \cup \{\bot\}$ *where* $\bot$ *denotes the absence of a measurement.*

In the following, we exemplify five measurements to illustrate how the decomposition and subprocess executions facilitate the detection of process variant differences. In particular, we consider the control flow, performance, and conformance perspective. To this end, we assume the following context for the remainder of this section: Let $\mathrm{SN} = (N, m_I, m_F), N = (P, T, F, l)$ be a WF-sytem net, $\mathcal{H} = (\mathbb{S}, E)$, $\mathbb{S} \subseteq \mathcal{P}(F)$ be a hierarchical decomposition of SN, $S \in \mathbb{S}$ be a subprocess, $\boldsymbol{\sigma} \in \mathcal{E}^*$ be a trace, and $\boldsymbol{\gamma} \in M^*_{\mathrm{ALL}}$ be an alignment of $\boldsymbol{\sigma}$ and SN.

Control flow First, we can measure whether a subprocess was activated, and how often it was executed:

$$\mu^{act}_{\mathrm{SN},S}(\boldsymbol{\gamma}) = \begin{cases} 1 & \text{if } \left|E^{exec}_{\mathrm{SN},\boldsymbol{\gamma},S}\right| > 0 \\ 0 & \text{else} \end{cases}, \quad \mu^{freq}_{\mathrm{SN},S}(\boldsymbol{\gamma}) = \left|E^{exec}_{\mathrm{SN},\boldsymbol{\gamma},S}\right|. \tag{7}$$

These measurements show differences with respect to the frequency of branching decisions and the number of repetitions.

In addition, the hierarchical nature of the decomposition facilitates the analysis of conditional decisions. Consider the following two event logs (not showing time for simplicity) that perfectly fit the Petri net N_3:

$$L_1 = \left[\langle a_1, a_2, b, c\rangle^{60}, \langle d, e\rangle^{14}, \langle d, f\rangle^{26}\right], \tag{8}$$

$$L_2 = \left[\langle a_1, a_2, b, c\rangle^{40}, \langle d, e\rangle^{27}, \langle d, f\rangle^{33}\right]. \tag{9}$$

In L_2, the activities e and f occur more frequently. Accordingly, the control flow measurements in Equation (7) show that, in L_2, executing S_8 and S_9 is more likely. However, if we respect the initial choice, the likelihood of observing f given that d was initially chosen is higher for L_1 (65% vs 55%). We can incorporate this into our frequency measurement by considering the parent subprocess S_7. We only consider whether S_8 and S_9 were activated if S_7 was activated. Assuming that S is not the root of $\mathcal{H}$, let $\bar{S}$ be the parent of S (i.e., $(\bar{S}, S) \in E$). We define the *conditional subprocess activation* measurement

$$\mu^{c.act}_{\mathrm{SN},S|\bar{S}}(\boldsymbol{\gamma}) = \begin{cases} \mu^{act}_{\mathrm{SN},S}(\boldsymbol{\gamma}) & \text{if } \mu^{act}_{\mathrm{SN},\bar{S}}(\boldsymbol{\gamma}) = 1 \\ \bot & \text{else} \end{cases}. \tag{10}$$

Conformance Process executions do not always comply with the process model. Thus, process variants might not only differ in how frequently they activate model elements but also in how they deviate from the prescribed control flow. An example is the *occurrence of log moves* measurement. For the subprocess S, we consider log moves on activities that are among the labels of the transitions T_S. In particular, we count log moves that occur outside the subprocess' executions:

$$\mu^{cc}_{\mathrm{SN},S}(\boldsymbol{\gamma}) = \left|\left\{i \in \{1,\dots,|\boldsymbol{\gamma}|\} \,|\, \boldsymbol{\gamma}_i \in M_{\mathrm{LM}} \wedge \exists t \in T_S \; l(t) = \pi^{act}\left(\pi^1\left(\boldsymbol{\gamma}_i\right)\right) \right.\right. \\ \left.\left. \wedge \forall I \in E^{exec}_{\mathrm{SN},\boldsymbol{\gamma},S} \; i \notin I\right\}\right|. \quad (11)$$

Differences with respect to this measurement show that, in one process variant, transitions of S are more likely to occur at unexpected positions. However, this measurement also illustrates a major challenge dealing with log moves—namely, duplicate transition labels. In case multiple transitions have the same label, we count log moves multiple times.

Performance Performance differences between process variants are often of major interest. Using our notion of process executions, one can define various performance measurements. In the following, we exemplify two complementary measurements and discuss their limitations. Given a complete execution interval $I \in E^{exec}_{\mathrm{SN},\boldsymbol{\gamma},S}$ of the subprocess S, we first consider the time series

$$\boldsymbol{\gamma}^{sync\_t}_{S,[I]} = \pi^{time}\left(\pi^1\left(\pi_{M_{\mathrm{SYNC}} \cap (\mathcal{E} \times T_S)}\left(\boldsymbol{\gamma}_{[I]}\right)\right)\right) \quad (12)$$

of synchronously executed subprocess transitions in I. Next, we define the *synchronous subprocess execution duration* as the time difference between the first and last synchronously executed transition that is adjacent to the subprocess:

$$\mu^{sync\_t}_{\mathrm{SN},S}(\boldsymbol{\gamma}) = \begin{cases} \bot & \text{if } |\boldsymbol{\gamma}^{sync\_t}_{S,[I]}| = 0 \\ & \text{for all } I \in E^{exec}_{\mathrm{SN},\boldsymbol{\gamma},S} \\ \operatorname{avg}\left(\left\{ \begin{array}{c} \max(\boldsymbol{\gamma}^{sync\_t}_{S,[I]}) \\ -\min(\boldsymbol{\gamma}^{sync\_t}_{S,[I]}) \end{array} \;\middle|\; \begin{array}{c} I \in E^{exec}_{\mathrm{SN},\boldsymbol{\gamma},S} \\ \wedge |\boldsymbol{\gamma}^{sync\_t}_{S,[I]}| > 0 \end{array} \right\}\right) & \text{else} \end{cases} \quad (13)$$

While this measurement provides insight into differences in the duration during which a subprocess was active, it does not account for delays prior or after the execution. For example, consider the subprocess S_3 in N_3 and the alignment

$$\boldsymbol{\gamma_3} = \frac{\gg \;\; b \;\; a_1 \;\; a_2 \;\; c}{t_1 \;\; t_6 \;\; t_3 \;\; t_4 \;\; t_9}, \quad (14)$$

where the activity executions' timestamps are depicted on top of the activities. Despite the first move marks S_3, the first activity is executed at time step 10. In contrast, $\mu^{sync\_t}_{N_3,S_3}(\boldsymbol{\gamma_3}) = 1$ suggests a fast execution of S_3. To compare initial delays, we can consider the *elapsed time since case start* measurement

$$\mu^{elap\_t}_{\mathrm{SN},S}(\boldsymbol{\gamma}, \boldsymbol{\sigma}) = \min_{I \in E^{exec}_{\mathrm{SN},\boldsymbol{\gamma},S}} \left(\min(\boldsymbol{\gamma}^{sync\_t}_{S,[I]})\right) - \pi^{act}(\boldsymbol{\sigma}_0) \quad (15)$$

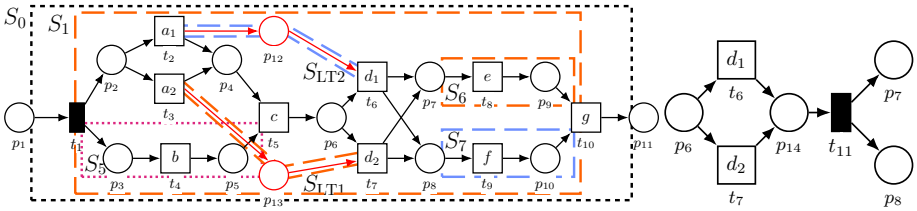
that extracts the time until the very first synchronous execution of a subprocess' transition. However, even if we consider both measurements, there are four major limitations: (i) for loops, we only consider the first delayed start. (ii) The *elapsed time since case start* measurement is monotonically increasing (e.g., $\mu^{elap\_t}_{N_3,(t_4,p_6)}(\gamma_3) = \mu^{elap\_t}_{N_3,S_3}(\gamma_3) + (11-10)$). Thus, the analyst needs to consider that differences can propagate to later parts of the process. (iii) The *synchronous subprocess execution duration* of subprocesses containing a single transition is either zero or undefined. (iv) For non-SESE subprocesses, there can be additional externally caused delays even after a subprocess' execution started. Therefore, one generally needs to consider multiple performance measurements depending on the performance aspect of interest and the subprocess under consideration. For example, to address the third limitation, we might consider the causal predecessors and successors of a transition in the run of the Petri net. For S_8, we would compute the time between firing t_2 and t_7. Yet, for the concurrent subprocesses S_3 and S_4, this means that their duration would be determined by t_1 and possibly t_9 if we consider the transition that removes the last token. In this case, we would get the same measurement value for both subprocesses.

Hypothesis Testing Applying the presented trace measurements, we extract zero ($\perp$) or one value per alignment. As we aim to compare subprocesses based on their executions, we first discard the irrelevant measurements (i.e., measurements having the value $\perp$). For each subprocess and perspective, we thereby obtain a population of real-valued measurements. To detect statistically significant differences between populations, we apply hypothesis testing under the null hypothesis that there is no significant difference. In doing so, we implicitly assume that the measurements are independent of each other. In practice, this assumption might not always hold (e.g., if a single resource handles multiple cases simultaneously). Nevertheless, case independence is a common assumption in the field of PC [10,8,25]. Moreover, we additionally assume that the measurements are approximately normally distributed; yet the populations might have different means and variances. Based on these assumptions, we apply Welch's t-test [29] with a p-value of 0.05 to test if two populations' mean values differ.

Besides the significance of a difference, the *effect size* assesses its strength. For large populations, even small differences in their means can become statistically significant. Therefore, we employ Cohen's d [11] to quantify a difference's effect size. Eventually, we determine the color of each subprocess in the decomposition based on whether there is a significant difference, whether the mean is larger for the left or right process variant, and the Cohen's d value.

4.3 Strategies for Decomposition

While our notion of WF-net decomposition is very generic, it is usually desirable that vertices in the hierarchy define logically coherent and independent subprocesses. In the literature, a structural characterization of such a subprocess is to



(a) WF-net N_2 with long-term dependencies

(b) Modeling Concurrency

Fig. 5: Limitations of a SESE-based decomposition. The WF-net shown in (a) extends the WF-net in Figure 3a by an additional long-term dependency (red). The colored polygons depict the non-trivial SESE subprocesses that constitute the RPST of this model. Using a SESE-based decomposition, we cannot analyze the subprocesses betweeen t_1 and t_5 (i.e., S_2 in Figure 3a) and p_6 and t_{10} (formerly S_3). The language preserving transformation depicted in (b) would allow to further decompose S_3 in N_1.

require that it has a single entry and a single exit vertex [27]. Thereby, each subprocess is self-contained interacting with the remaining net at its entry and exit node. This characterization serves as a basis for the RPST of a WF-net [27]. The RPST is a decomposition comprising a maximal set of vertices satisfying the following conditions: (i) each vertex is a fragment (i.e., a SESE subprocess); (ii) each fragment is *canonical*—that is, there exists no overlapping fragment (not necessarily contained in the RPST) that neither is a proper super- nor subset; and (iii) each vertex is a child of its smallest superset. These conditions imply that the edges of a net are the leaves of its RPST. Such an edge fragment is also called *trivial* (i.e., a fragment of size one). For example, excluding trivial fragments, Figure 3b shows the RPST of the WF-net in Figure 3a. For further details on (computing) RPSTs, we refer the reader to [27,24].

The property of defining self-contained subprocesses makes the RPST a promising decomposition technique for hierarchical PC. Therefore, in our implementation, we leverage an adapted version of a WF-net's RPST. We propose to adapt the RPST computation for the sake of PC because the strict SESE requirement makes the decomposition sensitive. For example, consider the WF-net N_2 depicted in Figure 5a which extends N_1 (Figure 3a) by two additional places. The highlighted places p_{12} and p_{13} couple the choices between a_1 and a_2 and between d_1 and d_2. This significantly changes the set of canonical fragments. Compared to N_1, the added places induce two additional canonical fragments S_{LT1} and S_{LT2} but violate the SESE property of six of the original canonical fragments. Thereby, the resulting RPST would neither allow us to explicitly analyze the choice between a_1 and a_2 nor could we distinguish the two main blocks of the workflow (i.e., the process before and after c).

WF-net Skeleton As shown in Figure 2, we propose an additional preprocessing step of the WF-net to create more SESE subprocesses. The idea is to remove places that connect structurally distant parts of the model preventing a more fine granular decomposition. As places constrain the behavior of the WF-net, removing places instead of transitions yields a net describing a relaxed process. In contrast, removing transitions can result in Petri nets without valid firing sequences. Thus, the RPST obtained by removing places describes a relaxed baseline process that we consider the *skeleton process* of the original model.

In order to later compute its RPST, we need to ensure that the *skeleton process* is a WF-net. Intuitively, this means that we can only remove constraints (i.e., places) that do not break the workflow. Accordingly, we can neither remove the source nor the sink. Moreover, we might only remove a place if, in the resulting net, each node is on a path between the source and the sink. For example, in N_2 (Figure 5a), the only candidates for removal are p_4, p_{12}, and p_{13}.

While removing p_{12} and p_{13} results in the original net that has a fine granular decomposition, this is not the case for p_4. After removing p_4, no further place can be removed resulting in a decomposition similar to the one before (S_{LT1} and S_{LT2} would be extended up to p_2). What distinguishes the places p_{12} and p_{13} from p_4 is that latter place constraints the *local* order of activities. In each firing sequence of the net that contains t_2, the number of steps in which p_4 is marked is less than for p_{12}. Therefore, we propose to preferably remove places that are marked for many steps.

Definition 13 (Place Marking Interval). *Let $N = (P, T, F, l)$ be a safe WF-net and $\mathbb{T} \in \mathcal{B}(T^*)$ be a multiset of valid firing sequences of N. Given a firing sequence $\boldsymbol{\sigma} = \langle t_1, \ldots, t_n \rangle \in \mathbb{T}$, the* token intervals *of a place $p \in P$ are*

$$\mathrm{ti}_{\boldsymbol{\sigma}}(p) = \{(i,j) \,|\, 1 \leq i < j \leq n, p \in t_i^\bullet, p \in {}^\bullet t_j, \forall k (i < k < j) \rightarrow p \notin {}^\bullet t_k \cup t_k^\bullet\}\,. \tag{16}$$

The average number of steps a place $p \in P$ is marked is

$$\mathrm{tt}_{\mathbb{T}}(p) = \mathrm{avg}([j - i \,|\, (i,j) \in \biguplus_{\boldsymbol{\sigma} \in \mathbb{T}} \mathrm{ti}_{\boldsymbol{\sigma}}(p)])\,. \tag{17}$$

This idea is inspired by an ILP Miner [31] variant that prefers adding places to a net where the token is consumed quickly. The safeness of the WF-net is crucial to uniquely match the transitions that produced and consumed a token. In contrast to structural place removal conditions, Definition 13 is not affected by implicit places (i.e., places that do not affect the valid firing sequences). For the sake of efficiency, in our implementation, we re-use the alignments of the event logs to evaluate Equation (17) rather than sampling the net.

Finally, we propose the following decomposition strategy for hierarchical PC: (i) sort the places according to Definition 13 in descending order, (ii) iterate over the sorted places removing places as long as the resulting net remains a WF-net (i.e., remains connected), (iii) compute the RPST of the resulting WF-net skeleton, and (iv) add the edges of the removed places as subprocesses under the root. Consider a set of firing sequences for N_2 (Figure 5a) such that p_{12} and p_{13}

are marked at least once. Depending on the position of t_4 in the firing sequence, the places p_{12} and p_{13} are marked for two or three steps. Each time one of these places is marked, the place p_4 is marked for at least one step less. Therefore, one can show that we have $\mathrm{tt}_S(p_4) < \mathrm{tt}_S(p_{12})$ or $\mathrm{tt}_S(p_4) < \mathrm{tt}_S(p_{13})$ (not necessarily both). Removing any other place would violate the WF-net constraint. Thus, we first remove p_{12} or p_{13} after which p_4 cannot be removed anymore.

While there are various ways to combine the RPST and the places removed, we simply add the associated sets of edges under the root. In contrast, one could add them under the smallest subprocess that contains the place's adjacent transitions. Adding the edges under the root has the advantage that the remaining tree is a proper RPST for the block skeleton. Assuming that the set of edges removed is usually small, we expect this design decision to have a minor impact.

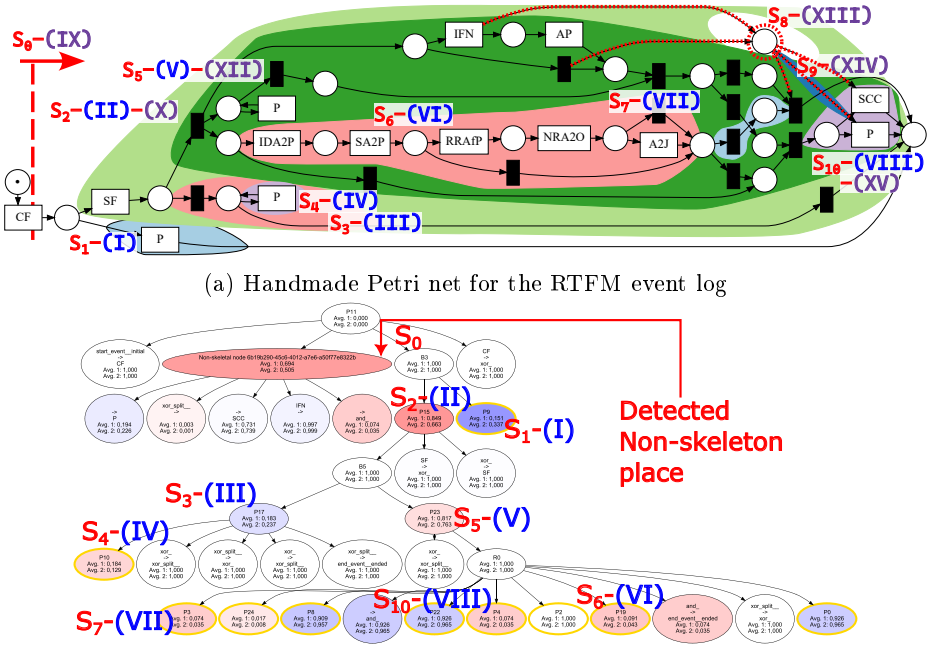
Our place removal approach might change the language of the WF-sytem net. Fortunately, this does not affect the analysis since we compute the alignments with respect to the original net. Besides, there are other language-preserving transformations that might improve a net's SESE decomposability. Figure 5b shows an example using a silent transition as a concurrency split. Since this does not change the model in terms of its language, we currently leave this to the modeler. Besides, approaches that transform other model notations into Petri nets often natively use silent transitions to create SESE subprocesses.

5 Case Study

We implemented our approach as a ProM plugin, available in the ProM nightly builds[3], and evaluate it in a case study on the real-life *Road Traffic Fine Management* (RTFM) event log [19]. To the best of our knowledge, from the model-based approaches discussed in Section 2, only Wynn et al. [30] provide the publicly available implementation *Profiler 3d*. Therefore, we qualitatively compare the results to *Profiler 3d*. Like existing works that consider this log [10,8,25], we split it into low fine cases—that is, the initial fine is less than €50—and high fine cases—that is, the fine is larger or equal to €50.

Based on the original log, we created a highly fitting BPMN model and transformed it into the Petri net depicted in Figure 6a. On a high level, a fine is first created (CF) and then either paid (P) or sent (SF). In the latter case, the offender may either pay it, or the case enters a subprocess concerned with additional penalization (AP) or appealing (IDA2P). We duplicated the payment transition (P) to precisely represent highly frequent variants where the fine was paid. Thereby, we can explicitly analyze the different ways of paying a fine. Finally, we add an implicit place (i.e., a place that does not constrain the behavior), highlighted red in Figure 6a. It creates additional boundary nodes in the additional penalization subprocess. Thereby, no proper fragment can contain this subprocess making it difficult to decompose. Despite the place being implicit, it is well-suited to investigate the time between inserting the fine notification (IFN) and paying (P) or collecting (SCC) it.

[3] `https://promtools.org/prom-6-nightly-builds/`



(a) Handmade Petri net for the RTFM event log

(b) Decomposition of the Petri net in (a) enhanced by differences in a subprocess' conditional activation likelihood

Fig. 6: Comparing low and high fine cases in terms of differences in the control flow. Subfigure (a) depicts a Petri net of the process. The decomposition depicted in (b) shows the differences detected using the *conditional subprocess activation* measurement. For significant differences, the shade depicts the effect size, and the color illustrates whether subprocesses are activated more likely for low (blue) or high (red) fines. Besides, we collapsed uninteresting subtrees (yellow outlines). The red annotations relate the nodes in the decomposition of the Petri net and to the descriptions in the text.

5.1 Results

Figure 6b shows the hierarchical decomposition of the Petri net in Figure 6a, and Figure 7 depicts the output of *Profiler 3d*. Our proposed decomposition strategy correctly identifies the additionally introduced place as a non-skeleton place. Next, we investigate control flow differences. Since the Petri net in Figure 6a contains a considerable number of choices, we employ the *conditional subprocess activation* measurement. Thereby, we can analyze subprocesses proceeding a choice irrespective of the choice's likelihood. Figure 6b shows that, for low fines, it is considerably more likely that we observe an immediate payment (**D**iagnostic D(I)). *Profiler 3d* also detect this difference. In contrast, high fines are often first sent (SF) to the offender (D(II)). After a fine was sent, an offender might pay, or—an interesting, frequent option present in the event log—the case

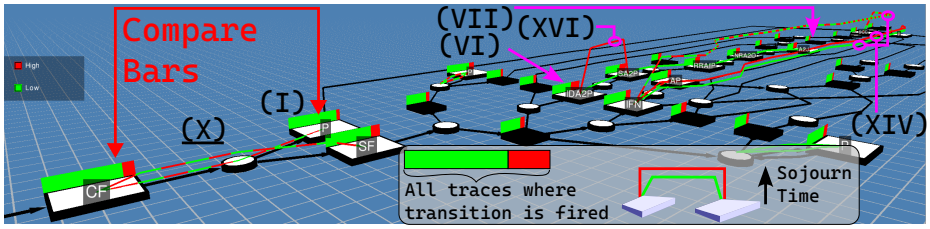


Fig. 7: Results obtained using *Profiler 3d* by Wynn et al. [30]. Due to limitations of the tool, the transitions depict *conditional firing likelihoods*. Given that the transition fires, the bar shows the fraction of low (green) and high (fine) cases. Detecting decision likelihood differences therefore requires comparing this fraction with the annotation of the transition labeled CF which is activated by all cases. The colored arcs on the edges depict the *median sojourn times* between pairs of activities, where the height of an edge scales with the sojourn time observed. The annotations refer to the differences described in the text. If the conclusions drawn using our method and the depicted approach differ, we underline the annotation.

may end. While latter option is more probable for low fines (D(III)), receiving a payment after sending the fine is more likely for high fines (D(IV)). Moreover, for high fines, the likelihood is higher that the case neither ends nor is paid but enters an extended subprocess (D(V)). This subprocess includes three concurrent strands of work. On the prefecture's site, the delayed payment leads to a notification (IFN) and an additional penalty (AP). Moreover, the prefecture might receive an appeal (IDA2P). On the offenders' side, they might decide to start paying the fine (possibly in multiple steps) even though an additional penalty will already be added due to the delay. For this subprocess, we observe two main differences. First, appeals are more likely for high fines (D(VI)). Second, given that an appeal was made, it is more probable to be successful for high fine cases (D(VII)). In particular, this difference might be quite interesting for stakeholders. Consequently, for low fine cases, it is slightly more likely that one eventually observes a payment or credit collection (D(VIII)). From the preceding differences, D(VI) and D(VII) are most noticeable in Figure 7. For the remaining differences, the fact that one needs to visually compare fractions of cases makes them very difficult to detect.

Performance Analyzing performance differences, we investigate the *synchronous subprocess execution duration* as well as the *elapsed time since case start*. Figure 8 depicts the projection of measurement differences onto the decomposition. First, high fine cases tend to have a longer overall cycle time (D(IX)). The average duration between a fine's creation and its last observed activity is approximately a year, while it is 263 days for low fine cases. Next, is noteworthy that, on average, low fines are sent 18 days earlier (D(X))—that is, 72 versus 90

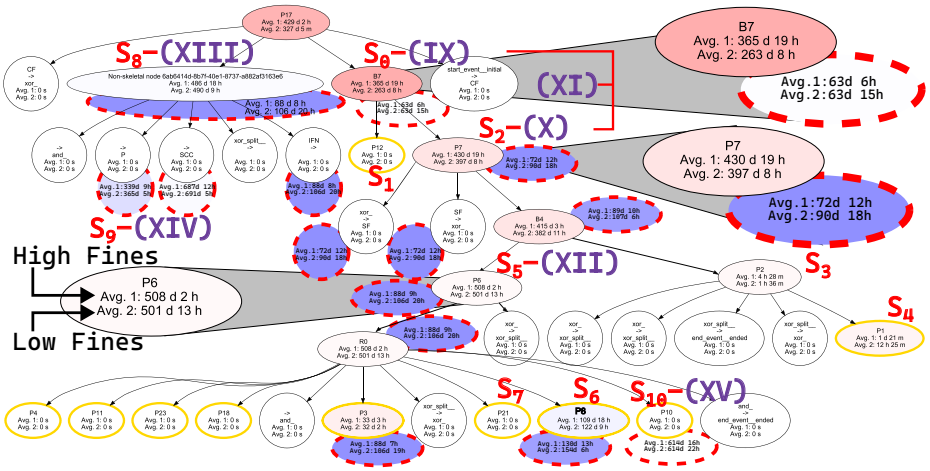


Fig. 8: Hierarchical comparison of low (avg. 2) and high fine (avg. 1) cases in terms of performance. In this decomposition of the Petri net in Figure 6a, we collapsed uninteresting subtrees (yellow outlines). For each subprocess (i.e., vertex), we compare its *synchronous subprocess execution duration*. For selected subprocesses (red dashed), we also depict the *elapsed time since case start*. The annotations depict the corresponding subprocesses in Figure 6a as well as labels for the diagnostics. We further enlarged important subprocesses (gray).

days. Since there is no difference regarding the *elapsed time since case start* for CF, we would expect that Figure 7 shows a sojourn time difference for the edge (CF, SF). Yet, this is not the case. Moreover, one can attribute all significant differences with respect to the elapsed time in the subtree rooted at S_2 to D(X). However, this requires an additional analysis of the average values. Due to the hierarchical decomposition, we can also easily see that the difference in the average duration of the subprocess S_2, entered by sending the fine, is smaller than for S_0—that is, 33 days vs 98 days (D(XI)). This suggests that the time savings for low fines are due to immediate payments (S_1). In fact, there is almost no difference regarding the duration of the appealing and additional fining subprocess S_5 (D(XII)). Finally, the introduced non-skeleton place gives additional insight into the time between inserting the notification (IFN) and the final payment (P) or credit collection (SCC). It shows that this duration does not differ significantly (D(XIII)). However, low fine cases that mark this place (i.e., cases that enter S_5) tend to be paid slightly earlier shown by a shorter *elapsed time since case start* (D(XIV)). Nevertheless, the overall time until the last payment is made or is collected forcefully, does not differ significantly (D(XV)). Finally, *Profiler 3d* detects increased sojourn times between IDA2P and SA2P for high fines. Similar to D(XIV), it also indicates that after a fine is inserted (IFN), the final payment is made slightly earlier for low fines (D(XVI)).

5.2 Discussion

Our evaluation shows that the *conditional subprocess activation* measurement is well-suited to compare process variants that considerably differ in the likelihoods of choices. In doing so, the hierarchical approach allows us to reason about subprocesses on different levels of granularity. For example, we not only observe that appeals are more likely for high fine cases (D(VI)), but also that these appeals a slightly more successful (D(VII)). Combined with the ability to consider duplicate labels, we could also compare the frequency of payments in different process contexts. Considering the performance, we identified differences in the execution duration of different subprocesses. Moreover, we gained additional insight by comparing subprocesses among the hierarchy (comp. D(X)). However, this requires the analyst to reason on top of the output of the method. Besides, one needs to consider and relate multiple performance metrics to paint the full picture. Finally, the proposed reduction of the net to its *skeleton process* enabled us to add an implicit place without negatively affecting the decomposition. This place could then be used to investigate a specific aspect of the process.

Compared to *Profiler 3d* [30], our approach shows differences more clearly. Moreover, the proposed method allows to reason about larger subprocesses and automatically analyzes subprocesses on different levels of granularity. While *Profiler 3d* also supports hierarchical Petri nets as input, it requires that the hierarchy is specified upfront.

6 Conclusion

In this paper, we leverage a shared Petri net to compare two event logs in a hierarchical manner. To this end, we decompose the model into a hierarchy of subprocesses. For each subprocess and case in the event logs, we then extract intervals during which the subprocess is considered being under execution. Based on these intervals, measurements that assess different aspects of the process can be defined. In this paper, we exemplify measurements that assess differences in the control flow, performance, and conformance. Furthermore, we propose a decomposition strategy based on Refined Process Structure Trees. In doing so, we introduce a new preprocessing step to improve the decomposability of the net. The case study shows how the proposed method allows to reason on larger subprocesses but also to drill down on interesting details.

For future work, we plan to investigate process-aware measurements that do not consider each case in isolation. The guidance provided by a good model might help to alleviate this limitation of existing process comparison approaches. Moreover, we intend to make the approach more interactive by allowing the user to assess performance metrics of flexibly defined sections during runtime.

Acknowledgments. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy-EXC-2023 Internet of Production-390621612. We also thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Process equivalence: Comparing two process models based on observed behavior. In: Business Process Management. pp. 129–144 (2006). https://doi.org/10.1007/11841760_10
2. van der Aalst, W.M.P.: Decomposing petri nets for process mining: A generic approach. Distributed and Parallel Databases **31**(4), 471–507 (2013). https://doi.org/10.1007/s10619-013-7127-5
3. van der Aalst, W.M.P., Stahl, C.: Information Systems. In: Modeling Business Processes: A Petri Net-Oriented Approach (2011). https://doi.org/10.7551/mitpress/8811.003.0003
4. van der Aalst, W.M.P., Tacke Genannt Unterberg, D., Denisov, V., Fahland, D.: Visualizing token flows using interactive performance spectra. In: Application and Theory of Petri Nets and Concurrency. pp. 369–380. Lecture Notes in Computer Science (2020). https://doi.org/10.1007/978-3-030-51831-8_18
5. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards robust conformance checking. In: Business Process Management Workshops. pp. 122–133 (2011). https://doi.org/10.1007/978-3-642-20511-8_11
6. Andrews, K., Wohlfahrt, M., Wurzinger, G.: Visual graph comparison. In: 13th International Conference Information Visualisation. pp. 62–67 (2009). https://doi.org/10.1109/IV.2009.108
7. van Beest, N.R.T.P., Dumas, M., García-Bañuelos, L., La Rosa, M.: Log delta analysis: Interpretable differencing of business process event logs. In: Business Process Management. pp. 386–405 (2015). https://doi.org/10.1007/978-3-319-23063-4_26
8. Bolt, A., de Leoni, M., van der Aalst, W.M.P.: A visual approach to spot statistically-significant differences in event logs based on process metrics. In: Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 9694, pp. 151–166 (2016). https://doi.org/10.1007/978-3-319-39696-5_10
9. Bouvier, P., Garavel, H., Ponce-de León, H.: Automatic decomposition of petri nets into automata networks – a synthetic account. In: Application and Theory of Petri Nets and Concurrency. pp. 3–23 (2020). https://doi.org/10.1007/978-3-030-51831-8_1
10. Cecconi, A., Augusto, A., Di Ciccio, C.: Detection of statistically significant differences between process variants through declarative rules. In: Business Process Management Forum. pp. 73–91 (2021). https://doi.org/10.1007/978-3-030-85440-9_5
11. Cohen, J.: Statistical Power Analysis for the Behavioral Sciences, chap. 2, pp. 20–27. 2 edn. (1988). https://doi.org/10.4324/9780203771587
12. Cordes, C., Vogelgesang, T., Appelrath, H.J.: A generic approach for calculating and visualizing differences between process models in multidimensional process mining. In: International Conference on Business Process Management. pp. 383–394. Springer (2014). https://doi.org/10.1007/978-3-319-15895-2_32
13. Eshuis, R.: Translating safe petri nets to statecharts in a structure-preserving way. In: FM 2009: Formal Methods. pp. 239–255 (2009). https://doi.org/10.1007/978-3-642-05089-3_16
14. Ivanov, S.Y., Kalenkova, A.A., van der Aalst, W.M.P.: BPMNDiffViz: A tool for BPMN models comparison. BPM reports **1507** (2015)
15. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. In: Proceedings of the ACM SIGPLAN 1994 confer-

ence on Programming language design and implementation. pp. 171–185 (1994). https://doi.org/10.1145/773473.178258
16. Karatkevich, A., Andrzejewski, G.: Hierarchical decomposition of petri nets for digital microsystems design. In: 2006 International Conference - Modern Problems of Radio Engineering, Telecommunications, and Computer Science. pp. 518–521 (2006). https://doi.org/10.1109/TCSET.2006.4404613
17. Kriglstein, S., Wallner, G., Rinderle-Ma, S.: A visualization approach for difference analysis of process models and instance traffic. In: Business Process Management, pp. 219–226 (2013). https://doi.org/10.1007/978-3-642-40176-3_18
18. Küster, J.M., Gerth, C., Förster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In: Business Process Management, Lecture Notes in Computer Science, vol. 5240, pp. 244–260 (2008). https://doi.org/10.1007/978-3-540-85758-7_19
19. de Leoni, M., Mannhardt, F.: Road traffic fine management process (2015). https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5
20. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Hierarchical conformance checking of process models based on event logs. In: Application and Theory of Petri Nets and Concurrency. pp. 291–310 (2013). https://doi.org/10.1007/978-3-642-38697-8_16
21. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Information Systems **46**, 102–122 (2014). https://doi.org/10.1016/j.is.2014.04.003
22. Nguyen, H., Dumas, M., La Rosa, M., ter Hofstede, A.H.M.: Multi-perspective comparison of business process variants based on event logs. In: Conceptual Modeling. pp. 449–459 (2018). https://doi.org/10.1007/978-3-030-00847-5_32
23. Pini, A., Brown, R., Wynn, M.T.: Process visualization techniques for multi-perspective process comparisons. In: Asia-Pacific Conference on Business Process Management. pp. 183–197. Springer (2015). https://doi.org/10.1007/978-3-319-19509-4_14
24. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: Web Services and Formal Methods. pp. 25–41 (2011). https://doi.org/10.1007/978-3-642-19589-1_2
25. Taymouri, F., La Rosa, M., Carmona, J.: Business process variant analysis based on mutual fingerprints of event logs. In: Advanced Information Systems Engineering. pp. 299–318 (2020). https://doi.org/10.1007/978-3-030-49435-3_19
26. Taymouri, F., Rosa, M.L., Dumas, M., Maggi, F.M.: Business process variant analysis: Survey and classification. Knowledge-Based Systems **211**, 106557 (2021). https://doi.org/10.1016/j.knosys.2020.106557
27. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Business Process Management. pp. 100–115 (2008). https://doi.org/10.1007/978-3-540-85758-7_10
28. Vidgof, M., Djurica, D., Bala, S., Mendling, J.: Interactive log-delta analysis using multi-range filtering. Software and Systems Modeling **21**, 847–868 (2022). https://doi.org/10.1007/s10270-021-00902-0
29. Welch, B.L.: The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. Biometrika **34**(1-2), 28–35 (1947). https://doi.org/10.1093/biomet/34.1-2.28
30. Wynn, M.T., Poppe, E., Xu, J., ter Hofstede, A.H.M., Brown, R., Pini, A., van der Aalst, W.M.P.: Processprofiler3d: A visualisation framework for log-based process performance comparison. Decision Support Systems **100**, 93–108 (2017). https://doi.org/10.1016/j.dss.2017.04.004

31. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Ilp-based process discovery using hybrid regions. In: Algorithms & Theories for the Analysis of Event Data. pp. 47–61. CEUR Workshop Proceedings (2015)
32. Zhong, C., He, W., Li, Z., Wu, N., Qu, T.: Deadlock analysis and control using petri net decomposition techniques. Information Sciences **482**, 440–456 (2019). https://doi.org/10.1016/j.ins.2019.01.029