Decomposing Petri Nets for Process Mining –A Generic Approach–

Wil M.P. van der Aalst[1,2]

[1] Architecture of Information Systems, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{w.m.p.v.d.aalst}@tue.nl
[2] International Laboratory of Process-Aware Information Systems, National Research University Higher School of Economics (HSE), 33 Kirpichnaya Str., Moscow, Russia.

Abstract. The practical relevance of *process mining* is increasing as more and more event data become available. Process mining techniques aim to discover, monitor and improve real processes by extracting knowledge from event logs. The two most prominent process mining tasks are: (i) *process discovery*: learning a process model from example behavior recorded in an event log, and (ii) *conformance checking*: diagnosing and quantifying discrepancies between observed behavior and modeled behavior. The increasing volume of event data provides both opportunities and challenges for process mining. Existing process mining techniques have problems dealing with large event logs referring to many different activities. Therefore, we propose a *generic approach to decompose process mining problems*. The decomposition approach is generic and can be combined with different existing process discovery and conformance checking techniques. It is possible to split computationally challenging process mining problems into many smaller problems that can be analyzed easily and whose results can be combined into solutions for the original problems.

1 Introduction

Process mining aims to *discover, monitor and improve real processes by extracting knowledge from event logs* readily available in today's information systems [1]. Starting point for any process mining task is an *event log*. Each event in such a log refers to an *activity* (i.e., a well-defined step in some process) and is related to a particular *case* (i.e., a *process instance*). The events belonging to a case are *ordered* and can be seen as one "run" of the process. It is important to note that an event log contains only *example behavior*, i.e., we cannot assume that all possible runs have been observed. In fact, an event log often contains only a fraction of the possible behavior [1].

The growing interest in process mining is illustrated by the *Process Mining Manifesto* [41] recently released by the *IEEE Task Force on Process Mining*. This manifesto is supported by 53 organizations and 77 process mining experts

contributed to it. The active contributions from end-users, tool vendors, consultants, analysts, and researchers illustrate the significance of process mining as a bridge between data mining and business process modeling.

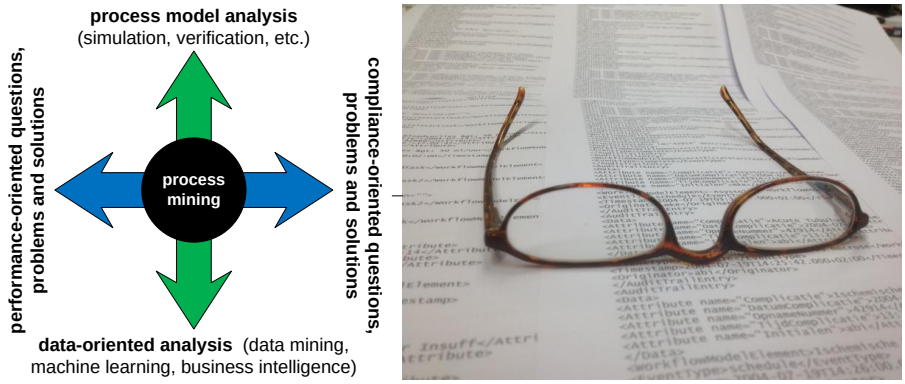


Fig. 1. Process mining combines process model analysis with data analysis to answer performance and compliance related questions (left-hand-side). Process models can be seen as the "glasses" required to be able to "see" event data (right-hand-side).

Figure 1 positions process mining. Traditional *data-oriented analysis* approaches such as data mining [38] and machine learning [49] do not consider processes, i.e., analysis focuses on particular decisions or patterns rather than the end-to-end processes. In contrast, *Business Process Management* (BPM) and *Workflow Management* (WFM) approaches focus on the analysis and improvement of end-to-end processes using knowledge from information technology and knowledge from management sciences [34, 65]. Process models play a central role in BPM/WFM. Examples of *process model analysis* approaches are simulation (for "what if" analysis) and verification (to find design errors). As shown in Figure 1, process mining combines both worlds to answer both performance and compliance related questions. The desire to link data and process is reflected by terms such as Business Process Intelligence (BPI) [36, 26]. However, only recently techniques and software have become available to systematically relate process models and event data [1].

Industry reports such as [46] and scientific studies [40] describe the incredible growth of data. The term "big data" illustrates the spectacular growth of data and the potential economic value of such data in different industry sectors. Most of the data that are generated refer to *events*, e.g., transactions in some financial system and actions of some automated system (e.g., X-ray machines, luggage-handling systems, or sensor networks). The incredible growth of event data provides new opportunities for process analysis. As more and more actions of people, organizations, and devices are recorded, there are ample opportunities to analyze processes based on the footprints they leave in event logs. In fact, we

believe that the analysis of purely *hand-made* process models will become less important given the omnipresence of event data.

The incredible growth of event data is also posing new challenges [58]. As event logs grow, process mining techniques need to become more efficient and highly scalable. Moreover, torrents of event data need to be distributed over multiple databases and large process mining problems need to be distributed over a network of computers (e.g., grids). Consider for example Philips Healthcare, a provider of medical systems that are often connected to the Internet to enable logging, maintenance, and remote diagnostics. More than 1500 Cardio Vascular (CV) systems (i.e., X-ray machines) are remotely monitored by Philips. On average each CV system produces 15,000 events per day, resulting in 22.5 million events per day for just their CV systems. The events are stored for many years and have many attributes. The error logs of ASML's lithography systems have similar characteristics and also contain about 15,000 events per machine per day. These numbers illustrate the fact that today's organizations are already storing terabytes of event data. Process mining techniques aiming at very precise results (e.g., guarantees with respect to the accuracy of the model or diagnostics), quickly become intractable when dealing with such real-life event logs.

Earlier applications of process mining in organizations such as Philips and ASML, show that there are various challenges with respect to performance (response times), capacity (storage space), and interpretation (discovered process models may be composed of thousands of activities). In this paper, we present a general approach to decompose existing process discovery and conformance checking problems.

Petri nets are often used in the context of process mining. Various algorithms employ Petri nets as the internal representation used for process mining. Examples are the region-based process discovery techniques [8, 16, 59, 24, 64], the α algorithm [9], and various conformance checking techniques [10, 50, 51, 57]. Other techniques use alternative internal representations (C-nets, heuristic nets, etc.) that can easily be converted to (labeled) Petri nets [1].

The process mining spectrum is quite broad and includes techniques like process discovery, conformance checking, model repair, role discovery, bottleneck analysis, predicting the remaining flow time, and recommending next steps. In this paper, we focus on *distributing* the following two main process mining problems:

- *Process discovery problem*: Given an event log consisting of a collection of traces (i.e., sequences of events), construct a Petri net that "adequately" describes the observed behavior.
- *Conformance checking problem*: Given an event log and a Petri net, diagnose the differences between the observed behavior (i.e., traces in the event log) and the modeled behavior (i.e., firing sequences of the Petri net).

Both problems are formulated in terms of Petri nets. However, other process notations could be used, e.g., BPMN models, BPEL specifications, UML activity diagrams, Statecharts, C-nets, and heuristic nets [7]. In fact, also different types of Petri nets can be employed, e.g., safe Petri nets, labeled Petri nets, free-choice

Petri nets, etc. Therefore, we would like to stress that the results presented in this paper are as general as possible and do not depend on specific Petri-net properties. Our approach is able to decompose any process composed using the most frequently used workflow patterns [7] without assuming a particular notation.

Process discovery and conformance checking are related problems. This becomes evident when considering *genetic* process discovery techniques [48, 19]. In each generation of models generated by the genetic algorithm, the conformance of every individual model in the population needs to be assessed (the so-called fitness evaluation). Models that fit well with the event log are used to create the next generation of candidate models. Poorly fitting models are discarded. The performance of genetic process discovery techniques will only be acceptable if dozens of conformance checks can be done per second (on the whole event log). This illustrates the need for efficient process mining techniques.

Dozens of process discovery [1, 8, 9, 14, 33, 16, 23, 24, 27, 35, 48, 59, 63, 64] and conformance checking [5, 10, 11, 13, 21, 28, 35, 50, 51, 57, 62] approaches have been proposed in literature. Despite the growing maturity of these approaches, the quality and efficiency of existing techniques leave much to be desired. State-of-the-art techniques still have problems dealing with large and/or complex event logs and process models. Therefore, we propose a *generic divide and conquer approach for process mining*:

- For *conformance checking*, we decompose the process model into smaller partly overlapping model fragments. If the decomposition is done properly, then any trace that fits into the overall model also fits all of the smaller model fragments and vice versa. Hence, metrics such as the fraction of fitting cases can be computed by only analyzing the smaller model fragments.
- To decompose *process discovery*, we split the set of activities into a collection of partly overlapping activity sets. For each activity set, we project the log onto the relevant events and discover a model fragment. The different fragments are glued together to create an overall process model. Again it is guaranteed that all traces in the event log that fit into the overall model also fit into the model fragments and vice versa.

Hence, we can decompose the two types of process mining tasks in a *generic* manner. The approach presented in this paper is generic because: (1) it does not assume a particular conformance checking or discovery algorithm, (2) different decompositions can be used, all resulting in valid results, and (c) the class of processes is not limited to particular types of Petri nets (e.g., free-choice nets or workflow nets). For example, the approach can be applied to a range of process discovery algorithms using different types of decompositions.

Our generic approach can also be used to *distribute* process mining problems over a network of computers (e.g., a grid or cloud infrastructure). The results in this paper can be viewed as a generalization of our earlier approach based on "passages" [2]. However, in this paper we do not use the notion of passages and many of the restrictions imposed in [2] no longer apply.

The remainder of this paper is organized as follows. Section 2 introduces various preliminaries (Petri nets, event logs, etc.). Section 3 discusses quality criteria for process mining and introduces the notion of alignments to compute the level of conformance. Section 4 defines various decomposition notions, e.g., valid decompositions of an overall model into model fragments. The section defines the requirements ensuring that conformance checking can be decomposed. Section 5 shows that also process discovery problems can be decomposed. The decomposition approach is independent of the underlying discovery technique and is proven to be correct. Section 6 provides pointers to concrete instantiations of the approach using the process mining framework *ProM*. Related work is discussed in Section 7. Section 8 concludes the paper.

2 Preliminaries: Petri Nets and Event Logs

This section introduces basic concepts related to Petri nets and event logs.

2.1 Multisets, Functions, and Sequences

Multisets are used to represent the state of a Petri net and to describe event logs where the same trace may appear multiple times.

$\mathcal{B}(A)$ is the set of all multisets over some set A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times element $a \in A$ appears in b. Some examples: $b_1 = [\,]$, $b_2 = [x, x, y]$, $b_3 = [x, y, z]$, $b_4 = [x, x, y, x, y, z]$, $b_5 = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. b_1 is the empty multiset, b_2 and b_3 both consist of three elements, and $b_4 = b_5$, i.e., the ordering of elements is irrelevant and a more compact notation may be used for repeating elements.

The standard set operators can be extended to multisets, e.g., $x \in b_2$, $b_2 \uplus b_3 = b_4$, $b_5 \setminus b_2 = b_3$, $|b_5| = 6$, etc. $\{a \in b\}$ denotes the set with all elements a for which $b(a) \geq 1$. $[f(a) \mid a \in b]$ denotes the multiset where element $f(a)$ appears $\sum_{x \in b | f(x) = f(a)} b(x)$ times.

A relation $R \subseteq X \times Y$ is a set of pairs. $\pi_1(R) = \{x \mid (x, y) \in R\}$ is the domain of R, $\pi_2(R) = \{y \mid (x, y) \in R\}$ is the range of R, and $\omega(R) = \pi_1(R) \cup \pi_2(R)$ are the elements of R. For example, $\omega(\{(a, b), (b, c)\}) = \{a, b, c\}$.

$f \in X \not\rightarrow Y$ is a partial function with domain $dom(f) \subseteq X$ and range $rng(f) = \{f(x) \mid x \in X\} \subseteq Y$. $f \in X \rightarrow Y$ is a total function, i.e., $dom(f) = X$. A partial function $f \in X \not\rightarrow Y$ is injective if $f(x_1) = f(x_2)$ implies $x_1 = x_2$ for all $x_1, x_2 \in dom(f)$.

Definition 1 (Function Projection). *Let $f \in X \not\rightarrow Y$ be a (partial) function and $Q \subseteq X$. $f{\restriction_Q}$ is the function projected on Q: $dom(f{\restriction_Q}) = dom(f) \cap Q$ and $f{\restriction_Q}(x) = f(x)$ for $x \in dom(f{\restriction_Q})$.*

The projection can also be used for bags, e.g., $[x^3, y, z^2]{\restriction_{\{x,y\}}} = [x^3, y]$.

$\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in X^*$ denotes a sequence over X of length n. $\langle\,\rangle$ is the empty sequence. Sequences are used to represent paths in a graph and traces in an event log. $\sigma_1 \cdot \sigma_1$ is the concatenation of two sequences and $\sigma{\restriction_Q}$ is the projection of σ on Q.

Definition 2 (Sequence Projection). *Let X be a set and $Q \subseteq X$ one of its subsets.* $\restriction_Q \in X^* \rightarrow Q^*$ *is a projection function and is defined recursively: (1)* $\langle\,\rangle\restriction_Q = \langle\,\rangle$ *and (2) for* $\sigma \in X^*$ *and* $x \in X$*:*

$$(\langle x\rangle \cdot \sigma)\restriction_Q = \begin{cases} \sigma\restriction_Q & \text{if } x \notin Q \\ \langle x\rangle \cdot \sigma\restriction_Q & \text{if } x \in Q \end{cases}$$

So $\langle y, z, y\rangle\restriction_{\{x,y\}} = \langle y, y\rangle$. Functions can also be applied to sequences: if $dom(f) = \{x, y\}$, then $f(\langle y, z, y\rangle) = \langle f(y), f(y)\rangle$.

Definition 3 (Applying Functions to Sequences). *Let $f \in X \not\rightarrow Y$ be a partial function. f can be applied to sequences of X using the following recursive definition (1)* $f(\langle\,\rangle) = \langle\,\rangle$ *and (2) for* $\sigma \in X^*$ *and* $x \in X$*:*

$$f(\langle x\rangle \cdot \sigma) = \begin{cases} f(\sigma) & \text{if } x \notin dom(f) \\ \langle f(x)\rangle \cdot f(\sigma) & \text{if } x \in dom(f) \end{cases}$$

Summation is defined over multisets and sequences, e.g., $\sum_{x \in \langle a,a,b,a,b\rangle} f(x) = \sum_{x \in [a^3,b^2]} f(x) = 3f(a) + 2f(b)$.

2.2 Petri Nets

We use Petri nets to formalize our main ideas and to demonstrate their correctness. However, as mentioned in Section 1 the results presented in the paper can be adapted for various other process modeling notations (BPMN models, BPEL specifications, UML activity diagrams, Statecharts, C-nets, heuristic nets, etc.). By using Petri nets we minimize the notational overhead allowing us the focus on the key ideas.

Definition 4 (Petri Net). *A Petri net is a tuple $N = (P, T, F)$ with P the set of places, T the set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation.*

Figure 2 shows a Petri net $N = (P, T, F)$ with $P = \{start, c1, \ldots, c9, end\}$, $T = \{t1, t2, \ldots, t11\}$, and $F = \{(start, t1), (t1, c1), (t1, c2), \ldots, (t11, end)\}$. The state of a Petri net, called *marking*, is a multiset of places indicating how many *tokens* each place contains. $[start]$ is the initial marking shown in Figure 2. Another potential marking is $[c1^{10}, c2^5, c4^5]$. This is the state with ten tokens in $c1$, five tokens in $c2$, and five tokens in $c4$.

Definition 5 (Marking). *Let $N = (P, T, F)$ be a Petri net. A marking M is a multiset of places, i.e., $M \in \mathcal{B}(P)$.*

A Petri net $N = (P, T, F)$ defines a directed graph with nodes $P \cup T$ and edges F. For any $x \in P \cup T$, ${}^{N}\bullet x = \{y \mid (y, x) \in F\}$ denotes the set of input nodes and $x^{N}\bullet = \{y \mid (x, y) \in F\}$ denotes the set of output nodes We drop the superscript N if it is clear from the context.

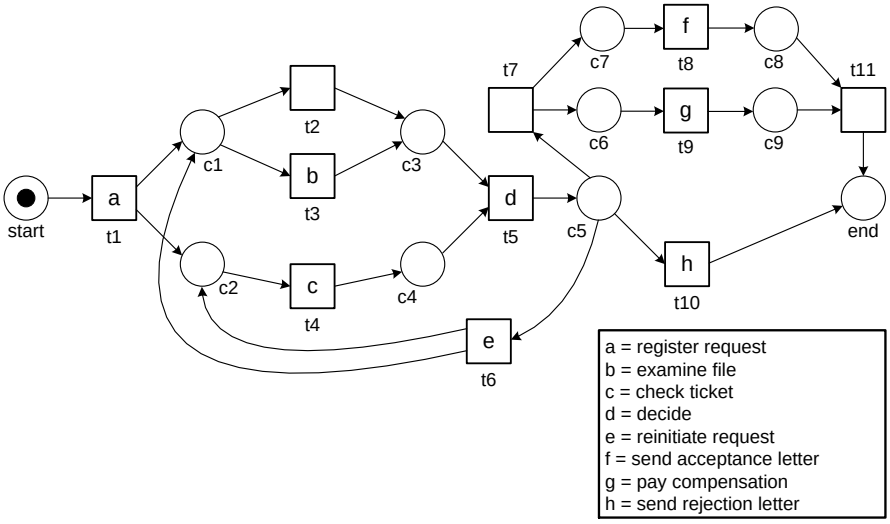


Fig. 2. A labeled Petri net.

A transition $t \in T$ is *enabled* in marking M of net N, denoted as $(N, M)[t\rangle$, if each of its input places $\bullet t$ contains at least one token. Consider the Petri net N in Figure 2 with $M = [c3, c4]$: $(N, M)[t5\rangle$ because both input places of $t5$ are marked.

An enabled transition t may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. Formally: $M' = (M \setminus \bullet t) \uplus t\bullet$ is the marking resulting from firing enabled transition t in marking M of Petri net N. $(N, M)[t\rangle(N, M')$ denotes that t is enabled in M and firing t results in marking M'. For example, $(N, [start])[t1\rangle(N, [c1, c2])$ and $(N, [c3, c4])[t5\rangle(N, [c5])$ for the net in Figure 2.

Let $\sigma = \langle t_1, t_2, \ldots, t_n\rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there is a set of markings $M_0, M_1, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $(N, M_i)[t_{i+1}\rangle(N, M_{i+1})$ for $0 \leq i < n$. A marking M' is *reachable* from M if there exists a σ such that $(N, M)[\sigma\rangle(N, M')$. For example, $(N, [start])[\sigma\rangle(N, [end])$ with $\sigma = \langle t1, t3, t4, t5, t10\rangle$ for the net in Figure 2.

Definition 6 (Labeled Petri Net). *A labeled Petri net $N = (P, T, F, l)$ is a Petri net (P, T, F) with labeling function $l \in T \not\to \mathcal{U}_A$ where $\mathcal{U}_A$ is some universe of activity labels. Let $\sigma_v = \langle a_1, a_2, \ldots, a_n\rangle \in {\mathcal{U}_A}^*$ be a sequence of activities. $(N, M)[\sigma_v \rhd (N, M')$ if and only if there is a sequence $\sigma \in T^*$ such that $(N, M)[\sigma\rangle(N, M')$ and $l(\sigma) = \sigma_v$ (cf. Definition 3).*

If $t \notin dom(l)$, it is called invisible. An occurrence of visible transition $t \in dom(l)$ corresponds to observable activity $l(t)$. The Petri net in Figure 2 is labeled. The labeling function is defined as follows: $dom(l) = \{t1, t3, t4, t5, t6, t8, t9, t10\}$, $l(t1) = a$ (a is a shorthand for "register request"), $l(t3) = b$ ("examine

file"), $l(t4) = c$ ("check ticket"), $l(t5) = d$ ("decide"), $l(t6) = e$ ("reinitiate request"), $l(t8) = f$ ("send acceptance letter"), $l(t9) = g$ ("pay compensation"), and $l(t10) = h$ ("send rejection letter"). Unlabeled transitions correspond to so-called "silent actions", i.e., transitions $t2$, $t7$, and $t11$ are unobservable.

Given the Petri net N in Figure 2: $(N, [start])[\sigma_v \rhd (N, [end])$ for $\sigma_v = \langle a, c, d, f, g \rangle$ because $(N, [start])[\sigma\rangle(N, [end])$ with $\sigma = \langle t1, t2, t4, t5, t7, t8, t9, t11 \rangle$ and $l(\sigma) = \sigma_v$.

In the context of process mining, we always consider processes that start in an initial state and end in a well-defined end state. For example, given the net in Figure 2 we are interested in so-called *complete* firing sequences starting in $M_{init} = [start]$ and ending in $M_{final} = [end]$. Therefore, we define the notion of a *system net*.

Definition 7 (System Net). *A system net is a triplet $SN = (N, M_{init}, M_{final})$ where $N = (P, T, F, l)$ is a labeled Petri net, $M_{init} \in \mathcal{B}(P)$ is the initial marking, and $M_{final} \in \mathcal{B}(P)$ is the final marking. $\mathcal{U}_{SN}$ is the* universe of system nets.

Definition 8 (System Net Notations). *Let $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$.*

- *$T_v(SN) = dom(l)$ is the set of* visible transitions *in SN,*
- *$A_v(SN) = rng(l)$ is the set of corresponding* observable activities *in SN,*
- *$T_v^u(SN) = \{t \in T_v(SN) \mid \forall_{t' \in T_v(SN)}\ l(t) = l(t') \ \Rightarrow\ t = t'\}$ is the set of* unique *visible transitions in SN (i.e., there are no other transitions having the same visible label), and*
- *$A_v^u(SN) = \{l(t) \mid t \in T_v^u(SN)\}$ is the set of corresponding* unique *observable activities in SN.*

Given a system net, $\phi(SN)$ is the set of all possible *visible* traces, i.e., complete firing sequences starting in M_{init} and ending in M_{final} projected onto the set of observable activities.

Definition 9 (Traces). *Let $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net. $\phi(SN) = \{\sigma_v \mid (N, M_{init})[\sigma_v \rhd (N, M_{final})\}$ is the set of* visible *traces starting in M_{init} and ending in M_{final}. $\phi_f(SN) = \{\sigma \mid (N, M_{init})[\sigma\rangle(N, M_{final})\}$ is the corresponding set of complete firing sequences.*

For Figure 2: $\phi(SN) = \{\langle a, c, d, f, g \rangle, \langle a, c, b, d, f, g \rangle, \langle a, c, d, h \rangle, \langle a, b, c, d, e, c, d, h \rangle, \ldots\}$ and $\phi_f(SN) = \{\langle t1, t2, t4, t5, t7, t8, t9, t11 \rangle, \langle t1, t3, t4, t5, t10 \rangle, \ldots\}$. Because of the loop involving transition $t6$ there are infinitely many visible traces and complete firing sequences.

In this paper, we need to compose and decompose process models. Therefore, we define the union of two system nets.

Definition 10 (Union of Nets). *Let $SN^1 = (N^1, M^1_{init}, M^1_{final}) \in \mathcal{U}_{SN}$ with $N^1 = (P^1, T^1, F^1, l^1)$ and $SN^2 = (N^2, M^2_{init}, M^2_{final}) \in \mathcal{U}_{SN}$ with $N^2 = (P^2, T^2, F^2, l^2)$ be two system nets.*

- $l^3 \in (T^1 \cup T^2) \not\to \mathcal{U}_A$ *with* $dom(l^3) = dom(l^1) \cup dom(l^2)$, $l^3(t) = l^1(t)$ *if* $t \in dom(l^1)$, *and* $l^3(t) = l^2(t)$ *if* $t \in dom(l^2) \setminus dom(l^1)$ *is the union of* l_1 *and* l_2,
- $N^1 \cup N^2 = (P^1 \cup P^2, T^1 \cup T^2, F^1 \cup F^2, l^3)$ *is the union of* N^1 *and* N^2, *and*
- $SN^1 \cup SN^2 = (N^1 \cup N^2, M^1_{init} \uplus M^2_{init}, M^1_{final} \uplus M^2_{final})$ *is the union of system nets* SN^1 *and* SN^2.

2.3 Event Log

As indicated earlier, *event logs* serve as the starting point for process mining. An event log is a multiset of *traces*. Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) in terms of the *activities* executed.

Definition 11 (Trace, Event Log). *Let* $A \subseteq \mathcal{U}_A$ *be a set of activities. A trace* $\sigma \in A^*$ *is a sequence of activities.* $L \in \mathcal{B}(A^*)$ *is an event log, i.e., a multiset of traces.*

An event log is a *multiset* of traces because there can be multiple cases having the same trace. In this simple definition of an event log, an event refers to just an *activity*. Often event logs store additional information about events. For example, many process mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order). In this paper, we abstract from such information. However, the results presented can easily be extended to event logs containing additional information.

An example log is $L_1 = [\langle a, c, d, f, g\rangle^{10}, \langle a, c, d, h\rangle^5, \langle a, b, c, d, e, c, d, g, f\rangle^5]$. L_1 contains information about 20 cases, e.g., 10 cases followed trace $\langle a, c, d, f, g\rangle$. There are $10 \times 5 + 5 \times 4 + 5 \times 9 = 115$ events in total.

The projection function $\restriction_X$ (cf. Definition 2) is generalized to event logs, i.e., for some event log $L \in \mathcal{B}(A^*)$ and set $X \subseteq A$: $L\restriction_X = [\sigma\restriction_X | \ \sigma \in L]$. For example, $L_1\restriction_{\{a,g,h\}} = [\langle a, g\rangle^{15}, \langle a, h\rangle^5]$. We will refer to these projected event logs as *sublogs*.

3 Conformance Checking

Conformance checking techniques investigate how well an event log $L \in \mathcal{B}(A^*)$ and a system net $SN = (N, M_{init}, M_{final})$ fit together. Note that SN may have been discovered through process mining or may have been made by hand. In any case, it is interesting to compare the observed example behavior in L with the potential behavior of SN.

Conformance checking can be done for various reasons. First of all, it may be used to audit processes to see whether reality conforms to some normative or descriptive model [6]. Deviations may point to fraud, inefficiencies, and poorly designed or outdated procedures. Second, conformance checking can be used to evaluate the performance of a process discovery technique. In fact, genetic

process mining algorithms use conformance checking to select the candidate models used to create the next generation of models [48].

There are four quality dimensions for comparing model and log: (1) *fitness*, (2) *simplicity*, (3) *precision*, and (4) *generalization* [1]. A model with good *fitness* allows for most of the behavior seen in the event log. A model has a perfect fitness if all traces in the log can be replayed by the model from beginning to end. The *simplest* model that can explain the behavior seen in the log is the best model. This principle is known as Occam's Razor. Fitness and simplicity alone are not sufficient to judge the quality of a discovered process model. For example, it is very easy to construct an extremely simple Petri net ("flower model") that is able to replay all traces in an event log (but also any other event log referring to the same set of activities). Similarly, it is undesirable to have a model that only allows for the exact behavior seen in the event log. Remember that the log contains only example behavior and that many traces that are possible may not have been seen yet. A model is *precise* if it does not allow for "too much" behavior. Clearly, the "flower model" lacks precision. A model that is not precise is "underfitting". Underfitting is the problem that the model over-generalizes the example behavior in the log (i.e., the model allows for behaviors very different from what was seen in the log). At the same time, the model should generalize and not restrict behavior to just the examples seen in the log. A model that does not *generalize* is "overfitting". Overfitting is the problem that a very specific model is generated whereas it is obvious that the log only holds example behavior (i.e., the model explains the particular sample log, but there is a high probability that the model is unable to explain the next batch of cases).

In the remainder, we will focus on fitness. However, the ideas are applicable to the other quality dimensions [5].

Definition 12 (Perfectly Fitting Log). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net. L is perfectly fitting SN if and only if $\{\sigma \in L\} \subseteq \phi(SN)$.*

Consider two event logs $L_1 = [\langle a, c, d, f, g\rangle^{10}, \langle a, c, d, h\rangle^5, \langle a, b, c, d, e, c, d, g, f\rangle^5]$ and $L_2 = [\langle a, c, d, f\rangle^{10}, \langle a, c, d, c, h\rangle^5, \langle a, b, d, e, c, d, g, f, h\rangle^5]$ and the system net SN of the Petri net depicted in Figure 2. Clearly, L_1 is perfectly fitting SN whereas L_2 is not. There are various ways to quantify fitness [1, 5, 10, 35, 48, 50, 51, 57].

To measure fitness, one needs to *align* traces in the event log to traces of the process model. Consider the following three alignments for the traces in L_1:

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline a & c & \gg & d & \gg & f & g & \gg \\ \hline a & c & \tau & d & \tau & f & g & \tau \\ t1 & t4 & t2 & t5 & t7 & t8 & t9 & t11 \\ \hline \end{array} \qquad \gamma_2 = \begin{array}{|c|c|c|c|c|} \hline a & c & \gg & d & h \\ \hline a & c & \tau & d & h \\ t1 & t4 & t2 & t5 & t10 \\ \hline \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & b & c & d & e & c & \gg & d & \gg & g & f & \gg \\ \hline a & b & c & d & e & c & \tau & d & \tau & g & f & \tau \\ t1 & t3 & t4 & t5 & t6 & t4 & t2 & t5 & t7 & t9 & t8 & t11 \\ \hline \end{array}$$

The top row of each alignment corresponds to "moves in the log" and the bottom two rows correspond to "moves in the model". Moves in the model are represented by the transition and its label. This is needed because there could be multiple transitions with the same label. If a move in the model cannot be mimicked by a move in the log, then a "$\gg$" ("no move") appears in the top row. For example, in the third position of γ_1 the log cannot mimic the invisible transition $t2$. The τ above $t2$ indicates that $t2 \notin dom(l)$. In the remainder, we write $l(t) = \tau$ if $t \notin dom(l)$. Note that all "no moves" (i.e., the seven $\gg$ symbols) in $\gamma_1 - \gamma_3$ are "caused" by invisible transitions.

Some example alignments for L_2 and SN:

$$\gamma_4 = \begin{array}{|c|c|c|c|c|c|c|c|}\hline a & c & \gg & d & \gg & f & \gg & \gg \\ \hline a & c & \tau & d & \tau & f & g & \tau \\ t1 & t4 & t2 & t5 & t7 & t8 & t9 & t11 \\ \hline \end{array} \qquad \gamma_5 = \begin{array}{|c|c|c|c|c|c|}\hline a & c & \gg & d & c & h \\ \hline a & c & \tau & d & \gg & h \\ t1 & t4 & t2 & t5 & & t10 \\ \hline \end{array}$$

$$\gamma_6 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline a & b & \gg & d & e & c & \gg & d & \gg & g & f & \gg & h \\ \hline a & b & c & d & e & c & \tau & d & \tau & g & f & \tau & \gg \\ t1 & t3 & t4 & t5 & t6 & t4 & t2 & t5 & t7 & t9 & t8 & t11 & \\ \hline \end{array}$$

Alignment γ_4 shows a "$\gg$" ("no move") in the top row that does not correspond to an invisible transition. The model makes a g move (occurrence of transition $t9$) that is not in the log. Alignment γ_6 has a similar move in the third position: the model makes a c move (occurrence of transition $t4$) that is not in the log. If a move in the log cannot be mimicked by a move in the model, then a "$\gg$" ("no move") appears in the bottom row. For example, in γ_5 the c move in the log is not mimicked by a move in the model and in γ_6 the h move in the log is not mimicked by a move in the model. Note that the "no moves" not corresponding to invisible transitions point to deviations between model and log.

A *move* is a pair $(x, (y, t))$ where the first element refers to the log and the second element refers to the model. For example, $(a, (a, t1))$ means that both log and model make an "a move" and the move in the model is caused by the occurrence of transition $t1$. $(\gg, (g, t9))$ means that the occurrence of transition $t9$ with label g is not mimicked by corresponding move of the log. $(c, \gg)$ means that the log makes an "c move" not followed by the model.

Definition 13 (Legal Moves). *Let $L \in \mathcal{B}(A^*)$ be an event log and let $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$. $A_{LM} = \{(x, (x, t)) \mid x \in A \wedge t \in T \wedge l(t) = x\} \cup \{(\gg, (x, t)) \mid t \in T \wedge l(t) = x\} \cup \{(x, \gg) \mid x \in A\}$ is the set of* legal moves.

An alignment is a sequence of legal moves such that after removing all $\gg$ symbols, the top row corresponds to a trace in the log and the bottom row corresponds to a firing sequence starting in M_{init} and ending M_{final}. Hence, the middle row corresponds to a visible path when ignoring the τ steps.

Definition 14 (Alignment). *Let $\sigma_L \in L$ be a log trace and $\sigma_M \in \phi_f(SN)$ a complete firing sequence of system net SN. An* alignment *of σ_L and σ_M is*

a sequence $\gamma \in {A_{LM}}^*$ such that the projection on the first element (ignoring $\gg$) yields σ_L and the projection on the last element (ignoring $\gg$ and transition labels) yields σ_M.

γ_1–γ_3 are examples of alignments for the traces in L_1 and their corresponding firing sequences in the system net of Figure 2. γ_4–γ_6 are examples of alignments for the traces in L_2 and complete firing sequences of the same system net. The projection of γ_6 on the first element (ignoring $\gg$) yields $\sigma_L = \langle a, b, d, e, c, d, g, f, h\rangle$ which is indeed a trace in L_2. The projection of γ_6 on the last element (ignoring $\gg$ and transition labels) yields $\sigma_M = \langle t1, t3, t4, t5, t6, t4, t2, t5, t7, t9, t8, t11\rangle$ which is indeed a complete firing sequence. The projection of γ_6 on the middle element (i.e., transition labels while ignoring $\gg$ and τ) yields $\langle a, b, c, d, e, c, d, g, f\rangle$ which is indeed a visible trace of the system net of Figure 2.

Given a log trace and a process model there may be many (if not infinitely many) alignments. Consider the following two alignments for $\langle a, c, d, f\rangle \in L_2$:

$$\gamma_4 = \begin{array}{|c|c|c|c|c|c|c|c|}\hline a & c & \gg & d & \gg & f & \gg & \gg \\ \hline a & c & \tau & d & \tau & f & g & \tau \\ \hline t1 & t4 & t2 & t5 & t7 & t8 & t9 & t11 \\ \hline \end{array} \qquad \gamma'_4 = \begin{array}{|c|c|c|c|c|c|c|}\hline a & c & \gg & d & \gg & f & \gg \\ \hline a & c & b & d & \tau & \gg & h \\ \hline t1 & t4 & t3 & t5 & t7 & & t10 \\ \hline \end{array}$$

γ_4 seems to be better alignment than γ'_4 because it has only one deviation (move in model only; $(\gg, (g, t9))$) whereas γ'_4 has three deviations: $(\gg, (b, t3))$, $(f, \gg)$, and $(\gg, (h, t11))$. To select the most appropriate one we associate *costs* to undesirable moves and select an alignment with the lowest total costs. To quantify the costs of misalignments we introduce a cost function δ.

Definition 15 (Cost of Alignment). *Cost function $\delta \in A_{LM} \to \mathbb{N}$ assigns costs to legal moves. The* cost *of an alignment $\gamma \in {A_{LM}}^*$ is the sum of all costs: $\delta(\gamma) = \sum_{(x,y)\in\gamma} \delta(x, y)$.*

Moves where log and model agree have no costs, i.e., $\delta(x, (x, t)) = 0$ for all $x \in A$. Moves in model only have no costs if the transition is invisible, i.e., $\delta(\gg, (\tau, t)) = 0$ if $l(t) = \tau$. $\delta(\gg, (x, t)) > 0$ is the cost when the model makes an "x move" without a corresponding move of the log (assuming $l(t) = x \neq \tau$). $\delta(x, \gg) > 0$ is the cost for an "x move" in just the log. These costs may depend on the nature of the activity, e.g., skipping a payment may be more severe than sending too many letters. However, in this paper we often use a standard cost function δ_S that assigns unit costs: $\delta(x, (x, t)) = 0$, $\delta(\gg, (\tau, t)) = 0$, and $\delta_S(\gg, (x, t)) = \delta_S(x, \gg) = 1$ for all $x \in A$. For example, $\delta_S(\gamma_1) = \delta_S(\gamma_2) = \delta_S(\gamma_3) = 0$, $\delta_S(\gamma_4) = 1$, $\delta_S(\gamma_5) = 1$, and $\delta_S(\gamma_6) = 2$ (simply count the number of $\gg$ symbols not corresponding to invisible transitions). Now we can compare the two alignments for $\langle a, c, d, f\rangle \in L_2$: $\delta_S(\gamma_4) = 1$ and $\delta_S(\gamma'_4) = 3$. Hence, we conclude that γ_4 is "better" than γ'_4.

Definition 16 (Optimal Alignment). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $SN \in \mathcal{U}_{SN}$ be a system net with $\phi(SN) \neq \emptyset$.*

- *For $\sigma_L \in L$, we define: $\Gamma_{\sigma_L, SN} = \{\gamma \in {A_{LM}}^* \mid \exists_{\sigma_M \in \phi_f(SN)}\ \gamma$ is an aligment of σ_L and $\sigma_M\}$.*

- *An alignment* $\gamma \in \Gamma_{\sigma_L,SN}$ *is* optimal *for trace* $\sigma_L \in L$ *and system net* SN *if for any* $\gamma' \in \Gamma_{\sigma_L,M}$*:* $\delta(\gamma') \geq \delta(\gamma)$.
- $\lambda_{SN} \in A^* \rightarrow {A_{LM}}^*$ *is a deterministic mapping that assigns any log trace* σ_L *to an optimal alignment, i.e.,* $\lambda_{SN}(\sigma_L) \in \Gamma_{\sigma_L,SN}$ *and* $\lambda_{SN}(\sigma_L)$ *is optimal.*
- $costs(L, SN, \delta) = \sum_{\sigma_L \in L} \delta(\lambda_{SN}(\sigma_L))$ *are the* misalignment costs *of the whole event log.*

$\gamma_1 - \gamma_6$ is are optimal alignments for the corresponding six possible traces in event logs L_1 and L_2. γ_4' is not an optimal alignment for $\langle a, c, d, f \rangle$. $costs(L_1, SN, \delta_S) = 10 \times \delta_S(\gamma_1) + 5 \times \delta_S(\gamma_2) + 5 \times \delta_S(\gamma_3) = 10 \times 0 + 5 \times 0 + 5 \times 0 = 0$. Hence, L_1 is perfectly fitting system net SN. $costs(L_2, SN, \delta_S) = 10 \times \delta_S(\gamma_4) + 5 \times \delta_S(\gamma_5) + 5 \times \delta_S(\gamma_6) = 10 \times 1 + 5 \times 1 + 5 \times 2 = 25$.

It is possible to convert misalignment costs into a fitness value between 0 (poor fitness, i.e., maximal costs) and 1 (perfect fitness, zero costs). We refer to [5, 10] for details. Misalignment costs can be related to Definition 12, because only perfectly fitting traces have costs 0 (assuming $\phi(SN) \neq \emptyset$).

Lemma 1 (Perfectly Fitting Log). *Event log* L *is perfectly fitting system net* SN *if and only if* $costs(L, SN, \delta) = 0$.

Once an optimal alignment has been established for every trace in the event log, these alignments can also be used as a basis to quantify other conformance notations such as precision and generalization [5]. For example, precision can be computed by counting "escaping edges" as shown in [50, 51]. Recent results show that such computations should be based on alignments [12]. The same holds for generalization [5]. Therefore, we focus on alignments when decomposing conformance checking tasks.

4 Decomposing Conformance Checking

Conformance checking can be time consuming as potentially many different traces need to be aligned with a model that may allow for an exponential (or even infinite) number of traces. Event logs may contain millions of events. Finding the best alignment may require solving many optimization problems [10] or repeated state-space explorations [57]. In worst case a state-space exploration of the model is needed per event. When using genetic process mining, one needs to check the fitness of every individual model in every generation [48]. As a result, thousands or even millions of conformance checks need to be done. For each conformance check, the whole event log needs to be traversed. Given these challenges, we are interested in reducing the time needed for conformance checking by decomposing the associated Petri net and event log. In this section, we show that it is possible to decompose conformance checking problems.

To decompose conformance checking problems we split a process model into model fragments. In terms of Petri nets: the overall system net SN is decomposed into a collection of subnets $\{SN^1, SN^2, \ldots, SN^n\}$ such that the union of these subnets yields the original system net. A decomposition is *valid* if the subnets

"agree" on the original labeling function (i.e., the same transition always has the same label), each place resides in just one subnet, and also each invisible transition resides in just one subnet. Moreover, if there are multiple transitions with the same label, they should reside in the same subnet. Only unique visible transitions (i.e., $T_v^u(SN)$, cf. Definition 8) can be shared among different subnets.

Definition 17 (Valid Decomposition). *Let $SN \in \mathcal{U}_{SN}$ be a system net with labeling function l. $D = \{SN^1, SN^2, \ldots, SN^n\} \subseteq \mathcal{U}_{SN}$ is a valid decomposition if and only if*

- *$SN^i = (N^i, M^i_{init}, M^i_{final})$ is a system net with $N^i = (P^i, T^i, F^i, l^i)$ for all $1 \leq i \leq n$,*
- *$l^i = l|_{T^i}$ for all $1 \leq i \leq n$,*
- *$P^i \cap P^j = \emptyset$ for $1 \leq i < j \leq n$,*
- *$T^i \cap T^j \subseteq T_v^u(SN)$ and $rng(l_i) \cap rng(l_j) \subseteq A_v^u(SN)$ for $1 \leq i < j \leq n$, and*
- *$SN = \bigcup_{1 \leq i \leq n} SN^i$.*

$\mathcal{D}(SN)$ is the set of all valid decompositions of SN.

Let $SN = (N, M_{init}, M_{final})$ with $N = (P, T, F, l)$ be a system net with valid decomposition $D = \{SN^1, SN^2, \ldots, SN^n\} \in \mathcal{D}(SN)$. We can observe the following properties:

- each place appears in precisely one of the subnets, i.e., for any $p \in P$: $|\{1 \leq i \leq n \mid p \in P^i\}| = 1$,
- each invisible transition appears in precisely one of the subnets, i.e., for any $t \in T \setminus T_v(SN)$: $|\{1 \leq i \leq n \mid t \in T^i\}| = 1$,
- visible transitions that do not have a unique label (i.e., there are multiple transitions with the same label) appear in precisely one of the subnets, i.e., for any $t \in T_v(SN) \setminus T_v^u(SN)$: $|\{1 \leq i \leq n \mid t \in T^i\}| = 1$,
- visible transitions having a unique label may appear in multiple subnets, i.e., for any $t \in T_v^u(SN)$: $|\{1 \leq i \leq n \mid t \in T^i\}| \geq 1$, and
- each edge appears in precisely one of the subnets, i.e., for any $(x, y) \in F$: $|\{1 \leq i \leq n \mid (x, y) \in F^i\}| = 1$.

Every system net has a trivial decomposition consisting of only one subnet, i.e., $\{SN\} \in \mathcal{D}(SN)$. However, we are interested in a *maximal decomposition* where the individual subnets are as small as possible. Figure 3 shows the maximal decomposition for the system net shown in Figure 2.

In the remainder, we assume that there are *no isolated nodes* in the original system net. Hence, for all $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ with $N = (P, T, F, l)$, we assume that $\omega(F) = P \cup T$. This is not a restriction: isolated places do not influence the behavior and can be removed, and for any transition t without input and output places we can add an initially marked self-loop place p_t. The self-loop place p_t does not restrict the behavior of t. By removing isolated nodes, we can simplify the construction of the maximal decomposition which is based on partitioning the edges in the original system net.

Definition 18 (Maximal Decomposition). *Let $SN \in \mathcal{U}_{SN}$ be a system net. The maximal decomposition $D_{max}(SN)$ is computed as follows:*

- *For any $(x, y) \in F$: $[\![(x, y)]\!] \subseteq F$ is the smallest set such that*

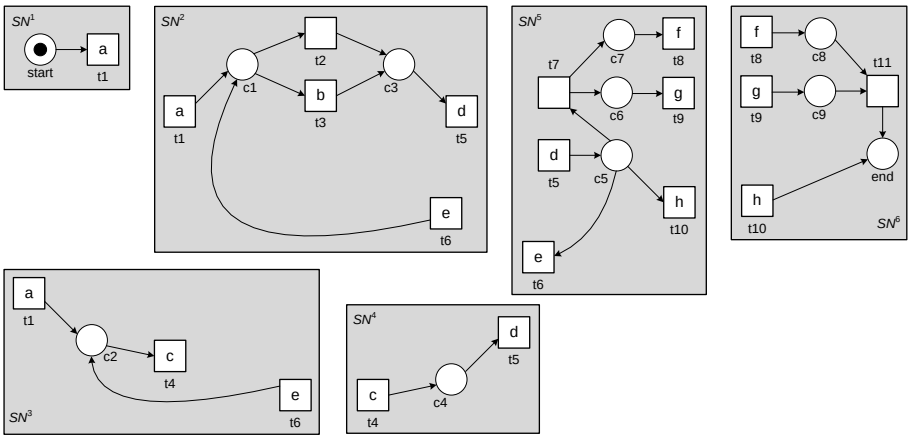


Fig. 3. Maximal decomposition of the system net shown in Figure 2 with $M_{init} = [start]$ and $M_{final} = [end]$. The initial and final markings are as follows: $M^1_{init} = [start]$ and $M^i_{init} = [\,]$ for $2 \leq i \leq 6$, $M^i_{final} = [\,]$ for $1 \leq i \leq 5$, and $M^6_{final} = [end]$.

  - $(x, y) \in [\![(x, y)]\!]$, *and*
  - $(x', y') \in [\![(x, y)]\!]$ *if* $(x', y') \in F$ *and* $\{x', y'\} \cap (\omega([\![(x, y)]\!]) \setminus T_v(SN)) \neq \emptyset$.
- *For any* $X \subseteq F$: $l_X = \{l(t) \mid t \in \omega(X) \cap T_v(SN)\}$.
- *For any* $(x, y) \in F$: $\langle\!\langle (x, y) \rangle\!\rangle \subseteq F$ *is the smallest set such that*
  - $[\![(x, y)]\!] \subseteq \langle\!\langle (x, y) \rangle\!\rangle$, *and*
  - $[\![(x', y')]\!] \subseteq \langle\!\langle (x, y) \rangle\!\rangle$ *if* $(x', y') \in F$ *and* $(l_{[\![(x',y')]\!]} \cap l_{\langle\!\langle (x,y) \rangle\!\rangle}) \not\subseteq A^u_v(SN)$.
- $P_{(x,y)} = \omega(\langle\!\langle (x, y) \rangle\!\rangle) \cap P$ *are the places of the subnet defined by edges* $\langle\!\langle (x, y) \rangle\!\rangle$.
- $T_{(x,y)} = \omega(\langle\!\langle (x, y) \rangle\!\rangle) \cap T$ *are the transitions of the subnet defined by edges* $\langle\!\langle (x, y) \rangle\!\rangle$.
- $D_{max}(SN) = \{((P_{(x,y)}, T_{(x,y)}, \langle\!\langle (x, y) \rangle\!\rangle, l \restriction_{T_{(x,y)}}), M_{init} \restriction_{P_{(x,y)}}, M_{final} \restriction_{P_{(x,y)}}) \mid (x, y) \in F\}$ *is the maximal decomposition of* SN.

The construction of the maximal decomposition is based on partitioning the edges. Each edge will end up in precisely one subnet. $[\![(x, y)]\!]$ are all edges that are "connected" to (x, y) via an undirected path involving places and invisible transitions. Consider for example $[\![(t1, c2)]\!] = \{(t1, c2), (c2, t4), (t6, c2)\}$ in Figure 2. $(c2, t4)$ and $(t6, c2)$ are added to $[\![(t1, c2)]\!]$ because $c2$ is not a visible transition. $(t6, c1)$ and $(c5, t6)$ are not added to $[\![(t1, c2)]\!]$ because $t6$ is a visible transition. After partitioning the edges into sets of connected edges, the sets that share *non-unique observable* activities are merged. In Figure 2 all visible transitions have a unique label, hence $\langle\!\langle (x, y) \rangle\!\rangle = [\![(x, y)]\!]$ for all edges $(x, y) \in F$. Based on the new partitioning of the edges, subnets are created. Consider for example $\langle\!\langle (t1, c2) \rangle\!\rangle = \{(t1, c2), (c2, t4), (t6, c2)\}$, the corresponding subnet SN^3 is shown in Figure 3. The six subnets $SN^1, SN^2, \ldots SN^6$ in Figure 3 form the maximal decomposition of the system net in Figure 2.

To illustrate the difference between $[\![(x, y)]\!]$ and $\langle\!\langle (x, y) \rangle\!\rangle$ let us consider the system net in Figure 2 where $t4$ is relabeled from c to b, i.e., there are two

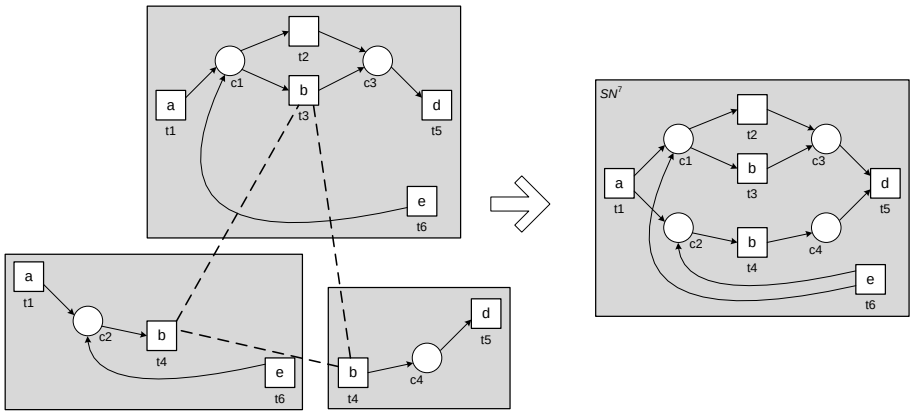


Fig. 4. Since b is a non-unique visible activity (transitions $t3$ and $t4$ have label b), three subnets need to be merged.

visible transitions having the same label: $l(t3) = l(t4) = b$. Let us call this modified system net SN_{mod}. Note that $T_v(SN_{mod}) = T_v(SN)$ and $T_v^u(SN_{mod}) = T_v(SN) \setminus \{t3, t4\}$, i.e., transitions $t3$ and $t4$ are still labeled but no longer unique. Now $\langle\!\langle (t1, c1) \rangle\!\rangle = [\![(t1, c1)]\!] \cup [\![(t1, c2)]\!] \cup [\![(t4, c4)]\!]$. In other words SN^2, SN^3 and SN^4 need to be merged into SN^7 as shown in Figure 4. Hence, the maximal decomposition of the modified system net SN_{mod} consists of four rather than six subnets: $D_{max}(SN_{mod}) = \{SN^1, SN^7, SN^5, SN^6\}$.

A straightforward algorithm to construct a maximal decomposition is quadratic in the number of edges. Hence, the complexity of partitioning is negligible to the complexity of the process mining problems that are being decomposed. Recall that most algorithms are linear in the size of the event log and exponential in the number of unique activities. A maximal decomposition is valid as is demonstrated next. However, please bear in mind that various valid decompositions can be used as the main results of the paper do not depend on particular decomposition technique.

Theorem 1 (Maximal Decomposition is Valid). *Let $SN \in \mathcal{U}_{SN}$ be a system net. $D_{max}(SN)$ is a valid decomposition, i.e., $D_{max}(SN) \in \mathcal{D}(SN)$.*

Proof. Let $D_{max}(SN) = \{SN^1, SN^2, \ldots, SN^n\}$. Clearly, all subnets SN^i are system nets that agree on a common labeling function: $l^i = l \restriction_{T^i}$ for all $1 \leq i \leq n$. Every edge (x, y) in SN is in precisely one subnet because of the way $[\![(x, y)]\!]$ and $\langle\!\langle (x, y) \rangle\!\rangle$ are constructed. Moreover, we assumed that there are no isolated nodes (can be removed through preprocessing). Therefore, also all nodes (places and transitions) and edges are included in at least one subnet. Hence, $SN = \bigcup_{1 \leq i \leq n} SN^i$. Remains to show that $P^i \cap P^j = \emptyset$ and $T^i \cap T^j \subseteq T_v^u(SN)$ for $1 \leq i < j \leq n$. The construction of $[\![(x, y)]\!]$ ensures that places and the edges connecting places end up in the same unique subnet. If $p \in P$, $p \in \omega([\![(x, y)]\!])$ and $p \in \omega([\![(x', y')]\!])$, then $[\![(x, y)]\!] = [\![(x', y')]\!]$. The same holds for invisible

transitions. Transitions having a visible label that is not unique need to end up in the same subnet. This is ensured by the construction of $\langle\langle(x,y)\rangle\rangle$ which merges subnets having shared visible transitions that are not unique. Hence, $T^i \cap T^j \subseteq T_v^u(SN)$ if $i \neq j$. □

After showing that we can decompose a system net into smaller subnets, we show that conformance checking can be done by locally inspecting the subnets using correspondingly projected event logs. To illustrate this, consider the following alignment for trace $\langle a,b,c,d,e,c,d,g,f\rangle$ and the system net in Figure 2:

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1&2&3&4&5&6&7&8&9&10&11&12\\\hline a&b&c&d&e&c&\gg&d&\gg&g&f&\gg\\\hline a&b&c&d&e&c&\tau&d&\tau&g&f&\tau\\ t1&t3&t4&t5&t6&t4&t2&t5&t7&t9&t8&t11\\\hline\end{array}$$

For convenience, the moves have been numbered. Now consider the following six alignments:

$$\gamma_3^1 = \begin{array}{|c|}\hline 1\\\hline a\\\hline a\\ t1\\\hline\end{array} \quad \gamma_3^2 = \begin{array}{|c|c|c|c|c|c|}\hline 1&2&4&5&7&8\\\hline a&b&d&e&\gg&d\\\hline a&b&d&e&\tau&d\\ t1&t3&t5&t6&t2&t5\\\hline\end{array} \quad \gamma_3^3 = \begin{array}{|c|c|c|c|}\hline 1&3&5&6\\\hline a&c&e&c\\\hline a&c&e&c\\ t1&t4&t6&t4\\\hline\end{array}$$

$$\gamma_3^4 = \begin{array}{|c|c|c|c|}\hline 3&4&6&8\\\hline c&d&c&d\\\hline c&d&c&d\\ t4&t5&t4&t5\\\hline\end{array} \quad \gamma_3^5 = \begin{array}{|c|c|c|c|c|c|}\hline 4&5&8&9&10&11\\\hline d&e&d&\gg&g&f\\\hline d&e&d&\tau&g&f\\ t5&t6&t5&t7&t9&t8\\\hline\end{array} \quad \gamma_3^6 = \begin{array}{|c|c|c|}\hline 10&11&12\\\hline g&f&\gg\\\hline g&f&\tau\\ t9&t8&t11\\\hline\end{array}$$

Each alignment corresponds to one of the six subnets $SN^1, SN^2, \ldots SN^6$ in Figure 3. The numbers are used to relate the different alignments. For example γ_3^6 is an alignment for trace $\langle a,b,c,d,e,c,d,g,f\rangle$ and subnets SN^6 in Figure 3. As the numbers 10, 11 and 12 indicate, γ_3^6 corresponds to the last three moves of γ_3.

To create sublogs for the different model fragments, we use the projection function introduced before. Consider for example the overall log $L_1 = [\langle a,c,d,f,g\rangle^{10}, \langle a,c,d,h\rangle^5, \langle a,b,c,d,e,c,d,g,f\rangle^5]$. $L_1^1 = L_1 \restriction_{\{a\}} = [\langle a\rangle^{20}]$, $L_1^2 = L_1 \restriction_{\{a,b,d,e\}} = [\langle a,d\rangle^{15}, \langle a,b,d,e,d\rangle^5]$, $L_1^3 = L_1 \restriction_{\{a,c,e\}} = [\langle a,c\rangle^{15}, \langle a,c,e,c\rangle^5]$, etc. are the sublogs corresponding to the subnets in Figure 3.

The following theorem shows that any trace that fits the overall process model can be decomposed into smaller traces that fit the individual model fragments. Moreover, if the smaller traces fit the individual model fragments, then they can be composed into an overall trace that fits into the overall process model. This result is the basis for decomposing process mining problems.

Theorem 2 (Conformance Checking Can be Decomposed). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $SN \in \mathcal{U}_{SN}$ be a system net with $A_v(SN) = A$. For any valid decomposition $D = \{SN^1, SN^2, \ldots, SN^n\} \in \mathcal{D}(SN)$: L is perfectly fitting system net SN if and only if for all $1 \leq i \leq n$: $L \restriction_{A_v(SN^i)}$ is perfectly fitting SN^i.*

Proof. Let $SN = (N, M_{init}, M_{final})$ be a system net with $N = (P, T, F, l)$. Let $D = \{SN^1, SN^2, \ldots, SN^n\}$ be a valid decomposition of SN with $SN^i = (N^i, M^i_{init}, M^i_{final})$, $N^i = (P^i, T^i, F^i, l^i)$, and $A^i = A_v(SN^i)$ for all i.
($\Rightarrow$) Let $\sigma_v \in L$ be such that there is a $\sigma \in T^*$ with $(N, M_{init})[\sigma\rangle(N, M_{final})$ and $l(\sigma) = \sigma_v$ (i.e., σ_v fits into the overall system net). For all $1 \leq i \leq n$, we need to prove that there is a σ_i with $(N^i, M^i_{init})[\sigma_i\rangle(N^i, M^i_{final})$ such that $l(\sigma_i) = \sigma_v{\restriction}_{A^i}$. This follows trivially because SN^i can mimic any move of SN with respect to transitions T^i: just take $\sigma_i = \sigma{\restriction}_{T^i}$.
($\Leftarrow$) Let $\sigma_v \in L$ be such that for each $1 \leq i \leq n$ there is a σ_i with (N^i, M^i_{init}) $[\sigma_i\rangle(N^i, M^i_{final})$ and $l(\sigma_i) = \sigma_v{\restriction}_{A^i}$. We need to prove that there is a $\sigma \in T^*$ such that $(N, M_{init})[\sigma\rangle(N, M_{final})$ and $l(\sigma) = \sigma_v$. The different σ_i sequences can be stitched together into an overall σ because the different subnets only interface via *unique* visible transitions and $SN = \bigcup_{1 \leq i \leq n} SN^i$. Transitions in one subnet can only influence other subnets through *unique* visible transitions and these can only move synchronously as defined by $\sigma_v \in L$. □

Theorem 2 shows that any trace in the log fits the overall model if and only if it fits each of the subnets.

Let us consider an example where not all visible transitions are unique. Let SN_{mod} be again the modified system net obtained by relabeling $t4$ from c to b in Figure 2. γ'_3 is an alignment for trace $\langle a, b, b, d, e, b, d, g, f\rangle$ and SN_{mod} and γ^7_3 is the corresponding alignment for SN^7 shown in Figure 4.

$$\gamma'_3 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1&2&3&4&5&6&7&8&9&10&11&12\\\hline a&b&b&d&e&b&\gg&d&\gg&g&f&\gg\\\hline a&b&b&d&e&b&\tau&d&\tau&g&f&\tau\\ t1&t3&t4&t5&t6&t4&t2&t5&t7&t9&t8&t11\\\hline\end{array} \qquad \gamma^7_3 = \begin{array}{|c|c|c|c|c|c|c|c|}\hline 1&2&3&4&5&6&7&8\\\hline a&b&b&d&e&b&\gg&d\\\hline a&b&b&d&e&b&\tau&d\\ t1&t3&t4&t5&t6&t4&t2&t5\\\hline\end{array}$$

Because γ'_3 is an alignment with costs 0, the decomposition of γ'_3 into γ^1_3, γ^5_3, γ^6_3, and γ^7_3 yields four alignments with costs 0 for the four subnets in $D_{max}(SN_{mod}) = \{SN^1, SN^5, SN^6, SN^7\}$. Moreover, γ^1_3, γ^5_3, γ^6_3, and γ^7_3 can be stitched back into γ'_3 provided that the four alignments are perfectly fitting their part of the overall model.

Let us now consider trace $\langle a, b, d, e, c, d, g, f, h\rangle$ which is not perfectly fitting the system net in Figure 2. An optimal alignment is:

$$\gamma_6 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1&2&3&4&5&6&7&8&9&10&11&12&13\\\hline a&b&\gg&d&e&c&\gg&d&\gg&g&f&\gg&h\\\hline a&b&c&d&e&c&\tau&d&\tau&g&f&\tau&\gg\\ t1&t3&t4&t5&t6&t4&t2&t5&t7&t9&t8&t11&\\\hline\end{array}$$

The alignment shows the two problems: the model needs to execute c whereas this event is not in the event log (position 3) and the event log contains g, f, and h whereas the model needs to choose between either g and f or h (position 13). The cost of this optimal alignment is 2. Optimal alignment γ_6 for the overall

model can be decomposed into alignments $\gamma_6^1 - \gamma_6^6$ for the six subnets:

$$\gamma_6^1 = \begin{array}{|c|}\hline 1 \\ \hline a \\ \hline a \\ \hline t1 \\ \hline \end{array} \quad \gamma_6^2 = \begin{array}{|c|c|c|c|c|c|}\hline 1 & 2 & 4 & 5 & 7 & 8 \\ \hline a & b & d & e & \gg & d \\ \hline a & b & d & e & \tau & d \\ \hline t1 & t3 & t5 & t6 & t2 & t5 \\ \hline \end{array} \quad \gamma_6^3 = \begin{array}{|c|c|c|c|}\hline 1 & 3 & 5 & 6 \\ \hline a & \gg & e & c \\ \hline a & c & e & c \\ \hline t1 & t4 & t6 & t4 \\ \hline \end{array}$$

$$\gamma_6^4 = \begin{array}{|c|c|c|c|}\hline 3 & 4 & 6 & 8 \\ \hline \gg & d & c & d \\ \hline c & d & c & d \\ \hline t4 & t5 & t4 & t5 \\ \hline \end{array} \quad \gamma_6^5 = \begin{array}{|c|c|c|c|c|c|c|}\hline 4 & 5 & 8 & 9 & 10 & 11 & 13 \\ \hline d & e & d & \gg & g & f & h \\ \hline d & e & d & \tau & g & f & \gg \\ \hline t5 & t6 & t5 & t7 & t9 & t8 & \\ \hline \end{array} \quad \gamma_6^6 = \begin{array}{|c|c|c|c|}\hline 10 & 11 & 12 & 13 \\ \hline g & f & \gg & h \\ \hline g & f & \tau & \gg \\ \hline t9 & t8 & t11 & \\ \hline \end{array}$$

Alignments γ_6^1 and γ_6^2 have costs 0. Alignments γ_6^3 and γ_6^4 have costs 1 (move in model involving c). Alignments γ_6^5 and γ_6^6 have costs 1 (move in log involving h). If we would add up all costs, we would get costs 4 whereas the costs of optimal alignment γ_6 is 2. However, we would like to compute an upper bound for the degree of fitness in a distributed manner. Therefore, we introduce an adapted cost function δ_Q.

Definition 19 (Adapted Cost Function). *Let $D = \{SN^1, SN^2, \ldots, SN^n\} \in \mathcal{D}(SN)$ be a valid decomposition of some system net SN and $\delta \in A_{LM} \to \mathbb{N}$ a cost function (cf. Definition 15). $c_Q(a,(a,t)) = c_Q(\gg,(a,t)) = c_Q(a,\gg) = |\{1 \leq i \leq n \mid a \in A_i\}|$ counts the number of subnets having a as an observable activity. The adapted cost function δ_Q is defined as follows: $\delta_Q(x,y) = \frac{\delta(x,y)}{c_Q(x,y)}$ for $(x,y) \in A_{LM}$ and $c_Q(x,y) \neq 0$.*

An observable activity may appear in multiple subnets. Therefore, we divide its costs by the number of subnets in which it appears: $\delta_Q(x,y) = \frac{\delta(x,y)}{c_Q(x,y)}$. This way we avoid counting misalignments of the same activity multiple times. For our example, $c_Q(\gg,(c,t4)) = |\{3,4\}| = 2$ and $c_Q(h,\gg) = |\{5,6\}| = 2$. Assuming the standard cost function δ_S this implies $\delta_Q(\gg,(c,t4)) = \frac{1}{2}$ and $\delta_Q(h,\gg) = \frac{1}{2}$. Hence the aggregated costs of $\gamma_6^1 - \gamma_6^6$ is 2, i.e., identical to the costs of the overall optimal alignment.

Theorem 3 (Lower Bound for Misalignment Costs). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$, $SN \in \mathcal{U}_{SN}$ be a system net, and δ a cost function. For any valid decomposition $D = \{SN^1, SN^2, \ldots, SN^n\} \in \mathcal{D}(SN)$:*

$$costs(L, SN, \delta) \geq \sum_{1 \leq i \leq n} costs(L{\restriction}_{A_v(SN^i)}, SN^i, \delta_Q)$$

Proof. For any $\sigma_v \in L$ there is an optimal alignment γ of σ_v and SN such that the projection on the last element yields a $\sigma \in T^*$ with $(N, M_{init})[\sigma\rangle(N, M_{final})$ and $l(\sigma) = \sigma_v$. As shown in the proof of Theorem 2 there is a σ_i with (N^i, M^i_{init}) $[\sigma_i\rangle(N^i, M^i_{final})$ such that $l(\sigma_i) = \sigma_v{\restriction}_{A^i}$ for any $1 \leq i \leq n$. In a similar fashion, γ can be decomposed in $\gamma_1, \gamma_2, \ldots, \gamma_n$ where γ_i is an alignment of σ_i and SN^i. The sum of the costs associated with these local alignments $\gamma_1, \gamma_2, \ldots, \gamma_n$ is exactly

the same as the overal alignment γ. However, there may be local improvements lowering the sum of the costs associated with these local alignments. Hence, $costs(L, SN, \delta) \geq \sum_{1 \leq i \leq n} costs(L{\restriction}_{A_v(SN^i)}, SN^i, \delta_Q)$. □

The sum of the costs associated to all selected optimal local alignments (using δ_Q) can never exceed the cost of an optimal overall alignment using δ. Hence, it can be used as an optimistic estimate, i.e., computing an upper bound for the overall fitness and a lower bound for the overall costs. More important, the fitness values of the different subnets provide valuable local diagnostics. *The subnets with the highest costs are the most problematic parts of the model. The alignments for these "problem spots" help to understand the main problems without having to look at very long overall alignments.*

Theorem 3 uses a rather sophisticated definition of fitness. We can also simply count the *fraction of fitting traces*. In this case the problem can be decomposed easily using the notion of valid decomposition.

Theorem 4 (Fraction of Perfectly Fitting Traces). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $SN \in \mathcal{U}_{SN}$ be a system net. For any valid decomposition $D = \{SN^1, SN^2, \ldots, SN^n\} \in \mathcal{D}(SN)$:*

$$\frac{|[\sigma \in L \mid \sigma \in \phi(SN)]|}{|L|} = \frac{|[\sigma \in L \mid \forall_{1 \leq i \leq n}\ \sigma{\restriction}_{A_v(SN^i)} \in \phi(SN^i)]|}{|L|}$$

Proof. Follows from the construction used in Theorem 2. A trace is fitting the overall model if and only if it fits all subnets. □

As Theorem 4 suggests, traces in the event log can be marked as fitting or non-fitting *per subnet*. These results can be merged easily and used to compute the *fraction of traces fitting the overall model*. Note that Theorem 4 holds for any decomposition of SN, including the maximal decomposition $D_{max}(SN)$.

Although the results presented only address the notion of fitness, it should be noted that alignments are the starting point for many other types of analysis. For example, *precision* can be computed by counting so-called "escaping edges" (sequences of steps allowed by the model but never happening in the event log) [50, 51]. This can be done at the level of subnets even though there is not a straightforward manner to compute the overall precision level. Note that relatively many escaping edges in a subnet suggest "underfitting" of that part of model. As shown in [12], alignments should be the basis for precision analysis. Therefore, the construction used in Theorems 2 and 3 can be used as a basis for computing precision. A similar approach can be used for generalization: many unique paths in a subnet may indicate "overfitting" of that part of the model [5].

The alignments can be used *beyond* conformance checking. An alignment γ_i (see proof of Theorem 3) relates observed events to occurrences of transitions in multiple subnets. If the event log contains *timestamps*, such alignments can be used to compute times in-between transition occurrences (waiting times, response times, service times, etc.) as shown in [1]. This way bottlenecks can be

identified. If the event log contains additional data (e.g., size of order, age of patient, or type of customer), then these local alignments can be used for *decision mining* [56]. For any decision point in a subnet (place with multiple output edges), one can create a decision tree based on the data available prior to the choice. Note that bottleneck analysis and decision point analysis provide local diagnostics and can be added to the overall model without any problems. Moreover, note that in our decomposition approach a place is always in precisely one subnet (together with its surrounding transitions). This allows us to analyze these aspects locally.

Assuming a process model having a decomposition consisting of many subnets, *the time needed for conformance checking can be reduced significantly.* There are two reasons for this. First of all, as our theorems show, larger problems can be decomposed into sets of independent smaller problems. Therefore, conformance checking can be distributed over multiple computers. Second, due to the exponential nature of most conformance checking techniques, the time needed to solve "many smaller problems" is less than the time needed to solve "one big problem". Existing conformance checking approaches use state-space analysis (e.g., in [57] the shortest path enabling a transition is computed) or optimization over all possible alignments (e.g., in [10] the A^* algorithm is used to find the best alignment). These techniques do *not* scale linearly in the number of activities. Therefore, decomposition is useful even if the checks per subnet are done on a single computer. Moreover, decomposing conformance checking is not just interesting from a performance point of view: decompositions can also be used to pinpoint the most problematic parts of the process (also in terms of performance) and provide localized diagnostics.

5 Decomposing Process Discovery

Process discovery, i.e., discovering a process model from event data, is highly related to conformance checking. This can be observed when considering genetic process mining algorithms that basically "guess" models and recombine parts of models that have good fitness to discover even better models [48, 20]. The fitness computation in genetic process mining is in essence a conformance check. Genetic process mining algorithms can be decomposed in various ways [3, 19]. However, in this paper we only consider the kind of decomposition described in the previous section. Moreover, we describe an approach that is not tailored towards a particular discovery algorithm. *We aim to distribute any process discovery algorithm by (1) decomposing the overall event log into smaller sublogs, (2) discovering a model fragment for each sublog, and (3) merging the discovered models into one overall process model.*

Since we focus on decomposing existing process discovery algorithms and do not propose a new algorithm, we use the following abstract definition of a process mining algorithm.

Definition 20 (Discovery Algorithm). *A discovery algorithm is a function* $disc \in \mathcal{B}(\mathcal{U}_A^*) \rightarrow \mathcal{U}_{SN}$ *that maps any event log* $L \in \mathcal{B}(A^*)$ *with* $A \subseteq \mathcal{U}_A$ *onto*

system net SN with $A_v(SN) = \{a \in \sigma \mid \sigma \in L\}$ *and an injective labeling function* l *(i.e.,* $T_v(SN) = T_v^u(SN)$*).*

In Section 7 we discuss related work and mention some of the many available process discovery algorithms [8, 9, 14, 16, 23, 24, 27, 35, 48, 59, 63, 64]. We assume the result to be presented as a system net. This implies that for algorithms not based on Petri nets an extra conversion step is needed. However, as mentioned before, our approach is not specific for Petri nets. We require that the labeling function of the generated system net is injective. Hence, the discovered system net cannot have duplicate activities, i.e., multiple transitions referring to the same activity. We make this assumption because it is not possible to compose model fragments with overlapping activities corresponding to different transitions: overlapping activities should be unique (cf. Definition 17) otherwise Theorem 2 does not hold. Note that very few process discovery algorithms allow for duplicate activities [39, 47]. Therefore, this is a reasonable assumption. However, it is possible that the system net has many silent transitions, i.e., transitions t such that $l(t) = \tau$. In principle, Theorem 2 also allows for duplicate activities within one fragment, but for simplicity we do not consider this case here. An example of a discovery algorithm is the α algorithm [9] which we will refer to as $disc_\alpha()$.

Given an event log L containing events referring to a set of activities A, we decompose discovery by distributing the activities over multiple sets $A^1, A^2, \ldots, A^n$. The same activity may appear in multiple sets as long as $A = A^1 \cup A^2 \cup \ldots \cup A^n$. For each activity set A_i, we discover a system net $\overline{SN}^i$ by applying a discovery algorithm to sublog $L{\restriction}_{A^i}$, i.e., the overall event log projected onto a subset of activities. Subsequently, the discovered system nets are massaged to avoid name clashes and to make sure that transitions with a visible label are merged properly. By combining the resulting system nets we obtain an overall system net $SN = SN^1 \cup SN^2 \cup \ldots \cup SN^n$ describing the behavior in the overall event log L.

Definition 21 (Decomposing Discovery). *Let* $L \in \mathcal{B}(A^*)$ *be an event log with* $A = \{a \in \sigma \mid \sigma \in L\} \subseteq \mathcal{U}_A$*. For any collection of activity sets* $\mathcal{C} = \{A^1, A^2, \ldots, A^n\}$ *such that* $A = \bigcup \mathcal{C}$ *and discovery algorithm* $disc \in \mathcal{B}(\mathcal{U}_A^*) \rightarrow \mathcal{U}_{SN}$*, we define:*

- $\overline{SN}^i = disc(L{\restriction}_{A^i})$ *for* $1 \leq i \leq n$*, i.e., a subnet is discovered per sublog,*
- *For all* $1 \leq i \leq n$*: system net* SN^i *is identical to system net* $\overline{SN}^i$ *after renaming places and transitions to avoid name clashes. Concretely, place* $p \in \overline{P}^i$ *is renamed to* p^i*, transition* $t \in \overline{T}^i \setminus T_v(\overline{SN})$ *is renamed to* t^i*, and transition* $t \in T_v(\overline{SN})$ *is renamed to* $\overline{l}(t)$*.*
- $distr\_disc(L, \mathcal{C}, disc) = \{SN^1, SN^2, \ldots, SN^n\}$*.*

To illustrate the idea consider the following event log:

$L_o = [\langle a, b, c, d, h\rangle^{30},$
$\langle a, c, b, d, e, b, c, d, j, g, f, k\rangle^{28},$
$\langle a, c, b, d, e, c, b, d, h\rangle^{28},$
$\langle a, c, b, d, e, c, b, d, j, f, g, k\rangle^{27},$
$\langle a, b, c, d, e, c, i, d, j, g, f, k\rangle^{26},$
$\langle a, b, c, d, e, i, c, d, h\rangle^{26},$

$\langle a, i, c, d, e, c, b, d, j, g, f, k\rangle^{25}$,
$\langle a, i, c, d, j, f, g, k\rangle^{24}$,
$\langle a, c, b, d, e, b, c, d, h\rangle^{24}$,
$\langle a, b, c, d, e, b, c, d, j, g, f, k\rangle^{24}$,
$\langle a, c, b, d, e, c, i, d, h\rangle^{22}$,
$\langle a, b, c, d, e, c, i, d, h\rangle^{22}$,
$\langle a, c, i, d, j, f, g, k\rangle^{21}$,
$\langle a, c, i, d, j, g, f, k\rangle^{20}$,
$\langle a, c, i, d, e, b, c, d, h\rangle^{18}$,
$\langle a, c, i, d, e, c, i, d, h\rangle^{17}$,
$\langle a, i, c, d, h\rangle^{17}$,
$\langle a, c, b, d, j, g, f, k\rangle^{17}$,
$\langle a, c, b, d, h\rangle^{15}$,
$\langle a, i, c, d, e, i, c, d, j, f, g, k\rangle^{14}$,
$\langle a, c, i, d, e, c, b, d, h\rangle^{14}$,
$\langle a, c, i, d, e, c, b, d, j, f, g, k\rangle^{12}$,
$\langle a, b, c, d, e, i, c, d, j, f, g, k\rangle^{12}$,
$\langle a, b, c, d, e, c, b, d, h\rangle^{11}$,
$\langle a, c, b, d, j, f, g, k\rangle^{10}$,
$\langle a, i, c, d, j, g, f, k\rangle^{9}$,
$\langle a, i, c, d, e, b, c, d, h\rangle^{9}$,
$\langle a, i, c, d, e, c, b, d, h\rangle^{9}$,
$\langle a, i, c, d, e, c, b, d, j, f, g, k\rangle^{8}$,
$\langle a, c, b, d, e, b, c, d, j, f, g, k\rangle^{8}$,
$\langle a, b, c, d, e, b, c, d, h\rangle^{7}$,
$\langle a, c, b, d, e, i, c, d, j, g, f, k\rangle^{7}$,
$\langle a, i, c, d, e, c, i, d, j, g, f, k\rangle^{6}$,
$\langle a, c, i, d, h\rangle^{6}$,
$\langle a, c, b, d, e, i, c, d, h\rangle^{6}$,
$\langle a, i, c, d, e, b, c, d, j, g, f, k\rangle^{6}$,
$\langle a, b, c, d, j, f, g, k\rangle^{5}$,
$\langle a, i, c, d, e, i, c, d, h\rangle^{5}$,
$\langle a, i, c, d, e, c, i, d, h\rangle^{4}$,
$\langle a, c, i, d, e, i, c, d, h\rangle^{4}$,
$\langle a, b, c, d, e, c, i, d, j, f, g, k\rangle^{4}$,
$\langle a, b, c, d, e, c, b, d, j, g, f, k\rangle^{4}$,
$\langle a, b, c, d, j, g, f, k\rangle^{3}$,
$\langle a, c, i, d, e, b, c, d, j, f, g, k\rangle^{3}$,
$\langle a, b, c, d, e, b, c, d, j, f, g, k\rangle^{3}$,
$\langle a, i, c, d, e, b, c, d, j, f, g, k\rangle^{3}$,
$\langle a, b, c, d, e, c, b, d, j, f, g, k\rangle^{3}$,
$\langle a, c, i, d, e, i, c, d, j, f, g, k\rangle^{2}$,
$\langle a, c, i, d, e, i, c, d, e, b, c, d, e, b, c, d, j, g, f, k\rangle^{2}$,
$\langle a, b, c, d, e, i, c, d, j, g, f, k\rangle^{2}$,
$\langle a, c, b, d, e, c, i, d, j, g, f, k\rangle^{2}$,
$\langle a, c, i, d, e, c, i, d, j, f, g, k\rangle^{2}$,
$\langle a, c, b, d, e, b, c, d, e, c, b, d, j, g, f, k\rangle^{2}$,
$\langle a, c, i, d, e, c, b, d, j, g, f, k\rangle^{2}$,
$\langle a, c, b, d, e, i, c, d, j, f, g, k\rangle^{2}$,
$\langle a, c, i, d, e, i, c, d, e, i, c, d, j, g, f, k\rangle^{1}$,
$\langle a, c, b, d, e, c, i, d, j, f, g, k\rangle^{1}$,
$\langle a, i, c, d, e, c, i, d, e, c, i, d, e, i, c, d, e, i, c, d, j, g, f, k\rangle^{1}$,
$\langle a, i, c, d, e, c, i, d, j, f, g, k\rangle^{1}$,
$\langle a, c, i, d, e, c, i, d, j, g, f, k\rangle^{1}]$.

Event log L_o consists of traces that originate from a process similar to Figure 2, but with the three invisible transitions ($t2$, $t7$, and $t11$) made visible. The corresponding new activities are i, j, and k. Since we are interested in process discovery, we assume we do not know this and try to rediscover the original process model by analyzing event log L_o. Note that $A = \{a \in \sigma \mid \sigma \in L_o\} = \{a, b, c, d, e, f, g, h, i, j, k\}$.

Let us decompose A into four subsets: $A^1 = \{a, b, d, e, i\}$, $A^2 = \{a, c, d, e\}$, $A^3 = \{d, e, h, j\}$, and $A^4 = \{f, g, h, j, k\}$. These sets are deliberately partly overlapping and each set contains causally related activities. As discussed later, there are different strategies to decompose A, but for the moment we assume these subsets to be given. We would like to apply the α algorithm to the corresponding four sublogs because it is well-known, simple, and deterministic. However, the α algorithm cannot deal with the situation where initial and final activities also appear in the middle of the event log. For example, consider the second trace in L_o projected onto A^2: $\langle a, c, d, e, c, d\rangle$. In this trace d appears in the middle and at the end and the α algorithm would produce nonsense. Therefore, we extend each trace in the event log with an artificial initial event

$\top$ and an artificial final event $\bot$. The resulting log is: $L'_o = [\langle\top\rangle \cdot \sigma \cdot \langle\bot\rangle \mid \sigma \in L_o] = [\langle\top, a, b, c, d, h, \bot\rangle^{30}, \langle\top, a, c, b, d, e, b, c, d, j, g, f, k, \bot\rangle^{28}, \langle\top, a, c, b, d, e, c, b, d, h, \bot\rangle^{28}, \langle\top, a, c, b, d, e, c, b, d, j, f, g, k, \bot\rangle^{27}, \langle\top, a, b, c, d, e, c, i, d, j, g, f, k, \bot\rangle^{26}, \ldots]$. The adapted four activity sets are $A^1 = \{\top, a, b, d, e, i, \bot\}$, $A^2 = \{\top, a, c, d, e, \bot\}$, $A^3 = \{\top, d, e, h, j, \bot\}$, and $A^4 = \{\top, f, g, h, j, k, \bot\}$. Each sublog is extended with these artificial events: $L_1 = L'_o{\restriction}_{\{\top,a,b,d,e,i,\bot\}}$, $L_2 = L'_o{\restriction}_{\{\top,a,c,d,e,\bot\}}$, $L_3 = L'_o{\restriction}_{\{\top,d,e,h,j,\bot\}}$, and $L_4 = L'_o{\restriction}_{\{\top,f,g,h,j,k,\bot\}}$.

After creating the four sublogs L_1, L_2, L_3, and L_4 we can discover the four corresponding system nets. The four process models in Figure 5 were constructed by applying the α algorithm to the four sublogs. We used the extended version of the α algorithm as implemented in *ProM*, i.e., unlike the classical eight-line algorithm [9], loops of length 2 are handled properly. Actually, the models shown in Figure 5 were generated using *ProM*. In the diagrams we deliberately do not show place and transition names as these are renamed anyway. The places are shaded using a particular pattern to be able to trace their origin when merging the four fragments (cf. Figure 6). Note that the addition of the artificial events had the desired effect: each subnet starts with a transition labeled $\top$ and ends with a transition labeled $\bot$. Note that L_1 is perfectly fitting SN^1, L_2 is perfectly fitting SN^2, etc.

Composing the four fragments shown in Figure 5 results in the overall system net shown in Figure 6. At a first glance this model may seem overly complicated. There are four initially marked source places ($p1$, $p2$, $p3$, and $p4$), four sink places ($p22$, $p23$, $p24$, and $p25$), and many other redundant places. Consider for example $p6$ which contains a token if and only if $p5$ contains a token while having identical connections. Hence, $p6$ can be removed while keeping $p5$. Also $p2$, $p3$, and $p4$ can be removed while keeping $p1$. Etc. These simplifications do not change the behavior.

In general a place is *redundant* (also called "implicit") if its removal does not change the behavior. A place is *structurally redundant* [17] if this can be decided on the structure of the net. By removing structurally redundant places and the artificially inserted $\top$ and $\bot$ transitions, we obtain the system net shown in Figure 7. This model adequately captures the behavior seen in the original event log L. In fact, L_o is perfectly fitting the overall system shown in Figure 7. This illustrates that discovery can indeed be decomposed into smaller problems.

The quality of the system net obtained by merging the process models discovered for the sublogs highly depends on the way the activities are decomposed and the characteristics of the discovery algorithm *disc*(). However, as the following theorem shows, the system nets discovered for the sublogs are always a valid decomposition of the overall model.

Theorem 5 (Discovery Can Be Decomposed). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A = \{a \in \sigma \mid \sigma \in L\} \subseteq \mathcal{U}_A$, $\mathcal{C} = \{A^1, A^2, \ldots, A^n\}$ with $A = \bigcup \mathcal{C}$, and $disc \in \mathcal{B}(\mathcal{U}_A^*) \rightarrow \mathcal{U}_{SN}$ a discovery algorithm. Let $distr\_disc(L, \mathcal{C}, disc) = \{SN^1, SN^2, \ldots, SN^n\}$ and $SN = \bigcup_{1 \leq i \leq n} SN^i$. $SN \in \mathcal{U}_{SN}$ and $\{SN^1, SN^2, \ldots, SN^n\}$ is a valid decomposition of SN, i.e., $distr\_disc(L, \mathcal{C}, disc) \in \mathcal{D}(SN)$.*

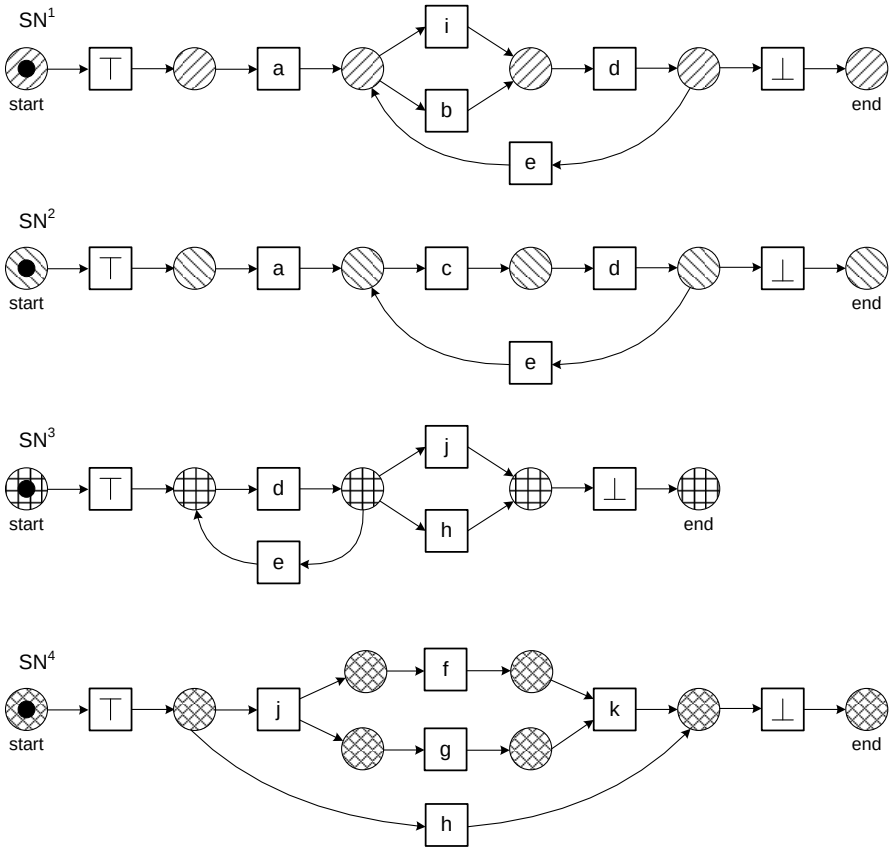


Fig. 5. System nets $SN^1 = disc_\alpha(L_1)$, $SN^2 = disc_\alpha(L_2)$, $SN^3 = disc_\alpha(L_3)$, and $SN^4 = disc_\alpha(L_4)$ discovered for respectively $L_1 = [\langle \top, a, b, d, \bot \rangle^{30}, \langle \top, a, b, d, e, b, d, \bot \rangle^{28}, \langle \top, a, b, d, e, b, d, \bot \rangle^{28}, \langle \top, a, b, d, e, b, d, \bot \rangle^{27}, \langle \top, a, b, d, e, i, d, \bot \rangle^{26}, \ldots]$, $L_2 = [\langle \top, a, c, d, \bot \rangle^{30}, \langle \top, a, c, d, e, c, d, \bot \rangle^{28}, \langle \top, a, c, d, e, c, d, \bot \rangle^{28}, \langle \top, a, c, d, e, c, d, \bot \rangle^{27}, \langle \top, a, c, d, e, c, d, \bot \rangle^{26}, \ldots]$, $L_3 = [\langle \top, d, h, \bot \rangle^{30}, \langle \top, d, e, d, j, \bot \rangle^{28}, \langle \top, d, e, d, h, \bot \rangle^{28}, \langle \top, d, e, d, j, \bot \rangle^{27}, \langle \top, d, e, d, j, \bot \rangle^{26}, \ldots]$, and $L_4 = [\langle \top, h, \bot \rangle^{30}, \langle \top, j, g, f, k, \bot \rangle^{28}, \langle \top, h, \bot \rangle^{28}, \langle \top, j, f, g, k, \bot \rangle^{27}, \langle \top, j, g, f, k, \bot \rangle^{26}, \ldots]$ using the α algorithm. Each net corresponds to a fragment of the overall process model and $M^i_{init} = [start]$ and $M^i_{final} = [end]$ for $i \in \{1, 2, 3, 4\}$.

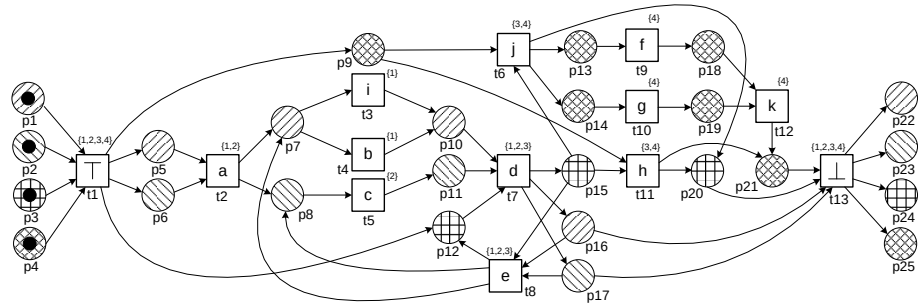


Fig. 6. The system net $SN = \bigcup_{1 \leq i \leq 4} SN^i$ with $M_{init} = [p1, p2, p3, p4]$ and $M^i_{final} = [p22, p23, p24, p25]$. SN contains many redundant places but these can be removed easily.

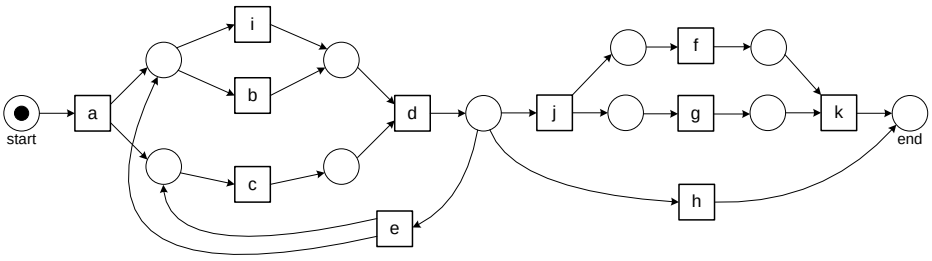


Fig. 7. The discovered system net SN obtained after removing redundant places and the artificially inserted $\top$ and $\bot$ transitions. ($M_{init} = [start]$ and $M_{final} = [end]$.) Event log L_o is perfectly fitting SN.

Proof. We need to show that $SN^i = (N^i, M^i_{init}, M^i_{final})$ is a system net with $N^i = (P^i, T^i, F^i, l^i)$ and $l^i = l{\restriction_{T^i}}$ for all $1 \leq i \leq n$, $P^i \cap P^j = \emptyset$ and $T^i \cap T^j \subseteq T^u_v(SN)$ for $1 \leq i < j \leq n$, and $SN = \bigcup_{1 \leq i \leq n} SN^i$. These properties follow directly from the definition of $distr\_disc(L, \mathcal{C}, disc)$ and the fact that each subnet has an injective labeling function l_i. □

The construction described in Definition 21 yields a net SN with valid decomposition $distr\_disc(L, \mathcal{C}, disc) = \{SN^1, SN^2, \ldots, SN^n\}$. This implies that we can apply Theorem 2 (Conformance Checking Can be Decomposed), Theorem 3 (Lower Bound for Misalignment Costs), and Theorem 4 (Fraction of Perfectly Fitting Traces).

Corollary 1. *Let $L \in \mathcal{B}(A^*)$ be an event log with $A = \{a \in \sigma \mid \sigma \in L\}$, $\mathcal{C} = \{A^1, A^2, \ldots, A^n\}$ with $A = \bigcup \mathcal{C}$, and $disc \in \mathcal{B}(\mathcal{U}^*_A) \rightarrow \mathcal{U}_{SN}$ a discovery algorithm. Let $distr\_disc(L, \mathcal{C}, disc) = \{SN^1, SN^2, \ldots, SN^n\}$ and $SN = \bigcup_{1 \leq i \leq n} SN^i$.*

- *L is perfectly fitting system net SN if and only if for all $1 \leq i \leq n$: $L{\restriction_{A_v(SN^i)}}$ is perfectly fitting SN^i,*

- $costs(L, SN, \delta) \geq \sum_{1 \leq i \leq n} costs(L{\restriction}_{A_v(SN^i)}, SN^i, \delta_Q)$ *for any cost function* δ, *and*
- $\frac{|[\sigma \in L \mid \sigma \in \phi(SN)]|}{|L|} = \frac{|[\sigma \in L \mid \forall_{1 \leq i \leq n} \ \sigma{\restriction}_{A_v(SN^i)} \in \phi(SN^i)]|}{|L|}$.

As the corollary points out: if the discovery algorithm *disc*() is able to create a perfectly fitting model for each sublog, then the overall model is also perfectly fitting. Moreover, if the discovery algorithm has problems finding a perfectly fitting model for a particular sublog, then the overall model will also exhibit these problems. For example, the fraction of traces fitting the overall model equals the fraction of traces fitting all individual models.

As indicated before, the quality of the resulting overall model heavily depends on the discovery algorithm used to discover a model for every sublog. In Section 3 we elaborated on the four quality dimensions for comparing model and log: (1) fitness, (2) simplicity, (3) precision, and (4) generalization [1]. If *disc*() has problems dealing with noisy or incomplete event logs and produces system nets that are unable to replay large parts of the event log, then the overall model will suffer from similar problems. An event log is noisy if it contains exceptional and very rare behavior. An event log is incomplete if only a fraction of the possible behavior has been observed. If the discovery algorithm *disc*() tends to produce overfitting, underfitting, or overly complex models for the sublogs, then the overall model will most likely also be overfitting, underfitting, or overly complex. Recall that the aim of this paper is to provide a generic approach to decompose process mining problems, and not to address the limitations of specific discovery algorithms described in literature [8, 9, 14, 16, 23, 24, 27, 35, 48, 59, 63, 64].

The quality of the resulting overall model also heavily depends on the way the set of activities A is decomposed into activity sets $\mathcal{C} = \{A^1, A^2, \ldots, A^n\}$ with $A = \bigcup \mathcal{C}$. A very naïve approach would be to split each activity into a separate activity set: $\mathcal{C} = \{\{a\} \mid a \in A\}$. The resulting model fragments would be completely disconnected and the composed system net would most likely be underfitting. This problem is illustrated by Figure 8. The other extreme would be $\mathcal{C} = \{A\}$, i.e., there is just one sublog being the original event log.

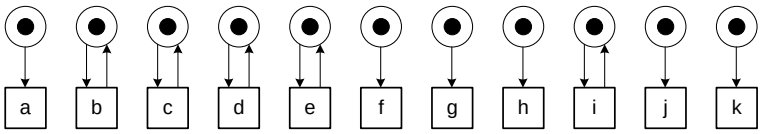


Fig. 8. A possible system net discovered when activity sets are singletons: $\mathcal{C} = \{\{a\} \mid a \in A\}$. The models discovered for subnets are disconnected because each activity appears in only one sublog.

There are various ways to decompose activity sets. For example, one can mine for frequent item sets to find activities that often happen together. Another, probably better performing, approach is to first create a *causal graph*

(A, R) where A is the set of activities and $R \subseteq A \times A$ is a relation on A. The interpretation of $(a_1, a_2) \in R$ is that there is "causal relation" between a_1 and a_2. Most process mining algorithms already build such a graph in a preprocessing step. For example, the α algorithm [9], the heuristic miner [63], and the fuzzy miner [37] scan the event log to see how many times a_1 is followed by a_2. If this occurs above a certain threshold, then it is assumed that $(a_1, a_2) \in R$. Even for large logs it is relatively easy to construct a casual graph (linear in the size of the event log). Moreover, counting the frequencies used to determine a casual graph can be distributed easily by partitioning the cases in the log. Also sampling (determining the graph based on representative examples) may be used to further reduce computation time.

Given a causal graph, we can view the decomposition as a graph partitioning problem [32, 42–44]. There are various approaches to partition the graph such that certain constraints are satisfied while optimizing particular metrics. For example, in [44] a vertex-cut based graph partitioning algorithm is proposed ensuring the balance of the resulting partitions while simultaneously minimizing the number of vertices that are cut (and thus replicated).

Some of the notions in graph partitioning are related to "cohesion and coupling" in software development [30]. Cohesion is the degree to which the elements of a module belong together. Coupling is the degree to which each module relies on the other modules. Typically, one aims at "high cohesion" and "low coupling". In terms of our problem this means that we would like to have activity sets that consist of closely related activities whereas the overlap between the different activity sets is as small as possible while still respecting the causalities.

The causal graph or corresponding Petri net can be decomposed using the so-called Refined Process Structure Tree (RPST) [54, 60] as shown in [53, 52]. The RPST allows for the construction of a hierarchy of SESE (Single-Exit-Single-Entry) components. Slicing the SESE at the desired level of granularity corresponds to a decomposition of the graph [52] that can be used for process mining.

The notion of "passages" defined in [2, 4] provides an alternative approach to decompose such a graph. A passage is a pair of two non-empty sets of activities (X, Y) such that the set of direct successors of X is Y and the set of direct predecessors of Y is X. As shown in [2], any Petri net can be partitioned using passages such that all edges sharing a source vertex or sink vertex are in the same set. This is done to ensure that splits and joins are not decomposed. Note that passages do not necessarily aim at high cohesion and low coupling.

In this paper, we do not investigate the different ways of decomposing activity sets. We just show that for any decomposition, we can distribute process discovery and conformance checking while ensuring correctness in the sense of Theorems 2, 3 and 4 and Corollary 1.

6 Instantiating The Approach

This paper provides a generic approach, i.e., the goal is not to decompose one particular process mining algorithm, but to decompose a large class of existing

process discovery and conformance checking techniques. Hence, the generic approach needs to be instantiated by selecting a particular decomposition approach and an existing process mining technique.

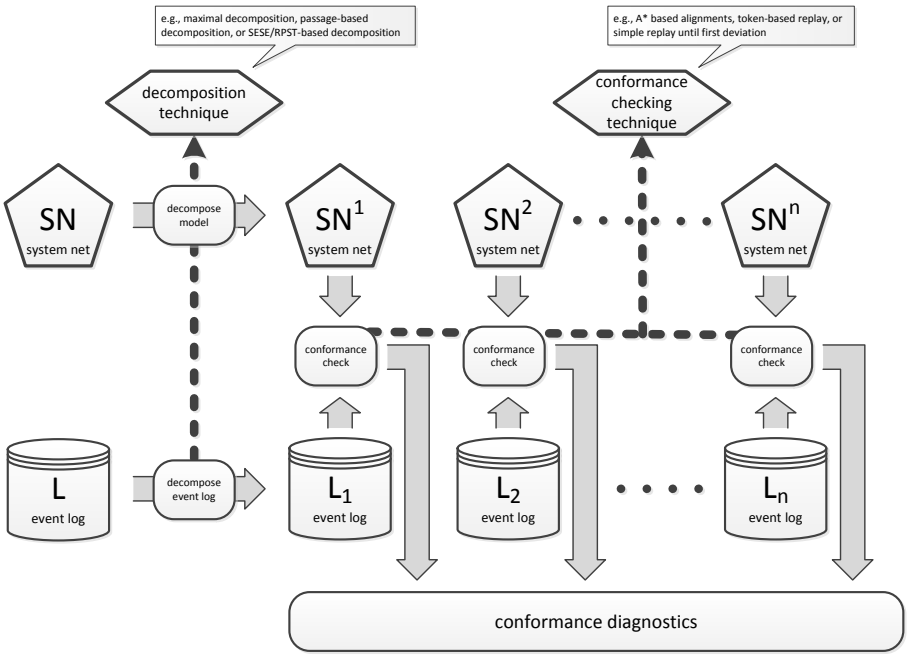
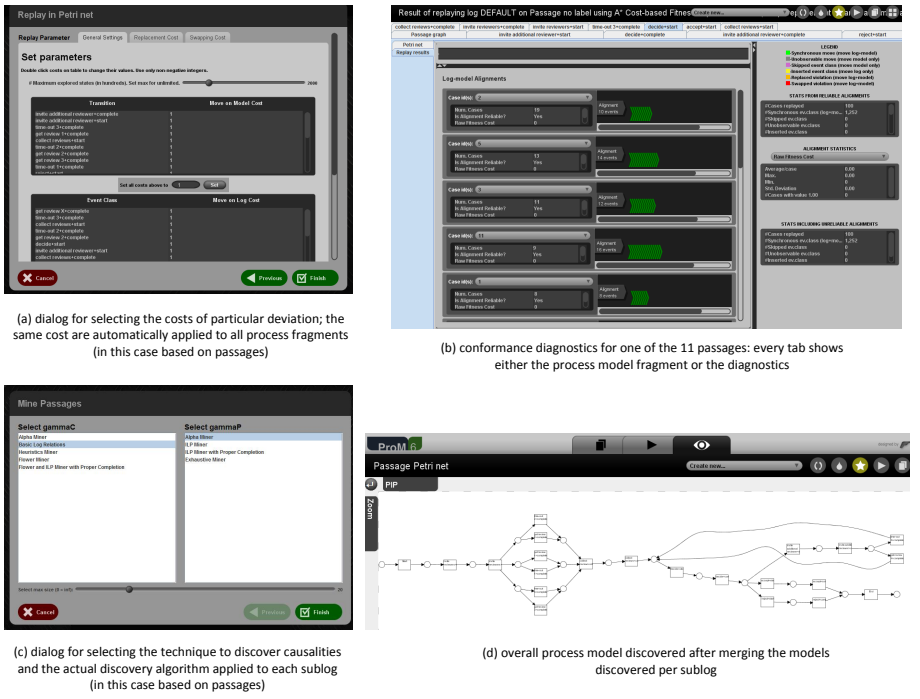


Fig. 9. Overview of decomposed conformance checking showing the configurable elements of the approach: (a) the *decomposition technique* used to split the overall model and event log, and (b) the *conformance checking technique* used to analyze the individual models and sublogs.

Figure 9 shows the configurable elements of our generic decomposed conformance checking approach. There are various ways to create a valid decomposition (cf. Definition 17). The maximal decomposition presented in Definition 18 is just an example. Next to the maximal decomposition, we have experimented with passage-based decomposition [2, 61] and SESE/RPST-based decompositions [53, 52] as described before. For the actual conformance checking of the individual process fragments we have been using the cost-based approach [5, 10] described in Section 3. However, in principle other techniques such as the ones described in [21, 28, 35, 57, 62] can be used. Most of these techniques have been implemented in *ProM*. Figure 10(a-b) shows two screenshots while using the passage-based approach for conformance checking. In case of using SESE/RPST-based decompositions, *ProM* provides a hierarchy of results following the iterative decomposition of SESE-components. The parts of the model with most conformance problems are highlighted [52].



(a) dialog for selecting the costs of particular deviation; the same cost are automatically applied to all process fragments (in this case based on passages)

(b) conformance diagnostics for one of the 11 passages: every tab shows either the process model fragment or the diagnostics

(c) dialog for selecting the technique to discover causalities and the actual discovery algorithm applied to each sublog (in this case based on passages)

(d) overall process model discovered after merging the models discovered per sublog

Fig. 10. Illustration of some of the *ProM* plug-ins using the general results presented in this paper.

Figure 10(c) shows the selection of the technique to create a causal graph (as mentioned in Section 5) and the selection of the actual discovery algorithm applied to each sublog. The result is shown in Figure 10(d). In this particular *ProM* plug-in passages [2] are used to decompose the event log. However, as mentioned before, this is just one example. Figure 11 shows the configurable elements of our generic decomposed process discovery approach. The overall approach allows for the selection of a decomposition technique to create the activity sets and the selection of a process discovery technique to find model fragments. For example, the passage-based *ProM* plug-in Figure 10(c) lists four discovery algorithms (e.g., the α miner and two language-based region miners).

For some initial experimental results using passage-based decomposition we refer to [61]. These experiments show that the actual speed-up varies greatly depending on the decomposition used. Indeed it becomes possible to check the conformance of models that could not be checked before. For example, in [61] it is shown that using the right decomposition the Business Process Intelligence Challenge (BPIC'12) event log can be checked. However, there are two important observations. In spaghetti-like models the largest passage typically consumes most computation time. Hence, it is important to try and balance the sizes of the different process fragments and not only use passage-based decompositions.

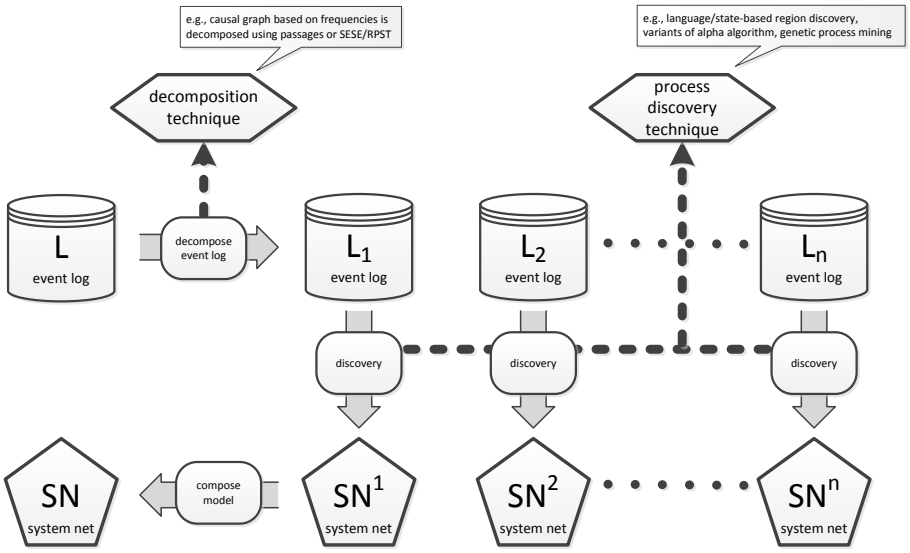


Fig. 11. Overview of decomposed process discovery showing the configurable elements of the approach: (a) the *decomposition technique* used to create the activity sets to create sublogs and (b) the *process discovery technique* used to infer models from sublogs.

Due to overhead, the decomposed approach may even take longer if an unsuitable decomposition is used. Moreover, the alignment-based approach [5, 10] used to check the fitness of process fragments may take longer if complex fragments have substantially more freedom because they are not longer embedded in a restrictive context. This is particular to the alignment-based approach that uses several optimizations to exclude paths that do not reach the final marking.

For experimental conformance checking results based on hierarchies of SESE (Single-Exit-Single-Entry) components computed using the Refined Process Structure Tree (RPST) [54, 60] we refer to [53, 52]. Also here we find several cases where it is possible to provide meaningful diagnostics where the overal alignment-based approach is unable to compute fitness values. Section 6 of [52] describes three models having hundreds of activities that could not be analyzed using conventional techniques, but that could be analyzed after decomposition. However, also here we would like to emphasize that results vary. For good-fitting event logs the speedup is negligible. Moreover, using a very fine-grained decomposition may result in optimistic fitness values (at the event level) as discussed in the context of Theorem 3.

Clearly, more experimentation is needed. However, as shown in this section, several instantiations of the approach have been implemented and the initial results are encouraging. Although Theorems 2, 3 and 4 and Corollary 1 are highly generic, the actual performance gains depend on the actual process mining algorithm that is decomposed and the decomposition selected.

7 Related Work

For an introduction to process mining we refer to [1]. For an overview of best practices and challenges, we refer to the Process Mining Manifesto [41]. The goal of this paper is to decompose challenging process discovery and conformance checking problems into smaller problems [3]. Therefore, we first review some of the techniques available for process discovery and conformance checking.

Process discovery, i.e., discovering a process model from a multiset of example traces, is a very challenging problem and various discovery techniques have been proposed [8, 9, 14, 16, 23, 24, 27, 35, 48, 59, 63, 64]. Many of these techniques use Petri nets during the discovery process and/or to represent the discovered model. It is impossible to provide an complete overview of all techniques here. Very different approaches are used, e.g., heuristics [27, 63], inductive logic programming [35], state-based regions [8, 24, 59], language-based regions [16, 64], and genetic algorithms [48]. Classical synthesis techniques based on regions [29] cannot be applied directly because the event log contains only example behavior. For state-based regions one first needs to create an automaton as described in [8]. Moreover, when constructing the regions, one should avoid overfitting. Language-based regions seem good candidates for discovering transition-bordered Petri nets for subnets [16, 64]. Unfortunately, these techniques still have problems dealing with infrequent/incomplete behavior.

As mentioned before, there are four competing quality criteria when comparing modeled behavior and recorded behavior: fitness, simplicity, precision, and generalization [1]. In this paper, we focused on fitness, but also precision and generalization can also be investigated per subnet. Various conformance checking techniques have been proposed in recent years [5, 10, 11, 13, 21, 28, 35, 50, 51, 57, 62]. Conformance checking can be used to evaluate the quality of discovered processes but can also be used for auditing purposes [6]. Most of the techniques mentioned can be combined with our decomposition approach. The most challenging part is to aggregate the metrics per model fragment and sublog into metrics for the overall model and log. We consider the approach described in [10] to be most promising as it constructs an optimal alignment given an arbitrary cost function. This alignment can be used for computing precision and generalization [5, 51]. However, the approach can be rather time consuming. Therefore, the efficiency gains obtained through decomposition can be considerable for larger processes with many activities and possible subnets.

Little work has been done on the decomposition and distribution of process mining problems [3]. In [55] MapReduce is used to scale event correlation as a preprocessing step for process mining. In [19] an approach is described to distribute genetic process mining over multiple computers. In this approach candidate models are distributed and in a similar fashion also the log can be distributed. However, individual models are not partitioned over multiple nodes. Therefore, the approach in this paper is complementary. Moreover, unlike [19], the decomposition approach in this paper is not restricted to genetic process mining.

More related are the divide-and-conquer techniques presented in [25]. In [25] it is shown that region-based synthesis can be done at the level of synchronized State Machine Components (SMCs). Also a heuristic is given to partition the causal dependency graph into overlapping sets of events that are used to construct sets of SMCs. In this paper we provide a different (more local) partitioning of the problem and, unlike [25] which focuses specifically on state-based region mining, we decouple the decomposition approach from the actual conformance checking and process discovery approaches.

Also related is the work on conformance checking of proclets [31]. Proclets can be used to define so-called artifact centric processes, i.e., processes that are not monolithic but that are composed of smaller interacting processes (called proclets). In [31] it is shown that conformance checking can be done per proclet by projecting the event log onto a single proclet while considering interface transitions in the surrounding proclets.

Several approaches have been proposed to distribute the verification of Petri net properties, e.g., by partitioning the state space using a hash function [18] or by modularizing the state space using localized strongly connected components [45]. These techniques do not consider event logs and cannot be applied to process mining.

Most data mining techniques can be distributed [22], e.g., distributed classification, distributed clustering, and distributed association rule mining [15]. These techniques often partition the input data and cannot be used for the discovery of Petri nets.

This paper generalizes the results presented in [2, 52, 53, 61] to arbitrary Petri net decompositions. In [2, 61] it is shown that so-called "passages" [4] can be used to decompose both process discovery and conformance checking problems. In [53, 52] it is shown that so-called SESE (Single-Exit-Single-Entry) components obtained through the Refined Process Structure Tree (RPST) [54, 60] can be used to decompose conformance checking problems. These papers use a particular particular decomposition structure. However, as shown in this paper, there are many ways to decompose process mining problems. Moreover, this can be done for any net structure and any collection of partly overlapping activity sets.

8 Conclusion

The practical relevance of process mining increases as more event data becomes available. More and more events are being recorded and already today's event logs provide massive amounts of process related data. However, as event logs and processes become larger, many computational challenges emerge. Most algorithms are linear in the size of the event log and exponential in the number of different activities [3]. Therefore, we aim at *decomposing large process mining problems* into collections of smaller process mining problems focusing on restricted sets of activities.

For *conformance checking* we decompose the process model into smaller fragments and split the event log into sublogs. As shown in this paper, conformance

can be analyzed per fragment and sublog. In fact, a trace in the log can be replayed by the overall model if and only if it can be replayed by each of the fragments. We have shown how to compute a *maximal decomposition*, i.e., how to split the overall model and log into fragments and sublogs that are as small as possible.

For *process discovery* we developed a similar approach and have formally proven its correctness. Given a decomposition into activity sets, we can discover a process model per sublog. These models can be stitched together to form an overall model. We would like to stress that we do not propose a new process discovery algorithm: instead, we show that any process discovery algorithm can be decomposed.

Our future work aims at investigating more metrics for conformance checking. In this paper we focused on the most dominant conformance notion: fitness. However, using the notion of alignments we would also like to explore other conformance notions related to precision and generalization. For discovery we now rely on an a priori decomposition of all activities into overlapping activity sets. We already discussed how to find such activity sets based on partitioning the causal graph learned over the model. However, this needs to be investigated in detail also using large scale experimentation. If the activity sets are overlapping too much, the performance gain is minimal. If the activity sets have too little overlap, the resulting model will most likely be underfitting.

Although there are many open questions, this paper lays the foundation for decomposing process mining problems. In general it is much more efficient to solve many smaller problems rather than one big problem. (Recall that most algorithms are linear in the size of the event log and exponential in the number of different activities.) Moreover, using parallel computing, these decomposed problems can be solved concurrently. Obviously, this is very attractive given modern computing infrastructures.

Acknowledgements

This work was supported by the Basic Research Program of the National Research University Higher School of Economics (HSE) in Moscow.

References

1. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.
2. W.M.P. van der Aalst. Decomposing Process Mining Problems Using Passages. In S. Haddad and L. Pomello, editors, *Applications and Theory of Petri Nets 2012*, volume 7347 of *Lecture Notes in Computer Science*, pages 72–91. Springer-Verlag, Berlin, 2012.
3. W.M.P. van der Aalst. Distributed Process Discovery and Conformance Checking. In J. de Lara and A. Zisman, editors, *International Conference on Fundamental Approaches to Software Engineering (FASE 2012)*, volume 7212 of *Lecture Notes in Computer Science*, pages 1–25. Springer-Verlag, Berlin, 2012.

4. W.M.P. van der Aalst. Passages in Graphs. BPM Center Report BPM-12-19, BPMcenter.org, 2012.
5. W.M.P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
6. W.M.P. van der Aalst, K.M. van Hee, J.M. van der Werf, and M. Verdonk. Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. *IEEE Computer*, 43(3):90–93, 2010.
7. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
8. W.M.P. van der Aalst, V. Rubin, H.M.W. Verbeek, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.
9. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
10. A. Adriansyah, B. van Dongen, and W.M.P. van der Aalst. Conformance Checking using Cost-Based Fitness Analysis. In C.H. Chi and P. Johnson, editors, *IEEE International Enterprise Computing Conference (EDOC 2011)*, pages 55–64. IEEE Computer Society, 2011.
11. A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Towards Robust Conformance Checking. In M. zur Muehlen and J. Su, editors, *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010)*, volume 66 of *Lecture Notes in Business Information Processing*, pages 122–133. Springer-Verlag, Berlin, 2011.
12. A. Adriansyah, J. Munoz-Gama, J. Carmona, B.F. van Dongen, and W.M.P. van der Aalst. Alignment Based Precision Checking. In B. Weber, D.R. Ferreira, and B. van Dongen, editors, *Workshop on Business Process Intelligence (BPI 2012)*, Tallinn, Estonia, 2012.
13. A. Adriansyah, N. Sidorova, and B.F. van Dongen. Cost-based Fitness in Conformance Checking. In *International Conference on Application of Concurrency to System Design (ACSD 2011)*, pages 57–66. IEEE Computer Society, 2011.
14. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer-Verlag, Berlin, 1998.
15. R. Agrawal and J.C. Shafer. Parallel Mining of Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, 1996.
16. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.
17. G. Berthelot. Transformations and Decompositions of Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 360–376. Springer-Verlag, Berlin, 1987.
18. M.C. Boukala and L. Petrucci. Towards Distributed Verification of Petri Nets properties. In *Proceedings of the International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS'07)*, pages 15–26. British Computer Society, 2007.

19. C. Bratosin, N. Sidorova, and W.M.P. van der Aalst. Distributed Genetic Process Mining. In H. Ishibuchi, editor, *IEEE World Congress on Computational Intelligence (WCCI 2010)*, pages 1951–1958, Barcelona, Spain, July 2010. IEEE.
20. J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In R. Meersman, S. Rinderle, P. Dadam, and X. Zhou, editors, *OTM Federated Conferences, 20th International Conference on Cooperative Information Systems (CoopIS 2012)*, volume 7565 of *Lecture Notes in Computer Science*, pages 305–322. Springer-Verlag, Berlin, 2012.
21. T. Calders, C. Guenther, M. Pechenizkiy, and A. Rozinat. Using Minimum Description Length for Process Mining. In *ACM Symposium on Applied Computing (SAC 2009)*, pages 1451–1455. ACM Press, 2009.
22. M. Cannataro, A. Congiusta, A. Pugliese, D. Talia, and P. Trunfio. Distributed Data Mining on Grids: Services, Tools, and Applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(6):2451–2465, 2004.
23. J. Carmona and J. Cortadella. Process Mining Meets Abstract Interpretation. In J.L. Balcazar, editor, *ECML/PKDD 210*, volume 6321 of *Lecture Notes in Artificial Intelligence*, pages 184–199. Springer-Verlag, Berlin, 2010.
24. J. Carmona, J. Cortadella, and M. Kishinevsky. A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In *Business Process Management (BPM2008)*, pages 358–373, 2008.
25. J. Carmona, J. Cortadella, and M. Kishinevsky. Divide-and-Conquer Strategies for Process Mining. In U. Dayal, J. Eder, J. Koehler, and H. Reijers, editors, *Business Process Management (BPM 2009)*, volume 5701 of *Lecture Notes in Computer Science*, pages 327–343. Springer-Verlag, Berlin, 2009.
26. M. Castellanos, F. Casati, U. Dayal, and M.C. Shan. A Comprehensive and Automated Approach to Intelligent Business Processes Execution Analysis. *Distributed and Parallel Databases*, 16(3):239–273, 2009.
27. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
28. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
29. P. Darondeau. Unbounded Petri Net Synthesis. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 413–438. Springer-Verlag, Berlin, 2004.
30. H. Dhama. Quantitative Models of Cohesion and Coupling in Software. *Journal of Systems and Software*, 29(1):65–74, 1995.
31. D. Fahland, M. de Leoni, B.F. van Dongen, and W.M.P. van der Aalst. Conformance Checking of Interacting Processes with Overlapping Instances. In S. Rinderle, F. Toumani, and K. Wolf, editors, *Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 345–361. Springer-Verlag, Berlin, 2011.
32. U. Feige, M. Hajiaghayi, and J. Lee. Improved Approximation Algorithms for Minimum-Weight Vertex Separators. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 563–572. ACM, New York, 2005.
33. W. Gaaloul, K. Gaaloul, S. Bhiri, A. Haller, and M. Hauswirth. Log-Based Transactional Workflow Mining. *Distributed and Parallel Databases*, 25(3):193–240, 2009.

34. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
35. S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research*, 10:1305–1340, 2009.
36. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business Process Intelligence. *Computers in Industry*, 53(3):321–343, 2004.
37. C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Berlin, 2007.
38. D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT press, Cambridge, MA, 2001.
39. J. Herbst. *Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen*. PhD thesis, Universität Ulm, November 2001.
40. M. Hilbert and P. Lopez. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025):60–65, 2011.
41. IEEE Task Force on Process Mining. Process Mining Manifesto. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer-Verlag, Berlin, 2012.
42. G. Karpis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
43. B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell Systems Technical Journal*, 49(2), 1970.
44. M. Kim and K. Candan. SBV-Cut: Vertex-Cut Based Graph Partitioning Using Structural Balance Vertices. *Data and Knowledge Engineering*, 72:285–303, 2012.
45. C. Lakos and L. Petrucci. Modular Analysis of Systems Composed of Semi-autonomous Subsystems. In *Application of Concurrency to System Design (ACSD2004)*, pages 185–194. IEEE Computer Society, 2004.
46. J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. Byers. Big Data: The Next Frontier for Innovation, Competition, and Productivity. McKinsey Global Institute, 2011.
47. A.K. Alves de Medeiros. *Genetic Process Mining*. Phd thesis, Eindhoven University of Technology, 2006.
48. A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.
49. T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
50. J. Munoz-Gama and J. Carmona. A Fresh Look at Precision in Process Conformance. In R. Hull, J. Mendling, and S. Tai, editors, *Business Process Management (BPM 2010)*, volume 6336 of *Lecture Notes in Computer Science*, pages 211–226. Springer-Verlag, Berlin, 2010.
51. J. Munoz-Gama and J. Carmona. Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pages 184–191, Paris, France, April 2011. IEEE.

52. J. Munoz-Gama, J. Carmona, and W.M.P. van der Aalst. Conformance Checking in the Large: Partitioning and Topology. BPM Center Report BPM-13-10, BPMcenter.org, 2013.
53. J. Munoz-Gama, J. Carmona, and W.M.P. van der Aalst. Hierarchical Conformance Checking of Process Models Based on Event Logs. In J. Desel and J.M. Colom, editors, *Applications and Theory of Petri Nets 2013*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2013.
54. A. Polyvyanyy, J. Vanhatalo, and H. Völzer. Simplified Computation and Generalization of the Refined Process Structure Tree. In M. Bravetti and T. Bultan, editors, *WS-FM 2010*, volume 6551 of *Lecture Notes in Computer Science*, pages 25–41. Springer-Verlag, Berlin, 2011.
55. H. Reguieg, F. Toumani, H. Motahari Nezhad, and B. Benatallah. Using MapReduce to Scale Events Correlation Discovery for Business Processes Mining. In A. Barros, A. Gal, and E. Kindler, editors, *International Conference on Business Process Management (BPM 2012)*, volume 7481 of *Lecture Notes in Computer Science*, pages 279–284. Springer-Verlag, Berlin, 2012.
56. A. Rozinat and W.M.P. van der Aalst. Decision Mining in ProM. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer-Verlag, Berlin, 2006.
57. 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.
58. A. Sheth. A New Landscape for Distributed and Parallel Data Management. *Distributed and Parallel Databases*, 30(2):101–103, 2012.
59. M. Sole and J. Carmona. Process Mining from a Basis of Regions. In J. Lilius and W. Penczek, editors, *Applications and Theory of Petri Nets 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245. Springer-Verlag, Berlin, 2010.
60. J. Vanhatalo, H. Völzer, and J. Koehler. The Refined Process Structure Tree. *Data and Knowledge Engineering*, 68(9):793–818, 2009.
61. H.M.W. Verbeek and W.M.P. van der Aalst. Decomposing Replay Problems: A Case Study. BPM Center Report BPM-13-09, BPMcenter.org, 2013.
62. J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens. A Robust F-measure for Evaluating Discovered Process Models. In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pages 148–155, Paris, France, April 2011. IEEE.
63. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.
64. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010.
65. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, Berlin, 2007.