Process Discovery and Conformance Checking Using Passages

W.M.P. van der Aalst
Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tue.nl

H.M.W. Verbeek
Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands.
h.m.w.verbeek@tue.nl

Abstract. The two most prominent *process mining* tasks are *process discovery* (i.e., learning a process model from an event log) and *conformance checking* (i.e., diagnosing and quantifying differences between observed and modeled behavior). The increasing availability of event data makes these tasks highly relevant for process analysis and improvement. Therefore, process mining is considered to be one of the key technologies for Business Process Management (BPM). However, as event logs and process models grow, process mining becomes more challenging. Therefore, we propose an approach to *decompose process mining problems* into smaller problems using the notion of *passages*. 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. Any Petri net can be partitioned using passages. Moreover, process discovery and conformance checking can be done *per passage* and the results can be aggregated. This has advantages in terms of efficiency and diagnostics. Moreover, passages can be used to distribute process mining problems over a network of computers. Passages are supported through *ProM* plug-ins that automatically decompose process discovery and conformance checking tasks.

Keywords: process mining, conformance checking, process discovery, distributed computing, business process management

Address for correspondence: Wil van der Aalst, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, PO Box 513, 5600 MB, Eindhoven, The Netherlands. E-mail: w.m.p.v.d.aalst@tue.nl. WWW: vdaalst.com.

1. Introduction

The term "Big Data" refers to the spectacular growth of data and the potential economic value such data has when analyzed using clever algorithms [28, 31]. The exponential growth of event data (e.g., from 2.6 exabytes in 1986 to 295 exabytes in 2007 according to [28]) 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, the analysis of *hand-made* process models will become less important given the omnipresence of event data. This is the reason why *process mining* is one of the "hot" topics in Business Process Management (BPM). Process mining aims to *discover, monitor and improve real processes by extracting knowledge from event logs* readily available in today's information systems [2].

The starting point for process mining 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 [2].

The growing interest in process mining is illustrated by the *Process Mining Manifesto* [29] 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.

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 [7, 15, 37, 21, 42], the α algorithm [8], and various conformance checking techniques [9, 33, 34, 36]. Other techniques use alternative internal representations (C-nets, heuristic nets, etc.) that can easily be converted to (labeled) Petri nets [2].

In this paper, we focus on the following two main process mining tasks:

- *Process discovery*: 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*: 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 tasks 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, heuristic nets, etc. In fact, also different types of Petri nets can be employed, e.g., safe Petri nets, labeled Petri nets, free-choice Petri nets, etc.

Process mining problems tend to be very challenging. There are obvious challenges that also apply to many other data mining and machine learning problems, e.g., dealing with noise, concept drift, and the need to explore a large and complex search space. For example, event logs may contain millions of events. Moreover, there are also some specific problems that make process discovery even more challenging:

- there are *no negative examples* (i.e., a log shows what has happened but does not show what could not happen);
- due to concurrency, loops, and choices the *search space has a complex structure* and the log typically contains only a *fraction* of all possible behaviors;
- there is *no clear relation between the size of a model and its behavior* (i.e., a smaller model may generate more or less behavior although classical analysis and evaluation methods typically assume some monotonicity property); and
- there is a need to balance between four (often) *competing quality criteria* (see Section 4): (1) *fitness* (be able to generate the observed behavior), (2) *simplicity* (avoid large and complex models), (3) *precision* (avoid "underfitting"), and (4) *generalization* (avoid "overfitting").

Process discovery and conformance checking are related problems. This becomes evident when considering *genetic* process discovery techniques [32, 17]. 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 [2, 7, 8, 13, 15, 20, 21, 23, 27, 32, 37, 40, 42] and conformance checking [5, 9, 10, 12, 18, 24, 27, 33, 34, 36, 38] 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 proposed a *divide-and-conquer approach for process mining*. This approach uses a new concept: *passages*. A passage is a pair of two sets of activity nodes (X, Y) such that $X\bullet = Y$ (i.e., the activity nodes in X influence the enabling of the activity nodes in Y) and $X = \bullet Y$ (i.e., the activity nodes in Y are influenced by the activity nodes in X). The notion of passages will be formalized in terms of graphs and labeled Petri nets. Passages can be used to *decompose process discovery and conformance checking problems into smaller problems*. By localizing process mining techniques to passages, more refined techniques can be used. Assuming that the event log and process model can be decomposed into many small passages, substantial speedups are possible. Moreover, passages can also be used to *distribute* process mining problems over a network of computers (e.g., a grid or cloud infrastructure).

This paper focuses on the theoretical foundations of process mining based on passages and extends our earlier conference paper [3] in various respects. For example, the results are generalized to a larger class of models and different passage partitionings. Moreover, we describe new ProM plug-ins based on this work and present experimental results.

The remainder is organized as follows. Section 2 introduces various preliminaries (Petri nets, event logs, etc.). Section 3 defines the notion of passages for arbitrary graphs and analyzes their properties. Section 4 discusses quality criteria for process mining and introduces the notion of alignments to compute the level of conformance. The notion of passages is used in Section 5 to decompose the overall conformance checking problem into a set of localized conformance checking problems. Section 6 shows how the same ideas can be used for process discovery, i.e., after determining the causal structure and related passages, the overall process discovery problem can be decomposed into a set of local process discovery problems. Section 7 shows, using a real-life event log, that dividing the problem into smaller problems using passages may indeed lead to significant speedups. Section 8 describes some of the ProM plug-ins that use passages to decompose process mining tasks into smaller tasks handled by conventional algorithms. Related work is discussed in Section 9. Section 10 concludes the paper.

2. Preliminaries

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

2.1. Graphs and Paths

First, we introduce basic graphs notations. We will use graphs to represent process models (i.e., Petri nets) and the causal structure (also referred to as "skeleton") of processes.

Definition 2.1. (Graph)
A graph is a pair $G = (N, E)$ comprising a set N of nodes and a set $E \subseteq N \times N$ of edges.

For a graph $G = (N, E)$ and $n \in N$, we define preset $\overset{G}{\bullet} n = \{n' \in N \mid (n', n) \in E\}$ (direct predecessors) and postset $n \overset{G}{\bullet} = \{n' \in N \mid (n, n') \in E\}$ (direct successors). This can be generalized to sets, i.e., for $X \subseteq N$: $\overset{G}{\bullet} X = \bigcup_{n \in X} \overset{G}{\bullet} n$ and $X \overset{G}{\bullet} = \bigcup_{n \in X} n \overset{G}{\bullet}$. The superscript G can be omitted if the graph is clear from the context.

Sequences are used to represent paths in a graph and traces in an event log. As defined next, $\sigma_1 \cdot \sigma_1$ is the concatenation of two sequences and $\sigma\upharpoonright_X$ is the projection of σ on X, e.g., $\langle a, b, c, c, b, a\rangle\upharpoonright_{\{a,b\}} = \langle a, b, b, a\rangle$.

Definition 2.2. (Sequences)
Let A be a set. $\sigma = \langle a_1, a_2, \dots, a_n\rangle \in A^*$ denotes a *sequence* over A of length n. For $\sigma_1, \sigma_2 \in A^*$: $\sigma_1 \cdot \sigma_2$ is the concatenation of two sequences, e.g., $\langle a\rangle \cdot \langle b, c\rangle = \langle a, b, c\rangle$. $\sigma\upharpoonright_X$ is the *projection* of σ on $X \subseteq A$, i.e., $\upharpoonright_X \in A^* \to X^*$ and is defined recursively: (1) $\langle\ \rangle\upharpoonright_X = \langle\ \rangle$, (2) for $a \in X$ and $\sigma \in A^*$: $(\langle a\rangle \cdot \sigma)\upharpoonright_X = \langle a\rangle \cdot \sigma\upharpoonright_X$, and (3) for $a \in A \setminus X$ and $\sigma \in A^*$: $(\langle a\rangle \cdot \sigma)\upharpoonright_X = \sigma\upharpoonright_X$.

A path in a graph is a sequence of nodes connected through edges. We use the notation $x \overset{\sigma:E'\#Q}{\rightsquigarrow} y$ to state that there is a non-empty path σ from node x to node y using edges in E' not visiting any nodes in Q.

Definition 2.3. (Path)
Let $G = (N, E)$ be a graph, $x, y \in N$, $E' \subseteq E$, and $Q \subseteq N$. $x \overset{\sigma:E'\#Q}{\rightsquigarrow} y$ if and only if σ is a sequence such that $\sigma = \langle n_1, n_2, \dots n_k\rangle$, $k > 1$, $x = n_1$, $y = n_k$, for all $1 \leq i < k$: $(n_i, n_{i+1}) \in E'$, and for all $1 < i < k$: $n_i \notin Q$. Derived notations:

- $x \overset{E'\#Q}{\rightsquigarrow} y$ if and only if there exists a sequence σ such that $x \overset{\sigma:E'\#Q}{\rightsquigarrow} y$,
- $nodes(x \overset{E'\#Q}{\rightsquigarrow} y) = \{n \in \sigma \mid \sigma \in N^* \wedge x \overset{\sigma:E'\#Q}{\rightsquigarrow} y\}$, and
- for $X, Y \subseteq N$: $nodes(X \overset{E'\#Q}{\rightsquigarrow} Y) = \bigcup_{(x,y) \in X \times Y} nodes(x \overset{E'\#Q}{\rightsquigarrow} y)$.

Consider the graph $G = (N, E)$ in Fig. 2 which is later used to introduce the notion of passages. $a \overset{E\#Q}{\rightsquigarrow} i$ holds for $Q = \{b, d, e, g\}$ because of the path $\sigma = \langle a, c, f, h, i\rangle$. $a \overset{E\#Q}{\rightsquigarrow} i$ does not hold if $Q = \{g, h\}$ because all paths connecting a to i need to visit g or h. If $Q = \{d, e, g\}$, then $nodes(a \overset{E\#Q}{\rightsquigarrow} i) = \{a, b, c, f, h, i\}$ because of the two paths connecting a to i not visiting any of the nodes in Q.

2.2. Multisets

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 *finite* 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., $x \in b_2$, $b_2 \uplus b_3 = b_4$, and $b_5 \setminus b_2 = b_3$. $|b|$ is the size of multiset b, e.g., $|b_1| = 0$ and $|b_5| = 6$. $\{a \in b\}$ denotes the set with all elements a for which $b(a) \geq 1$.

Let $\sigma = \langle a_1, a_2, \ldots, a_n\rangle \in A^*$ be a sequence and $b = [a_1, a_2, \ldots, a_n] \in \mathcal{B}(A)$ the corresponding multiset. $a \in \sigma$ if and only if $a \in b$. $\sum_{a\in\sigma} f(a) = \sum_{a\in b} f(a) = f(a_1) + f(a_2) + \ldots + f(a_n)$ for some function f.

Given a function $f \in A \to B$ and a multiset $b \in \mathcal{B}(A)$: $[f(a) \mid a \in b]$ denotes the multiset over B where element $f(a)$ appears $\sum_{x\in A | f(x)=f(a)} b(x)$ times.

2.3. Petri Nets

Most of the results presented in the paper, can be adapted for various process modeling notations. However, we use Petri nets to formalize the main ideas and to prove their correctness.

Definition 2.4. (Petri Net)
A *Petri net* is a tuple $PN = (P, T, F)$ having a finite set of places P, a finite set of transitions T, and a flow relation $F \subseteq (P \times T) \cup (T \times P)$.

Figure 1 shows an example Petri net $PN = (P, T, F)$ with $P = \{start, c1, \ldots, c5, end\}$, $T = \{a, b, \ldots, h\}$, and $F = \{(start, a), (a, c1), (a, c2), \ldots, (h, end)\}$. The state of a Petri net, called *marking*, is a multiset of places indicating how many *tokens* each place contains. Any $M \in \mathcal{B}(P)$ is a marking. $[start]$ is the initial marking shown in Fig. 1. 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 $c4$.

A Petri net $PN = (P, T, F)$ defines a graph $(P \cup T, F)$. Hence, for any $x \in P \cup T$, $\overset{PN}{\bullet} x = \{y \mid (y, x) \in F\}$ (input nodes) and $x \overset{PN}{\bullet} = \{y \mid (x, y) \in F\}$ (output nodes). As before, we drop the superscript if it is clear from the context.

A transition $t \in T$ is *enabled* in marking M, denoted as $M[t\rangle$, if each of its input places $\bullet t$ contains at least one token. Consider the Petri net in Fig. 1 with $M = [c3, c4]$: $M[e\rangle$ because both input places are marked.

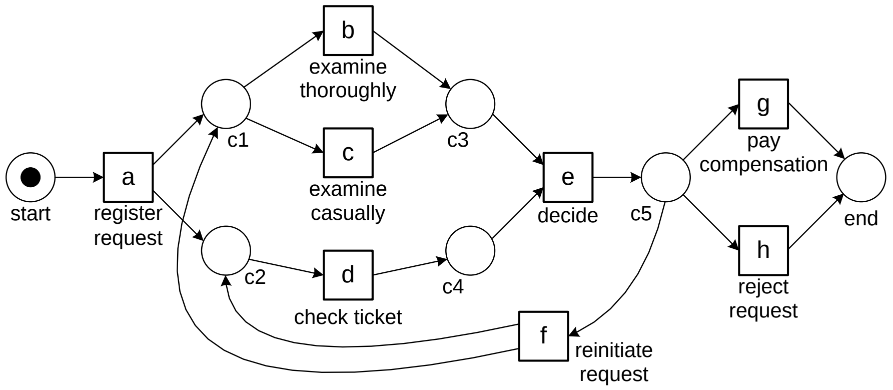

Figure 1. A Petri net.

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. $M[t\rangle M'$ denotes that t is enabled in M and firing t results in marking M'. For example, $[start][a\rangle[c1, c2]$ and $[c3, c4][e\rangle[c5]$ for the net in Fig. 1.

Let $\sigma = \langle t_1, t_2, \ldots, t_n\rangle \in T^*$ be a sequence of transitions. $M[\sigma\rangle M'$ denotes that there is a set of markings $M_0, M_1, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $M_i[t_{i+1}\rangle M_{i+1}$ for $0 \leq i < n$. A marking M' is *reachable* from M if there exists a σ such that $M[\sigma\rangle M'$. For example, $[start][\sigma\rangle[end]$ for $\sigma = \langle a, b, d, e, g\rangle$.

Definition 2.5. (Labeled Petri Net)
A *labeled* Petri net $PN = (P, T, F, T_v)$ is a Petri net (P, T, F) with visible transitions $T_v \subseteq T$. Let $\sigma_v = \langle t_1, t_2, \ldots, t_n\rangle \in T_v^*$ be a sequence of visible transitions. $M[\sigma_v \rhd M'$ if and only if there is a sequence $\sigma \in T^*$ such that $M[\sigma\rangle M'$ and the projection of σ on T_v yields σ_v, i.e., $\sigma_v = \sigma\upharpoonright_{T_v}$.

If we assume $T_v = \{a, e, g, h\}$ for the Petri net in Fig. 1, then $[start][\sigma_v \rhd [end]$ for $\sigma_v = \langle a, e, e, e, e, g\rangle$ (i.e., b, c, d, and f are invisible). Note that we consider a very limited form of labeling because any event in an event log need to deterministically refer to a transition.

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 Fig. 1 we are interested in firing sequences starting in $M_i = [start]$ and ending in $M_o = [end]$. Therefore, we define the notion of a *system net*.

Definition 2.6. (System Net)
A system net is a triplet $SN = (PN, M_i, M_o)$ where $PN = (P, T, F, T_v)$ is a Petri net with visible transitions T_v, $M_i \in \mathcal{B}(P)$ is the initial marking, and $M_o \in \mathcal{B}(P)$ is the final marking.

Given a system net, $\tau(SN)$ is the set of all possible visible full traces, i.e., firing sequences starting in M_i and ending in M_o projected onto the set of visible transitions.

Definition 2.7. (Traces)
Let $SN = (PN, M_i, M_o)$ be a system net. $\tau(SN) = \{\sigma_v \mid M_i[\sigma_v \rhd M_o\}$ is the set of visible full traces.

In the remainder, we will simply refer to such traces as the visible traces of SN. If we assume $T_v = \{a, e, f, g, h\}$ for the Petri net in Fig. 1, then $\tau(SN) = \{\langle a, e, g\rangle, \langle a, e, h\rangle, \langle a, e, f, e, g\rangle, \langle a, e, f, e, h\rangle, \ldots\}$.

Definition 2.8. (Connected)
Let $SN = (PN, M_i, M_o)$ with $PN = (P, T, F, T_v)$ be a system net. SN is *connected* if and only if $P \cup T = \mathit{nodes}(\{p \in M_i\} \overset{F\#\emptyset}{\rightsquigarrow} \{p \in M_o\})$ and $\tau(SN) \neq \emptyset$.

In a connected system net all nodes are on a path from some initially marked place to some place marked in the final marking. Moreover, there should be at least one trace leading from M_i to M_o (ensured by the requirement $\tau(SN) \neq \emptyset$). System nets that have no traces cannot be used for conformance checking and are not very meaningful from a process discovery point of view. When decomposing a net into passages we will often assume it is connected, e.g., the connectedness requirement ensures that all nodes end up in at least one passage in Theorem 5.4.

2.4. 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 2.9. (Trace, Event Log)
Let A be a finite set of activities. A *trace* $\sigma \in A^*$ is a finite sequence of activities. $L \in \mathcal{B}(A^*)$ is an *event log*, i.e., a finite 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 may 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 in this paper can easily be extended to event logs with more information.

An example log is $L_1 = [\langle a, e, g\rangle^{10}, \langle a, e, h\rangle^5, \langle a, e, f, e, g\rangle^3, \langle a, e, f, e, h\rangle^2]$. L_1 contains information about 20 cases, e.g., 10 cases followed trace $\langle a, e, g\rangle$. There are $10 \times 3 + 5 \times 3 + 3 \times 5 + 2 \times 5 = 70$ events in total.

The projection function $\upharpoonright_X$ (cf. Definition 2.2) is generalized to event logs, i.e., for some event log $L \in \mathcal{B}(A^*)$ and set $X \subseteq A$: $L\upharpoonright_X = [\sigma\upharpoonright_X \mid \sigma \in L]$. For example, $L_1\upharpoonright_{\{a,g,h\}} = [\langle a, g\rangle^{13}, \langle a, h\rangle^7]$. Note that all e and f events have been removed.

3. Passages

To decompose large process mining problems into smaller problems, we partition process models using the notion of *passages* introduced in this paper. A passage is a pair of non-empty sets of nodes (X, Y) such that the set of direct successors of X is Y and the set of direct predecessors of Y is X.

Definition 3.1. (Passage)
Let $G = (N, E)$ be a graph. $P = (X, Y)$ is a *passage* if and only if $\emptyset \neq X \subseteq N$, $\emptyset \neq Y \subseteq N$, $X \overset{G}{\bullet} = Y$, and $X = \overset{G}{\bullet} Y$. X is the set of *input nodes* of P, and Y is the set of *output nodes* of P. $pas(G)$ is the set of all passages of G.

Consider the sets $X = \{b, c, d\}$ and $Y = \{d, e, f\}$ in Fig. 2 (for the moment ignore the numbers in the graph). $X\bullet = \{b, c, d\}\bullet = \{d, e, f\} = Y$ and $X = \{b, c, d\} = \bullet\{d, e, f\} = \bullet Y$, so (X, Y) is indeed a passage.

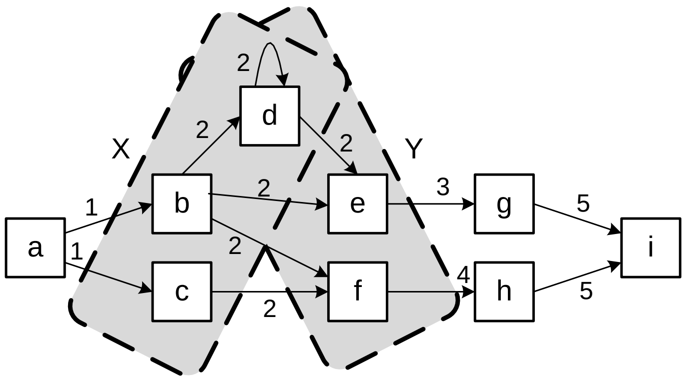

Figure 2. A graph with five minimal passages: $P_1 = (\{a\}, \{b, c\})$, $P_2 = (\{b, c, d\}, \{d, e, f\})$, $P_3 = (\{e\}, \{g\})$, $P_4 = (\{f\}, \{h\})$, and $P_5 = (\{g, h\}, \{i\})$. Passage P_2 is highlighted and edges carry numbers to refer to the minimal passage they belong to.

Definition 3.2. (Passages Operators)
Let $P_1 = (X_1, Y_1)$ and $P_2 = (X_2, Y_2)$ be two passages.

- $P_1 \leq P_2$ if and only if $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$,
- $P_1 < P_2$ if and only if $P_1 \leq P_2$ and $P_1 \neq P_2$,
- $P_1 \cup P_2 = (X_1 \cup X_2, Y_1 \cup Y_2)$,
- $P_1 \cap P_2 = (X_1 \cap X_2, Y_1 \cap Y_2)$,
- $P_1 \setminus P_2 = (X_1 \setminus X_2, Y_1 \setminus Y_2)$,
- $P_1 \rhd P_2$ if and only if $Y_1 \cap X_2 \neq \emptyset$, and
- $P_1 \# P_2$ if and only if $(X_1 \cap X_2) \cup (Y_1 \cap Y_2) = \emptyset$.

$P_1 \rhd P_2$ means that P_2 *follows* P_1, i.e., an output node of P_1 is an input node of P_2. Two passages P_1 and P_2 are called *disjoint* if $P_1 \# P_2$. Consider the following two concrete passages in Fig. 2: $P_4 = (\{f\}, \{h\})$ and $P_5 = (\{g, h\}, \{i\})$. $P_4 \rhd P_5$ because node h is an output node of P_4 and an input node of P_5. $P_4 \# P_5$ because $(\{f\} \cap \{g, h\}) \cup (\{h\} \cap \{i\}) = \emptyset$.

Lemma 3.3. (Relating Passages)
Let $G = (N, E)$ be a graph with passages $P_1 = (X_1, Y_1) \in pas(G)$ and $P_2 = (X_2, Y_2) \in pas(G)$.

- $P_3 = P_1 \setminus P_2$ is a passage if $P_3 \neq (\emptyset, \emptyset)$,
- $P_4 = P_2 \setminus P_1$ is a passage if $P_4 \neq (\emptyset, \emptyset)$,
- $P_5 = P_1 \cap P_2$ is a passage if $P_5 \neq (\emptyset, \emptyset)$, and

- $P_6 = P_1 \cup P_2$ is a passage.

Proof:
Let $P_3 = (X_3, Y_3) = P_1 \setminus P_2$, $P_4 = (X_4, Y_4) = P_2 \setminus P_1$, $P_5 = (X_5, Y_5) = P_1 \cap P_2$, and $P_6 = (X_6, Y_6) = P_1 \cup P_2$ as sketched in Fig. 3(a).

Assume $P_3 \neq (\emptyset, \emptyset)$. Remains to prove that $\emptyset \neq X_3 \subseteq N$, $\emptyset \neq Y_3 \subseteq N$, $X_3\bullet = Y_3$, and $X_3 = \bullet Y_3$. (Recall that $X_3 = X_1 \setminus X_2$ and $Y_3 = Y_1 \setminus Y_2$.)

- For any $x \in X_3$ there is at least one edge $(x, y') \in E$ because $x \in X_1$ and P_1 is a passage. For all such edges: $y' \in Y_3$ because $y' \in Y_1$ ($x \in X_1$ and P_1 is a passage) and $y' \notin Y_2$ (because P_2 is a passage and $x \notin X_2$). Note that if $(x, y') \in E$, $x \in X_3$, and $y' \in Y_2$, we find a contradiction (P_2 cannot be a passage without x as input node). Hence, $\emptyset \neq x\bullet \subseteq Y_3$ for $x \in X_3$ which implies that $X_3\bullet \subseteq Y_3$.
- For any $y \in Y_3$ there is at least one edge $(x', y) \in E$ because $y \in Y_1$ and P_1 is a passage. For all such edges: $x' \in X_3$ because $x' \in X_1$ ($y \in Y_1$ and P_1 is a passage) and $x' \notin X_2$ (because P_2 is a passage and $y \notin Y_2$). Hence, $\emptyset \neq \bullet y \subseteq X_3$ for $y \in Y_3$ which implies that $\bullet Y_3 \subseteq X_3$.
- Since $X_3\bullet \subseteq Y_3$, $\bullet Y_3 \subseteq X_3$, all nodes in X_3 have an outgoing edge, and all nodes in Y_3 have an incoming edge, we conclude: $X_3\bullet = Y_3$, and $X_3 = \bullet Y_3$. Hence, $P_3 = P_1 \setminus P_2$ is a passage.

In a similar fashion it can be shown that $P_4 = P_2 \setminus P_1$ is a passage if $P_4 \neq (\emptyset, \emptyset)$.

Assume $P_5 \neq (\emptyset, \emptyset)$. Remains to prove that $\emptyset \neq X_5 \subseteq N$, $\emptyset \neq Y_5 \subseteq N$, $X_5\bullet = Y_5$, and $X_5 = \bullet Y_5$. (Recall that $X_5 = X_1 \cap X_2$ and $Y_5 = Y_1 \cap Y_2$.)

- For any $x \in X_5$ there is at least one edge $(x, y') \in E$ because $x \in X_1$ and P_1 is a passage (and $x \in X_2$ and P_2 is a passage). For all such edges: $y' \in Y_5$ because $y' \in Y_1$ ($x \in X_1$ and P_1 is a passage) and $y' \in Y_2$ ($x \in X_2$ and P_2 is a passage). Hence, $\emptyset \neq x\bullet \subseteq Y_5$ for $x \in X_5$ which implies that $X_5\bullet \subseteq Y_5$.
- For any $y \in Y_5$ there is at least one edge $(x', y) \in E$ because $y \in Y_1$ and P_1 is a passage (and $y \in Y_2$ and P_2 is a passage). For all such edges: $x' \in X_5$ because $x' \in X_1$ ($y \in Y_1$ and P_1 is a passage) and $x' \in X_2$ ($y \in Y_2$ and P_2 is a passage). Hence, $\emptyset \neq \bullet y \subseteq X_5$ for $y \in Y_5$ which implies that $\bullet Y_5 \subseteq X_5$.
- Since $X_5\bullet \subseteq Y_5$, $\bullet Y_5 \subseteq X_5$, all nodes in X_5 have an outgoing edge, and all nodes in Y_5 have an incoming edge, we conclude: $X_5\bullet = Y_5$, and $X_5 = \bullet Y_5$. Hence, $P_5 = P_1 \cap P_2$ is a passage.

In a similar fashion it can be shown that $P_6 = P_1 \cup P_2$ is a passage. There is no need to require $P_6 \neq (\emptyset, \emptyset)$ because the union of two passages will always contain edges. □

Definition 3.4. (Edge Representation of Passages)
Let $G = (N, E)$ be a graph. For any $P = (X, Y)$ with $X \subseteq N$ and $Y \subseteq N$: $\widehat{P} = E \cap (X \times Y)$ denotes the set of edges of P.

Sets of edges can be used to fully characterize passages. In fact, any passage $P = (X, Y)$ with edges $Z = \widehat{P}$ is uniquely defined by (1) X, (2) Y, and (3) Z separately. If X is known, then we can derive $Y = X\bullet$ and $Z = E \cap (X \times Y)$ knowing that P is a passage. If Y is known, then $X = \bullet Y$ and $Z = E \cap (X \times Y)$. If Z is known, then $X = \{x \mid (x, y) \in Z\}$ and $Y = \{y \mid (x, y) \in Z\}$ because there cannot be input or output nodes without corresponding edges.

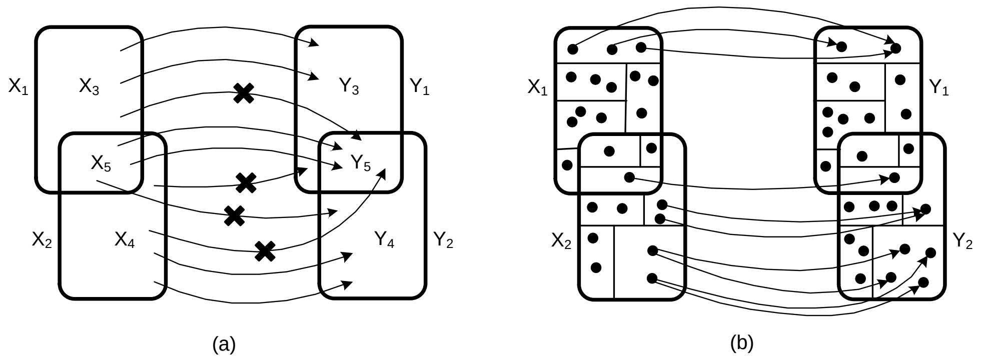

Figure 3. Understanding the fabric of passages: (a) only edges between pairs (X_3, Y_3), (X_4, Y_4), and (X_5, Y_5) are possible if (X_1, Y_1) and (X_2, Y_2) are passages, and (b) passages are composed of minimal passages. Note that input nodes X_i may overlap with output nodes Y_j but this is not shown to avoid cluttering the diagrams.

Lemma 3.5. (Relating Passages in Terms of Edges)
Let $G = (N, E)$ be a graph with passages $P_1, P_2 \in \mathit{pas}(G)$ and let $P_3 = P_1 \setminus P_2$, $P_4 = P_2 \setminus P_1$, $P_5 = P_1 \cap P_2$, and $P_6 = P_1 \cup P_2$. The following properties hold:

- $\widehat{P_3} = \widehat{P_1} \setminus \widehat{P_2}$,
- $\widehat{P_4} = \widehat{P_2} \setminus \widehat{P_1}$,
- $\widehat{P_5} = \widehat{P_1} \cap \widehat{P_2}$, and
- $\widehat{P_6} = \widehat{P_1} \cup \widehat{P_2}$.

Proof:
Note that the naming of passages is similar to Fig. 3(a). As shown in the proof of Lemma 3.3: $X_3\bullet = Y_3$, $X_3 = \bullet Y_3$, $X_4\bullet = Y_4$, $X_4 = \bullet Y_4$, and $X_5\bullet = Y_5$, $X_5 = \bullet Y_5$. Moreover, X_3, X_4, and X_5 partition X_6, and Y_3, Y_4, and Y_5 partition Y_6. These properties also hold when one of more sets are empty, e.g., if $X_3 = \emptyset$, then still $X_3\bullet = \emptyset\bullet = \emptyset = Y_3$. This implies that any edge in $E \cap (X_6 \times Y_6)$ belongs to $E \cap (X_3 \times Y_3)$, $E \cap (X_4 \times Y_4)$, or $E \cap (X_5 \times Y_5)$ as sketched in Fig. 3(a). The partitioning of the edges over these three sets can be used to prove the properties listed. $\square$

Corollary 3.6. (Comparing Passages in Terms of Edges)
Let $G = (N, E)$ be a graph with passages $P_1, P_2 \in \mathit{pas}(G)$. The following properties hold:

- $P_1 \leq P_2$ if and only if $\widehat{P_1} \subseteq \widehat{P_2}$,
- $P_1 < P_2$ if and only if $\widehat{P_1} \subset \widehat{P_2}$, and
- $P_1 \# P_2$ if and only if $\widehat{P_1} \cap \widehat{P_2} = \emptyset$.

To decompose process mining problems, we aim to partition a process model into smaller models using the notion of passages. Therefore, we define the notion of a *passage partitioning*. Consider for example the five passages shown in Fig. 2. The edges in Fig. 2 have numbers corresponding to the passage they belong to, e.g., edges (a, b) and (a, c) have a label "1" showing that they belong to passage $P_1 = (\{a\}, \{b, c\})$. Passages $\{P_1, P_2, P_3, P_4, P_5\}$ in Fig. 2 form a passage partitioning because the passages are pairwise disjoint and together they cover all edges.

Definition 3.7. (Passage Partitioning)
Let $G = (N, E)$ be a graph. $\{P_1, P_2, \ldots, P_n\}$ is a *passage partitioning* if and only if

1. $P_1, P_2, \ldots, P_n \in pas(G)$ are passages,
2. for all $1 \leq i < j \leq n$: $P_i \# P_j$, and
3. $E = \bigcup_{1 \leq i \leq n} \widehat{P_i}$.

Consider the graph in Fig. 2 and $P_6 = P_1 \cup P_2 = (\{a, b, c, d\}, \{b, c, d, e, f\})$ and $P_7 = P_3 \cup P_4 \cup P_5 = (\{e, f, g, h\}, \{g, h, i\})$. P_6 and P_7 are passages because the union of passages is guaranteed to be a passage (cf. Lemma 3.3). $\{P_6, P_7\}$ is a passage partitioning because $P_6 \# P_7$ and $\widehat{P_6} \cup \widehat{P_7} = E$. As Fig. 4 shows, relations between passages in a passage partitioning can be visualized using the follows relation.

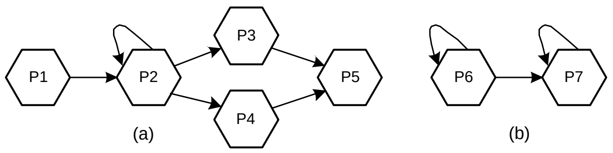

Figure 4. Two graphs based on the follows relation ($\rhd$) showing dependencies between passages in a passages partitioning, e.g., $P_2 \rhd P_4$ because $Y_2 \cap X_4 = \{f\} \neq \emptyset$.

A passage partitioning $\{P_1, P_2, \ldots, P_n\}$ defines an equivalence relation on the edges in a graph: $(x_1, y_1) \sim (x_2, y_2)$ if and only if there is a P_i such that $\{(x_1, y_1), (x_2, y_2)\} \subseteq \widehat{P_i}$. It is easy to see that $\sim$ is reflexive (i.e., $(x, y) \sim (x, y)$) , symmetric (i.e., $(x_1, y_1) \sim (x_2, y_2)$ if and only if $(x_2, y_2) \sim (x_1, y_1)$), and transitive (i.e., $(x_1, y_1) \sim (x_2, y_2)$ and $(x_2, y_2) \sim (x_3, y_3)$ implies $(x_1, y_1) \sim (x_3, y_3)$). Given passage partitioning $\{P_1, P_2, P_3, P_4, P_5\}$ in Fig. 2: $(b, d) \sim (b, e) \sim (b, f) \sim (c, f) \sim (d, d) \sim (d, e)$, i.e., the arcs having label "2" form an equivalence class.

To prove that a passage partitioning always exists we introduce the notion of *minimal passages*. A passage is *minimal* if it does not "contain" a smaller passage.

Definition 3.8. (Minimal Passage)
Let $G = (N, E)$ be a graph with passages $pas(G)$. $P \in pas(G)$ is *minimal* if there is no $P' \in pas(G)$ such that $P' < P$. $pas_{min}(G)$ is the set of minimal passages.

The five passages in Fig. 2 are minimal. Note that each edge belongs to precisely one minimal passage. In fact, a minimal passage is uniquely identified by any of its elements as is shown next.

Lemma 3.9. Let $G = (N, E)$ be a graph and $(x, y) \in E$. There is precisely one minimal passage $P_{(x,y)} = (X, Y) \in pas_{min}(G)$ such that $x \in X$ and $y \in Y$.

Proof:
Construct $P_{(x,y)} = (X, Y)$ as follows. Initially: $X := \{x\}$ and $Y := \{y\}$. Then repeat $X := X \cup \bullet Y$ and $Y := Y \cup X\bullet$ until X and Y do not change anymore. The algorithm will end because there are finitely many nodes. When it ends, $X = \bullet Y$ and $Y = X\bullet$. Hence, $P_{(x,y)} = (X, Y)$ is passage. No unnecessary elements are added to X and Y, so (X, Y) is minimal and there is precisely one such minimal passage for $(x, y) \in E$.

To prove the latter one can also consider all passages $Q = \{P_1, P_2, \ldots, P_n\}$ that contain (x, y). The intersection of all such passages $\bigcap Q$ contains edge (x, y) and is again a passage because of Lemma 3.3. Hence, $\bigcap Q = P_{(x,y)}$. $\square$

For any $\{(x, y), (x', y), (x, y')\} \subseteq E$: $P_{(x,y)} = P_{(x',y)} = P_{(x,y')}$, i.e., $P_{(x,y)}$ is uniquely determined by x and $P_{(x,y)}$ is also uniquely determined by y. The set of all minimal passages $pas_{min}(G) = \{P_{(x,y)} \mid (x, y) \in E\}$ forms a passage partitioning.

Corollary 3.10. (Any Graph Has a Passage Partitioning)
Any graph $G = (N, E)$ has a passage partitioning, e.g., the set of minimal passages $pas_{min}(G)$.

Later we will use this corollary to partition process mining problems into smaller problems. To control the granularity of composition it is important that passages can be combined to form larger passages (cf. Lemma 3.3) or split into minimal passages as shown by the following corollary.

Corollary 3.11. (Passages are Composed of Minimal Passages)
Let $G = (N, E)$ be a graph. For any passage $P \in pas(G)$, there exists a set of minimal passages $\{P_1, P_2, \ldots, P_n\} \subseteq pas_{min}(G)$ such that $P = P_1 \cup P_2 \cup \ldots \cup P_n$.

Figure 3(b) illustrates the "fabric" of passages and the role of minimal passages. The figure shows nodes as black dots and a few example edges are shown (just a sketch). The smaller areas correspond to minimal passages. Passage $P_1 = (X_1, Y_1)$ is composed of 8 minimal passages, $P_2 = (X_2, Y_2)$ is composed of 7 minimal passages. $P_5 = P_1 \cap P_2$ is composed of 3 minimal passages shared by P_1 and P_2. $P_6 = P_1 \cup P_2$ is composed of 12 minimal passages. Any edge belongs to precisely one minimal passage. Any node on the left-hand side ($X_1 \cup X_2$) and any node on the right-hand side ($Y_1 \cup Y_2$) belongs to precisely one minimal passage. Although not shown, note that the same node may appear on both sides.

4. Conformance Checking

Conformance checking techniques investigate how well an event log $L \in \mathcal{B}(A^*)$ and a system net $SN = (PN, M_i, M_o)$ fit together. Note that the process model 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 and 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 process discovery results. In fact, genetic process mining algorithms use conformance checking to select the candidate models used to create the next generation of models [32].

There are four quality dimensions for comparing model and log: (1) *fitness*, (2) *simplicity*, (3) *precision*, and (4) *generalization* [2]. 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.

Definition 4.1. (Perfectly Fitting Log)
Let $L \in \mathcal{B}(A^*)$ be an event log and let $SN = (PN, M_i, M_o)$ be a system net. L perfectly fits SN if and only if $\{\sigma \in L\} \subseteq \tau(SN)$.

Note that $\{\sigma \in L\}$ converts multiset L into a set of traces. Consider two event logs $L_1 = [\langle a, e, g\rangle^{10}, \langle a, e, h\rangle^5, \langle a, e, f, e, g\rangle^3, \langle a, e, f, e, h\rangle^2]$ and $L_2 = [\langle a, e, g\rangle^{10}, \langle a, g\rangle^3, \langle a, a, g, e, h\rangle^2]$ and the system net SN of the Petri net depicted in Fig. 1 with $T_v = \{a, e, f, g, h\}$. Clearly, L_1 perfectly fits SN whereas L_2 is not. There are various ways to quantify fitness [2, 5, 9, 27, 32, 33, 34, 36]. To illustrate that conformance checking tasks can be decomposed using passages, we focus on *alignments* as the basis for conformance checking.

To measure fitness, one needs to *align* traces in the event log to traces of the process model. Some example alignments for L_2 and SN:

$$\gamma_1 = \begin{array}{|c|c|c|} a & e & g \\ \hline a & e & g \end{array} \quad \gamma_2 = \begin{array}{|c|c|c|} a & \gg & g \\ \hline a & e & g \end{array} \quad \gamma_3 = \begin{array}{|c|c|c|c|c|} a & a & g & e & h \\ \hline a & \gg & \gg & e & h \end{array} \quad \gamma_4 = \begin{array}{|c|c|c|c|c|c|} a & a & \gg & g & e & h \\ \hline a & \gg & e & g & \gg & \gg \end{array}$$

The top row of each alignment corresponds to "moves in the log" and the bottom row corresponds to "moves in the model". 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_3 the model is unable to do the second a move and is unable to do g before e. 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 γ_2 the log did not do an e move whereas the model has to make this move to enable g and reach the end. Given a trace in the event log, there may be many possible alignments. To select the most appropriate one we associate *costs* to moves and select an alignment with the lowest total costs.

A *move* is a pair (x, y) where the first element refers to the log and the second element refers to the model. For example, (a, a) means that *both* log and model make an "a move". $(\gg, a)$ means that the model makes an "a move" without a corresponding move of the log. $(a, \gg)$ means that the log makes an "a move" not followed by the model.

Definition 4.2. (Alignment)
Let $L \in \mathcal{B}(A^*)$ be an event log and let SN be a system net with visible traces $\tau(SN) \subseteq A^*$. $A_{LM} = \{(a, a) \mid a \in A\} \cup \{(\gg, a) \mid a \in A\} \cup \{(a, \gg) \mid a \in A\}$ is the set of *legal moves*.

Let $\sigma_L \in L$ be a log trace and $\sigma_M \in \tau(SN)$ a model trace. 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 second element (again ignoring $\gg$) yields σ_M.

γ_1-γ_4 are examples of alignments for traces in L_2 and the net depicted in Fig. 1 with $T_v = \{a, e, f, g, h\}$. Clearly, γ_3 is a better alignment than γ_4. This can be quantified using a cost function δ.

Definition 4.3. (Cost of Alignment)
Cost function $\delta \in A_{LM} \to \mathbb{N}$ assigns costs to legal moves. The *cost* of an alignment γ is the sum of all costs: $\delta(\gamma) = \sum_{(x,y)\in\gamma} \delta(x, y)$ for $\gamma \in {A_{LM}}^*$.

Moves where log and model agree have no costs, i.e., $\delta(a, a) = 0$ for all $a \in A$. $\delta(\gg, a) > 0$ is the cost when the model makes an "a move" without a corresponding move of the log. $\delta(a, \gg) > 0$ is the cost for an "a 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: $\delta_S(a, a) = 0$ and $\delta_S(\gg, a) = \delta_S(a, \gg) = 1$ for all $a \in A$. For example, $\delta_S(\gamma_1) = 0$, $\delta_S(\gamma_2) = 1$, $\delta_S(\gamma_3) = 2$, and $\delta_S(\gamma_4) = 4$ (simply count the number of $\gg$ symbols).

Definition 4.4. (Optimal Alignment)
Let $L \in \mathcal{B}(A^*)$ be an event log and SN be a system net with $\tau(SN) \neq \emptyset$.

- For $\sigma_L \in L$, we define: $\Gamma_{\sigma_L,SN} = \{\gamma \in {A_{LM}}^* \mid \exists_{\sigma_M\in\tau(SN)}\ \gamma \textit{ is an aligment of } \sigma_L \textit{ and } \sigma_M\}$.
- An alignment $\gamma \in \Gamma_{\sigma_L,SN}$ is *optimal* for trace $\sigma_L \in L$ and system net SN if for any $\gamma' \in \Gamma_{\sigma_L,M}$: $\delta(\gamma') \geq \delta(\gamma)$.
- $\lambda_{SN} \in A^* \to {A_{LM}}^*$ 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.

γ_4 is not an optimal alignment for trace $\langle a, a, g, e, h\rangle$ and the net in Fig. 1 with $T_v = \{a, e, f, g, h\}$. γ_1, γ_2, and γ_3 are optimal alignments. Hence, $costs(L_2, SN, \delta_S) = 10\times\delta_S(\gamma_1)+3\times\delta_S(\gamma_2)+2\times\delta_S(\gamma_3) = 10 \times 0 + 3 \times 1 + 2 \times 2 = 7$.

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, 9] for details. Misalignment costs can be related to Definition 4.1, because only perfectly fitting traces have costs 0 (assuming $\tau(SN) \neq \emptyset$).

Lemma 4.5. (Perfectly Fitting Log)
Event log L perfectly fits 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 [33, 34]. Recent results show that such computations should be based on alignments [11]. The same holds for generalization [5]. Therefore, we focus on alignments when decomposing passages.

5. Distributed Conformance Checking

Conformance checking techniques 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 [9] or repeated state-space explorations [36]. When using genetic process mining, one needs to check the fitness of every individual model in every generation [32]. 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. Moreover, passages can be used to provide *local diagnostics* (per passage).

In this section, we show that it is possible to decompose and distribute conformance checking problems using the notion of *passages* defined in Section 3. In order to do this we focus on the visible transitions and create the so-called *skeleton* of the process model.

Definition 5.1. (Skeleton)
Let $PN = (P, T, F, T_v)$ be a labeled Petri net. The *skeleton* of PN is the graph $\mathit{skel}(PN) = (N, E)$ with $N = T_v$ and $E = \{(x, y) \in T_v \times T_v \mid x \overset{F\#T_v}{\rightsquigarrow} y\}$.

Figure 5 shows the skeleton of the net in Fig. 1 assuming that $T_v = \{a, e, f, g, h\}$. The resulting graph has two minimal passages.

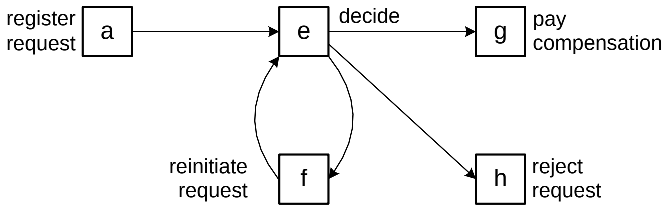

Figure 5. The skeleton of the labeled Petri net in Fig. 1 (assuming that $T_v = \{a, e, f, g, h\}$). There are two minimal passages: $(\{a, f\}, \{e\})$ and $(\{e\}, \{f, g, h\})$.

Note that only the visible transitions T_v appear in the skeleton. For example, if we assume that $T_v = \{a, g, h\}$ in Fig. 1, then the skeleton is $(\{a, g, h\}, \{(a, g), (a, h)\})$ and there is only one passage $(\{a\}, \{g, h\})$.

If there are multiple (minimal) passages in the skeleton, then we can decompose conformance checking problems into smaller problems *by partitioning the Petri net into net fragments and the event log into sublogs*. We will first show that each passage (X, Y) defines one *net fragment* $PN^{(X,Y)}$ (cf. Definition 5.2) and one *sublog* $L\restriction_{X \cup Y}$. Then we will prove that conformance can be checked per passage.

Consider event log $L = [\langle a, e, g\rangle^{10}, \langle a, e, h\rangle^{5}, \langle a, e, f, e, g\rangle^{3}, \langle a, e, f, e, h\rangle^{2}]$, the Petri net PN shown in Fig. 1 with $T_v = \{a, e, f, g, h\}$, and the skeleton shown in Fig. 5. There are two passages: $P_1 = (\{a, f\}, \{e\})$ and $P_2 = (\{e\}, \{f, g, h\})$. Based on this we define two net fragments PN_1 and PN_2 as shown in Fig. 6. Moreover, we define two sublogs: $L_1 = [\langle a, e\rangle^{15}, \langle a, e, f, e\rangle^{5}]$ and $L_2 = [\langle e, g\rangle^{10}, \langle e, h\rangle^{5}, \langle e, f, e, g\rangle^{3}, \langle e, f, e, h\rangle^{2}]$. To check the conformance of the overall event log on the overall model, we check the conformance of L_1 on PN_1 and L_2 on PN_2. Since L_1 perfectly fits PN_1 and L_2 perfectly fits PN_2, we can conclude that L perfectly fits PN.[1] This illustrates that conformance checking can be decomposed.

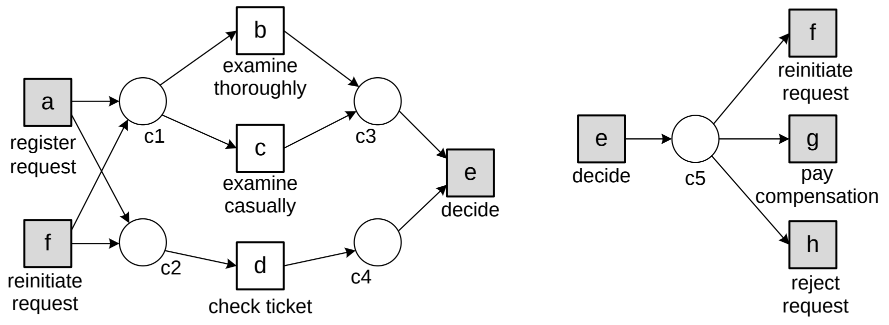

Figure 6. Two net fragments corresponding to the two passages of the skeleton in Fig. 5: $PN_1 = PN^{(\{a,f\},\{e\})}$ (left) and $PN_2 = PN^{(\{e\},\{f,g,h\})}$ (right). The visible transitions $T_v = \{a, e, f, g, h\}$ that form the boundaries of the fragments are highlighted.

In order to prove this, we first define the notion of a net fragment.

Definition 5.2. (Net Fragment)
Let $PN = (P, T, F, T_v)$ be a labeled Petri net. For any two sets of transitions $X, Y \subseteq T_v$, we define the net fragment $PN^{(X,Y)} = (P', T', F', T'_v)$ with:

- $Z = nodes(X \overset{F\#T_v}{\rightsquigarrow} Y) \setminus (X \cup Y)$ are the internal nodes of the fragment,
- $P' = P \cap Z$,
- $T' = (T \cap Z) \cup X \cup Y$,
- $F' = F \cap ((P' \times T') \cup (T' \times P'))$, and
- $T'_v = X \cup Y$.

Note that $PN_1 = PN^{(\{a,f\},\{e\})}$ in Fig. 6 has $Z = \{b, c, d, c1, c2, c3, c4\}$ as internal nodes.

Now we can prove the main result of this paper. Figure 7 illustrates our decomposition approach. A larger model can be decomposed into net fragments corresponding to passages. The event log can be decomposed in a similar manner and conformance checking can be done per passage.

The fragments corresponding to the passages are initially empty whereas the overall Petri net starts in a particular initial marking and ends in a particular final marking. Therefore, we extend the Petri net and event log to incorporate initialization and termination (cf. the dashed $\top$ and $\bot$ transitions in Figure 7).

Definition 5.3. (Extended System Net and Event Log)
Let $L \in \mathcal{B}(A^*)$ be an event log and $SN = (PN, M_i, M_o)$ be a system net with $PN = (P, T, F, T_v)$. Assume two fresh identifiers $\top$ and $\bot$ to represent an artificial start ($\top$) and an artificial complete ($\bot$).

- $\overline{L} = [\langle \top \rangle \cdot \sigma \cdot \langle \bot \rangle \mid \sigma \in L]$ is the event log extended with explicit start and complete events.
- $\overline{PN} = (P, T \cup \{\top, \bot\}, F \cup \{(\top, p) \mid p \in M_i\} \cup \{(p, \bot) \mid p \in M_o\}, T_v \cup \{\top, \bot\})$ is the Petri net extended with with start and complete transitions.
- $\overline{SN} = (\overline{PN}, [\,], [\,])$ is the *extended system net* having empty initial and final markings.

[1]Here we abstract from initialization and termination. These will be addressed by adding artificial start and complete transitions/events (cf. Definition 5.3).

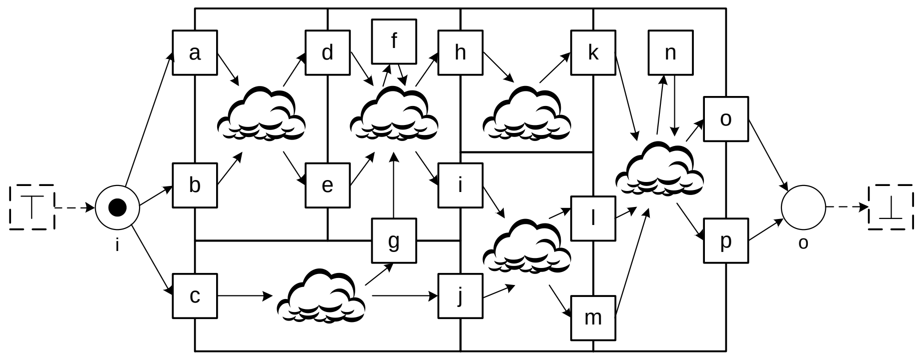

Figure 7. Petri net PN is decomposed into subnets $PN^{(X,Y)}$. The "clouds" model the internal structure of these subnets (places but possibly also hidden transitions). Due to the decomposition based on passages, one cloud can only influence another cloud through the visible interface transitions X and Y. Since the visible interface transitions are "controlled" by the event log, it is possible to check fitness locally per subnet.

The initial system net will often be a WF-net [1] as shown in Figure 7, i.e., there is one source place i and one sink place o with all nodes on a path from i to o and $M_i = [i]$ and $M_o = [o]$. However, the results presented apply to any connected net and not just WF-nets. Note that Definition 5.3 assumes that M_i and M_o are safe, i.e., these two markings have at most one token per place. However, the construction can easily be generalized by introducing arc weights. Figure 8(a-b) illustrates the net extension described in Definition 5.3.

To be able to project traces onto the visible nodes of a fragment, we define the following shorthand for a passage $P = (X, Y)$: $L\upharpoonright_P = L\upharpoonright_{X \cup Y}$, i.e., only the events corresponding to input or output nodes of P are retained in the resulting log.

Theorem 5.4. (Conformance Checking Can be Decomposed)
Let $L \in \mathcal{B}(A^*)$ be an event log and let $SN = (PN, M_i, M_o)$ be a connected system net. For any passage partitioning $\{P_1, P_2, \ldots, P_n\}$ of $skel(\overline{PN})$: L perfectly fits system net SN if and only if for all $1 \leq i \leq n$: $\overline{L}\upharpoonright_{P_i}$ perfectly fits $SN^i = (\overline{PN}^{P_i}, [\,], [\,])$.

Proof:
Note that SN is connected. This implies that in $\overline{PN}$ all nodes are on a path from $\top$ to $\bot$ and that all nodes and edges in $\overline{PN}$ (i.e., also the nodes in PN) are included in at least one net fragment $\overline{PN}^{P_i}$, i.e., $\bigcup_i \overline{PN}^{P_i} = \overline{PN}$.

Second, we argue that L perfectly fits system net SN if and only if $\overline{L}$ perfectly fits system net $\overline{SN}$. This can be learned from the observation that for any σ: $M_i[\sigma\rangle M_o$ in PN if and only if $[\,]\ [(\langle\top\rangle \cdot \sigma \cdot \langle\bot\rangle)\rangle\ [\,]$ in $\overline{PN}$. Hence, for $\sigma_v \in L$: $M_i[\sigma_v \triangleright M_o$ in PN if and only if $[\,]\ [(\langle\top\rangle \cdot \sigma_v \cdot \langle\bot\rangle)\triangleright\ [\,]$ in $\overline{PN}$.

Hence it suffices to prove that: $\overline{L}$ perfectly fits the extended system net $\overline{SN} = (\overline{PN}, [\,], [\,])$ if and only if for all i: $\overline{L}\upharpoonright_{P_i}$ perfectly fits $SN^i = (\overline{PN}^{P_i}, [\,], [\,])$. In the remainder we use the following notations: $\overline{PN} = (P, T, F, T_v)$ (note that $\{\top, \bot\} \subseteq T_v$ because $\overline{PN}$ is the extended Petri net) and $\overline{PN}^{P_i} = (P^i, T^i, F^i, T_v^i)$ (the net fragment corresponding to passage P_i, constructed using Definition 5.2).

($\Rightarrow$) Let $\sigma_v \in \overline{L}$ such that there is a $\sigma \in T^*$ with $[\,][\sigma\rangle[\,]$ in $\overline{PN}$ and $\sigma\upharpoonright_{T_v} = \sigma_v$ (i.e., σ_v fits into the

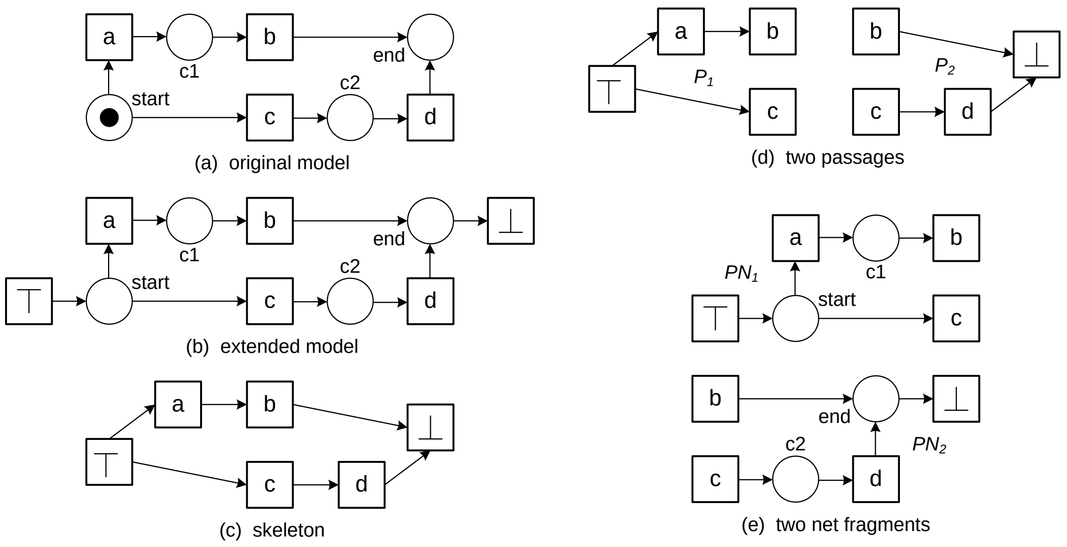

Figure 8. The system net $SN = (PN, [start], [end])$ shown in (a) is extended into system net $\overline{SN} = (\overline{PN}, [\,], [\,])$ by adding artificial start ($\top$) and complete ($\bot$) transitions as shown in (b). The skeleton $skel(\overline{PN})$ is shown in (c). A passage partitioning composed of $P_1 = (\{\top, a\}, \{a, b, c\})$ and $P_2 = (\{b, c, d\}, \{d, \bot\})$ is shown in (d) and the corresponding two net fragments are shown in (e).

overall extended system net $\overline{SN}$). For all $1 \leq i \leq n$, we need to prove that there is a σ_i with $[\,][\sigma_i\rangle[\,]$ in $\overline{PN}^{P_i}$ such that $\sigma_i\!\upharpoonright_{P_i} = \sigma_v\!\upharpoonright_{P_i}$. This follows trivially because SN^i can mimic any move of $\overline{SN}$ with respect to transitions T^i: just take $\sigma_i = \sigma\!\upharpoonright_{T^i}$.

($\Leftarrow$) Let $\sigma_v \in \overline{L}$ such that for each $1 \leq i \leq n$ there is a σ_i such that $[\,][\sigma_i\rangle[\,]$ in $\overline{PN}^{P_i}$ and $\sigma_i\!\upharpoonright_{P_i} = \sigma_v\!\upharpoonright_{P_i}$. We need to prove that there is a $\sigma \in T^*$ such that $[\,][\sigma\rangle[\,]$ in $\overline{PN}$ and $\sigma\!\upharpoonright_{T_v} = \sigma_v$. The different σ_i sequences can be stitched together into an overall σ because the different subnets only interface via visible transitions and $\bigcup_i \overline{PN}^{P_i} = \overline{PN}$. Take σ_v and extend it by adding the local events. Transitions in one subnet can only influence other subnets through visible transitions and these can only move synchronously as defined by $\sigma_v \in \overline{L}$. $\square$

Theorem 5.4 shows that any trace in the log fits the overall model if and only if it fits each of the passage-based fragments. Moreover, as shown next, an upper bound for the degree of fitness can be computed in a distributed manner. For this we introduce an adapted cost function δ_Q.

Definition 5.5. (Adapted Cost Function)
Let $Q = \{P_1, P_2, \ldots, P_n\}$ be a passage partitioning and $\delta \in A_{LM} \to \mathbb{N}$ a cost function (cf. Definition 4.3). $c_Q(x, y) = |\{1 \leq j \leq n \mid \{x, y\} \cap (X_j \cup Y_j) \neq \emptyset\}|$ counts the number of passages where x or y is an input or output node. The adapted cost function δ_Q is defined as follows: $\delta_Q(\top, \top) = 0$, $\delta_Q(\top, x) = \delta_Q(x, \top) = \infty$ if $x \neq \top$, $\delta_Q(\bot, \bot) = 0$, $\delta_Q(\bot, x) = \delta_Q(x, \bot) = \infty$ if $x \neq \bot$, $\delta_Q(x, y) = \frac{\delta(x,y)}{c_Q(x,y)}$ for $(x, y) \in A_{LM}$ and $c_Q(x, y) \neq 0$.

There should never be a move on log only or a move on model only involving $\top$ or $\bot$. This can be avoided by associating extremely high costs (denoted as ∞) to moves other than $(\top,\top)$ and $(\bot,\bot)$. A visible transition may appear in multiple passages. Therefore, we divide its costs by the number of passages 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.

Theorem 5.6. (Lower Bound for Misalignment Costs)
Let $L \in \mathcal{B}(A^*)$ be an event log and let $SN = (PN, M_i, M_o)$ be a connected system net. For any passage partitioning $Q = \{P_1, P_2, \ldots, P_n\}$ of $\mathit{skel}(\overline{PN})$:

$$\mathit{costs}(L, SN, \delta) \geq \sum_{1 \leq i \leq n} \mathit{costs}(\overline{L}{\upharpoonright_{P_i}}, SN^i, \delta_Q)$$

where $SN^i = (\overline{PN}^{P_i}, [\,], [\,])$.

Proof:
We can assume that the only moves involving the artificially added nodes are $(\top,\top)$ and $(\bot,\bot)$. Hence, no costs are added by extending the event log with $\top$ at the beginning and $\bot$ at end of each trace. For any $\sigma_v \in \overline{L}$ there is an optimal alignment γ of σ_v and $\overline{SN}$ such that the projection on the second element yields a trace σ_v' with $[\,][\sigma_v' \rhd [\,]$ in $\overline{PN}$, i.e., there is a trace σ with $[\,][\sigma\rangle[\,]$ in $\overline{PN}$ and $\sigma{\upharpoonright_{T_v}} = \sigma_v'$. As shown in the proof of Theorem 5.4 there is a σ_i with $[\,][\sigma_i\rangle[\,]$ in $\overline{PN}^{P_i}$ and $\sigma_i{\upharpoonright_{P_i}} = \sigma_v'{\upharpoonright_{P_i}}$ 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 $\sigma_v{\upharpoonright_{P_i}}$ and SN^i. The sum of the costs associated with these local alignments γ_i is exactly the same as the cost of the overall alignment γ. However, there may be local improvements lowering the sum of the costs associated with these local alignments. Hence, $\mathit{costs}(L, SN, \delta) \geq \sum_{1 \leq i \leq n} \mathit{costs}(\overline{L}{\upharpoonright_{P_i}}, SN^i, \delta_Q)$. □

Consider system net SN in Figure 8(a), passages $P_1 = (\{\top, a\}, \{a, b, c\})$ and $P_2 = (\{b, c, d\}, \{d, \bot\})$, and event logs $L_1 = [\langle a, b\rangle^{10}, \langle c, d\rangle^5]$, $L_2 = [\langle a, a, b\rangle]$, and $L_3 = [\langle a, b, c, d\rangle]$. The corresponding extended system net $\overline{SN}$ is shown in Figure 8(b) and the corresponding extended event logs are $\overline{L}_1 = [\langle \top, a, b, \bot\rangle^{10}, \langle \top, c, d, \bot\rangle^5]$, $\overline{L}_2 = [\langle \top, a, a, b, \bot\rangle]$, and $\overline{L}_3 = [\langle \top, a, b, c, d, \bot\rangle]$. $\mathit{costs}(L_1, SN, \delta_S) = \mathit{costs}(\overline{L}_1{\upharpoonright_{P_1}}, SN^1, \delta_Q) + \mathit{costs}(\overline{L}_1{\upharpoonright_{P_2}}, SN^2, \delta_Q) = 0$ and $\mathit{costs}(L_2, SN, \delta_S) = \mathit{costs}(\overline{L}_2{\upharpoonright_{P_1}}, SN^1, \delta_Q) + \mathit{costs}(\overline{L}_2{\upharpoonright_{P_2}}, SN^2, \delta_Q) = 1 + 0 = 1$, i.e., the costs of the optimal overall alignments are equal to the sums of the costs associated to all optimal local alignments. Consider for example the following overall optimal alignment for $\langle a, a, b\rangle$ and optimal local alignments for $\langle \top, a, a, b, \bot\rangle$:

$$\gamma = \begin{array}{||c|c|c|} a & a & b \\ \hline a & \gg & b \end{array} \qquad \gamma_1 = \begin{array}{|c|c|c|c|} \top & a & a & b \\ \hline \top & a & \gg & b \end{array} \qquad \gamma_2 = \begin{array}{|c|c|} b & \bot \\ \hline b & \bot \end{array}$$

The costs of the overall optimal alignment γ equals the costs of the two optimal local alignments γ_1 and γ_2. This does not hold for event log L_3: $\mathit{costs}(L_3, SN, \delta_S) = 2$, $\mathit{costs}(\overline{L}_3{\upharpoonright_{P_1}}, SN^1, \delta_Q) = 0.5$, and $\mathit{costs}(\overline{L}_3{\upharpoonright_{P_2}}, SN^2, \delta_Q) = 0.5$. Hence, the total costs are higher than the costs associated to the two optimal local alignments. To understand this, consider the following optimal alignments for $\langle a, b, c, d\rangle$ and $\langle \top, a, b, c, d, \bot\rangle$:

$$\gamma = \begin{array}{|c|c|c|c|} a & b & c & d \\ \hline a & b & \gg & \gg \end{array} \qquad \gamma' = \begin{array}{|c|c|c|c|} a & b & c & d \\ \hline \gg & \gg & c & d \end{array} \qquad \gamma_1 = \begin{array}{|c|c|c|c|} \top & a & b & c \\ \hline \top & a & b & \gg \end{array} \qquad \gamma_2 = \begin{array}{|c|c|c|c|} b & c & d & \bot \\ \hline \gg & c & d & \bot \end{array}$$

The cost of any of the two overall optimal alignments is $\delta_S(\gamma) = \delta_S(\gamma') = 2$. The cost of the optimal alignment γ_1 for passage P_1 is $\delta_Q(\gamma_1) = 0 + 0 + 0 + \delta_Q(c, \gg) = \frac{\delta_S(c,\gg)}{c_Q(c,\gg)} = \frac{1}{2} = 0.5$. The cost of the optimal alignment γ_2 for passage P_2 is $\delta_Q(\gamma_2) = \delta_Q(b, \gg) + 0 + 0 + 0 = \frac{\delta_S(b,\gg)}{c_Q(b,\gg)} = \frac{1}{2} = 0.5$. Hence, $\mathit{costs}(L_3, SN, \delta_S) = 2 > \mathit{costs}(\overline{L}_3 \upharpoonright_{P_1}, SN^1, \delta_Q) + \mathit{costs}(\overline{L}_3 \upharpoonright_{P_2}, SN^2, \delta_Q) = 1$. This shows that there may indeed be local improvements lowering the sum of the costs associated with local alignments.

Theorem 5.6 shows that 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 for 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 passages provide valuable local diagnostics. *The passages 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 5.6 shows just one of many possible definitions of fitness. We can also simply count the *fraction of fitting traces*. In this case the problem can be decomposed easily using the notation of passages.

Theorem 5.7. (Fraction of Perfectly Fitting Traces)
Let $L \in \mathcal{B}(A^*)$ be an event log and let $SN = (PN, M_i, M_o)$ be a connected system net. For any passage partitioning $Q = \{P_1, P_2, \ldots, P_n\}$ of $\mathit{skel}(\overline{PN})$:

$$\frac{|[\sigma \in L \mid \sigma \in \tau(SN)]|}{|L|} = \frac{|[\sigma \in \overline{L} \mid \forall_{1 \leq i \leq n}\, \sigma \upharpoonright_{P_i} \in \tau(SN^i)]|}{|\overline{L}|}$$

Proof:
Follows from the construction used in Theorem 5.4. A trace is fitting the overall model if and only if it fits all passages. □

As Theorem 5.7 suggests, traces in the event log can be marked as fitting or non-fitting *per passage*. These results can be merged easily and used to compute the *fraction of traces fitting the overall model*.

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) [33, 34]. This can be done at the level of passages even though there is not a straightforward manner to compute the overall precision level. Note that relatively many escaping edges in a passage suggest "underfitting" of that part of model. As shown in [11], alignments should be the basis for precision analysis. Therefore, the construction used in Theorems 5.4 and 5.6 can be used as a starting point. A similar approach can be used for generalization: many unique paths in a passage 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 5.6) relates observed events to occurrences of transitions of some passage P_i. 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 [2]. 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), these local alignments can be used for *decision mining* [35]. For any decision point in a passage (place with multiple output arcs), 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.

Assuming a process model with many passages, *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 [36] the shortest path enabling a transition is computed) or optimization over all possible alignments (e.g., in [9] 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 passage are done on a single computer. Moreover, passages are not just interesting from a performance point of view: they can also be used to pinpoint the most problematic parts of the process (also in terms of performance) and provide localized diagnostics.

6. Process Discovery: Divide and Conquer

In the previous section, we showed that we can decompose conformance checking tasks using passages. Instead of checking the conformance of the entire event log on the entire system net, we split up the log and the net into pairs of sublogs and net fragments, and check conformance on each of these pairs. Provided that we have a collection of passages, we can do something similar for discovery: Instead of discovering the whole system net in one go, we first split up the log into sublogs, then discover a net fragment for every sublog, and finally fold all net fragments into one overall system net.

Our approach builds on existing process discovery algorithms. There exist dozens of algorithms – ranging from the simple α miner [8] to the more sophisticated ILP miner [42] – that discover a Petri net from an event log. For passage-based discovery, we use such an algorithm, discover a fragment per passage, and apply Definition 5.2 in reverse direction. Γ_p denotes the *class* of algorithms that can be used to discover a Petri net $PN^{(X,Y)}$ for a passage (X, Y). In order to decompose a discovery problem using passages, we first need to derive a graph with causal dependencies from the event log. Γ_c denotes the *class* of algorithms that can be used to discover these dependencies.

Definition 6.1. (Γ_c algorithm)
Let $L \in \mathcal{B}(A^*)$ be an event log over A. A Γ_c algorithm is an algorithm that takes the event log L and returns a causal structure (i.e. a graph) with nodes $V \subseteq A$, that is, if γ_c is a Γ_c algorithm, then $\gamma_c(L) = (V, E)$ is a graph with $V \subseteq A$.

Note that many process discovery algorithms have an initial phase deriving these dependencies by scanning the event log, e.g., the $>$ ("directly follows"), $\rightarrow$ ("causality"), $\|$ ("concurrency"), and $\#$ ("choice") relationships inferred by the α miner [8]. The Heuristics miner [41, 39] derives similar relations while taking noise and incompleteness into account. These algorithms can easily be distributed as they simply count basic patterns in the event log. Basically, a Γ_c algorithm takes an event log as input, and returns a causal structure as output. Optionally, the discovered causal structure can be edited before computing the passages from it. After determining the passages, we discover a net fragment per passage using a Γ_p algorithm and merge the results.

Definition 6.2. (Γ_p algorithm)
Let $L \in \mathcal{B}(A^*)$ be an event log over A and let $X, Y \subseteq A$. A Γ_p algorithm is an algorithm that takes a passage (X, Y) and the corresponding sublog $L\!\upharpoonright_{X \cup Y}$ and returns a net fragment with visible transitions $X \cup Y$, that is, if γ_p is a Γ_p algorithm, then $\gamma_p(L\!\upharpoonright_{X \cup Y}, X, Y) = (P, T, F, X \cup Y)$ is a labeled Petri net.

Note that $\gamma_c \in \Gamma_c$ returns a causal structure that may include some (but not necessarily all) activities ($V \subseteq A$). This way, the algorithm can effectively remove, for example, infrequent activities that might only complicate the discovery process. The resulting causal structure can be inspected and modified by a domain expert. This domain expert can remove any causalities that she knows do not exist, and can add any causalities that she knows are missing. After the domain expert has thus massaged the causal structure, a passage partitioning is derived from it and used to decompose the whole event log into a collection of sublogs.

The Γ_c algorithm operates on the whole event log, whereas we are trying to speed-up discovery by splitting up the entire log into a collection of sublogs. However, we can use Γ_c algorithms that are very fast compared to more sophisticated process mining algorithms, e.g., algorithms based on state-based regions [7, 21, 37], language-based regions [15, 42], or genetic evolution [32] are much more time consuming. A typical example is the ILP miner [42], which creates an integer-linear programming problem that is exponential in the size of A, that is, in the number of activities. By using a fast Γ_c algorithm, passages can be used to quickly split up A into smaller sets; as a result the ILP miner can work much faster. Furthermore, the Γ_c algorithm need not take the entire log into account. Its purpose is to construct a causal structure, which is more abstract and high-level, whereas the Γ_p algorithm needs to fill in the nitty-gritty low-level details later. Therefore, it may suffice to use only a sample set of traces.

After having introduced the Γ_p and Γ_c classes of algorithms, we now introduce our passage-based discovery approach.

Definition 6.3. (Passage-based Discovery)
Let $L \in \mathcal{B}(A^*)$ be an event log over a set of activities A and let $\gamma_p \in \Gamma_p$ and $\gamma_c \in \Gamma_c$ be the two selected algorithms. Our passage-based discovery approach proceeds as follows:

1. Extend each trace in the event log with an artificial start event $\top$ and an artificial end event $\bot$ ($\{\top, \bot\} \cap A = \emptyset$). $\overline{L} = [\langle \top \rangle \cdot \sigma \cdot \langle \bot \rangle \mid \sigma \in L]$ is the resulting log over $\overline{A} = \{\top, \bot\} \cup A$.
2. Discover the causal structure using γ_c. $(A_{\gamma_c}, C_{\gamma_c}) = \gamma_c(\overline{L})$ is the resulting causal structure with $\{\top, \bot\} \subseteq A_{\gamma_c} \subseteq \overline{A}$ and $C_{\gamma_c} \subseteq A_{\gamma_c} \times A_{\gamma_c}$.
3. Optional: Have a domain expert inspect (and massage if needed) the causal structure $(A_{\gamma_c}, C_{\gamma_c})$. $(A'_{\gamma_c}, C'_{\gamma_c})$ is the resulting, possibly modified, causal structure.
4. Compute the set of minimal passages on the causal structure $(A'_{\gamma_c}, C'_{\gamma_c})$. $\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_k, Y_k)\} = \mathit{pas}_{\mathit{min}}(A'_{\gamma_c}, C'_{\gamma_c})$ is the resulting set of passages.
5. For every minimal passage (X_i, Y_i): Discover a net fragment using γ_p. $PN_i = (P_i, T_i, F_i, X_i \cup Y_i) = \gamma_p(\overline{L}\!\upharpoonright_{X_i \cup Y_i}, X_i, Y_i)$ is the resulting net fragment for passage i.
6. Merge the individual net fragments PN_i into one overall system net. $SN = (PN, M_i, M_o)$ with $PN = (P, T, F, T_v)$ is the resulting system net, where:
   - $P = \{\mathit{in}, \mathit{out}\} \cup \bigcup_{1 \le i \le k} P_i$,
   - $T = \cup_{1 \le i \le k} T_i$,

- $F = \{(in, \top), (\bot, out)\} \cup (\bigcup_{1 \leq i \leq k} F_i)$,
- $T_v = \bigcup_{1 \leq i \leq k} X_i \cup Y_i$,
- $M_i = [in]$, and
- $M_o = [out]$.

The log is extended by adding an artificial start event $\top$ and an artificial end event $\bot$ to every trace. This is just a technicality to ensure that there is a clearly defined start and end. Note that passages can be activated multiple times, e.g., in case of loops. Therefore, we add transitions $\top$ and $\bot$ and places in and out. If there is a unique start (end) event, then there is no need to add transition $\top$ ($\bot$). Ideally, the causal structure $(A'_{\gamma_c}, C'_{\gamma_c})$ has one source node $\top$, one sink node $\bot$, and all other nodes are on a path from $\top$ to $\bot$ (like in a WF-net [1]).

Note that in Step 4 minimal passages are computed since we aim to decompose the discovery problem in as many small independent problems as possible. However, in principle any passage partitioning may be used as illustrated by Theorems 5.4, 5.6, and 5.7.

To illustrate our divide-and-conquer approach, consider the event log $L = [\langle a, b, c, d\rangle^{40}, \langle b, a, c, d\rangle^{35}, \langle a, b, c, e\rangle^{30}, \langle b, a, c, e\rangle^{25}, \langle a, b, x, d\rangle^{1}, \langle a, b, e\rangle^{1}]$. The log describes 132 cases. We first add artificial events as described in Step 1: $\overline{L} = [\langle \top, a, b, c, d, \bot\rangle^{40}, \langle \top, b, a, c, d, \bot\rangle^{35}, \langle \top, a, b, c, e, \bot\rangle^{30}, \langle \top, b, a, c, e, \bot\rangle^{25}, \langle \top, a, b, x, d, \bot\rangle^{1}, \langle \top, a, b, e, \bot\rangle^{1}]$. Then we compute the causal structure using the $\gamma_c \in \Gamma_c$ algorithm of choice (Step 2). Assume that the causal structure shown in Fig. 9 is computed. Since x occurs only once whereas the other activities occur more than 50 times, x is excluded. The same holds for the dependency between b and e. Furthermore, we assume that the domain expert does not change the causal structure (Step 3).

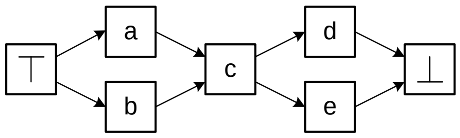

Figure 9. Causal structure $\gamma_c(\overline{L})$ discovered for the extended event log having four minimal passages.

The causal structure has four minimal passages (Step 4): $P_1 = (\{\top\}, \{a, b\})$, $P_2 = (\{a, b\}, \{c\})$, $P_3 = (\{c\}, \{d, e\})$, and $P_4 = (\{d, e\}, \{\bot\})$. Based on these passages, we create four corresponding sublogs: $L_1 = [\langle \top, a, b\rangle^{72}, \langle \top, b, a\rangle^{60}]$, $L_2 = [\langle a, b, c\rangle^{70}, \langle b, a, c\rangle^{60}, \langle a, b\rangle^{2}]$, $L_3 = [\langle c, d\rangle^{75}, \langle c, e\rangle^{55}, \langle d\rangle^{1}, \langle e\rangle^{1}]$, and $L_4 = [\langle d, \bot\rangle^{76}, \langle e, \bot\rangle^{56}]$. One transition-bordered Petri net is discovered per sublog using the $\gamma_p \in \Gamma_p$ algorithm of choice (Step 5). Figure 10 shows the net fragments discovered per passage. Note that infrequent behavior has been discarded by γ_p, i.e., trace $\langle a, b\rangle$ in L_2 is not possible according to PN_2 (does not end is desired end state), and traces $\langle d\rangle$ and $\langle e\rangle$ in L_3 are not possible according to PN_3.

In the last step of the approach, the four net fragments of Fig. 10 are merged into the overall net system shown in Figure 11 (Step 6). Note that this net system is indeed able to replay all frequent behavior. Two of the 132 cases cannot be replayed because they were treated as noise by the selected γ_c and γ_p algorithms.

Although the resulting model in Figure 11 is simple and has no loops, there are no limitations with respect to the control-flow patterns used and loops can be handled without any problems. The small example shows that we can use a divide-and-conquer approach when discovering process models. We

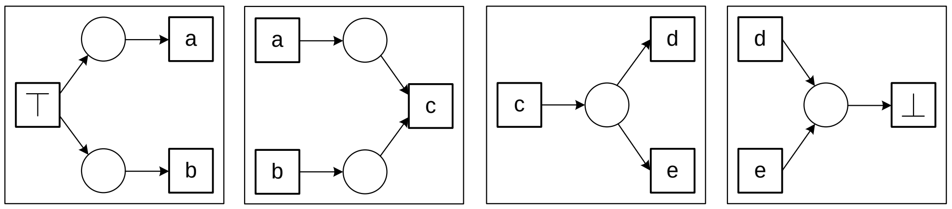

Figure 10. The Petri net fragments discovered for the four passages: PN_1, PN_2, PN_3, and PN_4.

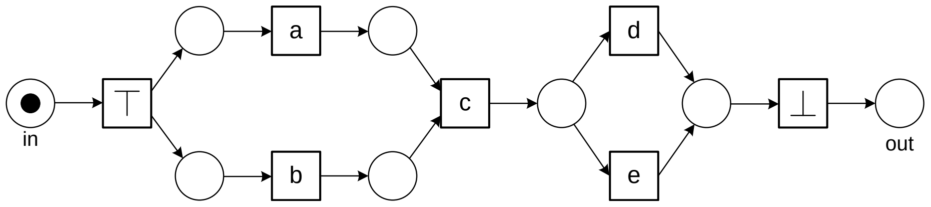

Figure 11. The Petri net obtained by merging the individual subsets.

deliberately did not restrict ourselves to specific Γ_c and Γ_p algorithms. The approach is generic and can be combined with existing process discovery techniques [2, 7, 8, 13, 15, 20, 21, 23, 27, 32, 37, 40, 42]. Moreover, the user can modify the causal structure to guide the discovery process.

By decomposing the overall discovery problem into a collection of smaller discovery problems, it is possible to do a more refined analysis and achieve significant speed-ups. The γ_c algorithm only needs to construct an abstract causal structure. Hence, it may take only a sample (say, 100 randomly chosen traces) of the event log into account. The γ_p algorithm is applied for every net passage, and needs to construct a detailed net fragment. Hence, it needs only to consider an event log consisting of just the activities involved in the corresponding passage. As a result, process discovery tasks can be distributed over a network of computers (assuming there are multiple passages). As most discovery algorithms are exponential in the number of activities, *the sequential discovery of all individual passages on one computer is often still faster than solving one big discovery problem.* If there are more minimal passages than computers, one can merge minimal passages into aggregate passages and use these for discovery and conformance checking (one passage per computer). However, in most situations, it will be more efficient to analyze the minimal passages sequentially.

7. Empirical Evaluation

In earlier sections we showed that process mining problems can be divided into smaller problems and that by doing this, in theory, significant speed-ups are possible. Since our passage-based decomposition approach is very general, it is not easy to evaluate this empirically. For example, we can choose from many different process discovery algorithms. Moreover, there are various algorithms that cannot benefit from a passage-based decomposition approach because they are linear in the size of the event log. For example, the α miner [8] and the heuristics miner [41] make one pass through the whole log while counting simple metrics like direct successions. Obvious such techniques will not benefit from passage-based decomposition. However, these algorithms have various limitations: they create models without

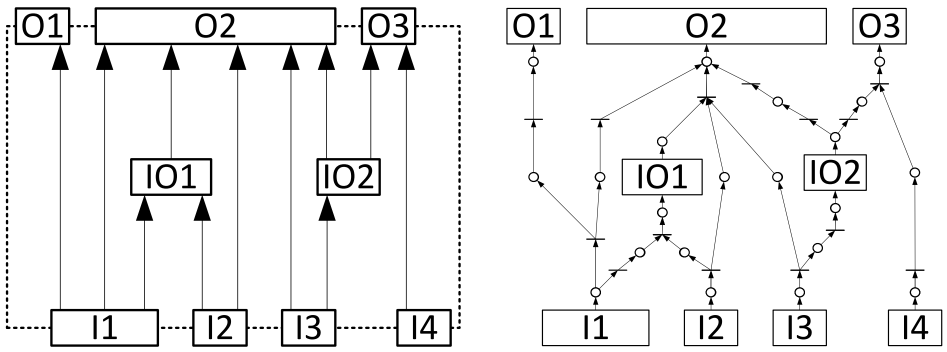

Figure 12. An example passage and a corresponding Petri net.

any guarantees, e.g., the resulting models may have deadlocks, have a poor fitness, and be very complex and under- or over-fitting. Only more expensive algorithms that replay the log and solve optimization problems can provide such guarantees.

Therefore, we use a particular setting to provide some insights into the speed-ups possible due to passage-based decomposition. We use a particular process discovery technique that heavily relies on conformance checking (replay). Section 7.1 presents the setting for the evaluation. Section 7.2 presents the empirical results obtained. These results clearly show that without using passages we would not be able to use the given γ_p algorithm, whereas with using passages it can be used on real-life logs. Section 7.3 discusses the main findings.

7.1. Setting

For the evaluation, we use a real-life event log[2] based on the BPI Challenge 2012 event log [26] and aim to discover a process model describing the most frequent behavior. This real-life log contains noise, which we will tackle by using the Heuristic Miner [41][3] as γ_c algorithm. This miner will provide us with a Heuristic net that shows generic causal relations between actions, but which does not show the specific transitions that represent these relations. To obtain these specific transitions we will use an exhaustive search algorithm as γ_p algorithm. This algorithm simply iterates over all possible sets of transitions that satisfy the causal relations, and takes in the end the set of transitions with maximal fitness.

The left-hand side of Figure 12 shows a passage containing six input nodes and five output nodes. Input node I2 is source to two causal relations: one to IO1 and one to O2. The splitting behavior of I2 can be captured by the exhaustive search algorithm in two ways: either there is an XOR-split between IO1 and O2, or there is an AND-split to both. The splitting behavior of I1 is slightly more complicated, as it involves three causal relations. This behavior can either be captured using (1) a single XOR-split, (2) a single AND-split, or (3) by a combination of an XOR-split between one and an AND-split for the other two, five possibilities in total.

[2]The actual event log used can be downloaded from `http://www.win.tue.nl/~hverbeek/downloads/preprints/Aalst13.xes.gz`.

[3]For sake of completeness: We used the Heuristics Miner with default settings, except for `Relative_to_best`, which we set to 0, and `Dependency`, which we set to 100.

Table 1 shows the numbers of partitions (Bell numbers) up to sets containing 7 relations. Please note that we explicitly partition the set of outgoing causal relations, that is, we do not allow multiple splitting transitions to capture the same outgoing causal relation. Reason for doing so is that the latter would allow for a Petri net that contains all possible splitting transitions, which would have maximal fitness by default. Instead, we partition the outgoing causal relations over the splitting transitions, and try to maximize the fitness thus. Mutatis mutandis, the same holds for incoming causal relations and joining transitions. As an example, there are 52 possible sets of transitions that capture the five incoming edges to the O2 output transition. In total, this particular passage would require $(5 \times 2 \times 2 \times 2) \times (2 \times 52 \times 2) = 8320$ possible sets of transitions. The right-hand side of Figure 12 shows a possible set of transitions.

To show the effect of passages, we will mine a Petri net for different log sizes (1%, 5%, 10%, 50%, and 100% of the $13,087$ traces of the original log) and for different numbers of passages (20, 15, 10, 7, and 5). Table 2 shows the characteristics of the system we used to run the evaluation on. For sake of completeness, we mention that we set the OpenXES shadow size to 16, which allows OpenXES to keep 16 buckets containing event log data in memory.

The log at hand contains 20 passages, of which the most complex passage requires 877 fitness checks, and 2062 fitness checks in total. Starting from these passages, we obtained less (but more complex) passages by combining pairs of passages into single passages. As a result, we obtained situations with 15, 10, 7, and 5 passages (see Table 3). Please note that the decision which passages to merge into new passages may have a huge impact on the resulting run times. For example, if we would merge two passages with 1 and 877 possibilities, then the resulting passage would have 877 possibilities, whereas if we would merge passages with 300 and 10 possibilities, the the resulting passage would have 3000 possibilities. With this in mind, we tried to merge passages that share some nodes in such a way that the remaining number of possibilities would not rise too quickly. For example, to obtain 15 passages from the initial 20 passages, we merged four passages with single-possibility passages, which does not increase the total number of possibilities, and we allowed only a minor increase for the fifth merger (from 25 and 5 to 125).

As for the situation with only 5 passages it was already a challenge to discover the model using only 1% of the traces in the log, we decided to stop at 5 passages. Clearly, it is impossible to apply our brute-force discovery approach to the overall log without any passages. As the results will show, the situation where we would only have a single passage, that is, the situation where we would try an exhaustive search for all possible transitions, would take about 3.46×10^{19} fitness checks, which would have taken eras to compute even if a single fitness check would take a fraction (say, a hundredth) of a second.

7.2. Results

Table 4 and Figure 13 show the results of the evaluation. These results clearly show that the run times are positively effected by an increase in the number of passages. For example, when using only 1 percent of the event log, it takes 879,634 seconds (more than 10 days) to discover the process when using 5 passages whereas this only takes 873 seconds (less than a quarter) to discover the process when using 20 passages.

Figure 14 shows the resulting Petri net from one of the experiments. Please note that not all experiments resulted in the same net, which is caused by the fact that we use random samples from the log to check the fitness on. From this example we can conclude that, at least for this log, many causal relations correspond to XOR-splits and/or XOR-joins, as the resulting net contains only two transitions with mul-

# Edges	1	2	3	4	5	6	7	...
# Possibilities	1	2	5	15	52	203	877	...

Table 1. The number of potential partitions given a set of k edges corresponds to the k-th Bell number

Key	**Value**
Computer	Dell Precision T5400
Processor	Intel® Xeon® CPU, E5430 @ 2.66Ghz (2 processors)
Installed memory (RAM)	16.0 GB
System type	64-bit Windows 7 Enterprise SP 1
JRE	64-bit jdk1.6.0_24
VM arguments	-ea -Xmx4G

Table 2. Basic information on the system used

	# Passages				
Passage	20	15	10	7	5
1	1	1			
2	1	1	1		
3	240				
4	1	240	240	240	240
5	300	300			
6	10	10	3000	3000	
7	25				
8	5	125	125	125	375,000
9	1	1			
10	2	2	2		
11	1				
12	30	30	30	60	60
13	2	2			
14	300	300	600		
15	1				
16	60	60	60	36,000	36,000
17	1	1			
18	877	877	877	877	
19	1				
20	203	203	203	203	178,031
Total	2062	2153	5138	40,505	589,331

Table 3. Numbers of possibilities per passage

# Passages	# Checks	1%	5%	10%	50%	100%
5	589, 331	879, 634	–	–	–	–
7	40, 505	36, 183	100, 852	198, 765	–	–
10	5138	5308	16, 853	25, 928	86, 116	139, 230
15	2153	1087	2882	4480	13, 995	24, 067
20	2062	873	2487	4040	12, 480	20, 414

Table 4. Obtained run times (in seconds). Due to lack of resources, we were unable to run the situations marked –.

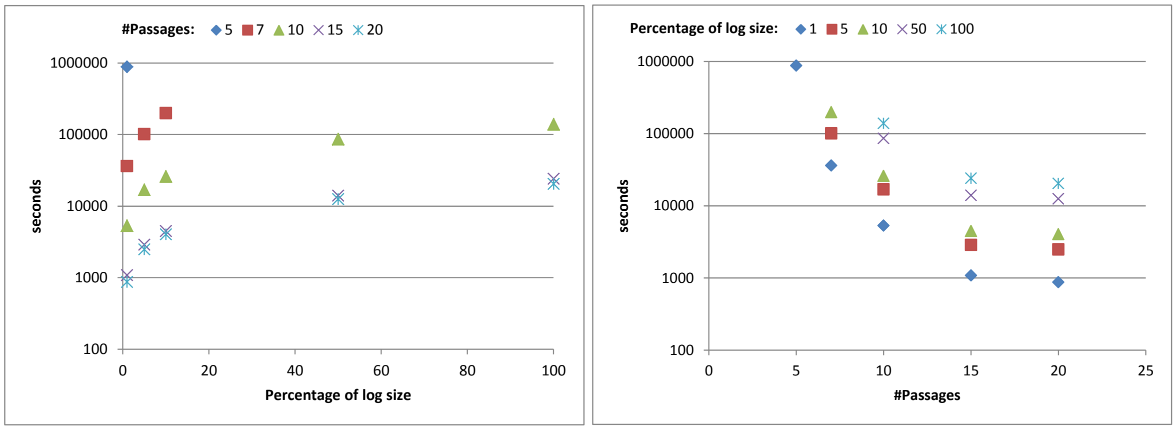

Figure 13. Obtained run times (in seconds) per log size (left) and number of passages (right). Note that run times are plotted on a logarithmic scale and for smaller numbers of passages we were only able to use a fraction of the event log.

tiple outputs and two transitions with multiple inputs. Also note that, using this technique, we were able to mine not only the transitions that do correspond to event classes in the log, but also many transitions that do not correspond to any event class in the log, that is, we were also able to discover many silent transitions.

7.3. Discussion of Experimental Results

Passage-based decomposition enabled us to discover a Petri net from a real-life log (based on the BPI Challenge 2012 log) using a brute-force approach. In a first phase, we have used the Heuristic Miner to extract the major causal relations from the log. In a second phase, we have split up these causal relations into passages, and have used an exhaustive search miner to convert these causal relations into transitions. As a result, we have obtained a Petri net able to explain the mainstream behavior. The resulting net contains 36 transitions that correspond to event classes in the log, 22 silent transitions, and 43 places. Note that the fact that the resulting Petri net contains silent transitions can be considered to be a plus, as there are only few techniques that can both handle noise in the log and come up with these silent transitions.

If we would not have been able to split up the causal relations into smaller parts (like the passages),

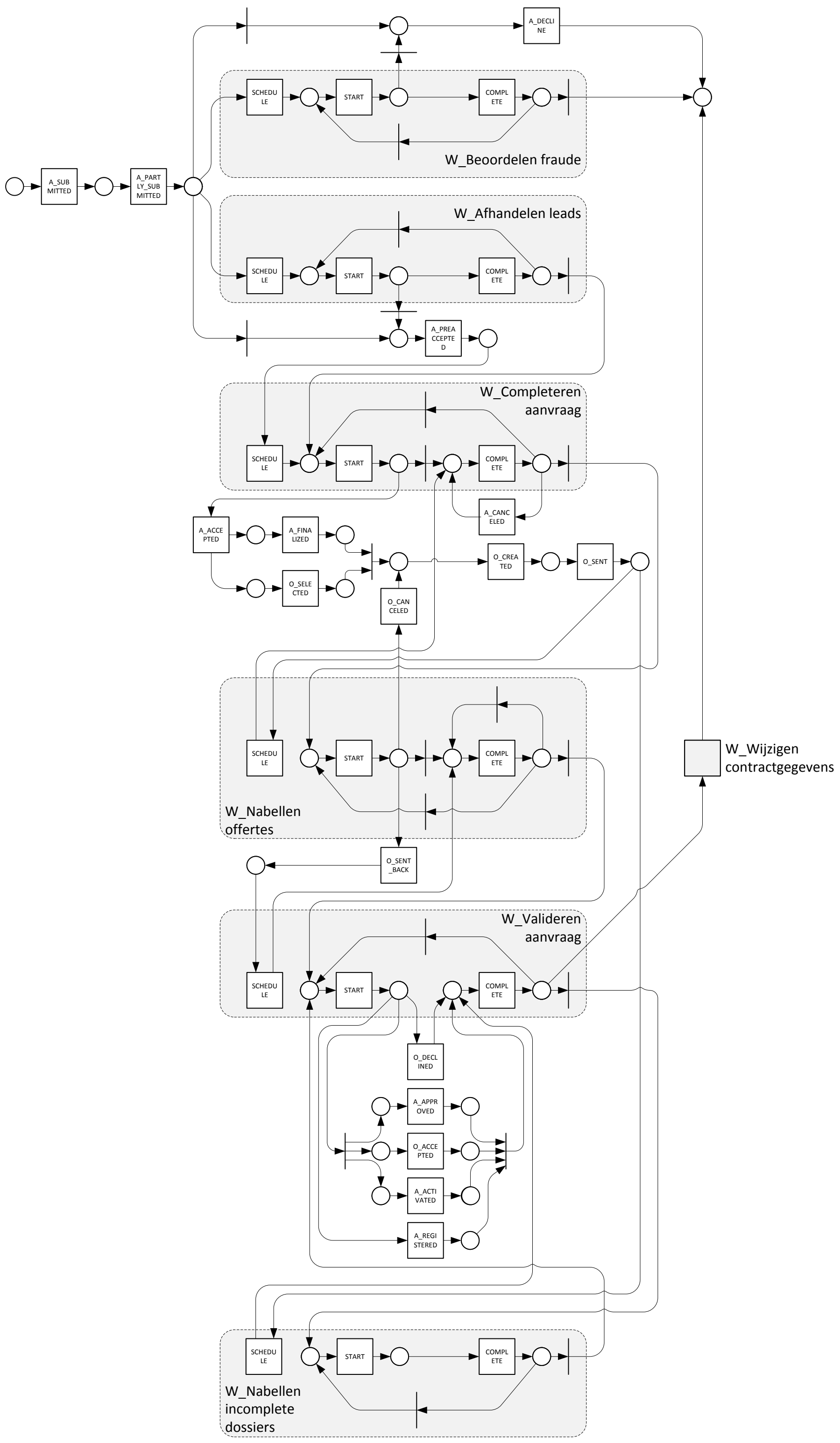

Figure 14. Obtained Petri net

then we would have had a hard time to convert these causal relations into transitions and places, as we would have to check 3.46×10^{19} possible combinations of transitions for the entire net.

The mining of the net took almost six hours when using the complete log to check the fitness for all 2062 possible transition sets for the 20 passages. The fewer passages we detect, the more complex they will be, the more possible transition sets there will be, and the more time it will take to check them all. Therefore, it is important to have as many passages as possible. In our example, our most complex passage corresponded to 877 possible transitions sets, and even this took almost six hours. Hence, it is vital to be able to break down passages into smaller passages in case the complexity of the passages is too high. More research is needed to create so-called "approximate passages", i.e., passages created by inserting artificial events or by leaving out edges that are less important. Obviously, there is a trade-off between the desire to include all possible causalities and breaking down larger passages. However, since one is often looking for understandable models that capture most of the observed behavior, it is valid to consider such trade-offs. Note that models obtained using large passages will typically contain complex fragments that are not understandable.

In this section, we focussed on discovery. However, the brute-force approach repeatedly computes fitness. Hence, the results shown in Table 4 and Figure 13 are also representative for conformance checking.

8. Implementation in ProM

The distributed conformance checking approach presented in Section 5 has been implemented as the "Replay Passages" plug-in in ProM 6.2, and the divide-and-conquer process discovery technique from Section 6 has been implemented as the "Mine Petri net using Passages" plug-in. Both plug-ins have been implemented in the "Passage" package, which is installed in ProM 6.2 by default.

The "Mine Petri net using Passages" plug-in is configured by selecting a $\gamma_c \in \Gamma_c$ algorithm and a $\gamma_p \in \Gamma_p$ algorithm, a maximum size on passages, and which activities to retain in the causal structure. ProM 6.2 supports the following Γ_c algorithms:

Alpha Miner Returns a causal structure based on the $\rightarrow$ ("causality") relation constructed by the α miner [8].

Basic Log Relations Constructs basic log relations (similar to those used by the α miner) from the event log and derives a causal structure from these relations.

Heuristics Miner Returns a causal structure using the Heuristics miner [41, 39]. The user is allowed to configure the Heuristics miner in the usual way (e.g., set thresholds for deriving causalities).

Flower Miner Creates a causal structure which results in two passages: One passage containing only the start ($\top$) and complete ($\perp$) events, and one passage containing all other (i.e., original) events. This algorithm allows one to run the chosen Γ_p algorithm on the original log (as the second passage contains all events from the original, i.e., not extended, log).

Flower and ILP Miner with Proper Completion First creates the two passages like the previous algorithm, then runs the ILP miner (with the proper completion option selected) on the second passage, and derives a causal structure from the mined net.

and the following Γ_p algorithms[4]:

[4]The exhaustive miner as used in Section 7 is not included in ProM 6.2, but is included as the **Exhaustive Miner** in the ProM 6 nightly build (See `http://www.promtools.org/prom6`) as of December 7, 2012.

Alpha Miner Returns a Petri net fragment per passage by applying the α miner [8] to each sublog.

ILP Miner Returns a Petri net fragment per passage by applying the ILP miner [42] to each sublog (using the default configuration).

ILP Miner with Proper Completion Returns a Petri net fragment per passage by applying the ILP miner [42] to each sublog (with the proper completion option selected).

The *maximum passage size* determines for which passages the Γ_p algorithm is used: If the size of a passage ($|X \cup Y|$ for a passage (X, Y)) exceeds this threshold, then a dummy net fragment containing only a transition for every event (i.e, no places) is constructed, otherwise the Γ_p algorithm is used to mine the net fragment. However, if this size is set to 0, then no maximum size applies, i.e., the Γ_p algorithm is used to mine every net fragment.

The current implementation does not allow for the interactive editing of the causal structure. The domain expert can only inspect the causal structure and select the set of activities to retain. After the plug-in has been configured, the passage-based discovery approach described in Definition 6.3 is used to construct the overall process model.

9. Related Work

For an introduction to process mining we refer to [2]. For an overview of best practices and challenges, we refer to the Process Mining Manifesto [29]. The goal of this paper is to decompose challenging process discovery and conformance checking problems into smaller problems [4]. 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 [7, 8, 13, 15, 20, 21, 23, 27, 32, 37, 40, 42]. Many of these techniques use Petri nets during the discovery process and/or to represent the discovered model. It is impossible to provide a complete overview of all techniques here. Very different approaches are used, e.g., heuristics [23, 40], inductive logic programming [27], state-based regions [7, 21, 37], language-based regions [15, 42], and genetic algorithms [32]. Classical synthesis techniques based on regions [25] 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 [7]. Moreover, when constructing the regions, one should avoid overfitting. Language-based regions seem good candidates for discovering transition-bordered Petri nets for passages [15, 42]. Unfortunately, these techniques still have problems dealing with infrequent/incomplete behavior.

As described in [2], there are four competing quality criteria when comparing modeled behavior and recorded behavior: fitness, simplicity, precision, and generalization. In this paper, we focused on fitness, but also precision and generalization can also be investigated per passage. Various conformance checking techniques have been proposed in recent years [5, 9, 10, 12, 18, 24, 27, 33, 34, 36, 38]. 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 applied to passages. The most challenging part is to aggregate the metrics per passage into metrics for the overall model and log. We consider the approach described in [9] 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, 34]. However, the approach can be rather time consuming. Therefore, the efficiency gains can be considerable for larger processes with many activities and passages.

Little work has been done on the decomposition and distribution of process mining problems [4]. In [17] 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 [17], the decomposition approach based on passages is not restricted to genetic process mining.

Most related are the divide-and-conquer techniques presented in [22]. In [22] 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. Passages provide a different (more local) partitioning of the problem and, unlike [22] which focuses specifically on state-based region mining, we decouple the decomposition approach from the actual conformance checking and process discovery approaches.

Several approaches have been proposed to distribute the verification of Petri net properties, e.g., by partitioning the state space using a hash function [16] or by modularizing the state space using localized strongly connected components [30]. These techniques do not consider event logs and cannot be applied to process mining.

Most data mining techniques can be distributed [19], e.g., distributed classification, distributed clustering, and distributed association rule mining [14]. These techniques often partition the input data and cannot be used for the discovery of Petri nets.

This paper is an extended version of a paper presented at Petri nets 2012 [3]. Many of the results have been generalized, e.g., from WF-nets to arbitrary nets and from minimal passages to arbitrary passage partitionings. Moreover, the properties of passages are now described in detail and the notion of passages is supported through various new ProM plug-ins. Unlike [3] we now also provide experimental results showing that speedups are indeed possible.

10. Conclusion

Computationally challenging process mining problems can be decomposed into smaller problems using the new notion of *passages*. As shown, conformance checking can be done per passage and the results per passage can be merged into useful overall conformance diagnostics using the observation that a trace is non-fitting if and only if it is non-fitting for at least one passage. The paper also presents a discovery approach where the discovery problem can be decomposed after determining the causal structure. The refined behavior can be discovered per passage and, subsequently, the discovered net fragments can be merged into an overall process model. Conformance checking and process discovery can be done much more efficiently using such decompositions. Moreover, the notion of passages can be used to localize process-related diagnostics. For example, it is easier to explore conformance-related problems per passage and passages provide a means to hierarchically structure discovered process models. Both approaches have been implemented in ProM 6.2 and can be used for decomposing a variety of conformance checking and process discovery algorithms.

Future work will focus on more large scale experiments demonstrating the performance gains when decomposing various process mining tasks. The experiments in this paper show that the actual speedup heavily depends on the number of passages and the size of the largest passage. If there are many smaller passages, orders of magnitude can be gained. However, in worst case, there is just one passage and no speed-up is possible. Ideally, we would like to use a passage partitioning $Q = \{P_1, P_2, \ldots, P_n\}$ such that n is as large as possible and the passages P_i are as small as possible. From a practical point of view, models with just a few large passages are less interesting as, by definition, they will be Spaghetti-like [2]. To ensure smaller passages, one may need to abstract from edges that are less important. We are currently investigating such "approximate passages". Clearly, there is a trade-off between the desire to include all possible causalities and minimizing the average passage size or the size of the biggest passage. Therefore, we would like to investigate Γ_c algorithms that try to minimize the number of passages without compromising accuracy too much.

References

[1] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[2] W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.

[3] 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.

[4] 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.

[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, 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] R. Agrawal and J.C. Shafer. Parallel Mining of Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, 1996.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] B. van Dongen. BPI Challenge 2012. Dataset. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f, 2012.

[27] 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.

[28] M. Hilbert and P. Lopez. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025):60–65, 2011.

[29] 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.

[30] C. Lakos and L. Petrucci. Modular Analysis of Systems Composed of Semiautonomous Subsystems. In *Application of Concurrency to System Design (ACSD2004)*, pages 185–194. IEEE Computer Society, 2004.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] A. Weijters and J. Ribeiro. Flexible Heuristics Miner (FHM). In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pages 310–317, Paris, France, April 2011. IEEE.

[40] 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.

[41] A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process Mining with the Heuristics Miner-algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven, 2006.

[42] 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.