Decomposing Process Mining Problems Using Passages

Wil M.P. van der Aalst

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands.
www.vdaalst.com

Abstract. Process discovery—discovering a process model from example behavior recorded in an event log—is one of the most challenging tasks in process mining. Discovery approaches need to deal with competing quality criteria such as *fitness*, *simplicity*, *precision*, and *generalization*. Moreover, event logs may contain low frequent behavior and tend to be far from complete (i.e., typically only a fraction of the possible behavior is recorded). At the same time, models need to have formal semantics in order to reason about their quality. These complications explain why dozens of process discovery approaches have been proposed in recent years. Most of these approaches are time-consuming and/or produce poor quality models. In fact, simply checking the quality of a model is already computationally challenging.
This paper shows that process mining problems can be *decomposed* into a set of smaller problems after determining the so-called *causal structure*. Given a causal structure, we partition the activities over a collection of *passages*. Conformance checking and discovery can be done *per passage*. The decomposition of the process mining problems has two advantages. First of all, the problem can be distributed over a network of computers. Second, due to the exponential nature of most process mining algorithms, decomposition can significantly reduce computation time (even on a single computer). As a result, conformance checking and process discovery can be done much more efficiently.

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

1 Introduction

A recent report by the McKinsey Global Institute (MGI) called "Big Data: The Next Frontier for Innovation, Competition, and Productivity" describes the spectacular growth of data and the potential economic value of such data in different industry sectors [31]. MGI estimates that enterprises globally stored more than 7 exabytes of new data on disk drives in 2010, while consumers stored more than 6 exabytes of new data on devices such as PCs and notebooks. Despite the growth of storage space, it impossible to store all event data. The global capacity to store data has been estimated in various studies. For example, a recent study in *Science* suggests that the total global storage capacity increased from 2.6 exabytes in 1986 to 295 exabytes in 2007 [27].

The incredible growth of event data provides new opportunities for process analysis. As more and more actions of people, organizations, and devices are recorded, there are

ample opportunities to analyze processes based on the footprints they leave in event logs. In fact, 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].

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* [28] 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 [6, 13, 36, 21, 39], the α algorithm [7], and various conformance checking techniques [8, 33–35]. 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 problems:

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

Both problems are formulated in terms of Petri nets. However, other process notations could be used, e.g., BPMN models, BPEL specifications, UML activity diagrams, Statecharts, C-nets, 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 3): (a) *fitness* (be able to generate the observed behavior), (b) *simplicity* (avoid large and complex models), (c) *precision* (avoid "underfitting"), and (d) *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, 6, 7, 11, 13, 20–22, 26, 32, 36, 38, 39] and conformance checking [3, 8–10, 18, 23, 26, 33–35, 37] 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 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. Section 2 introduces various preliminaries, including the new notion of passages on graphs, event logs, and Petri nets. Section 3 discusses quality criteria for process mining, e.g., the fitness notion is introduced. The notion of passages is used in Section 4 to decompose the overall conformance checking problem into a set of local conformance checking problems. Section 5 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. Related work is discussed in Section 6. Section 7 concludes the paper.

2 Preliminaries

This section introduces basic concept related to Petri nets, WF-nets, and event logs. Moreover, we introduce the notation of *passages* on arbitrary graphs. This notion will be used to decompose process mining problems into a set of smaller problems.

2.1 Graphs, Passages, 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 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 ${}^{G}\bullet n = \{n' \in N \mid (n', n) \in E\}$ (direct predecessors) and postset $n{}^{G}\bullet = \{n' \in N \mid (n, n') \in E\}$ (direct successors). This can be generalized to sets, i.e., for $X \subseteq N$: ${}^{G}\bullet X = \cup_{n \in X} {}^{G}\bullet n$ and $X {}^{G}\bullet = \cup_{n \in X} n{}^{G}\bullet$. The superscript G can be omitted if the graph is clear from the context.

To decompose process mining problems into smaller problems, we partition process models using the notion *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 2 (Passage). *Let $G = (N, E)$ be a graph. $P = (X, Y)$ is a passage if and only if $\emptyset \neq X \subseteq N$, $\emptyset \neq Y \subseteq N$, $X {}^{G}\bullet = Y$, and $X = {}^{G}\bullet Y$. $pas(G)$ is the set of all passages of G.*

Consider the sets $X = \{b, c, d\}$ and $Y = \{d, e, f\}$ in Fig. 1 (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.

A *weak passage* is a pair (X, Y) such that $\emptyset \neq X \cup Y \subseteq N$, $X {}^{G}\bullet \subseteq Y$, and ${}^{G}\bullet Y \subseteq X$, i.e., X may contain nodes without predecessors and Y may contain nodes without successors. Note that any passage is also a weak passage but not vice versa. In the remainder, we only consider passages.

Definition 3 (Operations on Passages). *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 $P_1 \neq P_2$,*
- *$P_1 \cup P_2 = (X_1 \cup X_2, Y_1 \cup Y_2)$,*
- *$P_1 \setminus P_2 = (X_1 \setminus X_2, Y_1 \setminus Y_2)$.*

The union of two passages $P_1 \cup P_2$ is again a passage. The difference of two passages $P_1 \setminus P_2$ is a passage if $P_2 < P_1$.

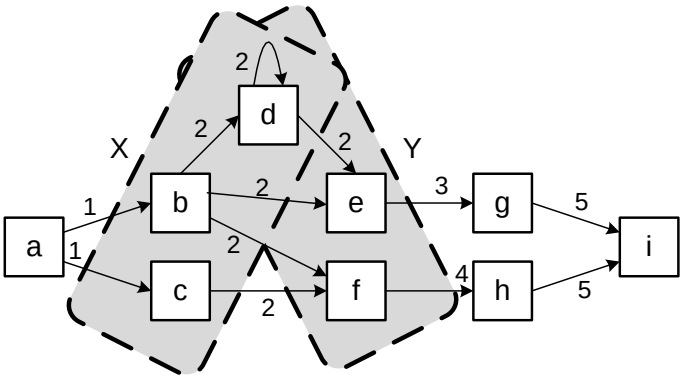

Fig. 1. A graph with five minimal passages: $P_1 = (\{a\}, \{b, c\})$, $P_2 = (\{b, c, d\}, \{d, e, f\})$, $P_3 = (\{e\}, \{g\})$, $P_4 = (\{f\}, \{h\})$, and $P_5 = (\{g, h\}, \{i\})$. Passage P_2 is highlighted and edges carry numbers to refer to the minimal passage they belong to.

Lemma 1 (Properties of Passages). *Let $G = (N, E)$ be a graph with passages $P_1, P_2 \in pas(G)$.*

- *$P_1 \cup P_2$ is a passage.*
- *If $P_2 < P_1$, then $P_1 \setminus P_2$ is a passage.*

Proof. Let $P_1 = (X_1, Y_1)$ and $P_2 = (X_2, Y_2)$ be two passages.

For $P_3 = (X_3, Y_3) = P_1 \cup P_2$ we need to prove: $\emptyset \neq X_3 \subseteq N$, $\emptyset \neq Y_3 \subseteq N$, $X_3\bullet = Y_3$, and $X_3 = \bullet Y_3$. This trivially holds because $X_3\bullet = (X_1 \cup X_2)\bullet = X_1\bullet \cup X_2\bullet = Y_1 \cup Y_2 = Y_3$ and $\bullet Y_3 = \bullet(Y_1 \cup Y_2) = \bullet Y_1 \cup \bullet Y_2 = X_1 \cup X_2 = X_3$.

Assume that $P_2 < P_1$ and $P_3 = (X_3, Y_3) = P_1 \setminus P_2$. Again we need to prove that $\emptyset \neq X_3 \subseteq N$, $\emptyset \neq Y_3 \subseteq N$, $X_3\bullet = Y_3$, and $X_3 = \bullet Y_3$. There is a $(x, y) \in E$ with $x \in X_3$ and $y \in Y_3$. Otherwise, $P_2 \not< P_1$. Hence, $X_3 \neq \emptyset$ and $Y_3 \neq \emptyset$. Observe that $X_2\bullet \cap X_3\bullet = \emptyset$ and $\bullet Y_2 \cap \bullet Y_3 = \emptyset$ because P_2 is a passage. Moreover, $X_3\bullet \subseteq Y_1$ and $\bullet Y_3 \subseteq X_1$. Hence, $X_3\bullet = (X_1 \setminus X_2)\bullet = X_1\bullet \setminus X_2\bullet = Y_1 \setminus Y_2 = Y_3$. $\bullet Y_3 = \bullet(Y_1 \setminus Y_2) = \bullet Y_1 \setminus \bullet Y_2 = X_1 \setminus X_2 = X_3$. Therefore, P_3 is indeed a passage. □

Since the union of two passages is again a passage, it is interesting to consider *minimal passages*. A passage is *minimal* if it does not "contain" a smaller passage.

Definition 4 (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.*

Figure 1 contains five minimal passages. The sets X and Y highlight minimal passage $P_2 = (\{b, c, d\}, \{d, e, f\})$. The edges in Fig. 1 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 $P_1 = (\{a\}, \{b, c\})$. Here we already use the property that an 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 2. *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. The procedure is deterministic. Hence, there is precisely one minimal passage for $(x, y) \in E$. □

Passages define an equivalence relation on the edges in a graph: $(x_1, y_1) \sim (x_2, y_2)$ if and only if $P_{(x_1,y_1)} = P_{(x_2,y_2)}$. 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 $(x_1, y_1) \sim (x_3, y_3)$). In Fig. 1 $(b, d) \sim (b, e) \sim (b, f) \sim (c, f) \sim (d, d) \sim (d, e)$, i.e., the arcs having label "2" form an equivalence class.

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. Moreover, $pas_{min}(G) = \{P_{(x,y)} \mid (x, y) \in E\}$.

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 in the graph $G = (N, E)$ where the set of intermediate nodes visited by path σ does not include any nodes in Q.

Definition 5 (Path). *Let $G = (N, E)$ be a graph with $x, y \in N$ and $Q \subseteq N$. $x \overset{\sigma:E\#Q}{\rightsquigarrow} y$ if and only if there is a sequence $\sigma = \langle n_1, n_2, \ldots n_k \rangle$ with $k > 1$ such that $x = n_1$, $y = n_k$, 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 path σ such that $x \overset{\sigma:E\#Q}{\rightsquigarrow} y$,*
- *$x \overset{\sigma:E}{\rightsquigarrow} y$ is a shorthand for $x \overset{\sigma:E\#Q}{\rightsquigarrow} y$ with $Q = \emptyset$,*
- *$nodes(x \overset{E\#Q}{\rightsquigarrow} y) = \{n \in \sigma \mid \exists_{\sigma \in N^*}\ x \overset{\sigma:E\#Q}{\rightsquigarrow} y\}$, and*
- *for $X, Y \subseteq N$: $nodes(X \overset{E\#Q}{\rightsquigarrow} Y) = \cup_{(x,y)\in X\times Y}\ nodes(x \overset{E\#Q}{\rightsquigarrow} y)$.*

Consider the graph $G = (N, E)$ in Fig. 1 to illustrate these notions. $a \overset{E\#Q}{\rightsquigarrow} i$ 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 $nodes(a \overset{E\#Q}{\rightsquigarrow} y) = \{a, b, c, f, h, i\}$ 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 multisets over some set A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times element $a \in A$ appears in b. Some examples: $b_1 = [\,]$,

$b_2 = [x, x, y]$, $b_3 = [x, y, z]$, $b_4 = [x, x, y, x, y, z]$, $b_5 = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. b_1 is the empty multiset, b_2 and b_3 both consist of three elements, and $b_4 = b_5$, i.e., the ordering of elements is irrelevant and a more compact notation may be used for repeating elements.

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

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 6 (Petri Net). *A Petri net is tuple $PN = (P, T, F)$ with P the set of places, T the set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation.*

Figure 2 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. $[start]$ is the initial marking shown in Fig. 2. Another potential marking is $[c1^{10}, c2^5, c4^5]$. This is the state with ten tokens in $c1$, five tokens in $c2$, and five tokens in $c4$.

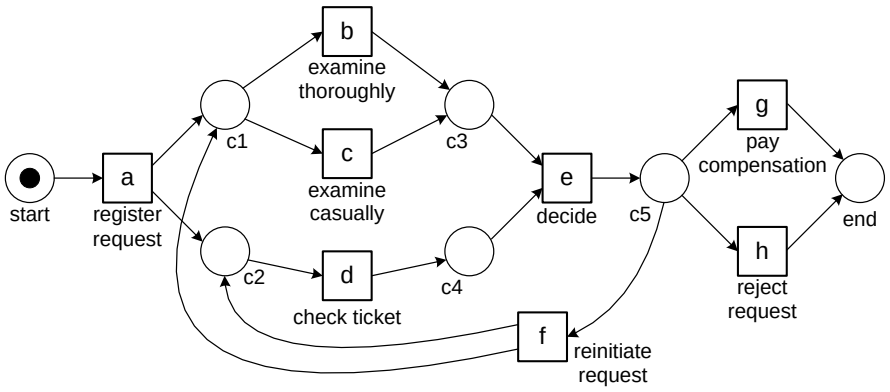

Fig. 2. A Petri net.

Definition 7 (Marking). *Let $PN = (P, T, F)$ be Petri net. A marking M is a multiset of places, i.e., $M \in \mathcal{B}(P)$.*

Like for graphs we define the preset and postset of a node. For any $x \in P \cup T$, ${}^{PN}\bullet x = \{y \mid (y,x) \in F\}$ (input nodes) and $x {}^{PN}\bullet = \{y \mid (x,y) \in F\}$ (output nodes). We drop the superscript PN 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. 2 with $M = [c3, c4]$: $M[e\rangle$ because both input places are marked.

An enabled transition t may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. Formally: $M' = (M \setminus \bullet t) \uplus t\bullet$ is the marking resulting from firing enabled transition t in marking M. $M[t\rangle M'$ 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. 2.

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 8 (Labeled Petri Net). *A labeled Petri net $PN = (P, T, F, T_v)$ is a Petri net (P, T, F) with visible labels $T_v \subseteq T$. 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., $\sigma_v = \sigma \restriction_{T_v}$).*

If we assume $T_v = \{a, e, g, h\}$ for the Petri net in Fig. 2, 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).

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. 2 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 9 (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 labels 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 10 (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 traces starting in M_i and ending in M_o.*

If we assume $T_v = \{a, e, f, g, h\}$ for the Petri net in Fig. 2, then $\tau(SN) = \{\langle a, e, g\rangle, \langle a, e, h\rangle, \langle a, e, f, e, g\rangle, \langle a, e, f, e, h\rangle, \ldots\}$.

2.4 WF-net

The Petri net in Fig. 2 has a designated source place ($start$), a designated source place (end), and all nodes are on a path from $start$ to end. Such nets are called *WF-nets* [1, 4].

Definition 11 (WF-net). *$WF = (PN, in, T_i, out, T_o)$ is a workflow net (WF-net) if*

- $PN = (P, T, F, T_v)$ *is a labeled Petri net,*
- $in \in P$ *is a source place such that* $\bullet in = \emptyset$ *and* $in\bullet = T_i$*,*
- $out \in P$ *is a sink place such that* $out\bullet = \emptyset$ *and* $\bullet out = T_o$*,*
- $T_i \subseteq T_v$ *is the set of initial transitions and* $\bullet T_i = \{in\}$*,*
- $T_o \subseteq T_v$ *is the set of final transitions and* $T_o\bullet = \{out\}$*, and*
- $nodes(in \overset{F}{\rightsquigarrow} out) = P \cup T$*, i.e., all nodes are on some path from source place* in *to sink place* out*.*

WF-nets are often used in the context of business process modeling and process mining. Compared to the standard definition of WF-nets [1, 4] we added the requirement that the initial and final transitions need to be visible.

A WF-net $WF = (PN, in, T_i, out, T_o)$ defines the system $SN = (PN, M_i, M_o)$ with $M_i = [in]$ and $M_o = [out]$. Ideally WF-nets are also *sound*, i.e., free of deadlocks, livelocks, and other anomalies [1, 4]. Formally, this means that for any state reachable from M_i it is possible to reach M_o.

Process models discovered using existing process mining techniques may be unsound. Therefore, we cannot assume/require all WF-nets to be sound.

2.5 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 12 (Trace, Event Log). *Let* A *be a set of activities. A trace* $\sigma \in A^*$ *is a sequence of activities.* $L \in \mathcal{B}(A^*)$ *is an event log, i.e., a multiset of traces.*

An event log is a *multiset* of traces because there can be multiple cases having the same trace. In this simple definition of an event log, an event refers to just an *activity*. Often event logs 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.

Definition 13 (Projection). *Let* A *be a set and* $X \subseteq A$ *a subset.* $\restriction_X \in A^* \rightarrow X^*$ *is a projection function and is defined recursively: (a)* $\langle\,\rangle \restriction_X = \langle\,\rangle$ *and (b) for* $\sigma \in A^*$ *and* $a \in A$*:*

$$(\sigma; \langle a\rangle) \restriction_X = \begin{cases} \sigma \restriction_X & \text{if } a \notin X \\ \sigma \restriction_X; \langle a\rangle & \text{if } a \in X \end{cases}$$

The projection function is generalized to event logs, i.e., for some event log $L \in \mathcal{B}(A^*)$ *and set* $X \subseteq A$*:* $L \restriction_X = [\sigma \restriction_X | \ \sigma \in L]$.

For the event log L_1: $L_1 \restriction_{\{a,g,h\}} = [\langle a, g\rangle^{13}, \langle a, h\rangle^7]$. Note that all e and f events have been removed.

3 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 of descriptive model [5]. Deviations may point to fraud, inefficiencies, and poorly designed or outdated procedures. Second, conformance checking can be used to evaluate the results of a process discovery techniques. 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: (a) *fitness*, (b) *simplicity*, (c) *precision*, and (d) *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 14 (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 is perfectly fitting SN if and only if $\{\sigma \in L\} \subseteq \tau(SN)$.*

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 $L_2 = [\langle a, e, g\rangle^{10}, \langle a, e, h\rangle^5, \langle a, g\rangle^3, \langle a, a, g, e, h\rangle^2]$ and the system net SN of the WF-net depicted in Fig. 2 with $T_v = \{a, e, f, g, h\}$. Clearly, L_1 is perfectly fitting SN and L_2 is not. There are various ways to quantify fitness [2, 3, 8, 26, 32–35], typically on a scale from 0 to 1 where 1 means perfect fitness. 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:

$$\gamma_1 = \begin{array}{|c|c|c|}\hline a & e & g \\ \hline a & e & g \\ \hline \end{array} \quad \gamma_2 = \begin{array}{|c|c|c|}\hline a & e & h \\ \hline a & e & h \\ \hline \end{array} \quad \gamma_3 = \begin{array}{|c|c|c|}\hline a & \gg & g \\ \hline a & e & g \\ \hline \end{array} \quad \gamma_4 = \begin{array}{|c|c|c|c|}\hline a & a & g & h \\ \hline a & \gg & e & h \\ \hline \end{array} \quad \gamma_5 = \begin{array}{|c|c|c|c|c|}\hline a & a & \gg & g & h \\ \hline a & \gg & e & g & \gg \\ \hline \end{array}$$

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_4 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_3 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. The goal is to find the alignment with the least number of $\gg$ elements, e.g., γ_4 is clearly better than γ_5. The number of $\gg$ elements can be used to quantify fitness. Moreover, once an optimal alignment has been established for every trace in the event log, these alignments can be used as a basis to quantify precision and generalization [3].

4 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 event infinite) number of traces. Event logs may contain millions of events. Finding the best alignment may require solving many optimization problems [8] or repeated state-space explorations [35]. 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.

In this section, we show that it is possible to decompose and distribute conformance checking problems using the notion of *passages* defined in Section 2.1. In order to do this we focus on the visible transitions and create the so-called *skeleton* of the process model.

Definition 15 (Skeleton). *Let $PN = (P, T, F, T_v)$ be a labeled Petri net. The* skeleton *of PN is the graph $skel(PN) = (N, E)$ with $N = T_v$ and $E = \{(x, y) \in T_v \times T_v \mid x \overset{F\#T_v}{\rightsquigarrow} y\}$.*

Figure 3 shows the skeleton of the WF-net in Fig. 2 assuming that $T_v = \{a, e, f, g, h\}$. The resulting graph has two minimal minimal passages.

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. 2, 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, we can decompose conformance checking problems into smaller problems *by partitioning the Petri net into net*

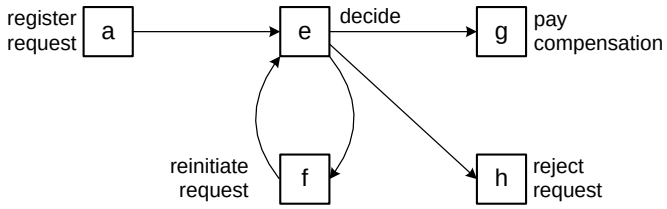

Fig. 3. The skeleton of the labeled Petri net in Fig. 2 (assuming that $T_v = \{a, e, f, g, h\}$). There are two minimal minimal passages: $(\{a, f\}, \{e\})$ and $(\{e\}, \{f, g, h\})$.

fragments and the event log into sublogs. Each passage (X, Y) defines one net fragment $PN^{(X,Y)}$ and one sublog $L \restriction_{X \cup Y}$. We will show that conformance can be checked per passage.

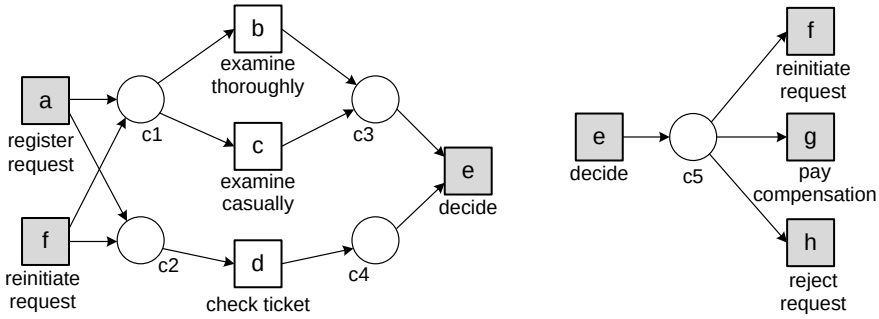

Fig. 4. Two net fragments corresponding to the two passages of the skeleton in Fig. 3: $PN_1 = PN^{(\{a,f\},\{e\})}$ (left) and $PN_2 = PN^{(\{e\},\{f,g,h\})}$ (right). The visible transitions that form the boundaries of a fragment are highlighted.

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 WF-net PN shown in Fig. 2 with $T_v = \{a, e, f, g, h\}$, and the skeleton shown in Fig. 3. 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. 4. 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 is perfectly fitting PN_1 and L_2 is perfectly fitting PN_2, we can conclude that L is perfectly fitting PN. This illustrates that conformance checking can be decomposed.

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

Definition 16 (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. 4 has $Z = \{b, c, d, c1, c2, c3, c4\}$ as internal nodes.

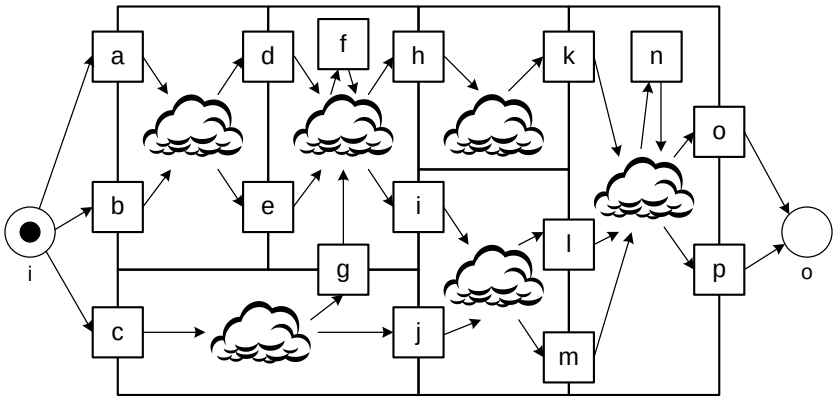

Fig. 5. WF-net WF is decomposed in subnets $PN^{(X,Y)}$. The "clouds" model the internal structure of these subnets (places but possibly also hidden transitions). Due to the decomposition based on passages, one cloud can only influence another cloud through the visible interface transitions X and Y. Since the visible interface transitions are "controlled" by the event log, it is possible to check fitness locally per subnet.

Now we can prove the main result of this paper. Figure 5 illustrates our decomposition approach. A larger model can be decomposed into net fragments corresponding to minimal passages. The event log can be decomposed in a similar manner and conformance checking can be done per passage.

Theorem 1 (Main Theorem). *Let $L \in \mathcal{B}(A^*)$ be an event log and let $WF = (PN, in, T_i, out, T_o)$ be a WF-net with $PN = (P, T, F, T_v)$.*

L is perfectly fitting system net $SN = (PN, [in], [out])$ if and only if

- *for any $\langle a_1, a_2, \ldots a_k \rangle \in L$: $a_1 \in T_i$ and $a_k \in T_o$, and*
- *for any $(X, Y) \in pas_{min}(skel(PN))$: $L \restriction_{X \cup Y}$ is perfectly fitting $SN^{(X,Y)} = (PN^{(X,Y)}, [\,], [\,])$.*

Proof. ($\Rightarrow$) Let $\sigma_v = \langle a_1, a_2, \ldots a_k \rangle \in L$ such that there is a $\sigma \in T^*$ with $[in][\sigma\rangle[out]$ and $\sigma \restriction_{T_v} = \sigma_v$ (i.e., σ_v fits into the overall WF-net). We need to prove the two properties listed above:

- $a_1 \in T_i$ and $a_k \in T_o$ because only transitions in T_i are enabled in the initial marking and only transitions in T_o can produce tokens for out. Moreover, when σ puts a token in place out all other places should be empty; otherwise σ cannot result in $[out]$ (property of WF-nets). Note that $T_i \subseteq T_v$ and $T_o \subseteq T_v$, so the first and last transition need to be visible.
- For any $(X,Y) \in pas_{min}(skel(PN))$: we define $PN^{(X,Y)} = (P', T', F', T'_v)$ and $\sigma' = \sigma \restriction_{T'}$. We need to prove that $[\,][\sigma'\rangle[\,]$ in $PN^{(X,Y)}$. This follows trivially because $SN^{(X,Y)}$ can mimic any move of SN with respect to transitions T'.

($\Leftarrow$) Let $\sigma_v = \langle a_1, a_2, \ldots a_k\rangle \in L$ such that $a_1 \in T_i$, $a_k \in T_o$, and assume that for any $(X,Y) \in pas_{min}(skel(PN))$ there is a sequence $\sigma_{(X,Y)}$ such that $[\,][\sigma_{(X,Y)}\rangle[\,]$ in $PN^{(X,Y)} = (P', T', F', T'_v)$ with $\sigma_{(X,Y)} \restriction_{X \cup Y} = \sigma_v \restriction_{X \cup Y}$. We need to prove that there is a $\sigma \in T^*$ such that $[in][\sigma\rangle[out]$ in PN with $\sigma \restriction_{T_v} = \sigma_v$. The different $\sigma_{(X,Y)}$ sequences can be stitched together into an overall σ because the different subnets only interface via visible transitions. Transitions in one subnet can only influence other subnets through visible transitions and these can only move synchronously as defined by $\sigma_v \in L$. □

Although the theorem only addresses the notion of perfect fitness, other conformance notions can be decomposed in a similar manner. Metrics can be computed per passage and then aggregated into an overall metric.

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 Theorem 1 shows, larger problems can be decomposed into a set 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 approaches use state-space analysis (e.g., in [35] the shortest path enabling a transition is computed) or optimization over all possible alignments (e.g., in [8] 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.*

5 Process Discovery: Divide and Conquer

As explained before, conformance checking and process discovery are closely related. Therefore, we can use the approach used in Theorem 1 for process discovery provided that some coarse *causal structure* (comparable to the skeleton in Section 4) is known. Based on the passages in the causal structure, multiple smaller discovery problems are formulated. This result in one net fragment per passage. These fragments can be folded into an overall model.

More concretely, we propose the following discovery approach:

1. Input is an event log $L_{raw} \in \mathcal{B}(A^*_{raw})$ over a set of activities A_{raw}.
2. Extend each trace in the event log with an artificial start event $\top$ and an artificial end event $\bot$ ($\{\top, \bot\} \cap A_{raw} = \emptyset$). $L_{ext} = [\langle\top\rangle; \sigma; \langle\bot\rangle \mid \sigma \in L_{raw}]$ is the resulting log over $A_{ext} = \{\top, \bot\} \cup A_{raw}$.

3. Discover the causal structure, i.e., we assume that there is an algorithm γ_c such that $\gamma_c(L_{ext}) = (A, C)$ with $\{\top, \bot\} \subseteq A \subseteq A_{ext}$ and $C \subseteq A \times A$. The causal structure may be inspected and modified by a domain expert.
4. Filter the event log using the selected set of activities A: $L = L_{ext} \restriction_A$.
5. Compute the set of passages on the graph $G = (A, C)$: $PS = pas_{min}(G) = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_k, Y_k)\}$. We assume that there is an algorithm γ_p, such that $\gamma_p(L \restriction_{X_i \cup Y_i}, X_i, Y_i) = PN_i = (P_i, T_i, F_i, X_i \cup Y_i)$ returns a Petri net with visible transitions $X_i \cup Y_i$. The discovered Petri nets only overlap with respect to visible transitions, i.e., for $1 \leq i < j \leq k$: $((P_i \cup T_i) \setminus (X_i \cup Y_i)) \cap ((P_j \cup T_j) \setminus (X_j \cup Y_j)) = \emptyset$. Moreover, each PN_i should respect the causal structure, i.e., visible transition $x \in X_i$ is connected to visible transition $y \in Y_i$ in PN_i if and only if $(x, y) \in C$.
6. Merge the individual subsets into one overall system net $SN = (PN, M_i, M_o)$ with $PN = (P, T, F, T_v)$ such that:
   - $P = \{in, out\} \cup \cup_{1 \leq i \leq k} P_i$,
   - $T = \cup_{1 \leq i \leq k} T_i$,
   - $F = \{(in, \top), (\bot, out)\} \cup (\cup_{1 \leq i \leq k} F_i)$,
   - $T_v = A$,
   - $M_i = [in]$, and
   - $M_o = [out]$.

The discovery process is *parameterized* by γ_c (the algorithm to find causal structure) and γ_p (the algorithm to find a local, transition bordered process model). Any combination of γ_c and γ_p can be used as the two main steps are decoupled by the causal structure. γ_c can also be used to filter out infrequent activities, noise, etc. Moreover, the user is able to edit the causal structure using domain knowledge or particular preferences. Experience shows that user feedback is vital to balance between overfitting and underfitting.

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 created in Step 3 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).

To illustrate the divide and conquer approach based on passages, consider the event log $L_{raw} = [\langle a, b, c, d\rangle^{40}, \langle b, a, c, d\rangle^{35}, \langle a, b, c, e\rangle^{30}, \langle b, a, c, e\rangle^{25}, \langle a, b, x, d\rangle^{1}, \langle a, b, e\rangle^{1}]$. The log describes 132 cases. We first add the artificial start and events (Step 2): $L_{ext} = [\langle \top, a, b, c, d, \bot\rangle^{40}, \langle \top, b, a, c, d, \bot\rangle^{35}, \langle \top, a, b, c, e, \bot\rangle^{30}, \langle \top, b, a, c, e, \bot\rangle^{25}, \langle \top, a, b, x, d, \bot\rangle^{1}, \langle \top, a, b, e, \bot\rangle^{1}]$. Then we compute the causal structure using γ_c (Step 3). Assume that the causal structure shown in Fig. 6 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. L is the log where x is removed (Step 4).

The causal structure has four minimal passages: $P_1 = (\{\top\}, \{a, b\})$, $P_2 = (\{a, b\}, \{c\})$, $P_3 = (\{c\}, \{d, e\})$, and $P_4 = (\{d, e\}, \{\bot\})$. Based on these passages we create four corresponding sublogs: $L_1 = [\langle \top, a, b\rangle^{72}, \langle \top, b, a\rangle^{60}]$, $L_2 = [\langle a, b, c\rangle^{70}, \langle b, a, c\rangle^{60}$,

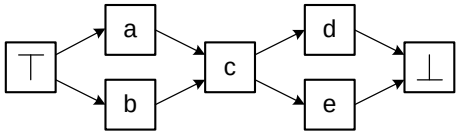

Fig. 6. Causal structure $\gamma_c(L_{ext})$ discovered for the extended event log having four minimal passages.

$\langle a, b\rangle^2]$, $L_3 = [\langle c, d\rangle^{75}, \langle c, e\rangle^{55}, \langle d\rangle^1, \langle e\rangle^1]$, and $L_4 = [\langle d, \bot\rangle^{76}, \langle e, \bot\rangle^{56}]$. One transition-bordered Petri net is discovered per sublog using γ_p (Step 5). Figure 7 shows the resulting net fragments. Note that infrequent behavior has been discarded, i.e., trace $\langle a, b\rangle$ in L_2 is not possible in PN_2, and traces $\langle d\rangle$ and $\langle e\rangle$ in L_3 are not possible in PN_3. What behavior is included and what not depends on γ_p.

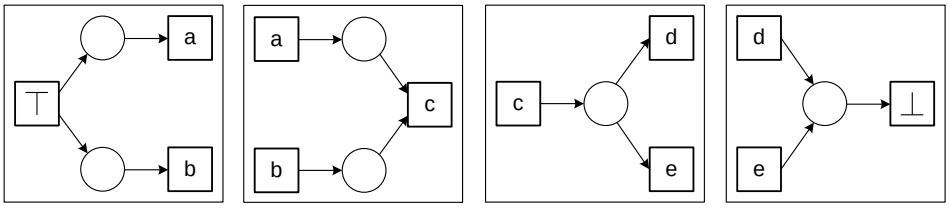

Fig. 7. The Petri net fragments discovered for the four passages: PN_1, PN_1, PN_3, and PN_4.

In the last step of the approach, the four net fragments of Fig. 7 are merged into the overall model shown in Figure 8 (Step 6). Note that this model is indeed able to replay all frequent behavior. Two of the 132 cases cannot be replayed because they were treated as noise by γ_c and γ_p.

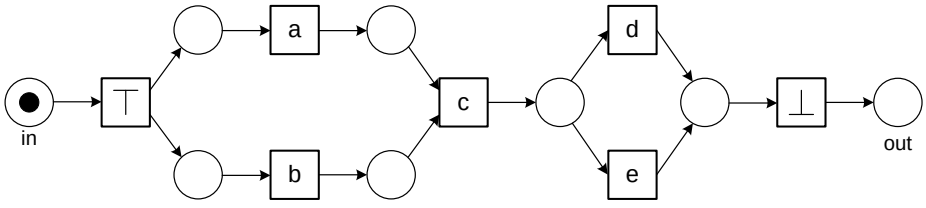

Fig. 8. The WF-net obtained by merging the individual subsets.

The small example shows that we can use a divide and conquer approach when discovering process models. We deliberately did not select concrete algorithms for γ_c and γ_p. The approach is generic and can be combined with existing process discovery

techniques [2, 6, 7, 11, 13, 20–22, 26, 32, 36, 38, 39]. Moreover, the user can modify the causal structure (i.e., the result of γ_c) 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 discovery algorithm γ_p is applied to an event log consisting of just the activities involved in the passage under investigation. Hence, process discovery tasks can be distributed over a network of computers (assuming there are multiple passages). Moreover, most discovery algorithms are exponential in the number of activities. Therefore, the sequential discovery of all individual passages is still faster than solving one big discovery problem.

6 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 [28]. The goal of this paper is to decompose challenging process discovery and conformance checking problems into smaller problems. 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 [6, 7, 11, 13, 20–22, 26, 32, 36, 38, 39]. Many of these techniques use Petri nets during the discovery process and/or to represent the discovered model. It is impossible to provide an complete overview of all techniques here. Very different approaches are used, e.g., heuristics [22, 38], inductive logic programming [26], state-based regions [6, 21, 36], language-based regions [13, 39], and genetic algorithms [32]. Classical synthesis techniques based on regions [14, 24, 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 [6]. Moreover, when constructing the regions, one should avoid overfitting. Language-based regions seem good candidates for discovering transition-bordered Petri nets for passages [13, 39]. 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 [3, 8–10, 18, 23, 26, 33–35, 37]. Conformance checking can be used to evaluate the quality of discovered processes but can also be used for auditing purposes [5]. 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 [8] 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 [3, 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. 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.

Several approaches have been proposed to distribute the verification of Petri net properties, e.g., by partitioning the state space using a hash function [15, 29] or by modularizing the state space using localized strongly connected components [16, 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 [12]. These techniques often partition the input data and cannot be used for the discovery of Petri nets.

7 Conclusion

Computationally challenging process mining problems can be decomposed in smaller problems using the new notion of *passages*. This paper shows that the fitness of the overall model can be analyzed per passage. The approach is independent of the particular conformance checking technique used. Moreover, the same idea can be applied to other conformance notions. 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 approach can be distributed over a network of computers.

This paper presents the idea of passages and provides a formal correctness proof showing that a log is perfectly fitting the overall model if and only if the property holds per passage. Future work will focus on large scale experiments demonstrating the performance gains on a variety of process mining problems. We anticipate that the actual speedup heavily depends on the number of passages. Therefore, it is important investigate this using real-life logs and models.

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, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2012. http://dx.doi.org/10.1002/widm.1045.
4. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. R. Agrawal and J.C. Shafer. Parallel Mining of Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, 1996.
13. 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.
14. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri Nets from Finite Partial Languages. *Fundamenta Informaticae*, 88(4):437–468, 2008.
15. M.C. Boukala and L. Petrucci. Towards Distributed Verification of Petri Nets properties. In *Prococeedings of the International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS'07)*, pages 15–26. British Computer Society, 2007.
16. M.C. Boukala and L. Petrucci. Distributed Verification of Modular Systems. In *Proceedings of CompoNet and SUMo 2011*, volume 726 of *Worksop Proceedings*, pages 1–15, Newcastle, UK, 2011. CEUR.
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.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.
23. 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.
24. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
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. 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.
27. M. Hilbert and P. Lopez. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(60), 2011.
28. IEEE Task Force on Process Mining. Process Mining Manifesto. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *BPM 2011 Workshops, Part I*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer-Verlag, Berlin, 2011.
29. L. Kristensen and L. Petrucci. An Approach to Distributed State Exploration for Coloured Petri Nets. In J. Cortadella and W. Reisig, editors, *Application and Theory of Petri Nets 2004*, volume 3099 of *Lecture Notes in Computer Science*, pages 474–483. Springer-Verlag, Berlin, 2004.
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)*, Paris, France, April 2011. IEEE.
35. 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.
36. 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.
37. 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.
38. 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.
39. 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.