Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 1–10

A General Divide and Conquer Approach for Process Mining

Wil M.P. van der Aalst
Architecture of Information Systems, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
International Laboratory of Process-Aware Information Systems,
National Research University Higher School of Economics (HSE),
33 Kirpichnaya Str., Moscow, Russia.
Email: w.m.p.v.d.aalst@tue.nl

***Abstract*—Operational processes leave trails in the information systems supporting them. Such event data are the starting point for process mining – an emerging scientific discipline relating modeled and observed behavior. The relevance of process mining is increasing as more and more event data become available. The increasing volume of such data ("Big Data") provides both opportunities and challenges for process mining. In this paper we focus on two particular types of process mining: *process discovery* (learning a process model from example behavior recorded in an event log) and *conformance checking* (diagnosing and quantifying discrepancies between observed behavior and modeled behavior). These tasks become challenging when there are hundreds or even thousands of different activities and millions of cases. Typically, process mining algorithms are linear in the number of cases and exponential in the number of different activities. This paper proposes a very general divide-and-conquer approach that decomposes the event log based on a partitioning of activities. Unlike existing approaches, this paper does not assume a particular process representation (e.g., Petri nets or BPMN) and allows for various decomposition strategies (e.g., SESE- or passage-based decomposition). Moreover, the generic divide-and-conquer approach reveals the core requirements for decomposing process discovery and conformance checking problems.**

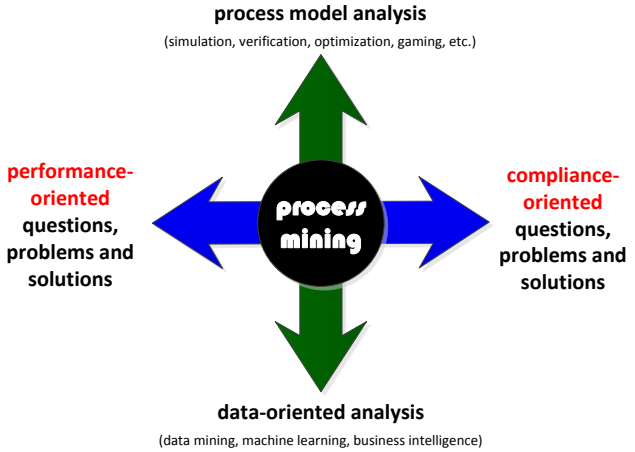

Fig. 1. Process mining is on the interface between process model analysis and data-oriented analysis and can be used to answer a variety of performance and compliance-related questions.

I. Introduction

RECENTLY, *process mining* emerged as a new scientific discipline on the interface between process models and event data [1]. Conventional *Business Process Management* (BPM) [2] and *Workflow Management* (WfM) [3] approaches and tools are mostly model-driven with little consideration for event data. *Data Mining* (DM) [4], *Business Intelligence* (BI), and *Machine Learning* (ML) [5] focus on data without considering end-to-end process models. Process mining aims to bridge the gap between BPM and WfM on the one hand and DM, BI, and ML on the other hand (cf. Figure 1).

The practical relevance of process mining is increasing as more and more event data become available (cf. the recent attention for "Big Data"). Process mining techniques aim to *discover, monitor and improve real processes by extracting knowledge from event logs*. The two most prominent process mining tasks are: (i) *process discovery*: learning a process model from example behavior recorded in an event log, and (ii) *conformance checking*: diagnosing and quantifying discrepancies between observed behavior and modeled behavior.

Starting point for any process mining task is an *event log*. Each *event* in such a log refers to an *activity* (i.e., a well-defined step in some process) and is related to a particular *case* (i.e., a *process instance*). The events belonging to a case are ordered, and can be seen as one "run" of the process. Such a run is often referred to as a *trace*. 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.

Lion's share of process mining research has been devoted to process discovery [1]. Here the challenge is to turn a multiset of example traces (observed cases) into a process model. Process representations allowing for concurrency and choice, e.g., Petri nets, BPMN models, UML activity diagrams, or EPCs, are preferred over low-level notations such as finite state machines of hidden Markov models [1].

Given a process model (discovered or made by hand) and an event log one can try to *align modeled and observed behavior*. An alignment relates a trace in a event log to its corresponding path in the model. If there is not a direct match, the trace is aligned with the closest or most likely path. Such alignments can be used to answer *performance-oriented* and *compliance-oriented* questions (cf. Figure 1). Alignments can be used to show how often paths are taken and activities are being executed. Moreover, events often bear a timestamp which

978-1-4673-4471-5/$25.00

can be used to compute flow times, waiting times, service times, etc. For example, alignments can be used to highlight bottlenecks in the process model. Similarly, alignments can be used to show where model and event log disagree. This is commonly referred to as *conformance checking*.

The incredible growth of event data is also posing new challenges [6]. As event logs grow, process mining techniques need to become more efficient and highly scalable. Moreover, torrents of event data need to be distributed over multiple databases and large process mining problems need to be distributed over a network of computers. Several approaches have been described in literature [7], [8], [9], [10], [11] (also see the related work described in Section VII). In this paper, we describe a *generic divide-and-conquer approach* based on a (valid) *partitioning* of the activities in sets. The activity sets should overlap if there is a direct dependency. We will illustrate the divide-and-conquer approach for the two main process mining tasks:

- For conformance checking, we decompose the process model into smaller partly overlapping submodels using projection. The event log is decomposed into sublogs, also using projection. Any trace that fits into the overall model also fits all submodels. The reverse only holds if the partitioning is *valid*. Metrics such as the fraction of fitting cases can be computed by checking the conformance of the submodels.
- To decompose process discovery, we first create an activity partitioning, i.e., we split the set of activities into a collection of partly overlapping activity sets. For each activity set, we project the log onto a sublog and discover a submodel for it. The different submodels can be merged to create an overall process model. Again it is guaranteed that all traces in the event log that fit into the overall model also fit into the submodels and vice versa.

Unlike existing papers [7], [8], [9], [10], [11], we abstract from a concrete process representation and do not select a particular decomposition strategy. Instead we focus on the core requirements to enable decomposition.

The remainder is organized as follows. Section II introduces preliminaries ranging from multisets to event logs. Section III provides abstract high-level definitions for process discovery and conformance checking. Section IV shows that any process mining problem can be decomposed trivially, but with a possible loss of precision. Section V shows that exact results can be obtained (not just bounds) if the activity partitioning is valid. Section VI discusses possible strategies to obtain valid (or otherwise suitable) activity partitionings. Related work is described in Section VII. Section VIII concludes the paper.

II. PRELIMINARIES

Before describing the two main process mining tasks and the ways in which these tasks can be distributed, we introduce some basic notations to reason about event logs and process models.

A. Multisets, Sequences and Projection

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., $z \in b_3$, $b_2 \uplus b_3 = b_4$, $b_5 \setminus b_2 = b_3$, $|b_5| = 6$, etc. Bags are compared in the usual manner, i.e., $b_2 \leq b_4$ and $b_2 \not\leq b_3$. $\{a \in b\}$ 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.

$\mathcal{P}(X)$ is the powerset of X, i.e., $Y \in \mathcal{P}(X)$ if $Y \subseteq X$.

$\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in X^*$ denotes a sequence over X. $|\sigma| = n$ is its length. $a \in \sigma$ if and only if $a \in \{a_1, a_2, \ldots, a_n\}$. $\langle\,\rangle$ is the empty sequence.

Projection is defined for sequences and sets or bags of sequences.

Definition 1 (Projection): Let $\sigma \in X^*$ and $Y \subseteq X$. $\sigma{\restriction_Y}$ is the projection of σ on Y, i.e., all elements in $X \setminus Y$ are removed (e.g., $\langle x, y, z, x, y, z \rangle{\restriction_{\{x,y\}}} = \langle x, y, x, y \rangle$). Projection is generalized to sets and bags. If $s \in \mathcal{P}(X^*)$, then $s{\restriction_Y} = \{\sigma{\restriction_Y} \mid \sigma \in s\}$ If $b \in \mathcal{B}(X^*)$, then $b{\restriction_Y} = [\sigma{\restriction_Y} \mid \sigma \in b]$. In the latter case frequencies are respected, e.g., $[\langle x, y, z, y \rangle^{10}, \langle z, y, z, y \rangle^5]{\restriction_{\{x,y\}}} = [\langle x, y, y \rangle^{10}, \langle y, y \rangle^5]$.

Without proof we mention some basic properties for sequences and projections.

Lemma 1 (Projection Properties): Let $\sigma \in X^*$, $Y \subseteq X$, $s \in \mathcal{P}(X^*)$, and $b \in \mathcal{B}(X^*)$.

- $\sigma \in s \;\Rightarrow\; \sigma{\restriction_Y} \in s{\restriction_Y}$,
- $\sigma \in b \;\Rightarrow\; \sigma{\restriction_Y} \in b{\restriction_Y}$,
- $\sigma \in s{\restriction_Y} \;\Leftrightarrow\; \exists_{\sigma' \in s}\ \sigma = \sigma'{\restriction_Y}$,
- $\sigma \in b{\restriction_Y} \;\Leftrightarrow\; \exists_{\sigma' \in b}\ \sigma = \sigma'{\restriction_Y}$, and
- for any $\sigma_1 \in {X_1}^*$ and $\sigma_2 \in {X_2}^*$: ${\sigma_1}{\restriction_{X_2}} = {\sigma_2}{\restriction_{X_1}} \;\Leftrightarrow\; \exists_{\sigma_3 \in (X_1 \cup X_2)^*}\ {\sigma_3}{\restriction_{X_1}} = \sigma_1 \;\wedge {\sigma_3}{\restriction_{X_2}} = \sigma_2$.

B. Activities, Traces, Event Logs, and Models

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. Process models are represented as sets of traces. As indicated earlier, we avoid restricting ourselves to a specific process notation. However, we will show some Petri nets and a BPMN model for illustration purposes.

Definition 2 (Universe of Activities, Universe of Traces): $\mathcal{A}$ is the universe of *activities*, i.e., the set of all possible and relevant activities. Other activities cannot be observed (or are

abstracted from). Elements of $\mathcal{A}$ may have *attributes*, e.g., costs, resource information, duration information, etc. A *trace* $\sigma \in \mathcal{A}^*$ is a sequence of activities found in an event log or corresponding to a run of some process model. $\mathcal{U} = \mathcal{A}^*$ is the universe of all possible traces over $\mathcal{A}$.

We assume that an activity is identified by attributes relevant for learning, i.e., irrelevant attributes are removed and attribute values may be coarsened. $|\mathcal{A}|$ is the number of unique activities. Process models with hundreds of activities (or more) tend to be unreadable. In the remainder we will refer to activities using a single letter (e.g. a), however, an activity could also be $decide(gold, manager, reject)$ to represent a decision to reject a gold customer's request by a manager.

In a process model a specific trace $\sigma \in \mathcal{U}$ is possible or not. Hence, a model can be characterized by its set of allowed traces.

Definition 3 (Process Model): A *process model* M is a non-empty collection of traces, i.e., $M \in \mathcal{P}(\mathcal{U})$ and $M \neq \emptyset$.

$A_M = \bigcup_{\sigma \in M} \{a \in \sigma\}$ is the set of activities possible in M. Figure 2 shows a process model M using the Business Process Model and Notation (BPMN) [12]. For this paper the representation itself is irrelevant. Trace $\langle a, b, d, e, f, c, d, g\rangle$ is one of the infinitely many possible traces of M.

An event log is a multiset of *sample* traces from a known or unknown process. The same trace can appear multiple times in the log. Moreover, the event log contains only example behavior. Often only few of the possible traces are observed [1].

Definition 4 (Event Log): An *event log* $L \in \mathcal{B}(\mathcal{U})$ is a multiset of observed traces.

$A_L = \bigcup_{\sigma \in L} \{a \in \sigma\}$ is the set of activities occurring in L. Note that projection (see Definition 1) is defined for both models and event logs.

$L = [\langle a, b, d, e, g\rangle^5, \langle a, c, d, e, h\rangle^4, \langle a, b, d, e, f, c, d, e, g\rangle]$ is an event log containing 10 traces that could have been generated by the BPMN model in Figure 2, e.g., five cases followed the path $\langle a, b, d, e, g\rangle$.

III. Process Discovery and Conformance Checking

In the introduction we already informally introduced the two main process mining tasks: *process discovery* (learning a model from a collection of example behaviors) and *conformance checking* (identifying mismatches between observed and modeled behavior). Using definitions 3 and 4 we can now formalize these notions at a high abstraction level.

Definition 5 (Process Discovery Technique): A process discovery technique $disc \in \mathcal{B}(\mathcal{U}) \rightarrow \mathcal{P}(\mathcal{U})$ is a function that produces a process model $disc(L) \in \mathcal{P}(\mathcal{U})$ for any event log $L \in \mathcal{B}(\mathcal{U})$.

Given an event log $L = [\langle a, c\rangle^5, \langle a, b, c\rangle^4, \langle a, b, b, b, c\rangle]$, the discovery technique may discover the process model that always starts with activity a, followed by zero or more b activities, and always ends with a c activity: $disc(L) = \{\langle a, c\rangle, \langle a, b, c\rangle, \langle a, b, b, c\rangle, \ldots\}$.

An example of a discovery algorithm is the α algorithm [13] that produces a Petri net based on the patterns identified in the event log. Many discovery techniques have been proposed in literature [14], [13], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24] and are supported by open source tools such as ProM and commercial tools such as Disco (Fluxicon), Perceptive Process Mining (also known as Futura Reflect), ARIS Process Performance Manager (Software AG), QPR ProcessAnalyzer, Interstage Process Discovery (Fujitsu), Discovery Analyst (StereoLOGIC), and XMAnalyzer (XMPro). It is impossible to provide an complete overview of all techniques here. Very different approaches can be followed, e.g., using heuristics [19], [23], inductive logic programming [20], state-based regions [14], [18], [22], language-based regions [16], [24], and genetic algorithms [21].

There are four quality dimensions for comparing model and log: (1) *fitness*, (2) *simplicity*, (3) *precision*, and (4) *generalization* [1]. A model with good *fitness* allows for most of the behavior seen in the event log. A model has a perfect fitness if all traces in the log can be replayed by the model from beginning to end. The *simplest* model that can explain the behavior seen in the log is the best model. This principle is known as Occam's Razor. Fitness and simplicity alone are not sufficient to judge the quality of a discovered process model. For example, it is very easy to construct an extremely simple Petri net ("flower model") that is able to replay all traces in an event log (but also any other event log referring to the same set of activities). Similarly, it is undesirable to have a model that only allows for the exact behavior seen in the event log. Remember that the log contains only example behavior and that many traces that are possible may not have been observed 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).

We often focus on fitness, e.g., event log $L = [\langle a, b, d, e, g\rangle^5, \langle a, c, d, e, h\rangle^4, \langle a, b, d, e, f, c, d, e, g\rangle]$ is perfectly fitting model M described in Figure 2.

Definition 6 (Conformance Checking Technique): A conformance checking technique $check \in (\mathcal{B}(\mathcal{U}) \times \mathcal{P}(\mathcal{U})) \rightarrow \mathcal{D}$ is a function that computes conformance diagnostics $check(L, M) \in \mathcal{D}$ (e.g., fitness or precision metrics) given an event log $L \in \mathcal{B}(\mathcal{U})$ and process model $M \in \mathcal{P}(\mathcal{U})$. $\mathcal{D}$ is the set of all possible diagnoses (e.g., a fitness value between 0 and 1) and depends on the metric chosen.

As indicated, we will often focus on fitness. Hence, we introduce some functions characterizing fitness.

Definition 7 (Conformance Checking Functions): Given an event log $L \in \mathcal{B}(\mathcal{U})$ and process model $M \in \mathcal{P}(\mathcal{U})$, we define the following functions:

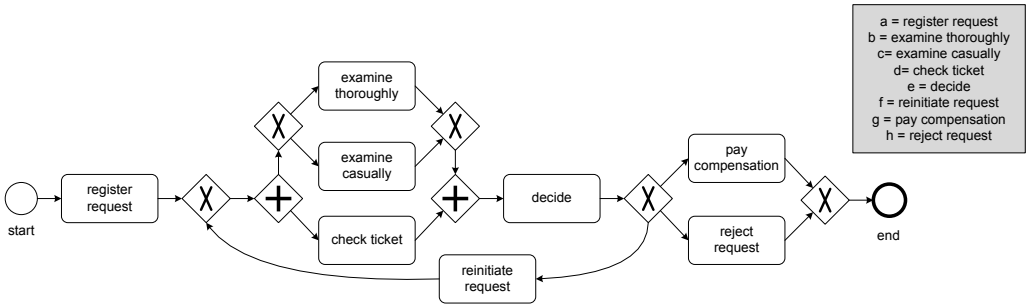

Fig. 2. A process model $M = \{\langle a,b,d,e,g\rangle, \langle a,c,d,e,g\rangle, \langle a,d,b,e,g\rangle, \langle a,d,c,e,g\rangle, \langle a,b,d,e,h\rangle, \langle a,c,d,e,h\rangle, \langle a,d,b,e,h\rangle, \langle a,d,c,e,h\rangle, \langle a,b,d,e,f,c,d,e,g\rangle, \langle a,c,d,e,f,b,d,e,h\rangle, \ldots\}$ expressed in terms of BPMN.

- $\mathit{fit}(L,M) = [\sigma \in L \mid \sigma \in M]$ is the multiset of fitting traces,
- $\mathit{nofit}(L,M) = [\sigma \in L \mid \sigma \notin M]$ is the multiset of non-fitting traces,
- $check_{pft}(L,M) = \frac{|\mathit{fit}(L,M)|}{|L|}$ is the fraction of traces in the event log perfectly fitting the model,
- $check_{bpf}(L,M) = (\mathit{fit}(L,M) = L)$ is a boolean function returning true if all traces in the event log fit.

Note that $\mathit{fit}(L,M) \uplus \mathit{nofit}(L,M) = L$. Clearly, $check_{pft}(L,M)$ and $check_{bpf}(L,M)$ are conformance checking functions in the spirit of Definition 6.

Simply counting the fraction of fitting cases is often too simplistic. Typically, one is interested in conformance metrics at the event level. For example, Petri-net based conformance checking approaches [25] may count the number of missing and remaining tokens during replay. Many conformance checking techniques have been proposed [26], [27], [28], [29], [30], [31], [20], [32], [33], [25], [34]. The alignment-based approach described in [26], [27] is much more sophisticated and provides much better diagnostics than simple metrics such as $check_{pft}(L,M)$. To illustrate the notion of alignments, let us consider the non-fitting trace $\langle a,b,c,e,g\rangle$ and process model M depicted in Figure 2. An optimal alignment is:

$$\gamma_1 = \left|\begin{array}{c|c|c|c|c|c} a & b & c & \gg & e & g \\ a & b & \gg & d & e & g \end{array}\right|$$

The $\gg$ symbols indicate the problems. The third column shows a "move on log only" $(c, \gg)$ indicating that the observed c event cannot be matched with a move of the model. The fourth column shows a "move on model only" $(\gg, d)$ indicating that the necessary execution of d in the model cannot be matched with an event in the log. All other columns refer to "synchronous moves", i.e., model and log agree. Alignment γ_1 has lowest costs assuming equal costs to all non-synchronous moves, i.e., simply count the number of $\gg$ symbols. A non-optimal alignment is:

$$\gamma_2 = \left|\begin{array}{c|c|c|c|c|c|c|c|c|c} a & b & \gg & \gg & \gg & c & \gg & e & g \\ a & b & d & e & f & c & d & e & g \end{array}\right|$$

Alignment γ_2 has four non-synchronous moves, i.e., double the number of non-synchronous moves compared to γ_1. Therefore, it is not optimal and not considered during analysis.

The alignment-based approach is very flexible because it can deal with arbitrary cost functions and any model representation. For example, one can associate costs to activities that are executed too late or by the wrong person. Alignments can also be used for computing precision and generalization [26], [33]. However, the approach can be rather time consuming. Therefore, the efficiency gains obtained through decomposition can be considerable for larger processes.

For simplicity we will focus in the remainder on the fraction of perfectly fitting traces $check_{pft}(L,M)$. However, as illustrated by the results in [7], we can use our decomposition approach also for more sophisticated alignment-based approaches.

IV. Decomposing Models and Logs

As events logs and process models grow in size, process mining may become a time consuming activity. Conformance checking may become intractable when many different traces need to be aligned with a model that allows 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 or repeated state-space explorations. In worst case, a state-space exploration of the model is needed per event. When using genetic process mining, one needs to check the fitness of every individual model in every generation. As a result, millions of conformance checks may be required. For each conformance check, the whole event log needs to be traversed.

For process discovery there are similar problems. Now the model is not given and the challenge is to find a model that scores good with respect to different objectives, e.g., fitness, simplicity, precision, and generalization. Depending on the representational bias, there may be infinitely many candidate models.

Given these challenges, we are interested in reducing the time needed for process mining tasks by decomposing the associated event log. In this section, we show that it is possible to decompose any process mining problem by *partitioning the set of activities*.

Definition 8 (Activity Partitioning): $P = \{A_1, A_2, \ldots, A_n\}$ is an *activity partitioning* of a set A if $A = \bigcup_{1 \leq i \leq n} A_i$. The activity sets may overlap. $\tilde{A}_i = A_i \cap \bigcup_{j \neq i} A_j$ are all activities that A_i shares with other activity sets. $A_i^I = A_i \setminus \tilde{A}_i$

are the internal activities of A_i. $\overline{A}_i = A \setminus (A_i^I) = \bigcup_{j \neq i} A_j$ are the non-internal activities of A_i.

Note that $\overline{A}_i \cap A_i = \tilde{A}_i$. A possible activity partitioning for the activities used in Figure 2 is $P = \{\{a, b, c, d, e, f\}, \{e, f, g, h\}\}$. Note that both activity sets in P share activities e and f.

Given an activity partitioning P, activity set $A \in P$, trace $\sigma \in \mathcal{U}$, model $M \in \mathcal{P}(\mathcal{U})$, and event log $L \in \mathcal{B}(\mathcal{U})$, we define the following terms:

- $\sigma{\restriction_A}$ is a *subtrace* of σ,
- $M{\restriction_A} \in \mathcal{P}(\mathcal{U})$ is a *submodel* of M, and
- $L{\restriction_A} \in \mathcal{B}(\mathcal{U})$ is a *sublog* of L.

Given an activity partitioning P consisting of n activity sets, we can partition the overall model into n submodels and the overall event log into n sublogs.[1]

It is easy to see that any trace that fits the overall model also fits any submodel (use the first property of Lemma 1). The reverse does not need to hold in case the behavior of one submodel may depend on internal behavior of another submodel. Nevertheless, we can compute bounds for conformance and use this insight for decomposed process discovery.

Definition 9 (Alternative Conformance Functions): Let $M \in \mathcal{P}(\mathcal{U})$ be a process model, $P = \{A_1, A_2, \ldots, A_n\}$ an activity partitioning of A_M, and $L \in \mathcal{B}(\mathcal{U})$ an event log. We define variants of the functions in Definition 7 that only use submodels and sublogs.

- $\mathit{fit}^P(L, M) = [\sigma \in L \mid \forall_{1 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}]$,
- $\mathit{check}^P_{pft}(L, M) = \frac{|\mathit{fit}^P(L,M)|}{|L|}$, and
- $\mathit{check}^P_{bpf}(L, M) = (\mathit{fit}^P(L, M) = L)$.

Theorem 1 (Conformance Bounds): Let $M \in \mathcal{P}(\mathcal{U})$ be a process model and $P = \{A_1, A_2, \ldots, A_n\}$ an activity partitioning of A_M. For any event log $L \in \mathcal{B}(\mathcal{U})$:

- $\mathit{fit}(L, M) \leq \mathit{fit}^P(L, M)$,
- $\mathit{check}_{pft}(L, M) \leq \mathit{check}^P_{pft}(L, M)$, and
- $\mathit{check}_{bpf}(L, M) \Rightarrow \mathit{check}^P_{bpf}(L, M)$.

Proof: Let $\sigma \in \mathit{fit}(L, M)$, i.e., $\sigma \in L$ and $\sigma \in M$. Using Lemma 1 we can deduce that $\sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}$ for any i. Hence, $\sigma \in \mathit{fit}^P(L, M)$. This proves that $\mathit{fit}(L, M) \leq \mathit{fit}^P(L, M)$. The two other statements follow directly from this. ■

Consider activity partitioning $P = \{A_1, A_2\}$ with $A_1 = \{a, b, c, d, e, f\}$, and $A_2 = \{e, f, g, h\}$ for model M in Figure 2 and $L = [\langle a, b, d, e, g\rangle^5, \langle a, c, d, e, h\rangle^4, \langle a, b, d, e, f, c, d, e, g\rangle]$. $M{\restriction_{A_1}} = \{\langle a, b, d, e\rangle, \langle a, c, d, e\rangle, \langle a, d, b, e\rangle, \langle a, d, c, e\rangle, \langle a, b, d, e, f, c, d, e\rangle, \langle a, c, d, e, f, b, d, e\rangle, \ldots\}$ and $M{\restriction_{A_2}} = \{\langle e, g\rangle, \langle e, h\rangle, \langle e, f, e, g\rangle, \langle e, f, e, h\rangle, \ldots\}$. $L{\restriction_{A_1}} = [\langle a, b, d, e\rangle^5, \langle a, c, d, e\rangle^4, \langle a, b, d, e, f, c, d, e\rangle]$ and $L{\restriction_{A_2}} = [\langle e, g\rangle^5, \langle e, h\rangle^4, \langle e, f, e, g\rangle]$. $\mathit{fit}(L, M) = \mathit{fit}^P(L, M) = L$, i.e., all traces fit into the overall model and all submodels. Hence, $\mathit{check}_{pft}(L, M) = \mathit{check}^P_{pft}(L, M) = 1$, and $\mathit{check}_{bpf}(L, M) = \mathit{check}^P_{bpf}(L, M) = \mathit{true}$.

[1]Note that we use the term "partition" in a loose manner. As indicated before, activity sets may overlap and hence submodels and sublogs can also overlap in terms of events/activities.

V. Valid Activity Partitioning

Consider the following four event logs each containing 100 traces: $L_1 = [\langle a, c, d\rangle^{50}, \langle b, c, e\rangle^{50}]$, $L_2 = [\langle a, c, d\rangle^{25}, \langle a, c, e\rangle^{25}, \langle b, c, d\rangle^{25}, \langle b, c, e\rangle^{25}]$, $L_3 = [\langle a, c, d\rangle^{49}, \langle a, c, e\rangle^{1}, \langle b, c, d\rangle^{1}, \langle b, c, e\rangle^{49}]$, and $L_4 = [\langle a, c, d\rangle^{25}, \langle d, c, a\rangle^{25}, \langle b, c, e\rangle^{25}, \langle e, c, b\rangle^{25}]$. Also consider the process models $M_1 = \{\langle a, c, d\rangle, \langle b, c, e\rangle\}$ and $M_2 = \{\langle a, c, d\rangle, \langle a, c, e\rangle, \langle b, c, d\rangle, \langle b, c, e\rangle\}$. Figure 3 shows both models in terms of a Petri net. M_1 is the model *with* places p_1 and p_2 and M_2 is the model *without* these places. Note that the Petri net is just shown for illustration purposes. None of the definitions or results in this paper depends on a particular representation (see Definition 3).

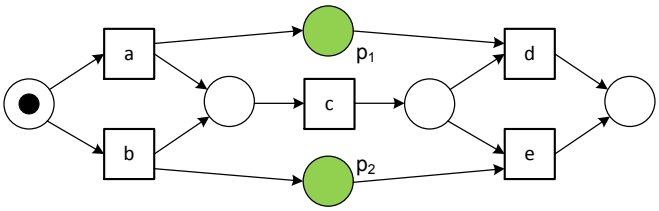

Fig. 3. Two Petri nets used to illustrate valid activity partitioning. $M_1 = \{\langle a, c, d\rangle, \langle b, c, e\rangle\}$ is the model with places p_1 and p_2 and $M_2 = \{\langle a, c, d\rangle, \langle a, c, e\rangle, \langle b, c, d\rangle, \langle b, c, e\rangle\}$ is the model without these places.

Given these three event logs and two process models, we can compute the following fractions of fitting traces:

$\mathit{check}_{pft}(L_1, M_1) = 1$	$\mathit{check}_{pft}(L_1, M_2) = 1$
$\mathit{check}_{pft}(L_2, M_1) = 0.50$	$\mathit{check}_{pft}(L_2, M_2) = 1$
$\mathit{check}_{pft}(L_3, M_1) = 0.98$	$\mathit{check}_{pft}(L_3, M_2) = 1$
$\mathit{check}_{pft}(L_4, M_1) = 0.50$	$\mathit{check}_{pft}(L_4, M_2) = 0.50$

Consider now the activity partitioning $P = \{A_1, A_2\}$ with $A_1 = \{a, b, c\}$ and $A_2 = \{c, d, e\}$. Using this partitioning more events logs will perfectly fit the individual submodels:

$\mathit{check}^P_{pft}(L_1, M_1) = 1$	$\mathit{check}^P_{pft}(L_1, M_2) = 1$
$\mathit{check}^P_{pft}(L_2, M_1) = 1$	$\mathit{check}^P_{pft}(L_2, M_2) = 1$
$\mathit{check}^P_{pft}(L_3, M_1) = 1$	$\mathit{check}^P_{pft}(L_3, M_2) = 1$
$\mathit{check}^P_{pft}(L_4, M_1) = 0.50$	$\mathit{check}^P_{pft}(L_4, M_2) = 0.50$

This illustrates that in general $\mathit{check}_{pft}(L, M) \leq \mathit{check}^P_{pft}(L, M)$, but both values do not need to be the same, e.g., $\mathit{check}_{pft}(L_2, M_1) \neq \mathit{check}^P_{pft}(L_2, M_1)$. In this case the difference is caused by the dependency between the two choices in M_1 which is not "communicated" via the activities in $A_1 \cap A_2 = \{c\}$. This example triggers the question when $\mathit{check}_{pft}(L, M) = \mathit{check}^P_{pft}(L, M)$, or more precisely, what properties must activity partitioning P have such that $\mathit{fit}(L, M) = \mathit{fit}^P(L, M)$?

An activity partitioning is *valid* if the "internal behavior" of one set of activities A_i does not depend on the internals of

another activity set A_j. It other words: the activity sets need to synchronize on non-local phenomena.

Definition 10 (Valid Activity Partitioning): Let $M \in \mathcal{P}(\mathcal{U})$ be a process model with activities A_M and $P = \{A_1, A_2, \ldots, A_n\}$ an activity partitioning of the set A_M. P is *valid* for M if and only if $M = \{\sigma \in {A_M}^* \mid \forall_{1 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}\}$.

Activity partitioning $P = \{A_1, A_2\}$ with $A_1 = \{a, b, c\}$ and $A_2 = \{c, d, e\}$ is valid for M_2 but not for M_1. Note that $\langle a, c, e\rangle{\restriction_{A_1}} = \langle a, c\rangle \in M_1{\restriction_{A_1}}$ and $\langle a, c, e\rangle{\restriction_{A_2}} = \langle c, e\rangle \in M_1{\restriction_{A_2}}$, but $\langle a, c, e\rangle \notin M_1$.

The following theorem shows that a valid activity partitioning allows us to compute conformance per submodel without loosing precision, i.e., we get exact values rather than bounds.

Theorem 2 (Conformance Checking Can Be Decomposed): Let $M \in \mathcal{P}(\mathcal{U})$ be a process model and $P = \{A_1, A_2, \ldots, A_n\}$ a valid activity partitioning of A_M. For any event log $L \in \mathcal{B}(\mathcal{U})$:

- $\mathit{fit}(L, M) = \mathit{fit}^P(L, M)$,
- $\mathit{check}_{pft}(L, M) = \mathit{check}^P_{pft}(L, M)$, and
- $\mathit{check}_{bpf}(L, M) = \mathit{check}^P_{bpf}(L, M)$.

Proof: $\mathit{fit}(L, M) = [\sigma \in L \mid \sigma \in M] = [\sigma \in L \mid \sigma \in \{\sigma' \in {A_M}^* \mid \forall_{1 \leq i \leq n}\ \sigma'{\restriction_{A_i}} \in M{\restriction_{A_i}}\}] = [\sigma \in L \mid \forall_{1 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}] = \mathit{fit}^P(L, M)$. This proves the first statement. The two other statements follow directly from this. ■

Theorem 2 shows that conformance checking can be decomposed. Next we show that also discovery can be decomposed. Here we only have an event log to start with. However, while constructing the overall model we can simply assume independence and thus ensure a valid activity partitioning.

Corollary 1 (Process Discovery Can Be Decomposed): Let $L \in \mathcal{B}(\mathcal{U})$ be an event log and $P = \{A_1, A_2, \ldots, A_n\}$ an activity partitioning of A_L. Let $\mathit{disc} \in \mathcal{B}(\mathcal{U}) \rightarrow \mathcal{P}(\mathcal{U})$ be a discovery algorithm used to obtain the submodels $M_i = \mathit{disc}(L{\restriction_{A_i}})$ with $i \in \{1, \ldots, n\}$. $M = \{\sigma \in U \mid \forall_{1 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M_i\}$ is the overall model constructed by merging the discovered submodels.

- P is a valid activity partitioning for M,
- $\mathit{fit}(L, M) = \mathit{fit}^P(L, M)$,
- $\mathit{check}_{pft}(L, M) = \mathit{check}^P_{pft}(L, M)$, and
- $\mathit{check}_{bpf}(L, M) = \mathit{check}^P_{bpf}(L, M)$.

By applying the construction of Corollary 1 we can decompose process discovery. If all the sublogs fit perfectly, then the overall event log will also fit the overall model perfectly. However, if activity sets are not overlapping sufficiently, the model may be underfitting (too general).

One may try to relax the validity requirement. For example, by considering one activity set A_i and its complete environment $\overline{A_i}$.

Definition 11 (Weakly Valid Activity Partitioning): Let $M \in \mathcal{P}(\mathcal{U})$ be a process model with activities A_M and $P = \{A_1, A_2, \ldots, A_n\}$ an activity partitioning of the set A_M. P is *weakly valid* if and only if $M = \{\sigma \in {A_M}^* \mid \exists_{1 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}} \wedge \sigma{\restriction_{\overline{A_i}}} \in M{\restriction_{\overline{A_i}}}\}$.

Clearly, any valid activity partitioning is also weakly valid. If $\sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}$ and $\sigma{\restriction_{\overline{A_i}}} \in M{\restriction_{\overline{A_i}}}$, then we can apply Lemma 1 to show that $\sigma{\restriction_{A_j}} \in M{\restriction_{A_j}}$ for $j \neq i$. The reverse does not hold. Consider for example $M = \{\langle a, b, d, e, g\rangle, \langle a, c, d, f, g\rangle$ and $P = \{A_1, A_2, A_3, A_4\}$ with $A_1 = \{a, b, c\}$, $A_2 = \{b, c, d\}$, $A_3 = \{d, e, f\}$, and $A_4 = \{e, f, g\}$. $P = \{A_1, A_2, A_3, A_4\}$ is not a valid activity partitioning because the traces $\langle a, c, d, e, g\rangle$ and $\langle a, b, d, f, g\rangle$ are not in M. However, P is weakly valid. This example also shows that Theorem 2 in general does not hold for weakly valid activity partitionings.

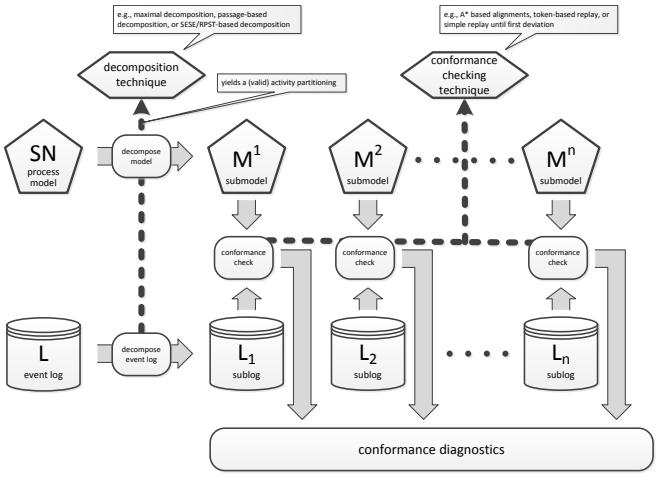

Fig. 4. Overview of decomposed conformance checking showing the configurable elements of the approach: (a) the *decomposition technique* yielding the activity partitioning that is used to split the overall model and event log, and (b) the *conformance checking technique* used to analyze the individual models and sublogs.

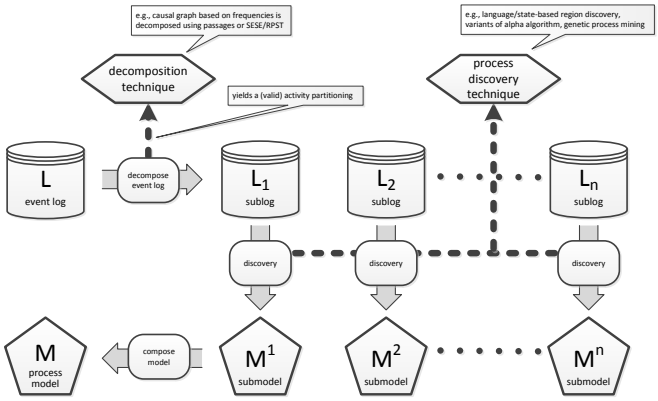

Fig. 5. Overview of decomposed process discovery showing the configurable elements of the approach: (a) the *decomposition technique* yielding the activity partitioning that is the basis for creating sublogs and (b) the *process discovery technique* used to infer submodels from sublogs.

Figures 4 and 5 summarize the overall approach proposed in this paper. Moreover, these figures point out the configurable elements. For conformance checking we need to find an activity partitioning P and a conformance checking technique

$check \in (\mathcal{B}(\mathcal{U}) \times \mathcal{P}(\mathcal{U})) \rightarrow \mathcal{D}$ (see Definition 6). For process discovery we need to find an activity partitioning P (based on only the event log since there is no initial model) and a process discovery technique $disc \in \mathcal{B}(\mathcal{U}) \rightarrow \mathcal{P}(\mathcal{U})$ (see Definition 5).

VI. Finding a (Valid) Activity Partitioning

Theorems 1 and 2 are very general, but stand or fall with a suitable activity partitioning. Consider the following two extreme activity partitionings for an activity set A: $P_{one} = \{A\}$ (just one activity set containing all activities) and $P_{all} = \{\{a\} \mid a \in A\}$ (one activity set per activity). Both are not very useful. P_{one} does not decompose the problem, i.e., there is still one big task. P_{all} decomposes the set of activities into singleton activity sets. P_{all} considers all activities in isolation, hence conformance checking and discovery are only considering frequencies of activities and not their order. For conformance checking P_{all} is typically not valid and decomposed discovery using P_{all} will most likely result in a severely underfitting model.

For conformance checking we can exploit the structure of the process model when searching for a (valid) activity partitioning. For example, the process model (e.g., a Petri net) can be decomposed using the so-called Refined Process Structure Tree (RPST) [35], [36] as shown in [10], [9]. The RPST allows for the construction of a hierarchy of SESE (Single-Exit-Single-Entry) components. Slicing the SESE at the desired level of granularity corresponds to a decomposition of the graph [9] that can be used for process mining.

In [7] an algorithm providing the so-called "maximal decomposition" of a Petri net is given. The construction of the maximal decomposition is based on partitioning the edges. Each edge will end up in precisely one submodel. Edges are taken together if they are connected through a place, through an internal transition (invisible action), or through multiple transitions having the same label. The algorithm iterates until no edges need to joined. Any labeled Petri net has a unique maximal decomposition and this decomposition defines a valid activity partitioning.

The notion of "passages" defined in [8], [37] provides an alternative approach to decompose a Petri net. A passage is a pair of two non-empty sets of activities (X, Y) such that the set of direct successors of X is Y and the set of direct predecessors of Y is X. As shown in [8], any Petri net can be partitioned using passages such that all edges sharing a source vertex or sink vertex are in the same set. This is done to ensure that splits and joins are not decomposed. Note that passages do not necessarily aim at high cohesion and low coupling. Nevertheless, they define a valid activity partitioning.

For discovery we cannot exploit the structure of the model to ensure the validity or suitability of an activity partitioning. Therefore, often an intermediate step is used [7]. For example, one can mine for frequent item sets to find activities that often happen together. Another, probably better performing, approach is to first create a *causal graph* (A, R) where A is the set of activities and $R \subseteq A \times A$ is a relation on A. The interpretation of $(a_1, a_2) \in R$ is that there is a "causal relation" between a_1 and a_2. Most process mining algorithms already build such a graph in a preprocessing step. For example, the α algorithm [13], the heuristic miner [23], and the fuzzy miner [38] scan the event log to see how many times a_1 is followed by a_2. If this occurs above a certain threshold, then it is assumed that $(a_1, a_2) \in R$. Even for large logs it is relatively easy to construct a causal graph (linear in the size of the event log). Moreover, counting the frequencies used to determine a causal graph can be distributed easily by partitioning the cases in the log. Also sampling (determining the graph based on representative examples) may be used to further reduce computation time.

Given a causal graph, one can view decomposition as a graph partitioning problem [39], [40], [41], [42]. There are various approaches to partition the graph such that certain constraints are satisfied while optimizing particular metrics. For example, in [42] a vertex-cut based graph partitioning algorithm is proposed ensuring the balance of the resulting partitions while simultaneously minimizing the number of vertices that are cut (and thus replicated).

Some of the notions in graph partitioning are related to "cohesion and coupling" in software development [43]. Cohesion is the degree to which the elements of a module belong together. Coupling is the degree to which each module relies on the other modules. Typically, one aims at "high cohesion" and "low coupling". In terms of our problem this means that we would like to have activity sets that consist of closely related activities whereas the overlap between the different activity sets is as small as possible while still respecting the causalities.

Definition 10 also suggests to investigate correlations between activity sets. An activity set $A_i \in P$ may be influenced through $\tilde{A}_i = A_i \cap \bigcup_{j \neq i} A_j$ but not through $\overline{A}_i \setminus A_i$. The goal is to find suitable "milestone activities", i.e., shared activities "decoupling" two activity sets.

The ideas mentioned above have only been explored superficially, but nicely illustrate that there are many promising directions for future research.

Interestingly, we can merge activity sets without jeopardizing validity. This allows us to decompose process mining problems at different levels of granularity or to provide a "tree view" on the process and its conformance.

Theorem 3 (Hierarchy Preserves Validity): Let $M \in \mathcal{P}(\mathcal{U})$ be a process model and $P = \{A_1, A_2, \ldots, A_n\}$ a valid activity partitioning of A_M with $n \geq 2$. Activity partitioning $P' = \{A_1 \cup A_2, A_3 \ldots, A_n\}$ is also valid.

Proof: Since P is valid $M = \{\sigma \in {A_M}^* \mid \forall_{1 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}\}$. We need to prove: $\{\sigma \in {A_M}^* \mid \sigma{\restriction_{A_1 \cup A_2}} \in M{\restriction_{A_1 \cup A_2}} \wedge \forall_{3 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}\} = M$. $\sigma{\restriction_{A_1 \cup A_2}} \in M{\restriction_{A_1 \cup A_2}}$ implies that $\sigma{\restriction_{A_1}} \in M{\restriction_{A_1}}$ and $\sigma{\restriction_{A_2}} \in M{\restriction_{A_2}}$ (Lemma 1). Hence, $\{\sigma \in {A_M}^* \mid \sigma{\restriction_{A_1 \cup A_2}} \in M{\restriction_{A_1 \cup A_2}} \wedge \forall_{3 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}\} \subseteq \{\sigma \in {A_M}^* \mid \forall_{1 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}\} = M$. Moreover, for any $\sigma \in M$, $\sigma{\restriction_{A_1 \cup A_2}} \in M{\restriction_{A_1 \cup A_2}} \wedge \forall_{3 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}$ trivially holds. The observation that $M \subseteq \{\sigma \in {A_M}^* \mid \sigma{\restriction_{A_1 \cup A_2}} \in M{\restriction_{A_1 \cup A_2}} \wedge \forall_{3 \leq i \leq n}\ \sigma{\restriction_{A_i}} \in M{\restriction_{A_i}}\} \subseteq M$ completes the proof. ■

Theorem 3 can be applied iteratively. Hence, any combination of activity sets originating from a valid activity partitioning yields another valid activity partitioning. This allows us to coarsen any valid activity partitioning. Note that if $P = \{A_1, A_2, \ldots, A_n\}$ is not valid, then $P' = \{A_1 \cup A_2, A_3 \ldots, A_n\}$

may still be valid. Therefore, we can try to merge problematic activity sets in order to get a valid activity partitioning and better (i.e., more precise) process mining results. For example, when we are using alignments as described in [26], [27] we can diagnose the activity sets that disagree. We can also give preference to alignments that to not disagree on the interface of different activity sets. Last but no least, we can create a hierarchy of conformance/discovery results, similar to a dendrogram in hierarchical clustering.

VII. Related Work

For an introduction to process mining we refer to [1]. For an overview of best practices and challenges, we refer to the Process Mining Manifesto [44]. Also note the availability of open source tools such as ProM and commercial tools such as Disco (Fluxicon), Perceptive Process Mining (also known as Futura Reflect), ARIS Process Performance Manager (Software AG), QPR ProcessAnalyzer, Interstage Process Discovery (Fujitsu), Discovery Analyst (StereoLOGIC), and XMAnalyzer (XMPro).

The goal of this paper is to decompose challenging process discovery and conformance checking problems into smaller problems [6]. 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 [14], [13], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. 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 [19], [23], inductive logic programming [20], state-based regions [14], [18], [22], language-based regions [16], [24], and genetic algorithms [21]. Classical synthesis techniques based on regions [45] 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 [14]. Moreover, when constructing the regions, one should avoid overfitting. Language-based regions seem good candidates for discovering transition-bordered Petri nets that can serve as submodels [16], [24]. Unfortunately, these techniques still have problems dealing with infrequent/incomplete behavior.

There are four competing quality criteria when comparing modeled behavior and recorded behavior: fitness, simplicity, precision, and generalization [1]. In this paper, we focused on fitness, but also precision and generalization can also be investigated per submodel. Various conformance checking techniques have been proposed in recent years [26], [27], [28], [29], [30], [31], [20], [32], [33], [25], [34]. Conformance checking can be used to evaluate the quality of discovered processes but can also be used for auditing purposes [46]. Most of the techniques mentioned can be combined with our decomposition approach. The most challenging part is to aggregate the metrics per model fragment and sublog into metrics for the overall model and log. We consider the approach described in [27] 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 [26], [33]. However, the approach can be rather time consuming. Therefore, the efficiency gains obtained through decomposition can be considerable for larger processes with many activities and possible subnets.

Little work has been done on the decomposition and distribution of process mining problems [6], [7]. In [47] MapReduce is used to scale event correlation as a preprocessing step for process mining. In [48] 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 [48], the decomposition approach in this paper is not restricted to genetic process mining.

More related are the divide-and-conquer techniques presented in [49]. In [49] it is shown that region-based synthesis can be done at the level of synchronized State Machine Components (SMCs). Also a heuristic is given to partition the causal dependency graph into overlapping sets of events that are used to construct sets of SMCs. In this paper we provide a different (more local) partitioning of the problem and, unlike [49] which focuses specifically on state-based region mining, we decouple the decomposition approach from the actual conformance checking and process discovery approaches.

Also related is the work on conformance checking of proclets [50]. Proclets can be used to define so-called artifact centric processes, i.e., processes that are not monolithic but that are composed of smaller interacting processes (called proclets). In [50] it is shown that conformance checking can be done per proclet by projecting the event log onto a single proclet while considering interface transitions in the surrounding proclets.

Several approaches have been proposed to distribute the verification of Petri net properties, e.g., by partitioning the state space using a hash function [51] or by modularizing the state space using localized strongly connected components [52]. These techniques do not consider event logs and cannot be applied to process mining.

Most data mining techniques can be distributed [53], e.g., distributed classification, distributed clustering, and distributed association rule mining [54]. These techniques often partition the input data and cannot be used for the discovery of Petri nets.

This paper generalizes the results presented in [7], [8], [9], [10], [11] to arbitrary decompositions (Petri-net based or not). In [8], [55], [11] it is shown that so-called “passages” [37] can be used to decompose both process discovery and conformance checking problems. In [10], [9] it is shown that so-called SESE (Single-Exit-Single-Entry) components obtained through the Refined Process Structure Tree (RPST) [35], [36] can be used to decompose conformance checking problems. These papers use a particular particular decomposition strategy. However, as shown in [7], there are many ways to decompose process mining problems.

The results in [7] are general but only apply to Petri nets. This paper further generalizes the divide and conquer approach beyond Petri nets. This allows us to simplify the presentation

and clearly show the key requirements for decomposing both process discovery and conformance checking problems.

VIII. Conclusion

In this paper we provided a high-level view on the decomposition of process mining tasks. Both conformance checking and process discovery problems can be divided into smaller problems that can be distributed over multiple computers. Moreover, due to the exponential nature of most process mining techniques, the time needed to solve "many smaller problems" is less than the time needed to solve "one big problem". Therefore, decomposition is useful even if the smaller tasks are done on a single computer. Moreover, decomposing process mining problems is not just interesting from a performance point of view. Decompositions can also be used to pinpoint the most problematic parts of the process (also in terms of performance) and provide localized diagnostics. This also helps us to better understand the limitations of existing conformance checking and process discovery techniques.

In this paper we discussed a very general divide-and-conquer approach without focusing on a particular representation or decomposition strategy. Nevertheless, it provided new and interesting insights with respect to the essential requirements of more concrete approaches. The paper also provides pointers to approaches using Petri nets as a representational bias and SESEs [10], [9], passages [8], [11], or maximal decompositions [7] as a decomposition strategy. It is clear that these are merely examples of the broad spectrum of possible techniques to decompose process mining problems. Given the incredible growth of event data, there is an urgent need to explore and investigate the entire spectrum in more detail.

Acknowledgements

This work was supported by the Basic Research Program of the National Research University Higher School of Economics (HSE). The author would like to thank Eric Verbeek, Jorge Munoz-Gama and Joseph Carmona for their joint work on decomposing Petri nets for process mining (e.g., using passages and SESEs).

References

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

[2] ——, "Business Process Management: A Comprehensive Survey," *ISRN Software Engineering*, pp. 1–37, 2013, doi:10.1155/2013/507984.

[3] W. van der Aalst and K. van Hee, *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2004.

[4] D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. MIT press, Cambridge, MA, 2001.

[5] T. Mitchell, *Machine Learning*. McGraw-Hill, New York, 1997.

[6] W. van der Aalst, "Distributed Process Discovery and Conformance Checking," in *International Conference on Fundamental Approaches to Software Engineering (FASE 2012)*, ser. Lecture Notes in Computer Science, J. Lara and A. Zisman, Eds., vol. 7212. Springer-Verlag, Berlin, 2012, pp. 1–25.

[7] ——, "Decomposing Petri Nets for Process Mining: A Generic Approach," BPM Center Report BPM-12-20 (accepted for Distributed and Parallel Databases), BPMcenter.org, 2012.

[8] ——, "Decomposing Process Mining Problems Using Passages," in *Applications and Theory of Petri Nets 2012*, ser. Lecture Notes in Computer Science, S. Haddad and L. Pomello, Eds., vol. 7347. Springer-Verlag, Berlin, 2012, pp. 72–91.

[9] J. Munoz-Gama, J. Carmona, and W. van der Aalst, "Conformance Checking in the Large: Partitioning and Topology," in *International Conference on Business Process Management (BPM 2013)*, ser. Lecture Notes in Computer Science, F. Daniel, J. Wang, and B. Weber, Eds., vol. 8094. Springer-Verlag, Berlin, 2013, pp. 130–145.

[10] ——, "Hierarchical Conformance Checking of Process Models Based on Event Logs," in *Applications and Theory of Petri Nets 2013*, ser. Lecture Notes in Computer Science, J. Colom and J. Desel, Eds., vol. 7927. Springer-Verlag, Berlin, 2013, pp. 291–310.

[11] H. Verbeek and W. van der Aalst, "Decomposing Replay Problems: A Case Study," BPM Center Report BPM-13-09, BPMcenter.org, 2013.

[12] OMG, "Business Process Model and Notation (BPMN)," Object Management Group, formal/2011-01-03, 2011.

[13] W. van der Aalst, A. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[14] W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Günther, "Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting," *Software and Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2010.

[15] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining Process Models from Workflow Logs," in *Sixth International Conference on Extending Database Technology*, ser. Lecture Notes in Computer Science, vol. 1377. Springer-Verlag, Berlin, 1998, pp. 469–483.

[16] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process Mining Based on Regions of Languages," in *International Conference on Business Process Management (BPM 2007)*, ser. Lecture Notes in Computer Science, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714. Springer-Verlag, Berlin, 2007, pp. 375–383.

[17] J. Carmona and J. Cortadella, "Process Mining Meets Abstract Interpretation," in *ECML/PKDD 210*, ser. Lecture Notes in Artificial Intelligence, J. Balcazar, Ed., vol. 6321. Springer-Verlag, Berlin, 2010, pp. 184–199.

[18] J. Carmona, J. Cortadella, and M. Kishinevsky, "A Region-Based Algorithm for Discovering Petri Nets from Event Logs," in *Business Process Management (BPM2008)*, 2008, pp. 358–373.

[19] J. Cook and A. Wolf, "Discovering Models of Software Processes from Event-Based Data," *ACM Transactions on Software Engineering and Methodology*, vol. 7, no. 3, pp. 215–249, 1998.

[20] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, "Robust Process Discovery with Artificial Negative Events," *Journal of Machine Learning Research*, vol. 10, pp. 1305–1340, 2009.

[21] A. Medeiros, A. Weijters, and W. van der Aalst, "Genetic Process Mining: An Experimental Evaluation," *Data Mining and Knowledge Discovery*, vol. 14, no. 2, pp. 245–304, 2007.

[22] M. Sole and J. Carmona, "Process Mining from a Basis of Regions," in *Applications and Theory of Petri Nets 2010*, ser. Lecture Notes in Computer Science, J. Lilius and W. Penczek, Eds., vol. 6128. Springer-Verlag, Berlin, 2010, pp. 226–245.

[23] A. Weijters and W. van der Aalst, "Rediscovering Workflow Models from Event-Based Data using Little Thumb," *Integrated Computer-Aided Engineering*, vol. 10, no. 2, pp. 151–162, 2003.

[24] J. van der Werf, B. van Dongen, C. Hurkens, and A. Serebrenik, "Process Discovery using Integer Linear Programming," *Fundamenta Informaticae*, vol. 94, pp. 387–412, 2010.

[25] A. Rozinat and W. van der Aalst, "Conformance Checking of Processes Based on Monitoring Real Behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, 2008.

[26] W. 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*, vol. 2, no. 2, pp. 182–192, 2012.

[27] A. Adriansyah, B. van Dongen, and W. van der Aalst, "Conformance Checking using Cost-Based Fitness Analysis," in *IEEE International Enterprise Computing Conference (EDOC 2011)*, C. Chi and P. Johnson, Eds. IEEE Computer Society, 2011, pp. 55–64.

[28] ——, "Towards Robust Conformance Checking," in *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010)*, ser. Lecture Notes in Business Information Process-

ing, M. Muehlen and J. Su, Eds., vol. 66. Springer-Verlag, Berlin, 2011, pp. 122–133.

[29] A. Adriansyah, N. Sidorova, and B. van Dongen, "Cost-based Fitness in Conformance Checking," in *International Conference on Application of Concurrency to System Design (ACSD 2011)*. IEEE Computer Society, 2011, pp. 57–66.

[30] T. Calders, C. Guenther, M. Pechenizkiy, and A. Rozinat, "Using Minimum Description Length for Process Mining," in *ACM Symposium on Applied Computing (SAC 2009)*. ACM Press, 2009, pp. 1451–1455.

[31] J. Cook and A. Wolf, "Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model," *ACM Transactions on Software Engineering and Methodology*, vol. 8, no. 2, pp. 147–176, 1999.

[32] J. Munoz-Gama and J. Carmona, "A Fresh Look at Precision in Process Conformance," in *Business Process Management (BPM 2010)*, ser. Lecture Notes in Computer Science, R. Hull, J. Mendling, and S. Tai, Eds., vol. 6336. Springer-Verlag, Berlin, 2010, pp. 211–226.

[33] ——, "Enhancing Precision in Process Conformance: Stability, Confidence and Severity," in *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, N. Chawla, I. King, and A. Sperduti, Eds. Paris, France: IEEE, April 2011, pp. 184–191.

[34] J. Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A Robust F-measure for Evaluating Discovered Process Models," in *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, N. Chawla, I. King, and A. Sperduti, Eds. Paris, France: IEEE, April 2011, pp. 148–155.

[35] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, "Simplified Computation and Generalization of the Refined Process Structure Tree," in *WS-FM 2010*, ser. Lecture Notes in Computer Science, M. Bravetti and T. Bultan, Eds., vol. 6551. Springer-Verlag, Berlin, 2011, pp. 25–41.

[36] J. Vanhatalo, H. Völzer, and J. Koehler, "The Refined Process Structure Tree," *Data and Knowledge Engineering*, vol. 68, no. 9, pp. 793–818, 2009.

[37] W. van der Aalst, "Passages in Graphs," BPM Center Report BPM-12-19, BPMcenter.org, 2012.

[38] C. Günther and W. van der Aalst, "Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics," in *International Conference on Business Process Management (BPM 2007)*, ser. Lecture Notes in Computer Science, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714. Springer-Verlag, Berlin, 2007, pp. 328–343.

[39] U. Feige, M. Hajiaghayi, and J. Lee, "Improved Approximation Algorithms for Minimum-Weight Vertex Separators," in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. ACM, New York, 2005, pp. 563–572.

[40] G. Karpis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[41] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *The Bell Systems Technical Journal*, vol. 49, no. 2, 1970.

[42] M. Kim and K. Candan, "SBV-Cut: Vertex-Cut Based Graph Partitioning Using Structural Balance Vertices," *Data and Knowledge Engineering*, vol. 72, pp. 285–303, 2012.

[43] H. Dhama, "Quantitative Models of Cohesion and Coupling in Software," *Journal of Systems and Software*, vol. 29, no. 1, pp. 65–74, 1995.

[44] IEEE Task Force on Process Mining, "Process Mining Manifesto," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, F. Daniel, K. Barkaoui, and S. Dustdar, Eds., vol. 99. Springer-Verlag, Berlin, 2012, pp. 169–194.

[45] P. Darondeau, "Unbounded Petri Net Synthesis," in *Lectures on Concurrency and Petri Nets*, ser. Lecture Notes in Computer Science, J. Desel, W. Reisig, and G. Rozenberg, Eds., vol. 3098. Springer-Verlag, Berlin, 2004, pp. 413–438.

[46] W. van der Aalst, K. van Hee, J. van der Werf, and M. Verdonk, "Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor," *IEEE Computer*, vol. 43, no. 3, pp. 90–93, 2010.

[47] H. Reguieg, F. Toumani, H. M. Nezhad, and B. Benatallah, "Using MapReduce to Scale Events Correlation Discovery for Business Processes Mining," in *International Conference on Business Process Management (BPM 2012)*, ser. Lecture Notes in Computer Science, A. Barros, A. Gal, and E. Kindler, Eds., vol. 7481. Springer-Verlag, Berlin, 2012, pp. 279–284.

[48] C. Bratosin, N. Sidorova, and W. van der Aalst, "Distributed Genetic Process Mining," in *IEEE World Congress on Computational Intelligence (WCCI 2010)*, H. Ishibuchi, Ed. Barcelona, Spain: IEEE, July 2010, pp. 1951–1958.

[49] J. Carmona, J. Cortadella, and M. Kishinevsky, "Divide-and-Conquer Strategies for Process Mining," in *Business Process Management (BPM 2009)*, ser. Lecture Notes in Computer Science, U. Dayal, J. Eder, J. Koehler, and H. Reijers, Eds., vol. 5701. Springer-Verlag, Berlin, 2009, pp. 327–343.

[50] D. Fahland, M. de Leoni, B. van Dongen, and W. van der Aalst, "Conformance Checking of Interacting Processes with Overlapping Instances," in *Business Process Management (BPM 2011)*, ser. Lecture Notes in Computer Science, S. Rinderle, F. Toumani, and K. Wolf, Eds., vol. 6896. Springer-Verlag, Berlin, 2011, pp. 345–361.

[51] M. 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)*. British Computer Society, 2007, pp. 15–26.

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

[53] 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*, vol. 34, no. 6, pp. 2451–2465, 2004.

[54] R. Agrawal and J. Shafer, "Parallel Mining of Association Rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 962–969, 1996.

[55] H. Verbeek and W. van der Aalst, "An Experimental Evaluation of Passage-Based Process Discovery," in *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2012)*, ser. Lecture Notes in Business Information Processing, M. Rosa and P. Soffer, Eds., vol. 132. Springer-Verlag, Berlin, 2013, pp. 205–210.