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 points for process mining—a new 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.

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 an 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
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 use partitioning to show where model and event log disagree. This is valid. Section VI discusses possible strategies to see the related work described in Section VII). In this paper, we contrast existing papers [7], [8], [9], [10], [11], we abstract the conformance checking 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: \( b_1 = [1] \), \( b_2 = [x, x, y] \), \( b_3 = [x, y, z] \), \( b_4 = [x, x, y, x, y, z] \), \( b_5 = [x^2, y^3, z] \) 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., \( \in, \subseteq, \cup \) between sets, \( \in, \subseteq, \cup \) between multisets.

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.

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

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 \( b \in \mathcal{B}(X^*) \).

\[ \sigma \in s \Rightarrow \sigma|_Y \in s|_Y, \]
\[ \sigma \in b \Rightarrow \sigma|_Y \in b|_Y, \]
\[ \sigma \in s|_Y \Leftrightarrow \exists \sigma' \in s \sigma = \sigma'|_Y, \]
\[ \sigma \in b|_Y \Leftrightarrow \exists \sigma' \in b \sigma = \sigma'|_Y, \]
\[ \sigma \in X^* \Rightarrow \exists \sigma \in X^* \sigma = \sigma|_Y \] and
\[ \sigma \in X^* \Rightarrow \exists \sigma \in X^* \sigma = \sigma|_Y \] for any \( \sigma_1 \in X^* \) and \( \sigma_2 \in X^* \).

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

We assume that an activity is identified by attributes relevant for learning, i.e., irrelevant attributes are removed and attribute values may be coarsened. $|A|$ 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 $\sigma \in U$ 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., $M \subseteq \mathcal{P}(U)$ and $M \neq \emptyset$.

$A_M = \bigcup_{\sigma \in A} \{a \in \sigma\}$ 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 $L \in \mathcal{B}(U)$ 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.

$L = \{(a, b, d, e, g)^5, (a, c, d, e, h)^4, (a, h, d, e, f, c, d, e, g)\}$ 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 $disc \in \mathcal{B}(U) \rightarrow \mathcal{P}(U)$ is a function that produces a process model $disc(L) \in \mathcal{P}(U)$ for any event log $L \in \mathcal{B}(U)$.

Given an event log $L = \{(a, c)^5, (a, b, c)^4, (a, b, b, b, b, c)\}$, 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: $disc(L) = \{(a, c), (a, b, c), (a, b, b, c), \ldots\}$.

An example of a discovery algorithm is the $\alpha$ 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], [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 $L = [(a, b, d, e, g)^5, (a, c, d, e, h)^4, (a, b, d, e, f, c, d, e, g)]$ is perfectly fitting model $M$ described in Figure 2.

Definition 6 (Conformance Checking Technique): A conformance checking technique $check \in (\mathcal{B}(U) \times \mathcal{P}(U)) \rightarrow D$ is a function that computes conformance diagnostics $check(L, M) \in D$ (e.g., fitness or precision metrics) given an event log $L \in \mathcal{B}(U)$ and process model $M \in \mathcal{P}(U)$. $D$ 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 $L \in \mathcal{B}(U)$ and process model $M \in \mathcal{P}(U)$, we define the following functions:
**PROCEEDINGS OF THE FEDCSIS. KRAK ´OW , 2013**

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 \( \text{check}_{pft}(L, M) \). 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 event 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 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):** Let \( A = \{A_1, A_2, \ldots, A_n\} \) be an activity partitioning of a set \( A \) if \( A = \bigcup_{1 \leq i \leq n} A_i \). The activity sets may overlap. \( A_i = A_i \cap \bigcup_{j \neq i} A_j \) are all activities that \( A_i \) shares with other activity sets. \( A_i' = A_i \setminus A_i \).
are the internal activities of $A_i, \overline{A_i} = A \setminus (A_i^\ell) = \bigcup_{j \neq i} A_j$ are the non-internal activities of $A_i$.

Note that $\overline{A_i} \cap A_i = \emptyset$. 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 properties $e$ 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:

- $\sigma|_A$ a subtrace of $\sigma$.
- $M|_A \in \mathcal{P}(\mathcal{U})$ is a submodel of $M$.
- $L|_A \in \mathcal{B}(\mathcal{U})$ 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.

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 hold in general since the event log can change. 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 the following functions.

- $\text{fit}^P(L, M) = [\sigma \in L : \forall 1 \leq i \leq n \sigma|_{A_i} \in \langle M|_{A_i} \rangle]$.
- $\text{check}_{pfit}^P(L, M) = \frac{\text{fit}^P(L, M)}{|L|}$.
- $\text{check}_{bpf}^P(L, M) = (\text{fit}^P(L, M) = L)$.

**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 $L \in \mathcal{B}(\mathcal{U})$:

- $\text{fit}^P(L, M) \leq \text{fit}^P(L, M)$.
- $\text{check}_{pfit}(L, M) \leq \text{check}_{pfit}(L, M)$.
- $\text{check}_{bpf}(L, M) \Rightarrow \text{check}_{bpf}(L, M)$.

**Proof:** Let $\sigma \in \text{fit}^P(L, M)$, i.e., $\sigma \in L$ and $\sigma \in M$. Using Lemma 1 we can deduce that $\forall 1 \leq i \leq n \sigma|_{A_i} \in \langle M|_{A_i} \rangle$. Hence, $\sigma \in \text{fit}^P(L, M)$. This proves that $\text{fit}^P(L, M) \leq \text{fit}^P(L, M)$. 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 $L = \{(a, b, d, e, g), (a, c, d, e, h)\}$. $M|_{A_1} = \{a, b, c, d, e, f\}$, $M|_{A_2} = \{e, f, g, h\}$, and $L|_{A_1} = \{(a, b, d, e, g), (a, c, d, e, h)\}$ and $L|_{A_2} = \{(e, f, g, h)\}$.

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

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

Activity partitioning $P = \{A_1, A_2\}$ with $A_1 = \{a, b, c\}$ and $A_2 = \{d, e, f\}$ is valid for $M_2$ but not for $M_1$. Note that $(a, c, e)[A_2] = (a, c) \in M_1[A_1]$ and $(a, c, e)[A_2] = (c, c) \in M_1[A_2]$ but $(a, c, e) \notin M_1$.

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

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

- $\text{fit}(L, M) = \text{fit}^P(L, M)$,
- $\text{check}_{P^{\text{st}}}(L, M) = \text{check}^P_{P^{\text{st}}}(L, M)$, and
- $\text{check}_{P^{\text{st}}}(L, M) = \text{check}^P_{P^{\text{st}}}(L, M)$.

Proof: $\text{fit}(L, M) = \{\sigma \in L | \sigma \in M \} = \{\sigma \in L | \sigma[A_i] \in M[A_i] \forall 1 \leq i \leq n.\sigma[A_i] \in M[A_i] \}$. 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 $L \in \mathcal{B}(U)$ be an event log and $\mathcal{P} = \{A_1, A_2, \ldots , A_n\}$ an activity partitioning of $A_M$. Let $\text{disc} \in \mathcal{B}(U) \rightarrow \mathcal{P}(U)$ be a discovery algorithm used to obtain the submodels $M_i = \text{disc}_i(L[A_i])$ with $i \in \{1, \ldots , n\}$.

$M = \{\sigma \in U | \forall 1 \leq i \leq n. \sigma[A_i] \in M[A_i]\}$ is the overall model constructed by merging the discovered submodels.

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

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 $M \in \mathcal{P}(U)$ be a process model with activities $A_M$ and $\mathcal{P} = \{A_1, A_2, \ldots , A_n\}$ an activity partitioning of the set $A_M$. $P$ is weakly valid if and only if $M' = \{\sigma \in A_M | \exists 1 \leq i \leq n. \sigma[A_i] \in M'[A_i] \land \sigma[A_j] \in M'[A_j]\}$.

Clearly, any valid activity partitioning is also weakly valid. If $\sigma[A_i] \in M'[A_i]$ and $\sigma[A_j] \in M'[A_j]$, then we can apply Lemma 1 to show that $\sigma[A_i] \in M'[A_i]$ for $j \neq i$. The reverse does not hold. Consider for example $M = \{(a, b, d, e, g), (a, c, d, f, g)\}$ and $\mathcal{P} = \{A_1, A_2, A_3, A_4\}$ 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 $(a, c, d, e, g)$ and $(a, b, d, f, g)$ 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 configural 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 configural 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 configural elements. For conformance checking we need to find an activity partitioning $P$ and a conformance checking technique
check ∈ (B(U) × P(U)) → 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 \( dsc \in B(U) \rightarrow P(U) \) (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 \( A : P_{\text{one}} = \{\{a\} | a \in A\} \) (just one activity set containing all activities) and \( P_{\text{all}} = \{\{a\} | \) (one activity set per activity). Both are not very useful. \( P_{\text{one}} \) does not decompose the problem, i.e., there is still one big task. \( P_{\text{all}} \) decomposes the set of activities into singleton activity sets. \( P_{\text{all}} \) considers all activities in isolation, hence conformance checking and discovery are only considering frequencies of activities and not their order. For conformance checking \( P_{\text{all}} \) is typically not valid and decomposed discovery using \( P_{\text{all}} \) 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 \(\alpha\) 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]. Coherence 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 \( A_i = A_i \cap \bigcup_{j \neq i} A_j \), but not through \( A_i \setminus A_i \). 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 \( M \in P(U) \) 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 \( M = \{\sigma \in A_M^* | \forall 1 \leq i \leq n \sigma | A_i \in M | A_i\} \). We need to prove: \( \{\sigma \in A_M^* | \sigma | A_{1 \cup 2} \in M | A_{1 \cup 2} \} \wedge \forall 1 \leq i \leq n \sigma | A_i \in M | A_i\} = M \). \( \sigma | A_{1 \cup 2} \in M | A_{1 \cup 2} \) implies that \( \sigma | A_i \in M | A_i \) and \( \sigma | A_2 \in M | A_2 \) (Lemma 1). Hence, \( \{\sigma \in A_M^* | \sigma | A_{1 \cup 2} \in M | A_{1 \cup 2} \} \wedge \forall 1 \leq i \leq n \sigma | A_i \in M | A_i\} \subseteq \{\sigma \in A_M^* | \forall 1 \leq i \leq n \sigma | A_i \in M | A_i\} = M \). Moreover, for any \( \sigma \in M \), \( \sigma | A_{1 \cup 2} \in M | A_{1 \cup 2} \) \wedge \forall 1 \leq i \leq n \sigma | A_i \in M | A_i \) trivially holds. The observation that \( M \subseteq \{\sigma \in A_M^* | \sigma | A_{1 \cup 2} \in M | A_{1 \cup 2} \wedge \forall 1 \leq i \leq n \sigma | A_i \in M | A_i\} \subseteq M \) 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 proclcts [50]. Proclcts 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 proclcts). In [50] it is shown that conformance checking can be done per proclct by projecting the event log onto a single proclct while considering interface transitions in the surrounding proclcts.

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
