How Much Event Data Is Enough? A Statistical Framework for Process Discovery

Martin Bauer[1], Arik Senderovich[2], Avigdor Gal[2], Lars Grunske[1], and Matthias Weidlich[1(✉)]

[1] Humboldt-Universität zu Berlin, Berlin, Germany
{bauermax,grunske,weidlima}@informatik.hu-berlin.de
[2] Technion – Israel Institute of Technology, Haifa, Israel
sariks@technion.ac.il, avigal@ie.technion.ac.il

Abstract. With the increasing availability of business process related event logs, the scalability of techniques that discover a process model from such logs becomes a performance bottleneck. In particular, exploratory analysis that investigates manifold parameter settings of discovery algorithms, potentially using a software-as-a-service tool, relies on fast response times. However, common approaches for process model discovery always parse and analyse all available event data, whereas a small fraction of a log could have already led to a high-quality model. In this paper, we therefore present a framework for process discovery that relies on statistical pre-processing of an event log and significantly reduce its size by means of sampling. It thereby reduces the runtime and memory footprint of process discovery algorithms, while providing guarantees on the introduced sampling error. Experiments with two public real-world event logs reveal that our approach speeds up state-of-the-art discovery algorithms by a factor of up to 20 .

Keywords: Process discovery · Log pre-processing · Log sampling

1 Introduction

Process mining emerged as a discipline that targets analysis of business processes based on event logs [1]. Such logs are recorded by information systems during the conduct of a process, such that each event denotes the timestamped execution of a business activity for a particular instance of the process. One of the essential tasks in process mining is process discovery—the construction of a model of the process from an event log.

In recent years, a plethora of algorithms have been proposed for process discovery [2]. These algorithms differ along various dimensions, such as the imposed assumptions on the notion of an event log (e.g., atomic events vs. interval events [3,4]), the applied representational bias (e.g., constructing transition

J. Krogstie and H. A. Reijers (Eds.): CAiSE 2018, LNCS 10816, pp. 239–256, 2018.
https://doi.org/10.1007/978-3-319-91563-0_15

systems [5], Petri-nets [6], or BPMN models [7]), the handling of log incompleteness and noise (e.g., heuristics to balance over-fitting and under-fitting [8] or filter noise [9]), and the formal guarantees given for the resulting model (e.g., ensuring deadlock-freedom [10]).

Regardless of the specific algorithm chosen, efficiency becomes an increasingly important requirement for process discovery, due to several reasons. First, the increasing pervasiveness of data sensing and logging mechanisms led to a broad availability of large event logs in business contexts, reaching up to billions of events [11]. Hence, the manifesto of the IEEE Task Force on Process Mining concludes that *'additional efforts are needed to improve performance and scalability'* [12]. Second, discovery algorithms typically have multiple parameters that need to be configured. Since these parameters have a large impact on the resulting model [13], discovery becomes an exploratory analysis rather than a one-off procedure. Third, companies started to offer software-as-a-service solutions for process mining, see Signavio[1] or Lana-Labs[2], which are trouble-some to use if analysis needs to be preceded by an upload of very large logs.

Acknowledging the need for more efficient process discovery, divide-and-conquer strategies enable the decomposition of a discovery problem into smaller ones [14,15]. Also, models for distributed computation may be exploited to increase efficiency [16,17]. All these techniques fundamentally consider *all* available data, conducting a *complete* scan of the event log. Yet, in practice, we observe that a small fraction of a log may already enable discovery of a high-quality model. This holds in particular as logs are not trustworthy and discovery algorithms apply heuristics to cope with incomplete information and noise in the data. Hence, small deviations between models discovered from a complete log and a partial log, respectively, may be considered a result of the inherent uncertainty of the process discovery setting. Following this line, the challenge then becomes to answer the following question: *How to systematically determine how much of data of an event log shall be considered when discovering a process model?*

In this paper, we set out to answer the above question with a framework for statistical pre-processing of an event log. More specifically, this paper contributions are as follows:

(1) A statistical framework for process discovery that is based on an incremental pre-processing of an event log. For each trace, we assess whether it yields new information for the process at hand. By framing this procedure as a series of binomial experiments, we establish well-defined bounds on the error introduced by considering only a sample of the log.
(2) We instantiate this framework for control-flow discovery. That is, we show how discovery algorithms that exploit the directly-follows graph of a log such as the Inductive Miner [9,10] can benefit from this framework.
(3) We handle the enrichment of discovered process models with information on execution times, e.g., to estimate the cycle time of a process. That is, we elaborate on different variants to incorporate the respective information in our framework.

[1] https://www.signavio.com/.
[2] https://lana-labs.com/.

In the remainder, we first introduce preliminary notions and notations (Sect. 2), before presenting our statistical framework for process discovery and its instantiation (Sect. 3). An evaluation of our techniques with two real-world datasets reveals that they indeed reduce the runtime and memory footprint of process discovery significantly (Sect. 4). We close with a discussion of related work (Sect. 5) and conclusions (Sect. 6).

2 Preliminaries

This section defines event logs, as well as directly-follows graphs and process trees as two process modelling formalisms, and elaborates on their discovery from event logs.

Event Logs. Following [4,18], we adopt an interval-based notion of an event log. Let $\mathcal{A}$ be a set of activity identifiers (activities, for short) and $\mathcal{T}$ be a totally ordered time domain, e.g., given by the positive real numbers. Then, an event e recorded by an information system is characterised by at least the following information: an activity $e.a \in \mathcal{A}$; a start time $e.s \in \mathcal{T}$; and a completion time $e.c \in \mathcal{T}$ with $e.s \leq e.c$. We say that an event e is instantaneous, if $e.s = e.c$. Furthermore, let $\mathcal{E}$ denote the universe of all possible events. A single execution of a process, called a trace, is modelled as a set of events, $\xi \subseteq \mathcal{E}$. However, no event can occur in more than one trace.

Trace	Activity	Start	Complete
1	R: Receive Claim	9:05	9:05
1	F: Fetch Previous Claim	9:05	9:10
1	P: Plausibility Check	9:08	9:20
1	U: Update Claim Status	9:20	9:22
1	U: Update Claim Status	9:46	9:50
2	R: Receive Claim	9:51	9:51
3	R: Receive Claim	9:55	9:55
2	P: Plausibility Check	9:57	10:03
2	F: Fetch Previous Claim	10:01	10:06
3	F: Fetch Previous Claim	10:12	10:17
3	P: Plausibility Check	10:13	10:22
3	U: Update Claim Status	10:22	10:25

Fig. 1. Example event log.

An event log is modelled as a set of traces, $L \subseteq 2^{\mathcal{E}}$. To illustrate this notion, Fig. 1 depicts the log of an example claim handling process. It contains three traces, each comprising events that denote the execution of particular activities. Each event has a start and completion time. As a short-hand, we define $A_L = \{a' \in \mathcal{A} \mid \exists\, \xi \in L, e \in \xi : e.a = a'\}$ as the set of activities referenced by events in the log. Using the activity identifiers in Fig. 1, we have $A_L = \{R, F, P, U\}$ for the example.

Process Models. A rather simple formalism to capture the behaviour of a process, which is widely used in process mining tools in practice, is a directly-follows graph (DFG). A DFG is a directed graph $G = (V, E)$. Vertices V denote the activities of a process. Directed edges $E \subseteq V \times V$ model that the target activity can be executed immediately after the source activity in a process instance. A plain DFG may be extended by marking some vertices $V_<, V_> \subseteq V$ as start and completion vertices. Then, a path in the DFG from a start to a completion vertex represents a

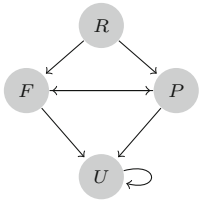


Fig. 2. Example DFG.

possible execution sequence of the process. An example DFG is given in Fig. 2. With $V_< = \{R\}$ and $V_> = \{U, F\}$, it defines that $\langle R, F, U\rangle$ would such an execution sequence.

Based on DFGs, richer types of models that feature explicit concepts, e.g., for repetitive behaviour and concurrent execution can be constructed. In this work, we consider the formalism of a process tree [10,18]. In a process tree, leaf nodes denote activities or a specific silent activity τ. Non-leaf nodes are control-flow operators, such as sequence ($\rightarrow$), exclusive choice ($\times$), concurrency ($\wedge$), interleaving ($||$) and structured loops ($\circlearrowleft$). Given a process tree, a set of execution sequences of activities is constructed recursively. For a leaf node, this set contains a single execution sequence, consisting of the respective activity (the sequence is empty for the silent activity). For non-leaf nodes, semantics is induced by a function that joins the execution sequences of the subtrees of the node. The process tree given in Fig. 3, e.g., defines that $\langle R, P, F, U, U\rangle$ would be an execution sequence of the process.

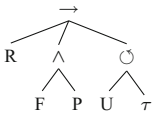


Fig. 3. Process tree.

Models such as process trees may also be annotated with additional information, such as times or costs of activity execution. Adopting the model from [4], for instance, each activity is assigned a duration in terms of a cumulative distribution function (CDF). Semantics of such a timed process tree are no longer given in terms of execution sequences of activities, but in terms of events, which, as defined above, associate activity executions with start and completion times. To this end, an execution sequence of the untimed tree is enriched by constructing the start time for each activity execution based on the completion time of the previous activity, while the completion time is then determined by drawing a duration from the respective CDF.

Process Discovery. Process discovery constructs a process model from an event log. With $\mathcal{L}$ and $\mathcal{M}$ as the universe of event logs and process models, respectively, a discovery algorithm can thus be seen as a function $\rho : \mathcal{L} \rightarrow \mathcal{M}$.

As mentioned earlier, a large number of specific algorithms have been proposed in recent years [1,2]. Referring to the above formalisms for process models, a DFG may trivially be constructed from a log L, once a total order $\succ_\xi$ has been defined for the events of each trace $\xi \in L$. For an interval-based log, this order may be derived from either the start times or the completion times (breaking ties, if needed). Then, a DFG is discovered as $(A_L, \succ)$ with $\succ = \bigcup_{\xi \in L} \succ_\xi$. This simple construction of a DFG may be adapted through frequency-based techniques for noise filtering [5]. The DFG in Fig. 2 is obtained from the log in Fig. 1, when ordering events by their completion time.

Process trees may be discovered by various variants of the Inductive Miner. In its basic version, this algorithm relies on the DFG and iteratively identifies cuts, specific sets of vertices, in the graph to build a process tree [10]. The tree in Fig. 3 would be discovered for the log in Fig. 1, using the DFG in Fig. 2. Variants of the miner target robustness against noise [9] or exploit start and completion times to distinguish interleaved from concurrent execution [18] and detect delays between activities [4].

3 A Statistical Framework for Process Discovery

This section introduces our approach to statistical pre-processing of an event log. By reducing a log's volume through sampling of traces until no new information is encountered, the efficiency of process discovery algorithms is improved. Below, we first give an overview of the general framework (Sect. 3.1), before we instantiate it for control-flow discovery (Sect. 3.2) and extraction of performance information (Sect. 3.3).

3.1 Statistical Pre-processing of Event Logs

The general idea of our approach is to avoid a full scan of an event log during process discovery. This is to cope with the phenomena of traces that contain highly redundant information. Specifically, even though each event in a trace constitutes original information, process discovery typically works only on abstractions of events and traces. As a consequence, many events and traces are considered to be equivalent or highly similar by a discovery algorithm. Hence, it is reasonable to assume that process discovery based on a representative subset of traces (and thus events) of a log can be expected to yield a highly similar result compared to the model obtained by processing a complete log. Based on this assumption, we propose the following statistical approach to reason on the 'discovery sufficiency' of a given subset of traces.

A Statistical View on Log Sampling. When parsing a log trace-by-trace, some traces may turn out to be similar to those already encountered. As motivated above, this similarity is assessed in terms of the information used by a discovery algorithm. On a conceptual level, this is captured by a trace abstraction function $\psi : 2^{\mathcal{E}} \to \mathcal{X}$ that extracts from a trace, information of some domain $\mathcal{X}$ that is relevant for discovery. This information may, for example, relate to the occurrence of activities, their ordering dependencies, or quantitative information, and needs to be instantiated for a specific discovery algorithm.

Based on the trace abstraction function, one can define a random Boolean predicate $\gamma(L', \xi)$ that captures whether a trace $\xi \in 2^{\mathcal{E}}$ provides new information with respect to an event log $L' \subseteq 2^{\mathcal{E}}$. Here, in a strict sense, trace ξ provides novel information solely if its abstraction is not part of those jointly derived for all traces in L':

$$\gamma(L', \xi) \Leftrightarrow \psi(\xi) \notin \bigcup_{\xi' \in L'} \psi(\xi'). \tag{1}$$

In some cases, however, the abstraction function of a trace yields information of rather fine granularity, e.g., numerical values that represent the time or cost of activity execution. If so, it is reasonable to also consider a relaxed notion of new information. Assuming that the distance between the abstract information provided by traces can be quantified by a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ and given a relaxation parameter $\epsilon \in \mathbb{R}_0^+$, we therefore also consider the following predicate:

$$\gamma^{\epsilon}(L', \xi) \Leftrightarrow d\left(\psi(\xi), \bigcup_{\xi' \in L'} \psi(\xi')\right) > \epsilon. \tag{2}$$

In other words, if the observed abstraction $\psi(\xi')$ is ϵ far from L', then $\psi(\xi')$ adds new information beyond L'.

Next, we define the *discovery sufficiency* property for an event log $L' \subseteq L$, with respect to an abstraction ψ, a relaxation parameter ϵ, and some probability measure δ.

Definition 1 (Discovery Sufficiency). *An event log $L' \subseteq L$ is called (δ, ϵ, ψ)-discovery sufficient, if for a newly sampled trace $\xi : \xi \in (L \setminus L')$, it holds that:*

$$p_{\gamma}(L', \xi) = P(\gamma(L', \xi) = 1) < \delta, \tag{3}$$

with P being a standard probability measure.

Essentially, discovery sufficiency of L' requires that the chance of a newly sampled trace to add new information beyond L' is bounded by δ. Our goal is to come up with a (δ, ϵ, ψ)-discovery sufficient log L' based on the original log L. When it is clear from the context, we shall use discovery sufficiency without its parameters.

We now formulate a statistical hypothesis testing procedure to assess when *enough* information has been extracted from the log, thereby giving a criterion to terminate the construction of L'. Then, we shall present an algorithm that is based on the procedure.

Given $L' \subseteq L$, we would like to test whether it is discovery sufficient. We assume under the null hypothesis that L' is sufficient; or not, under the alternative hypothesis. Next, we consider an independent and identically distributed (i.id.) sample of N traces $\xi_1, \ldots, \xi_N$ from L, such that $\xi_i \neq L'$. We denote $p = p_i = p_{\gamma}(L', \xi_i)$ the probability that ξ_i will yield new information, i.e., the probability that $\gamma(L', \xi_i) = 1$. By the i.id. assumption, the probability p_i to have new information in ξ_i is the same for all ξ_i. Hence, under the null hypothesis, it holds $p = 0$; under the alternative, we hypothesise $p > 0$. Further, let k be the number of traces of the N samples that bring new information. Under our assumptions, k is binomially distributed, with parameters N and $1 - p$.

To assure that L' is discovery sufficient, we wish to bound the probability $p_{\gamma}(L', \xi)$ by δ for new ξ. In statistical terms, we wish to bound the probability that the null hypothesis is not rejected, given that the alternative is true for L'. Furthermore, we want to provide a statistically significant answer, with significance level α. To ensure α, and bound $p_{\gamma}(L', \xi)$ with δ, we must select an appropriate sample size N_{min}, which is referred to as *minimum sample size* [19]. Using the normal approximation to the binomial distribution, the minimal sample size is given by,

$$N_{min} \geq \frac{1}{2\delta}\left(-2\delta^2 + z^2 + \sqrt{z}\right), \tag{4}$$

with z corresponding to the realisation of a standardised normal random variable for $1 - \alpha$ (one-sided hypothesis test). For $\alpha = 0.01$ and $\delta = 0.05$, for example, we

obtain $N_{min} \geq 128$. After seeing 128 traces without new information, sampling can be stopped knowing with 0.99 confidence that the probability of finding new information in the remaining log is less than 0.05. We apply this idea in what follows.

Algorithm 1: Statistical Framework for Process Discovery

input : L, an event log,
α, a significance value,
δ, a similarity value,
γ, a predicate that holds true, if a trace provides new information,
ρ, a discovery algorithm.
output : $\hat{m}$, a process model.

1 $L', \hat{L} \leftarrow \emptyset$; /* The sampled logs, overall and for current experiment */
2 $i \leftarrow 0$; /* The number of current iterations without new information */
/* Determine the number of failed trails to be observed, based on Eq. 4 */
3 $N \leftarrow 1/(2\delta)\left(-2\delta^2 + z^2 + \sqrt{z}\right)$ with z of $(1-\alpha)$;
4 **repeat** /* Repeat until N traces without new information have been seen */
5 $\xi \leftarrow select(L \setminus \hat{L})$; /* Sample a single trace */
6 **if** $\gamma(\xi, L')$ **then** /* If ξ provides new information */
7 $i \leftarrow 0$; /* Reset the counter */
8 $\hat{L} \leftarrow \emptyset$; /* Reset sampled log for current experiment */
9 **else**
10 $i \leftarrow i + 1$; /* Increment the counter */
11 $\hat{L} \leftarrow \hat{L} \cup \{\xi\}$; /* Add trace to sampled log for current experiment */
12 **end**
13 $L' \leftarrow L' \cup \{\xi\}$; /* Add trace to overall sampled log */
14 **until** $i \geq N \vee \hat{L} = L$;
15 **return** $\hat{m} = \rho(L')$; /* Return model discovered from overall sampled log */

Statistical Framework for Process Discovery. The above observations are exploited as formalised in Algorithm 1. This statistical framework for process discovery takes as input an event log, a significance level α, a sufficiency bound δ, a predicate to determine whether a trace provides new information, and a discovery algorithm. After initialisation, in line 3, the algorithm computes the number of trails that need to fail, i.e., the number of consecutive traces that do not provide new information, according to Eq. 4. Then, it samples traces from the log (line 4–line 14). For each trace, it determines whether new information has been obtained (line 6). If so, an iteration counter is reset. Once the counter indicates that N traces without new information have been sampled, the procedure terminates by applying the mining algorithm to the sampled log.

From a technical point of view, the framework in Algorithm 1 is independent of the chosen discovery algorithm, which is applied only once a sampled log has been obtained. However, it relies on a predicate to decide whether a trace provides new information over a set of traces. This predicate, in turn, shall be devised with respect to the properties of a trace that are exploited by the discovery algorithm. Consequently, the effectiveness of the approach is not only affected by the configuration of Algorithm 1 in terms of statistical significance

and sufficiency bound, but depends also on the definition of the respective predicate. How often it holds true is influenced by the relation of the richness of the employed trace abstraction and the variability of the event data to the overall size of the original log. With richer abstractions and higher variability, more traces will provide new information, so that the size of the sampled log approaches the size of the original log.

3.2 Instantiation for Control-Flow Discovery

Next, we turn to the instantiation of the above framework for control-flow discovery, which yields a process model comprising a set of activities along with their execution dependencies. According to the model introduced in Sect. 3.1, this requires us to define a suitable trace abstraction function. Based thereon, using predicate $\gamma(L', \xi)$ of Eq. 1, it is determined, whether a trace ξ provides new information over a log L'.

As a particular example, we consider the Inductive Miner [10] for the definition of this predicate. As detailed in Sect. 2, this miner constructs a DFG and then, deterministically, builds a process tree from this graph. Hence, a trace provides new information, if the sets of vertices or edges of the DFG change. Considering also explicit start and completion vertices (not exploited by the Inductive Miner, but useful when employing the DFG directly), we define a trace abstraction function as follows. Given a total order of events in a trace (see Sect. 2), the function yields a tuple of the contained activities, their order, and the minimal and maximal elements induced by that order:

$$\psi_{IM}(\xi) \mapsto \left(A_{\{\xi\}}, \succ_\xi, \min_{\succ_\xi}(\xi), \max_{\succ_\xi}(\xi) \right) \tag{5}$$

Lifting union of sets and containment of elements in sets to tuples of sets, the predicate $\gamma(L', \xi)$ based on the trace abstraction function ψ_{IM} holds true, if, intuitively, trace ξ shows a new activity, a new order dependency, or new information about activities starting or completing the process.

As an example, consider the information extracted by the above function from the first two traces of the log in Fig. 1. Here, $\psi_{IM}(\xi_1) \cup \psi_{IM}(\xi_2)$ yields the following information in terms of known activities, their order dependencies, and the minimal and maximal elements: $(\{R, F, P, U\}, \{(R, F), (R, P), (F, P), (P, F), (P, U), (U, U)\}, \{R\}, \{F, U\})$. We note that this information is sufficient to construct the DFG in Fig. 2. Hence, adding the third trace of the example log does not provide any new information, as $\psi_{IM}(\xi_3) = (\{R, F, P, U\}, \{(R, F), (F, P), (P, U)\}, \{R\}, \{U\})$.

When aiming at process discovery with other variants of the Inductive Miner, the above trace abstraction may also be applied, even though the respective discovery algorithms exploit further information. For instance, the Inductive Miner Infrequent [9] incorporates relative frequencies of edges in the DFG to cope with noise in event logs. On the one hand, these frequencies may be neglected during sampling from the event log (applying the above abstraction) and only be

derived from the selected sample of traces. In that case, the guarantee in terms of δ-similarity of the sampled log would not cover these frequencies, though. A different approach, thus, is to consider the relative edge frequencies in the abstraction, so that they influence the predicate to assess whether a trace contains new information. As these relative frequencies can be expected to change slightly with each sampled trace, the relaxed predicate, see Eq. 2, needs to be applied. Then, the relative edge frequencies stabilise during sampling and traces that incur only minor changes in these frequencies are not considered to provide new information.

3.3 Instantiation for Performance Discovery

We complement the above discussion with an instantiation of the framework that incorporates additional aspects of a trace, beyond execution dependencies. That is, we consider the extraction of performance details in terms of the cycle time, i.e., the time from start to completion of the process. However, further aspects of a trace, such as costs induced by a trace or the involved roles, may be integrated in a similar manner.

The cycle time of a process is commonly captured as a numerical value at a rather fine granularity. Consequently, observations of the cycle time of particular traces will show some variability—if at all, only very few traces will have the same cycle time. According to the model presented in Sect. 3.1, this suggests to rely on the relaxed predicate $\gamma^{\epsilon}(L', \xi)$, see Eq. 2. It allows for a certain deviation, bound by ϵ, of the information provided by ξ from the one known already from L', while still considering ξ as providing no new information. Without this relaxation, all traces with different cycle times would add new information, even if the absolute difference of these times is negligible. To incorporate information on cycle time and instantiate predicate $\gamma^{\epsilon}(L', \xi)$, we need to define a respective trace abstraction function and a distance function, as detailed below.

Model-Based and Activity-Based Cycle Time Approximation. The estimation of the average cycle time of a process based on a log may be approached in different ways. One solution is to compute the average over the cycle times observed for all traces, considering each trace as an independent observation. Another solution is to compute the cycle time analytically based on the average durations of activities. The latter approach builds a performance annotated process model such as the timed process tree discussed in Sect. 2, fitting a distribution for the duration of an activity based on all observations.

Either approach is realised in our framework by specific trace abstraction functions. If the cycle time is considered per trace, referred to as *model-based approximation*, the abstraction captures the time between the start of the first event and the completion time of the last event of a trace (again, $\succ_\xi$ is the total order of events in trace ξ):

$$\psi_{tCT}(\xi) \mapsto \{e_2.c - e_1.s\} \quad \text{with} \quad e_1 = \min_{\succ_\xi}(\xi) \quad \text{and} \quad e_2 = \max_{\succ_\xi}(\xi). \tag{6}$$

If the cycle time is captured on the level of activities (*activity-based approximation*), the trace abstraction function yields the observed durations per activity:

$$\psi_{aCT}(\xi) \mapsto \bigcup_{a' \in A_{\{\xi\}}} \left\{ \left(a, \operatorname*{avg}_{e \in \xi, e.a = a'} (e.c - e.s) \right) \right\} \tag{7}$$

For the first trace in the log in Fig. 1, these abstractions yield $\psi_{tCT}(\xi_1) = 45$ (minutes) for the model-based approximation, and $\psi_{aCT}(\xi_1) = \{(R,0),(F,5),(P,12),(U,3)\}$ for activity-based approximation, where $(U,3)$ stems from two events related to U in ξ_1.

Difference of Cycle Time Approximations. Using one of the above trace abstractions, instantiation of the predicate $\gamma^{\epsilon}(L',\xi)$ further requires the definition of a distance function, to assess whether the information provided by ξ deviates with less than ϵ from the information provided by L'. To this end, we compare the cycle times computed based on the measurements from L' and computed based on L' and ξ. Specifically, while adding traces to L', the cycle time computation is expected to converge, so that the difference will continuously fall below ϵ at some point.

We illustrate the definition of this difference for the trace abstraction function ψ_{tCT}. Then, given a trace ξ, the singleton set with its cycle time $\psi_{tCT}(\xi) = CT_{\xi}$, and the cycle times of all traces in the sampled log $\bigcup_{\xi \in L'} \psi_{tCT}(\xi) = CT_{L'}$, this difference is measured as the change in the average of cycle times, if trace ξ is incorporated:

$$d(CT_{\xi}, CT_{L'}) \mapsto \left| \operatorname*{avg}_{ct \in CT_{L'}} (ct) - \operatorname*{avg}_{ct \in CT_{L'} \cup CT_{\xi}} (ct) \right|. \tag{8}$$

If this difference is smaller than ϵ, the cycle time measurement provided by trace ξ is negligible and, thus, the trace is considered to not provide any new information. It is worth to mention that sequence of average cycle time measurements obtained when adding traces to L' corresponds to a Cauchy-sequence, as the sequence converges and the space of possible cycle times is a complete metric space. Hence, it can be concluded, for any chosen ϵ, that after a certain number of traces have been incorporated, adding additional traces will not increase the difference above ϵ again.

Referring to our running example, the log in Fig. 1, consider the case of having sampled the first and second trace already. Then, with $L' = \{\xi_1, \xi_2\}$, we obtain $CT_{L'} = \{45, 15\}$. Trace ξ_3 with $CT_{\xi_3} = \{30\}$ does not add new information, as the average of values in $CT_{L'}$ and the average of values in $CT_{L'} \cup CT_{\xi_3}$ are equivalent.

The definition of the difference function for the case of ψ_{aCT} uses the same principle, applied to individual activities. For each activity duration, we assess, whether the average of values observed in L' is changed by more than ϵ, if the values in ξ are incorporated.

Inspecting the third trace ξ_3 of the log in Fig. 1, again, it would not provide new information: Activity R is instantaneous and F has the same duration

(5 min) in all traces. For activities P and U, the durations in ξ_3 correspond to the averages of durations observed in traces ξ_1 and ξ_2.

Using the above functions means that the effectiveness of sampling depends on the selection of parameter ϵ and the variability of cycle times (or activity durations) in the log. A high value for ϵ means that the information provided by more traces will be considered to be negligible, leading to a smaller sampled log as the basis for discovery. We later explore this aspect in our experimental evaluation.

4 Experimental Evaluation

This section reports on an experimental evaluation of our statistical framework for process discovery. Using two real-world datasets, we instantiated the framework with the Inductive Miner Infrequent, a state-of-the-art algorithm, for the construction of timed process trees. Our results indicate that the runtime of the algorithm can be reduced by a factor of up to 20. Below, we elaborate on datasets and the experimental setup (Sect. 4.1), before turning to the actual results (Sect. 4.2).

4.1 Datasets and Experimental Setup

Datasets. We relied on two event logs that have been published as part of the Business Process Intelligence (BPI) Challenge and are, therefore, publicly available.

BPI-2012 [20] is a log of a process for loan or overdraft applications at a Dutch financial institute. It contains 262,200 events of 13,087 traces.

BPI-2014 [21] is the log of an incident management process as run by Rabobank Group ICT. For the experiments, we employed the event log of incidence activities, which comprises 343,121 events of 46,616 traces.

Discovery Algorithms. For our experiments, we adopted the Inductive Miner Infrequent (IMI) [9] as a discovery algorithm. This choice is motivated as follows: IMI discovers a process tree and, thus, supports common control-flow structures, such as exclusive choices and concurrency. Second, IMI features a frequency-based handling of noise in event logs, making it suitable for the application to real-world event logs.

In the experiments, the IMI is applied to construct a process tree using a 20% noise filtering threshold. In addition, we collect the performance details needed to quantify the cycle time of the process from the log. Hence, our framework is instantiated with a combination of two trace abstraction functions, one focusing on control-flow information (Eq. 5) and one extracting performance details. Regarding the latter, we employ the model-based cycle time approximation (Eq. 6) as a default strategy. In one experiment, however, we also compare this strategy with the activity-based approximation.

When sampling the event log, we set $\alpha = 0.01$ and $\delta = 0.99$, so that the information loss is expected to be small. Due to the consideration of performance details, we use the relaxed version of the predicate to indicate new information (Eq. 2). Hence, the relaxation parameter ϵ of this predicate becomes a controlled variable in our experiments.

Measures. We measure the effectiveness of our statistical pre-processing by the size of the sampled log, i.e., the *number of traces*, and relate it to the size of the original log.

As the effectiveness of sampling is an analytical, indirect measure for the efficiency of our approach, we also quantify its actual efficiency as implemented in ProM [22]. To this end, we measure the total *runtime* (in ms) and total *memory footprint* of the complete discovery procedure, contrasting both measures with the plain IMI implementation.

To explore a potential loss of information incurred by our approach, we also compare the quality of the models obtained with IMI and our approach. Here, the control-flow dimensions is considered by measuring the *fitness* [23] between the model and the event log, while we consider performance information in terms of the *approximated cycle time*.

For all measures, we report the mean average of 100 experimental runs, along with the 10th and 90th percentiles.

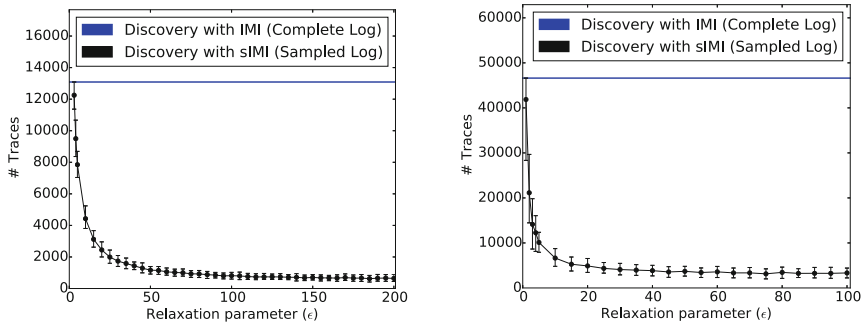
Experimental Setup. We implemented our approach based on the IMI implementation within ProM [22]. Specifically, we developed two ProM plugins, one targeting the direct application of our approach by users and one conducting the experiments reported in the remainder of this section. Both plugins are publicly available at github.[3]

Data on the efficiency of our statistical framework for process discovery has been obtained on a PC (Dual-Core, 2.5 GHz, 8 GB RAM) running Oracle Java 1.8.

4.2 Experimental Results

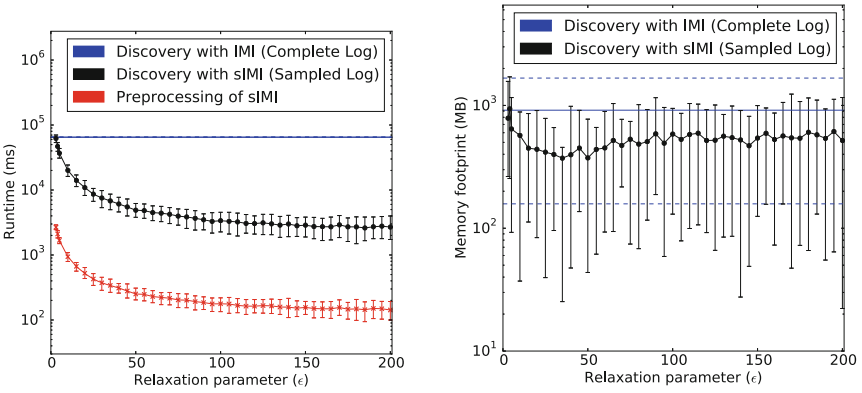
The first experiment concerned the effectiveness of the statistical pre-processing of the event log. For both datasets, Fig. 4 illustrates the number of traces that are included in the sampled log by the statistical version of IMI (denoted by sIMI), when varying the relaxation parameter ϵ. As a reference point, the flat blue line denotes the size of the complete event log as used by the plain version of the IMI. Choosing the smallest value for the relaxation parameter means that virtually every trace provides new information, so that pre-processing has no effect. For any slightly higher value, however, the sampled log is significantly smaller than the original one. Already for ϵ values of around 20 (corresponding to deviations in performance details of 20 min for a process that runs for 9 days on average), the sampled log contains only a small fraction of all traces. This provides us

[3] https://github.com/Martin-Bauer/StatisticalInductiveMinerInfrequent.



(a) BPI-2012 (b) BPI-2014

Fig. 4. Effectiveness of process discovery: # Traces vs. relaxation parameter. (Color figure online)



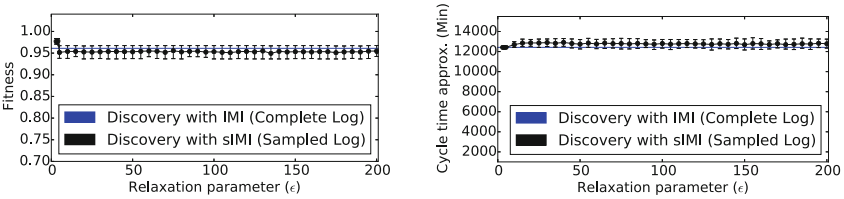
(a) Runtime (log) vs. relaxation parameter. (b) Memory (log) vs. relaxation parameter.

Fig. 5. Efficiency of process discovery for the BPI-2012 event log. (Color figure online)

with evidence that our approach can indeed reduce the volume of event data in process discovery significantly.

The drastic reduction of the size of the event log immediately improves the actual efficiency of process discovery. Figure 5 shows the runtime and memory footprint measured when running the statistical and the plain version of the IMI in ProM for the BPI-2012 dataset (BPI-2014 yields the very same trends). Runtime drops drastically when increasing the relaxation parameter (note the logarithmic scale). Compared to the runtime of the plain algorithm, we observe speed-ups by a factor of 20. Figure 5a further illustrates that the time spent in the pre-processing of the event log is relatively small, compared to the overall runtime of the discovery procedure.

In terms of memory consumption, the statistical variant of the IMI is more efficient than the plain one, see Fig. 5b. While there is a large variability in the



(a) Fitness vs. relaxation parameter. (b) Cycle time approx. vs. relaxation parameter.

Fig. 6. Quality of the resulting models in terms of fitness and cycle time approximation for the BPI-2012 event log.

measurements, we see that this variability is also observed for the implementation of the plain IMI in ProM (blue dashed lines denote the 10th and 90th percentiles). However, the statistical variant of the IMI has a consistently smaller memory footprint.

Next, Fig. 6 illustrates results on the quality of the obtained models. For the BPI-2012 dataset, the fitness measured between the discovered model and the event log is shown in Fig. 6a, also in relation to the fitness of the model mined by the IMI for the complete log. The results demonstrate that, while the exact same fitness value is not guaranteed to be obtained due to noise filtering by the IMI, the observed deviations are arguably negligible. For the BPI-2014 dataset (not visualised), the plain IMI and the statistical variant achieve a fitness value of 1.0 in all configurations and experiment runs.

Figure 6b further demonstrates, for the BPI-2012 dataset, that also the cycle times approximations obtained with the models discovered from the sampled log are relatively close to the one derived from the complete log, even for very large relaxation parameters. The same trend also materialised for the BPI-2014 dataset.

Finally, we turn to the two strategies for cycle time estimation, see Sect. 3.3. For the BPI-2012 dataset, Fig. 7 compares the model-based strategy with the activity-based one in terms of the sampling effectiveness. Both variants show the same trend. It turns out, though, that the activity-based strategy yields a smaller log for the same ϵ value. This is explained by the fact that this strategy considers performance details at a more fine-granular level. That is, deviations of size ϵ need to be observed on average for all activities, instead of the complete trace, in order to mark a trace as containing new information.

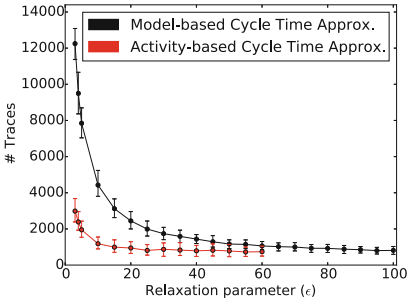


Fig. 7. Cycle time approx. strategies.

5 Related Work

Having discussed some traditional approaches to process discovery in Sect. 1, below, we review literature on incremental and online process discovery, and sampling methods.

Incremental and Online Discovery. In the Big Data era, it is essential to consider process discovery from streams of data. Mining procedural and declarative process models from event streams was first proposed in [24], and [25], respectively. These works propose novel discovery algorithms for settings where continuously arriving data cannot be fully stored in memory. Subsequently, a more generic event processing architecture was proposed by [26], thus allowing to re-use existing discovery algorithms such as the α-miner [27], and the inductive miner [9,10], on streams of events. In this paper, we apply our statistical sampling method to finite historical data, which is large enough to incur high performance costs, yet exists as a batch.

While most online methods for process discovery rely on incremental processing, the latter is relevant also in the context of event logs. For instance, it has been argued that process discovery based on region theory and Petri net synthesis shall be conducted incrementally, to avoid full re-computation of the model when a new log becomes available [28]. Incremental techniques are also motivated by performance considerations. As mentioned earlier, divide-and-conquer strategies to decompose discovery problems have been explored for mining based on directly follows graphs [14,15] as well as for region-based approaches [29]. Unlike our approach, however, these methods always consider the complete event log.

Sampling Sequences. The statistical part of our approach is based on sampling from sequence databases, i.e., datasets that contain traces, similarly to process logs. In process discovery, it has been suggested to sample Parikh vector of traces from a log to increase processing efficiency [30]. Yet, this approach does not give sample size guarantees, as detailed in our work. In the related field of specification mining, the notion of k-confidence has been introduced to capture the probability that a log is complete [33], yet without providing statistical guarantees. The latter was addressed by Busany and Maoz [31], who developed theoretical lower bounds on the number of samples required to perform inference. The approach was instantiated for the k-tails algorithm [32] and mining of Markov Chains. Our work adopts this general idea, yet focuses on trace abstractions used by common process discovery algorithms.

6 Conclusion

In this paper, we developed an approach to increase the efficiency of common process discovery algorithms. We argued that, instead of parsing and analysing all available event data, a small fraction of a log can be sufficient to discover a process model of high quality. Following this line, we presented a statistical framework for process discovery. In a pre-processing step, a sampled log is obtained,

while still providing guarantees on the introduced error in terms of the probability of missing information. We instantiated this framework for a state-of-the-art algorithm for control-flow discovery, namely the Inductive Miner Infrequent, and also showed how the framework is used when discovering performance details about a process. Our experiments with two publicly available real-world event logs revealed that the pre-processing indeed yields a drastic decrease in runtime and reduces the memory footprint of a ProM-based implementation of the respective mining algorithm. At the same time, the resulting models show only minor deviations compared to the model mined from the complete log.

Note that the presented instantiation of our framework, in terms of trace abstraction functions, is applicable for a broad range of mining algorithms. Our abstraction for control-flow information relies on directly-follows graphs (DFG), a model employed, e.g., by the family of α-algorithms [34] or mining based on activity dependencies [35]. Similarly, the abstraction for performance information captures what is incorporated by common performance-aware variants of heuristic discovery, e.g., as implemented in Disco.[4] For algorithms that extract further information from traces, the respective abstractions need to be adapted. For instance, for delay-aware process discovery as introduced in [4], the abstraction would include interval relations between activities.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer, Heidelberg (2016)
2. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: review and benchmark. CoRR abs/1705.02288 (2017)
3. Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J.: A novel approach for process mining based on event types. J. Intell. Inf. Syst. **32**(2), 163–190 (2009)
4. Senderovich, A., Weidlich, M., Gal, A.: Temporal network representation of event logs for improved performance modelling in business processes. In: Carmona, J., Engels, G., Kumar, A. (eds.) BPM 2017. LNCS, vol. 10445, pp. 3–21. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65000-5_1
5. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering workflow models from event-based data using little thumb. Integr. Comput. Aided Eng. **10**(2), 151–162 (2003)
6. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 226–245. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13675-7_14
7. Conforti, R., Dumas, M., García-Bañuelos, L., Rosa, M.L.: BPMN miner: automated discovery of BPMN process models with hierarchical structure. Inf. Syst. **56**, 284–303 (2016)
8. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Avoiding over-fitting in ILP-based process discovery. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 163–171. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_10

[4] https://fluxicon.com/disco/.

9. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_6
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
11. van der Aalst, W.M.P.: Data scientist: the engineer of the future. In: Mertins, K., Bénaben, F., Poler, R., Bourrières, J.-P. (eds.) Enterprise Interoperability VI. PIC, vol. 7, pp. 13–26. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04948-9_2
12. van der Aalst, W.M.P.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011, Part I. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19
13. Solé, M., Carmona, J.: Region-based foldings in process discovery. IEEE Trans. Knowl. Data Eng. **25**(1), 192–205 (2013)
14. van der Aalst, W.M.P., Verbeek, H.M.W.: Process discovery and conformance checking using passages. Fundam. Inform. **131**(1), 103–138 (2014)
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery with guarantees. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (eds.) CAiSE 2015. LNBIP, vol. 214, pp. 85–101. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19237-6_6
16. Wang, S., Lo, D., Jiang, L., Maoz, S., Budi, A.: Scalable Parallelization of Specification Mining Using Distributed Computing, pp. 623–648. Morgan Kaufmann, Boston (2015)
17. Evermann, J.: Scalable process discovery using map-reduce. IEEE TSC **9**(3), 469–481 (2016)
18. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 204–217. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_17
19. Fleiss, J.L., Levin, B., Paik, M.C.: Statistical Methods for Rates and Proportions. Wiley, New York (2013)
20. Van Dongen, B.: BPI Challenge 2012 (2012). https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
21. Van Dongen, B.: BPI Challenge 2014 (2014). https://doi.org/10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35
22. Verbeek, E., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Prom 6: the process mining toolkit. In: BPM Demos. CEUR, vol. 615. CEUR-WS.org (2010)
23. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisc. Rew. Data Mining Knowl. Discov. **2**(2), 182–192 (2012)
24. Burattin, A., Sperduti, A., van der Aalst, W.M.P.: Control-flow discovery from event streams. In: IEEE CEC, pp. 2420–2427. IEEE (2014)
25. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online discovery of declarative process models from event streams. IEEE TSC **8**(6), 833–846 (2015)
26. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Event stream-based process discovery using abstract representations. CoRR abs/1704.08101 (2017)

27. Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE TKDE **16**(9), 1128–1142 (2004)
28. Badouel, E., Schlachter, U.: Incremental process discovery using petri net synthesis. Fundam. Inform. **154**(1–4), 1–13 (2017)
29. Solé, M., Carmona, J.: Incremental process discovery. TOPNOC **5**, 221–242 (2012)
30. Carmona, J., Cortadella, J.: Process mining meets abstract interpretation. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010. LNCS (LNAI), vol. 6321, pp. 184–199. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15880-3_18
31. Busany, N., Maoz, S.: Behavioral log analysis with statistical guarantees. In: ICSE, pp. 877–887. ACM (2016)
32. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. IEEE Trans. Comput. **21**(6), 592–597 (1972)
33. Cohen, H., Maoz, S.: Have we seen enough traces? In: ASE. IEEE CS, pp. 93–103 (2015)
34. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. **15**(2), 145–180 (2007)
35. Song, W., Jacobsen, H., Ye, C., Ma, X.: Process discovery from dependence-complete event logs. IEEE TSC **9**(5), 714–727 (2016)