Process Discovery from Low-Level Event Logs

Bettina Fazzinga[2], Sergio Flesca[1], Filippo Furfaro[1], and Luigi Pontieri[2](✉)

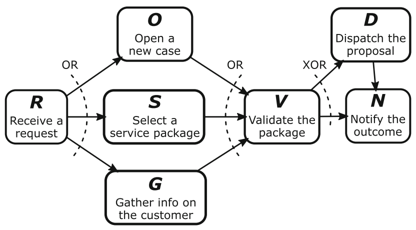
[1] DIMES, University of Calabria, Rende, Italy
{flesca,furfaro}@dimes.unical.it
[2] ICAR-CNR, Rende, Italy
{fazzinga,pontieri}@icar.cnr.it

Abstract. The discovery of a control-flow model for a process is here faced in a challenging scenario where each trace in the given log L_E encodes a sequence of low-level events without referring to the process' activities. To this end, we define a framework for inducing a process model that describes the process' behavior in terms of both activities and events, in order to effectively support the analysts (who typically would find more convenient to reason at the abstraction level of the activities than at that of low-level events). The proposed framework is based on modeling the generation of L_E with a suitable Hidden Markov Model (HMM), from which statistics on precedence relationships between the hidden activities that triggered the events reported in L_E are retrieved. These statistics are passed to the well-known *Heuristics Miner* algorithm, in order to produce a model of the process at the abstraction level of activities. The process model is eventually augmented with probabilistic information on the mapping between activities and events, encoded in the discovered HMM. The framework is formalized and experimentally validated in the case that activities are "atomic" (i.e., an activity instance triggers a unique event), and several variants and extensions (including the case of "composite" activities) are discussed.

Keywords: Process discovery · Log abstraction · Bayesian reasoning

1 Introduction

Thanks to the diffusion of automated process management and tracing platforms, many process logs (i.e., collections of execution traces) are now available. Log data can be used to analyze and improve a process, by possibly using process mining techniques [2], and in particular *process discovery* (PD) techniques. PD aims at inducing a model that compactly describes the behavior of the process in terms of ad-hoc formalisms (such as *Workflow Nets*, *Heuristics Nets*, etc.), where temporal dependencies between the actions performed during the process enactments are represented. In order to make the model usable, the activities described in it should correspond to some high-level view of the steps of the



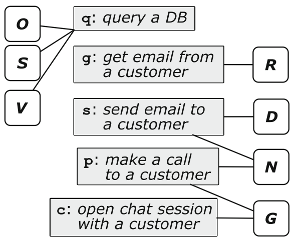

O
S
V
q: query a DB
g: get email from a customer
R
s: send email to a customer
D
p: make a call to a customer
N
c: open chat session with a customer
G

Fig. 1. (*a*) The activity flow of the service-activation process; (*b*) The mapping between high-level activities (upper-case symbols) and low-level events (lower-case symbols).

process, with which analysts are familiar. However, all PD techniques require each event reported in the log to coincide with (or be univocally mapped to) these high-level activities. Unfortunately, often this assumption does not hold in practice. As a matter of fact, in the logs of many real systems, the recorded events just represent low-level operations, with no clear reference to the business activities that were carried out through these operations, as shown in the following example.

Example 1. Consider the case of a phone company, where a *service activation* process is carried out. The process performs the flow of activities depicted in Fig. 1(*a*), and each activity requires the execution of a low-level operation. The mapping between activities and operations is *many to many*, as depicted in Fig. 1(*b*): different executions of the same activity can result in different low-level operations, and, vice versa, the same operation can be the result of performing different activities. For instance, activity G can be accomplished by performing either `c` or `p`. Analogously, the event of performing `p` can be generated by the execution of either activity G or N.

Each enactment of the process is monitored by a tracing system, which will store a log trace consisting of a sequence of low-level events, capturing each the execution of a low-level operation. Thus, a process instance consisting in the execution of sequence $R\,S\,G\,V\,N$ may be recorded in the log as the traces `g q p q s q`, or `g q c q p q`, depending which low-level event is triggered by each activity. □

It is worth noticing that a situation like the one sketched in the example above is not rare when analyzing the logs of knowledge-intensive processes and of legacy applications based on messaging and/or document management systems (e.g., SCM/PDM systems), as well as the events triggered by human activity detection systems (as in the Lifelogging analysis scenario of [22]). More generally, the relevance of a scenario where the tracing system provides low-level event logs rather than high-level activity logs is witnessed by several recent research works aiming at defining process mining tools for these kinds of logs. In particular, some "event abstraction" techniques [5,6,12,13,17,22] have been proposed for translating low-level traces into sequences of high-level activities, so that "traditional" process mining tools [2] working with activity logs can be eventually exploited on the translated logs. Unfortunately, most of these techniques require some knowledge of several aspects of the process behavior (e.g., procedural/declarative process models [5,12,17], event-oriented disambiguation rules [6], activity-annotated example traces [22]), that is not available in typical process discovery settings. Moreover, as these techniques often cannot automatically dissolve every possible ambiguity in the event-activity mapping, an expert's intervention is required to eventually bring the original log at the abstraction level of the activities.

We address the PD problem in this complex setting, and introduce a framework that, starting from a low-level event log L_E, produces a *two-level process model* W^2, where the process behavior is described at both the abstraction levels of activities and events. The proposed framework can work in the presence of a limited domain knowledge, where only the alphabets of the types of activities and events are required to be specified, along with the indication of a candidate mapping between activity types and event types (some preliminary knowledge on activity dependencies can be also exploited by the framework, even if it is not mandatory). In particular, our approach is structured according to the following steps (depicted in detail in Fig. 3):

1: learn a stochastic model capable of reproducing the generation of the input log L_E: this yields a *Hidden Markov Model* (HMM), where the observations encode the low-level events, and the underlying hidden states the activities. The HMM's structure is built on the basis of what is known about the activity/event mapping and the activity dependencies, while its parameters are learned using L_E's traces as training sequences;

2: extract statistics from the stochastic model: statistics on precedence relationships between the hidden activities that triggered the events reported in L_E are obtained by looking into the HMM resulting from step 1;

3: produce a process model in terms of activities: the statistics retrieved at step 2 are given as input to a minor modification of the well-known *Heuristics Miner* algorithm (whose standard implementation would extract these statistics from an activity-aware log), that produces a model of the process at the abstraction level of activities;

4: augment the process model with the events: the control-flow model produced at step 3 is enriched by associating each activity with a probability distribution (pdf) over the low-level events that the activity execution can trigger (see Fig. 2). This pdf is built using the emission probabilities of the learned HMM.

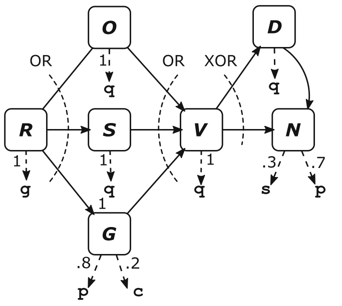


Fig. 2. A 2-level model

The framework can straightforwardly embed other PD techniques, in place of *Heuristics Miner*, and is both formalized and tested in the case that the activity/event mapping is many-to-many, but the activities are "simple" (i.e. the execution of an activity triggers exactly one event)—as also assumed in [7,21]. In a setting (like that of log abstraction works [5,6,12,17]) featuring "composite" activities, one might preprocess the log with event clustering methods, as suggested in [7]. Anyway, for the sake of generality, Sect. 6 discusses how our framework can be extended to handle directly logs of processes with composite activities.

Plan of the Paper. The next section introduces preliminary notions and notations, and recalls the fundamentals of Hidden Markov Models and *Heuristics Miner*. Section 3 introduces our framework (Sects. 3.1–3.4 regard steps 1–4 above). Section 4 reports the experimental evaluation we performed, Sect. 5 discusses some related work and Sect. 6 discusses some limitations and possible extensions of our framework.

2 Preliminaries

Logs, Traces, Processes, Activities and Events. A log is a set of *traces*. Each trace Φ describes a process instance at the abstraction level of basic *low-level events*, each generated by the execution of a *high-level activity*. That is, a process instance w is the execution of a sequence $A_1, \ldots, A_m$ of activities; in turn, the execution of each activity A_i generates an event E_j; hence, the trace Φ_w describing w is a sequence of events $E_1, \ldots, E_m$. Activities will be denoted with upper-case letters, possibly adorned with subscripts, and events specifically with letter E, possibly adorned with subscripts.

We assume that we are given a low-level event log L_E, whose traces can have different lengths (we denote as T the maximum length of a trace in L_E). We also assume that the alphabets $\mathcal{A}$ of activities and $\mathcal{E}$ of events are given, along with a *candidate mapping* $\mu : \mathcal{A} \rightarrow 2^{\mathcal{E}}$ associating each activity A with a superset $\mathcal{E}'$ of the set of events that can be the result of an execution of A. As it will be clear later, μ is used as a starting point and will be refined using a learning procedure exploiting L_E, thus resulting in a probabilistic mapping between activities and events (see Sect. 6 - point e), where possible ways to preliminarily obtain a mapping μ are discussed.

Observe that we allow different activities to result in the same event, and different events to be the result of the execution of the same activity (this corresponds to a rather general scenarios, where shared functionalities are allowed).

Hidden Markov Models (HMMs) and the Baum-Welch (BW) Algorithm. A *Hidden Markov Model* (HMM) is a statistical Markov model in which the system being modeled is a Markov process with unobserved (i.e. hidden) states, meaning that only the observations are visible. Specifically, an HMM is a tuple $\langle Q, \pi, O, \alpha, \beta \rangle$, where: $Q = q_1, ..., q_N$ is a set of states, $\pi = \pi_1, ..., \pi_N$ is an initial probability distribution over the states (where each π_i represents the probability that q_i is the initial state of the Markov chain), α is an $N \times N$ transition matrix, where each α_{ij} represents the probability of moving from state q_i to state q_j, $O = o_1, ..., o_K$ is a set of possible observations, and β is a $K \times N$ matrix, where each β_{ij} (also called *emission probability*) expresses the probability that the state q_j generates the observation o_i.

A classical problem concerning HMMs is their training, that is the learning of the parameters of an HMM given a sequence of observations $\boldsymbol{\omega}$. The standard algorithm for HMM training is the *Baum-Welch* algorithm (BW), a special case of the *Expectation-Maximization* (EM) algorithm. BW iteratively refines the transition and emission probabilities of the HMM, with the aim of maximizing the likelihood that the sequence of observations $\boldsymbol{\omega}$ is generated. Specifically, it starts with initializing the α and β matrices, and then, at each iteration, it performs the expectation and the maximization step. The expectation step uses α and β to compute, for each time point, (i) an $N \times N$ matrix ξ_t, s.t. each cell $\xi_t(i,j)$ contains the probability of being in state q_i at time t and in q_j at $t+1$, and (ii) a vector γ_t of size N, s.t. each $\gamma_t(i)$ is the probability of being in q_i at t. Then, the maximization step starts from ξ_t and γ_t and recomputes α, β, and π, whose new values are given as input of the next iteration. At the end of the iterations (whose number is an input parameter), the algorithm outputs α and β.

The main reason for adopting BW is that it is the *de facto* standard learning algorithm for HMMs, at least when the HMM is not "very large" (BW's complexity is $O(N \cdot |\omega| \cdot Q^{\max})$, where $Q^{\max}$ is the maximum state outdegree). In our scenario, the number of HMM states quadratically depends on the number of activities, that is typically not large, so that the feasibility is not jeopardized. Anyway, BW is easy to parallelize, and efficient sampling-based variants exist that can deal with settings yielding a very large number of states (as it may happen with the extension to composite activities – see Sect. 6). More details on BW and its variants can be found in [19].

Process Discovery Algorithms: Heuristics Miner. Among the many techniques for inducing a control-flow model from an activity log L (see, e.g., [1,2,4,23]), we here consider the simple and popular *Heuristics Miner* [24] algorithm, which has been widely used in real-life process mining projects, owing to its fastness and robustness to noise. The relevance of *Heuristics Miner* is also witnessed by recent efforts to extend it to deal with large logs [10,11] and online discovery settings [8]. However, as discussed in Sect. 6, the framework is orthogonal to the PD technique invoked at one of its steps. In particular, the possibility of using techniques guaranteeing the correctness of the returned model, that is not guaranteed by *Heuristics Miner*, is worth mentioning.

Heuristics Miner relies on the measures $|A > B|$ and $|A \gg B|$, indicating how many times the sub-sequences AB and ABA occur in the log, respectively. Furthermore, it considers also how many times each activity A appears at the beginning/end of a trace, mainly to decide if A is a starting/final activity. On the basis of these statistics, *Heuristics Miner* builds a process representation in the form of a "Heuristics Net".

In Sect. 6, it is discussed how further statistics that can be optionally used by *Heuristics Miner* can be inferred in our framework.

3 A Framework for Process Discovery

In brief, the problem to be addressed is that of inferring a model of the process behavior from a log L_E describing, at the abstraction level of low-level events, the actions performed during the process enactments. As an output, we intend to provide a model W^2 describing the control-flow at both the abstraction levels of events and activities, so that typical analysts (who are generally more familiar with high-level activities, rather than with low-level events) are allowed to reason through W^2 on the behavioral aspects of the process from different standpoints.

The starting point is the well-established algorithm *Heuristics Miner*. As recalled in Sect. 2, *Heuristics Miner* is a popular technique for inferring a process model from a log, in the case that the log and the desired model are at the same level of abstraction (i.e., both refer to high-level process activities). Our proposal is a framework that uses the algorithmic core of *Heuristics Miner* as an intermediate step, as shown in Fig. 3. Basically, the process for inferring a model works as follows.

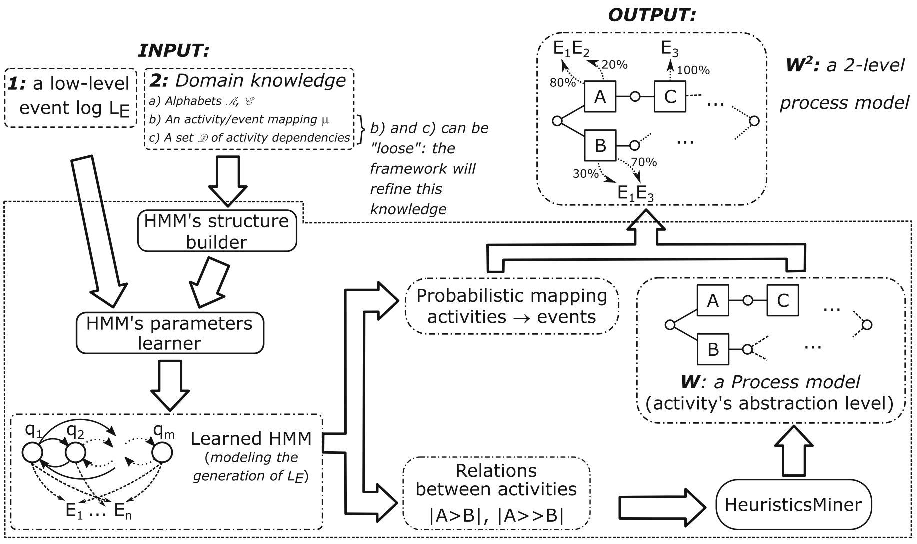

INPUT:
1: a low-level event log L_E
2: Domain knowledge
a) Alphabets $\mathcal{A}$, $\mathcal{E}$
b) An activity/event mapping μ
c) A set $\mathcal{D}$ of activity dependencies
b) and c) can be "loose": the framework will refine this knowledge
OUTPUT:
W^2: a 2-level process model
E_1E_2
E_3
80%
20%
100%
30%
70%
E_1E_3
A
B
C
HMM's structure builder
HMM's parameters learner
q_1 q_2 q_m
$E_1 \dots E_n$
Learned HMM (modeling the generation of L_E)
Probabilistic mapping activities → events
Relations between activities |A>B|, |A>>B|
HeuristicsMiner
W: a Process model (activity's abstraction level)

Fig. 3. The phases of our approach for inferring a model from a low-level event log

The input consists of a low-level event log L_E along with some domain knowledge, covering: (a) the alphabets $\mathcal{A}$, $\mathcal{E}$, and (b) the candidate activity/event mapping μ, and (c) a set $\mathcal{D}$ of dependencies between activities, such as "*A always/never precedes/is preceded by B*". Points (b) and (c) can be specified even "loosely": μ is not required to be tight (as discussed in Sect. 2), and $\mathcal{D}$ can be incomplete (in fact, in our experiments, $\mathcal{D} = \emptyset$). This aspect will be further addressed in Sect. 6 (point ***a***), after having made clearer the details of the framework. The knowledge encoded by (a), (b), (c) is used to define the structure of an HMM, that is in turn given as input, along with the log L_E, to the BW algorithm. This way, an HMM reproducing the generation of the sequences of events in L_E is obtained. The structure of the HMM given to BW is specifically designed so that it can encode different forms of dependencies between activities: this enables the result of BW to be used to extract the statistics on the precedence relations between the activities that are needed by *Heuristics Miner* to infer a model.

Then, *Heuristics Miner* is run on these statistics, and a high-level model W of the process is obtained. In other words, we circumvent the unavailability of an activity-aware log, from which the "standard" *Heuristics Miner* would extract statistics on the mutual dependencies exhibited by the activities, with two steps: (1) we learn an HMM modeling the sequences of events recorded in L_E as the *observed* counterparts of the sequences of *hidden* activities executed during the process enactments; (2) we extract the needed activity-level statistics from this HMM, and use them to feed *Heuristics Miner*.

Now, the model W returned by *Heuristics Miner* describes the control-flow at the abstraction level of the activities. Thus, as a final step, we augment W with a description of the behavior exhibited at the abstraction level of events. In particular, the augmented model W^2 returned by the framework is obtained by suitably embedding into W the probabilistic mapping between activities and events encoded in the learned HMM.

Let us now discuss some major aspects of our process discovery approach in detail.

3.1 Structure of the HMM

The semantics of HMMs suggests a natural way of modeling the generation of a low-level event log, as the result of executing sequences of high-level activities. In fact, the scenario is the following: (1) what is observed (i.e., the log) is a set of traces, each consisting of a sequence of events; (2) what generate the events are the activities, that provide a high-level description of each step of the process, but are not explicitly represented in the log. These arguments back the use of an HMM's structure where the observations are the events in $\mathcal{E}$, and the hidden states are the activities in $\mathcal{A}$ (see Fig. 4(a)).

However, a major requirement of the HMM in our context is that it must allow the information needed by *Heuristics Miner* to be easily extracted, once the parameters of the same HMM have been learned. Since *Heuristics Miner* needs to know $|A \gg B|$ for each pair of activities A, B, the above-discussed naïve model is unsuitable, as it does not represent sequences of three activities (in fact, states represent single activities and arcs pairs of consecutive activities). Hence, a more suitable model is that of associating each hidden state q with a pair of activities $A^{pre}A^{cur}$ (referred to as $q.A^{pre}$ and $q.A^{cur}$), where A^{cur} represents the current activity, and A^{pre} the activity performed immediately before A^{cur}. For representing the starting of a process (corresponding to an activity execution preceded by no other activity), we use states q where $q.A^{pre}$ is the "null" activity

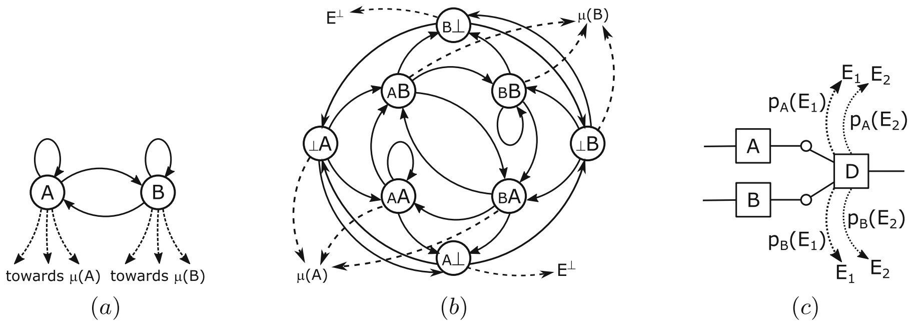


Fig. 4. A naïve HMM (a) and an HMM with memory of the previous state (b). Associating an activity with a pdf over the resulting events in W^2 (c)

(denoted as "$\perp$"). Analogously, a state q where $q.A^{cur}$ = "$\perp$" indicates the end of a process instance. In particular, emissions and transitions are set as follows:

i. every state q has one emission for each $E \in \mu(a.A^{cur})$. As a special case, we assume that the "fake" activity $\perp$ has the unique possible outcome $\mu(\perp) = \{E^{\perp}\}$, where $E^{\perp}$ is an event registered at the end of each trace to indicate the termination of the corresponding process instance;
ii. transitions are put from a state q_1 to a state q_2 if and only if $q_1.A^{cur} = q_2.A^{pre}$ and if the possibility that $q_1.A^{cur}$ precedes $q_2.A^{cur}$ in a process execution is not forbidden by the dependencies in $\mathcal{D}$. Herein, the transition from a state q with $q.A^{cur}$ = "$\perp$" to a q' with $q'.A^{pre}$ = "$\perp$" corresponds to moving from the end of a process instance to the start of another process instance.

Before explaining how to train the HMM (thus, how to set the transitions' and emissions' probabilities), and how to derive $|A > B|$ and $|A \gg B|$ from it, let us show an example, in Fig. 4(b), with two activities and three events, assuming $\mathcal{D} = \emptyset$ (as done in our tests). In this figure and in what follows, a state q with $q.A^{pre} = A$ and $q.A^{cur} = B$ is denoted as $q = {}_{A}B$, where the smaller font refers to the previous activity.

3.2 Training the HMM

The HMM's parameters are learned using L_E as set of training sequences. In particular, we use the BW algorithm, that is invoked on a starting configuration of the HMM where:

(1) for each state, the outgoing transitions are equiprobable, as well as the emissions;
(2) the pdf π indicating the initial states is uniform over the states q with $q.A^{pre} = \perp$, and assigns 0 to all the other states.

Once the parameters of the HMM have been learned, the HMM will have a subset of the states, of the transitions and of the emissions of the starting configuration. For instance, if the BW algorithm has learned that an execution of A, when preceded by B, is never followed by C, then the transition from the state $\textsc{b}A$ to $\textsc{a}C$ is deleted. Analogously, if BW has learned that A is never followed by B, then the state $\textsc{a}B$ and all of its ingoing and outgoing transitions are deleted, along with its emissions. Thus, the learning step refines the domain knowledge used to define the HMM's structure, as it can learn dependencies not initially specified in $\mathcal{D}$ and make the initial candidate mapping μ probabilistic. In fact, for each activity A, every $E \in \mu(A)$ will be associated with an emission probability representing the likelihood that E is the outcome of A (a 0 emission probability means removing E from the set of candidate events).

3.3 Extracting the Statistics Needed by *Heuristics Miner* from the HMM

From the learned HMM, the statistics to be given as input to *Heuristics Miner* are evaluated as follows. First of all, we assume that the BW algorithm returns, along with the probabilities of the transitions and the emissions, also the vector $\boldsymbol{\gamma} = \gamma_1, \ldots, \gamma_T$, where each γ_i is a pdf assigning each state q the probability of being the actual state of the process at time i, considering all the process enactments encoded in L_E.[1] Then, for each pair of activities A, B, the number of times that an execution of A is followed by an execution of B is estimated as the sum of the probabilities that at the different steps the process is in the state $\textsc{a}B$ of the HMM, that is: $|A > B| = \Sigma_{i\in[1..T]}\gamma_i(\textsc{a}B)$.

Analogously, the number of occurrences of ABA is estimated by summing the probabilities that, at each step, the process has performed activity B after having performed activity A, and then will perform A again. This means taking probabilities at the various steps that the process is in state $\textsc{a}B$ and then moves to $\textsc{b}A$, that is: $|A \gg B| = \Sigma_{i\in[1..T]}\gamma_i(\textsc{a}B) \cdot a_{(\textsc{a}B)(\textsc{b}A)}$ where $a_{(\textsc{a}B)(\textsc{b}A)}$ is the entry of the transition matrix associated with the transition between the states $\textsc{a}B$ and $\textsc{b}A$.

3.4 Invoking *Heuristics Miner* and Building a Two-Level Process Model W^2

The values of $|A > B|$ and $|A \gg B|$, computed for each activity pair, are passed as argument to *Heuristics Miner*, that uses them to build a process model.

The output of *Heuristics Miner* is a model W of the process at the abstraction level of activities (in particular, in our experiments we leveraged an implementation of the algorithm available in ProM 6.7 that returns a *Heuristics Net*).

Starting from W, a two level process model W^2 is composed, that augments W by associating each activity A occurring in W with a pdf over the events into which the execution of A can result (see Fig. 3 for an example of shapes of W

[1] As recalled in Sect. 2, $\boldsymbol{\gamma}$ is computed by the standard BW algorithm to perform the maximization step. In our implementation, $\boldsymbol{\gamma}$ is not disposed when the algorithm ends, as it is useful as a statistics on the states of the process.

and W^2). This pdf is suitably extracted from the emission probabilities in the learned HMM. In particular, for each activity A in W, the emission probabilities that are associated with A in W^2 are those of the states q in the HMM with $q.A^{cur} = A$ and such that $q.A^{pre}$ is an activity that precedes A in W. For instance, consider the case that $\mathcal{A} = \{A, B, C, D\}$ and that, according to W, activity D can be preceded by A or B, but not by C, and it cannot be the initial activity. Then, the emission pdf associated with D will incorporate the emission probabilities of states ${}_AD$ and ${}_BD$, but not that of ${}_CD$. This is depicted in Fig. 4(c), where the pdf of D is "split" into $p_A(E)$ and $p_B(E)$, denoting the probabilities that D triggers the event E when the execution of D is preceded by A and B, respectively. The pdfs of A and B are not distinguished based on the previous activity, meaning that the event produced by these activities does not depend on the previous activity.

Observe that it can happen in practice that a state q is present in the HMM, but the models W and W^2 do not allow the possibility that $q.A^{pre}$ precedes $q.A^{cur}$. In fact, *Heuristics Miner* may infer that $q.A^{pre} \Rightarrow q.A^{cur}$ does not hold since the cardinality of $|A > B|$, although greater than 0, is "small". In this case, *Heuristics Miner* concludes that the occurrence of the subsequence AB in some process enactment is symptomatic of noise and thus it must not be encoded in the resulting process model.

4 Experiments

The Datasets. The empirical validation was conducted over noisy datasets generated from the process models W_1 = "*parallel5*", W_2 = "*a12*", W_3 = "*herbstFig3p4*", W_4 = "*herbstFig6p18*", W_5 = "*herbstFig6p41*", selected from a popular set of benchmark logs used in many previous works (e.g., [18,23]), and featuring different control-flow constructs (sequences, choices, parallelism, short loops, and invisible tasks).

For each W_i, we generated 20 low-level event logs, each denoted as L^s_{ij}, with $s \in \{1.2, 1.4, 1.6, 1.8, 2.0\}$ and $j = [1..4]$, where s is a parameter denoting the average number of activities onto which an event can be mapped (i.e., the average number of activities that possibly result in the same event). Thus, s measures the amount of shared functionalities and, intuitively enough, gives a measure of the uncertainty to be dealt with (in particular, $s = 1$ means that each event corresponds to one activity exactly).

Specifically, each L^s_{ij} was generated as follows. First, we used the model W_i (that contains references to high-level activities only) to generate a noise-free activity log NFL^A_i conforming to W_i. Then, a noisy activity log L^A_i was obtained by perturbing each trace in NFL^A_i by randomly switching pairs of consecutive activities, or replacing an activity occurrence with another activity, or deleting an activity occurrence. Perturbations were applied with probability 2% for each step in each trace. After obtaining the (noisy) activity log L^A_i, we generated 20 mappings, denoted as μ^s_{ij} (with $s \in \{1.2, 1.4, 1.6, 1.8, 2.0\}$ and $j = [1..4]$), between the activities in W_i and a set of events with the same cardinality as $|\mathcal{A}|$.

Specifically, for each value of s, we generated 4 different mappings $\mu^s_{i1}, \ldots, \mu^s_{i4}$, all ensuring that, on the average, each event is mapped to s activities. Observe that, since $|\mathcal{E}| = |\mathcal{A}|$, s measures also the average number of events that are the possible outcomes of the same activity. Starting from μ^s_{ij}, we obtained a probabilistic mapping $\tilde{\mu}^s_{ij}$, by interpreting the events in μ^s_{ij} as the possible outcomes of a random variable following a Zeta distribution (in particular, the events in $\tilde{\mu}^s_{ij}(A)$ were associated with a progressive rank k, and their probabilities were set proportional to $1/k$). Finally, from the activity log L^A_i, for each $s \in \{1.2, 1.4, 1.6, 1.8, 2.0\}$ and $j = [1..4]$, we obtained an event log L^s_{ij} as follows: for each occurrence of an activity A in L^A_i, an event E was sampled from $\tilde{\mu}^s_{ij}(A)$, and the occurrence of A was replaced with E. This way, we obtained 20 event logs for each model W_i.

Measuring the Effectiveness. To validate our framework, we performed the following tests, for each log L^s_{ij}. First, we ran our prototype on L^s_{ij} and obtained a two-level model $W^2(L^s_{ij})$. Then, the effectiveness was evaluated by measuring how far $W^2(L^s_{ij})$ is from the actual model originating L^s_{ij}, that is the combination of the activity model W_i and the mapping $\tilde{\mu}^s_{ij}$. This was accomplished using three metrics: $\mathcal{F}$, $\mathcal{P}$, and *Err*.

$\mathcal{F}$ is a *fitness* metric returning a sort of "recall" score quantifying the capability of W^2 to capture the behavior of the ground-truth model W_i manifested in the (noise-free) activity log NFL^A_i. The metric is evaluated by using the fitness calculation method in [20][2], to compare the log NFL^A_i and a Petri-net representation of the activity level of $W^2(L^s_{ij})$, i.e., the model obtained from $W^2(L^s_{ij})$ by disregarding events' emissions.

$\mathcal{P}$ is instead and empirical metric of *precision*, computed in a symmetric way by first generating an activity log L' (of 5000 traces) from a Petri-net representation of the activity level of $W^2(L^s_{ij})$, and then computing a fitness score (still using the method of [20]) for the ground-truth model W_i (represented as a Petri net) and L'.

Finally, *Err* provides an *error* measurement for the emission probabilities represented in $W^2(L^s_{ij})$, computed as the maximum difference between the probability associated with an emission in $W^2(L^s_{ij})$ and that in $\tilde{\mu}^s_{ij}$.

The Setting. The HMM to be learned was constructed according to what said in Sect. 3.1, assuming $\mathcal{D} = \emptyset$ (thus, benefiting from no knowledge of the process behavior in terms of activity dependencies) and assuming the presence of a domain expert providing as candidate mapping the actual sets of possible outcomes of every activity (however, no information on the probability associated with each event was assumed, thus the emission probabilities were set uniformly

[2] This method mainly relies on a non-blocking replay (where tokens can be put artificially into any place of the net when the latter cannot reproduce a trace step), and penalizes both unexpected events and improper completion (based on how many tokens were created artificially and left unconsumed, respectively). It can be hence applied to a not sound process model (as were those returned by *Heuristics Miner* on some of the low-level logs produced with $s > 1.6$).

and had to be refined by BW). The benefits that could arise from using a non-empty $\mathcal{D}$ are discussed in Sect. 6.

Our results are easily reproducible, as our prototype uses a standard implementation of the BW algorithm, and the implementations of algorithm Heuristics Miner and of the fitness algorithm [20] available in ProM 6.7 and in ProM 5.2, respectively—precisely, when computing each fitness score we set a maximum search depth of 1 over the invisible transitions (if any) of the net. Moreover, we used the default parameter setting for *Heuristics Miner*, except for the parameters *DependendencyDivisor* and *relativeToBest* that were set equal to 20 (corresponding to about 0.4% the number of analyzed log traces) and 10%, instead of 1 and 5%, respectively.

The results were obtained using only 2 iterations of the BW algorithm, as we noticed that in most cases further iterations did not improve the quality of the discovered process models substantially—in fact, allowing BW to perform more steps can help improve the learned HMM, and refine the dependency statistics for *Heuristics Miner*, but such refinements may be unnecessary to reckon the structure of the process well enough.

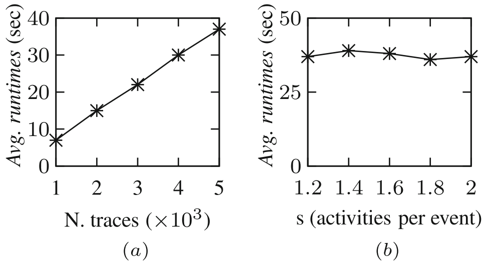
Discussion of the Results. The results in Table 1 show that our framework is rather effective in finding a model that is representative of the behavior of the processes analyzed. In particular, the closeness of the two-level model to the actual processes' behavior is witnessed by the high values of $\mathcal{F}$, $\mathcal{P}$, and $1 - Err$. As expected, these measures are affected by s: the higher s (i.e., the average number of activities that could generate the same event), the lower $\mathcal{F}$, $\mathcal{P}$, and $1 - Err$. In fact, intuitively enough, larger values of s correspond to dealing with a higher level of uncertainty when mapping each log event to the actual activity generating it, as the average number of candidate activities is s. Interestingly, the lower efficacy of the framework at the highest value of s is more evident for the models W_3 and W_4, that exhibit a more complex structure than W_1, W_2, and W_5 (containing choices and parallelism, but not mixed with loops). All the results in Table 1 must be read considering that invoking *Heuristics Miner* on the activity logs $L_1^A, \ldots, L_5^A$ (from which all the low-level event logs were derived) returned the correct models $W_1, \ldots, W_5$. This means that the values of $\mathcal{F}$ less than 1 are due to the uncertainty introduced by the many-to-many mapping between activities and events.

Figure 5 shows the sensitivity of the run times vs. various parameters. For every run, the time spent by *Heuristics Miner* was negligible compared with that taken by the BW algorithm; thus we do not distinguish between the time shares of these two components.

The diagram in Fig. 5(a) reports, for each $X \in \{1000, 2000, 3000, 4000, 5000\}$, the mean values of the run times over all the event logs $L_{i1}^s(X)$ (for all $s \in \{1.2, 1.4, 1.6, 1.8, 2.0\}$ and $i \in [1..5]$), where $L_{i1}^s(X)$ consists of the first X traces in L_{i1}^s. This diagram shows that the run time is linear with the number of traces, which is in line with the fact that BW is linear in the number of observations. The diagram in Fig. 5(b) reports the average run times over all the L_{i1}^s logs grouped by s, and shows that the computation time does not depend on s.

Table 1. Effectiveness of the framework (using 2 invocations of BW algorithm). For each model W_i and each value of s, the averages of the results obtained over the logs $L^s_{i1}, \ldots, L^s_{i4}$ are shown.

	W_1(*parallel5*)			W_2(*a12*)			W_3(*herbstFig3p4*)			W_4(*herbstFig6p18*)			W_5(*herbstFig6p41*)		
	$\mathcal{F}$	$\mathcal{P}$	*Err*	$\mathcal{F}$	$\mathcal{P}$	*Err*	$\mathcal{F}$	$\mathcal{P}$	*Err*	$\mathcal{F}$	$\mathcal{P}$	*Err*	$\mathcal{F}$	$\mathcal{P}$	*Err*
s = 1.2	0.97	0.97	2%	0.90	0.92	2%	0.93	0.89	1%	0.96	0.97	2%	0.92	0.94	3%
s = 1.4	0.98	0.96	5%	0.88	0.81	6%	0.94	0.92	4%	0.95	0.96	2%	0.96	0.81	5%
s = 1.6	0.95	0.97	6%	0.89	0.77	8%	0.88	0.84	5%	0.90	0.83	3%	0.94	0.88	4%
s = 1.8	0.92	0.88	8%	0.88	0.72	9%	0.84	0.77	8%	0.90	0.86	4%	0.92	0.86	6%
s = 2.0	0.89	0.92	9%	0.88	0.75	9%	0.78	0.80	10%	0.83	0.72	8%	0.88	0.79	8%



Model	$\|\mathcal{A}\|$	Avg. trace length	Runtimes (sec)
W_1	10	10	8
W_2	14	≈ 8	30
W_3	12	≈ 28	42
W_4	10	≈ 20	16
W_5	16	13	88

(c)

Fig. 5. Run times vs. number of traces (a), activities per events (b) and models (c)

This result is obvious for HM (that does not deal with the events). For BW, it follows from the fact that its computation complexity mainly depends on the number of states and observation types, and not on their mutual correlations. Finally, the table in Fig. 5(c) reports the average run times vs. the models used to produce the logs, and characteristics of both the models and logs.

5 Related Work

The problem of discovering a control-flow model has been deeply addressed in the last years, and a wide variety of solutions have been proposed (see [2,4,23] for recent surveys). A method for inducing a dependency graph was first presented in [3]. Similar graph-like models also underly subsequent methods [18,24], which also introduced mechanisms for describing the semantics of join/split nodes. In most of these methods, including algorithm *Heuristics Miner* [24], activity dependencies are derived from pairwise ordering statistics over the activities, extracted from the log, collectively referred to as “log abstractions”.

By contrast, in other works the discovery problem was conceptually stated as a search over more expressive classes of process models, such as Petri nets (as done in [25]), or block-structured workflow models (as done in [15]). This search has been faced using various strategies, ranging from ad-hoc algorithms, to approximated optimization methods. An approach that was shown quite effective (cf. [4]) is the *Inductive Miner* algorithm [15], which founds on iteratively refining a grammar-like representation of the process, while recursively splitting the events in the log.

However, as empirically observed in [23], *Heuristics Miner* often manages to produce effective and readable process models, compared to many later approaches, and to reach a good trade-off between effectiveness (owing to its capability to deal with both complex constructs and noise) and fastness/scalability. In particular, the usage of compact log abstractions (quadratic in the number of activities, and independent of the log size), that can be computed in linear time (possibly in a parallelized way), makes it a natural candidate to implement scalable process discovery approaches [10], and even to deal with streaming logs [8]—by contrast, most current process discovery methods need to keep the log in main memory [16]. To the best of our knowledge, this feature is shared with only one advanced process discovery algorithm [16], where the search procedure of Inductive Miner [15] is not performed by recursing on the log, but rather on a matrix storing directly-follow measurements (the same as *Heuristics Miner*).

All the methods above cannot be applied to our problem setting, as they assume that each log event refers (or can be deterministically mapped) to a process activity.

Other related approaches are those addressed at interpreting low-level event logs [5,6,12,13,17,22], and eventually translating event traces into sequences of high-level activities. In principle, such techniques could be used as a preprocessing tool, to enable the application of activity-oriented process discovery algorithms to the abstracted version of a low-level log. However, most of these techniques need to be provided with explicit and rather deep knowledge on the process' behavior, such as: procedural/declarative process models (as in [5,12,17]), event-oriented disambiguation rules (like the attribute and event context conditions in [6]), documents containing activity descriptions [5,6], activity-annotated traces for training a sequence-labelling model (as in [22]). Moreover, since this does not usually suffice to resolve all ambiguities, the intervention of an expert [5,6] is needed to distill the "right" translation.

6 Conclusions: Discussion of Limitations and Extensions

Our framework for performing the process discovery task starting from a low-level event log has been proved effective in returning accurate two-level process models when tested over logs adhering to different real-life models taken from repositories popular in the process mining community. In what follows, we discuss some critical points, limitations and possible extensions of our approach.

(a) Domain knowledge: how much? Our experimental results suggest that the framework can be effective even in the presence of a rather limited amount of domain knowledge: models of good quality were obtained despite $\mathcal{D}$ was assumed empty. Obviously, there are limits to what can be learned in the absence of knowledge of the process behavior. For instance, activities O, S, V in Fig. 1 are indistinguishable from their low-level counterparts (i.e., the event q), thus no mechanism is likely to effectively solve the ambiguity when interpreting an occurrence of q in the log. However, in this example, the knowledge of the dependency "*V always precedes D or N*" (that can be straightforwardly encoded in the HMM's structure) can dramatically reduce the uncertainty, so that an accurate model becomes obtainable (in fact, taking into account this dependency, only the last occurrence of q in a trace could be interpreted as the outcome of V). The possibility of exploiting knowledge from a domain expert (in the spirit of [14]) also places the proposed framework as the core of an iterative scheme, where, after a model is produced, the expert is called for validating it and possibly enriching $\mathcal{D}$ with dependencies that s/he believes can help solve ambiguities. This progressive refinement would also address possible limits of the high-level PD algorithm embedded in the framework, that might yield models that are unsound or with deadlocks, and thus need some revising (as may happen with *Heuristics Miner*, that does not guarantee the correctness of the produced model).

(b) Dealing with composite activities. The extension to the case that activities are composite is straightforward if no interleaving is allowed between the sequences of events of two activities (but interleaving sequences of activities are still allowed). In this case, each state ${}_AB$ can be represented by three states *start*${}_AB$, *intermediate*${}_AB$, and *final*${}_AB$ with the events in $\mu(B)$ as possible emissions. The general case where the sequences of events of two activities A and B can be interleaved can still be modelled, by putting into each state a bitstring encoding the set of activities that are still "active". The point is that this can dramatically increase the size of the HMM, thus more scalable variants of the BW algorithm (for instance, sampling-based ones) must be tried.

(c) Dealing with non-free choices. In order to allow *Heuristics Miner* to discover non-free choices in the process behavior, we need to extract statistics of the form $|A \ggg B|$, representing how many times activity B followed activity A in the log, at any distance. These statistics cannot be obtained directly from the HMM, but can be inferred by (i) running the Viterbi algorithm over the trained HMM and every trace of the log, in order to obtain the most probable sequence of activities for each trace, and (ii) computing $|A \ggg B|$ by analyzing each of the so obtained sequences of activities.

(d) Exploiting PD algorithms other than Heuristics Miner. Techniques such as α [1] and $\alpha+$ [9] can be straightforwardly used in place of *Heuristics Miner*, as the information they need as input is obtainable from our trained HMM and the values of γ. The same holds for the "single-pass" scalable version of *Inductive Miner* [16]. As a matter of fact, we have conducted some preliminary experiments using *Inductive Miner* over the data sets generated from the process models W_1 and W_2. Interestingly, we obtained models with higher fitness but lower precision (compared to those obtained with *Heuristics Miner*). However, this result must be read along with the fact that the models produced by *Inductive Miner* were all sound (while some of those returned by *Heuristics Miner* were not). Further experiments are needed to investigate the sensitivity of this behavior to the parameters of *Inductive Miner* and to the process features.

PD techniques taking as input information different from the statistics taken by *Heuristics Miner* can be also embedded in the framework: after learning the HMM, the event traces can be converted into activity traces by means of the Viterbi algorithm, and then the required information can be extracted from the converted log.

(e) Obtaining the candidate mapping via semi-automatic approaches. The initial candidate mapping μ, that is refined by the learning phase into a probabilistic event/activity mapping, can be provided by a domain expert. Otherwise, existing event-to-activity matching techniques, like those in [5,6], can help distillate a small subset of all possible mappings between activities and events (at the level of types), by leveraging background knowledge on the process behavior and/or textual descriptions available for both the activities and the events. By only considering the $\langle$activity, event$\rangle$ pairs returned by these techniques as admissible mappings, it is possible to initialize more precisely the emission probabilities of the HMM model, as emissions supported by no evidence (but otherwise considered as candidate mappings) can be preliminarily discarded.

References

1. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE TKDE **16**(9), 1128–1142 (2004)
2. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st edn. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19345-3
3. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Schek, H.-J., Alonso, G., Saltor, F., Ramos, I. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 467–483. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0101003
4. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: review and benchmark. arXiv preprint arXiv:1705.02288 (2017)
5. Baier, T., Di Ciccio, C., Mendling, J., Weske, M.: Matching events and activities by integrating behavioral aspects and label analysis. Softw. Syst. Model., 1–26 (2017)
6. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Inf. Syst. **46**, 123–139 (2014)
7. Baier, T., Rogge-Solti, A., Weske, M., Mendling, J.: Matching of events and activities - an approach based on constraint satisfaction. In: Frank, U., Loucopoulos, P., Pastor, Ó., Petrounias, I. (eds.) PoEM 2014. LNBIP, vol. 197, pp. 58–72. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45501-2_5
8. Burattin, A., Sperduti, A., van der Aalst, W.M.: Control-flow discovery from event streams. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 2420–2427. IEEE (2014)
9. de Medeiros, A.K.A., van Dongen, B.F., van der Aalst, W.M.P., Weijters, A.J.M.M.: Process mining: extending the α-algorithm to mine short loops. Technical report, University of Technology, Eindhoven (2004). bETA Working Paper Series, WP 113

10. Evermann, J.: Scalable process discovery using map-reduce. IEEE Trans. Serv. Comput. **9**(3), 469–481 (2016)
11. Fazzinga, B., Flesca, S., Furfaro, F., Masciari, E., Pontieri, L.: A compression-based framework for the efficient analysis of business process logs. In: Proceedings of 27th International Conference on Scientific and Statistical Database Management, SSDBM 2015, pp. 6:1–6:12 (2015)
12. Fazzinga, B., Flesca, S., Furfaro, F., Masciari, E., Pontieri, L.: Efficiently interpreting traces of low level events in business process logs. Inf. Syst. **73**, 1–24 (2018)
13. Fazzinga, B., Flesca, S., Furfaro, F., Pontieri, L.: Online and offline classification of traces of event logs on the basis of security risks. J. Intell. Inf. Syst. **50**(1), 195–230 (2018)
14. Greco, G., Guzzo, A., Lupia, F., Pontieri, L.: Process discovery under precedence constraints. ACM Trans. Knowl. Discov. Data **9**(4), 32:1–32:39 (2015)
15. 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
16. 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
17. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: From low-level events to activities - a pattern-based approach. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 125–141. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_8
18. Medeiros, A.K., Weijters, A.J., Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Disc. **14**, 245–304 (2007)
19. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. In: Waibel, A., Lee, K.F. (eds.) Readings in Speech Recognition, pp. 267–296. Morgan Kaufmann (1990)
20. Rozinat, A., van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
21. Senderovich, A., Rogge-Solti, A., Gal, A., Mendling, J., Mandelbaum, A.: The ROAD from sensor data to process instances via interaction mining. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 257–273. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39696-5_16
22. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Event abstraction for process mining using supervised learning techniques. In: Bi, Y., Kapoor, S., Bhatia, R. (eds.) IntelliSys 2016. LNNS, vol. 15, pp. 251–269. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-56994-9_18
23. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Inf. Syst. **37**(7), 654–676 (2012)
24. Weijters, A., van Der Aalst, W.M., De Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Technical report WP 166, pp. 1–34 (2006)
25. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundamenta Informaticae **94**, 387–412 (2009)