Is my event log complete? - A probabilistic approach to process mining

Citation for published version (APA):

DOI:
10.1109/RCIS.2011.6006848

Document status and date:
Published: 01/01/2011

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 16. Sep. 2023
Is My Event Log Complete? —
A Probabilistic Approach to Process Mining

Kees M. van Hee, Zheng Liu, Natalia Sidorova
Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{k.m.v.hee, z.liu3, n.sidorova}@tue.nl

Abstract—Process mining is a technique for extracting process models from event logs recorded by information systems. Process mining approaches normally rely on the assumption that the log to be mined is complete. Checking log completeness is known to be a difficult issue. Except for some trivial cases, checkable criteria for log completeness are not known. We overcome this problem by taking a probabilistic point of view. In this paper, we propose a method to compute the probability that the event log is complete. Our method provides a probabilistic lower bound for log completeness for three subclasses of Petri nets, namely, workflow nets, T-workflow nets, and S-workflow nets. Furthermore, based upon the complete log obtained by our methods, we propose two specialized mining algorithms to discover T-workflow nets and S-workflow nets, respectively. We back up our theoretical work with empirical studies that show that the probabilistic bounds computed by our method are reliable.

Keywords—process mining, workflow management, event log, Petri nets, probabilistic analysis.

I. INTRODUCTION

Many processes today are supported by computer systems which log their events. Originally these logs were used to trace back the events and to detect whether errors or exceptions had occurred. Today the challenge has moved to the area of deriving more sophisticated information from event logs. Process mining is a branch of data mining that is focused on the discovery of the underlying process model, i.e., the model of the process that generated the event log [2–8, 12, 13].

Process mining algorithms are often based on the assumption that the underlying process handles individual cases, which can be handling a claim in an insurance company, the application of a medical protocol to a patient in a hospital or the maintenance of a car in a repair shop. A log typically consists of events that can be grouped in sequences per case. For mining purposes we often assume that these cases are independent of each other, which means that the order of events per case is only determined by the characteristics of the case and not by the accidental presence of other cases. Certainly the duration of an event, waiting times and the total cycle time of a case might be influenced by the presence of other cases due to queuing phenomena, but the specific event order is not.

Given an event log and a process mining algorithm that computes a process model from this log, an important question is: “Can we claim that the discovered process model is identical (actually bisimilar) to the process model that generated the events?”, which requires answering the question “Is my event log complete?”. This question plays an essential role in standard applications of process mining, when companies want to find out with process mining what their actual processes are, and this question becomes even more important in the domain of automated and continuous audit, when the auditing company checks that the process behaviour represented in the log complies to the rules and the auditors want to be certain that the log they have obtained contains enough information about the behaviour of the process, i.e. that their compliance check on the log implies that the process that generated the log is compliant to the rules and no uncompliant behaviour will be encountered if the organisation continues using this process.

There are some theoretical results for log completeness, i.e., [3, 10, 17]. These works are, however, based on a number of assumptions which cannot be verified in practice, for example, the assumption that every two events that may happen directly one after another according to the model implemented in the system have also occurred in the log directly one after another. One can only check this assumption if the process model is known, but this is exactly the problem of process mining, which results in the circular reasoning.

In this paper we take a probabilistic approach in order to guarantee, with some probability, that the event log is complete and the discovered process is bisimilar to the generating process. We consider each case as a random variable and each event log as a sequence of independent samples from this random variable. We first investigate the problem of log completeness in the context of the $\alpha$-algorithm [3] for process mining, which mines Petri nets, in particular workflow nets [1], as process models. We derive a probabilistic bound for log completeness for the class of structured sound workflow nets.

At the next step we try and improve the mining algorithm and come up with a more precise probabilistic bound for completeness by taking into account available information about the process structure. We consider two simple but important classes of structured sound workflow nets: S-workflows and T-workflows. While S-workflows do not support parallelism but allow modeling choices, T-workflows do the opposite: they provide support for modelling parallelism but do not
allow for having choices in the model. S-workflows model in principle all sequential processes, which are often used for processing cases in such organisations as insurance companies, financial and governmental institutions. T-workflows are in fact standardly used as project planning models in large construction or maintenance projects in e.g. oil, building and aircraft industries.

In case we know that the process that produced the log works as an S-workflow, or a T-workflow, we can employ this information in our mining procedure. Due to the use of this information in computing the probabilistic bound, we achieve more precise estimations and we are able to claim that the log is complete with a significantly higher probability than in case we would apply the probabilistic bound we have for structured sound workflow nets to S-nets and T-nets.

The rest of this paper is organised as follows. Section II gives basic concepts from probability and set theory. Section III introduces concepts of Petri nets and process mining do-...
Note that S-workflow nets and T-workflow nets both belong to the class of structured sound workflow nets.

Workflow nets reflecting the actual behaviour of a system can be discovered from event logs, which are sets of traces, with every trace being a sequence of events for a completed process instance.

A standard technique used in process mining for finding a workflow model is the discovery of causal dependencies between the events of the system: If a task is always followed by another task it is likely that there is a causal relation between them. In order to describe these log-based ordering relations, we introduce the following notations:

Definition 6 (Log-based ordering relations). Let $L$ be an event log and $a, b$ be events.

1. $a \triangleright L b$ iff $\exists \sigma \in L, i \in \{1, 2, \ldots, n\} : \sigma(i) = a \land \sigma(i + 1) = b$, i.e. $a$ directly follows $b$ in at least one trace.
2. $a \ntriangleright L b$ iff $a \triangleright L b$ and $\neg (b \triangleright L a)$, i.e. $b$ is a possible direct successor of $a$ and not another way around.
3. $a \parallel L b$ iff $a \triangleright L b$ and $b \triangleright L a$, which indicates potential parallelism.
4. $a \# b$ iff $\neg (a \triangleright b, b \triangleright a)$, gives pairs of transitions that never follow each other directly, implying that direct successor relations and parallelism are not possible.

If we consider $L$ to be the set of all possible traces of the process, then we drop the subscript $L$ in Definition 6, i.e., we write $a \triangleright b$ as $a \triangleright b$. The set of all traces can contain an infinite number of traces in case of loops in the process.

The $\alpha$-algorithm (Algorithm 1) is a process mining algorithm that is able to extract a workflow model based on a complete log generated by a sound workflow net by examining the log-based ordering relations observed between tasks [3]:

Definition 7 (Complete event log). Let $N = (P, T, F, i, f)$ be a sound workflow net. Let $L$ be an event log of $N$. $L$ is complete if and only if (1) $\triangleright \subseteq \triangleright L$, and (2) for any $t \in T$ there is a $\sigma \in L$ such that $t \in \sigma$.

Algorithm 1 Construct a workflow net by the $\alpha$-algorithm

1. $T_L := \{t \in T \mid \exists \sigma \in L : t \in \sigma\}$
2. $T_I := \{t \in T \mid \exists \sigma \in L : t = \sigma(1)\}$
3. $T_F := \{t \in T \mid \exists \sigma \in L : t = \sigma(|\sigma|)\}$
4. $X_L := \{(A, B) \mid A \subseteq T_L \land B \subseteq T_L \land \forall a \in A \land \forall b \in B : a \rightarrow b \land \forall a_1, a_2 \in A \land b_1, b_2 \in B : b_1 \# b_2\}$
5. $Y_L := \{(A, B) \in X_L \mid \forall (A', B') \in X_L : A \subseteq A' \land B \subseteq B' \rightarrow (A, B) = (A', B')\}$
6. $P_L := \{p_{(A, B)}(A, B) \in Y_L \} \cup \{i_L, f_L\}$
7. $F_L := \{(a, p_{(A, B)}) \mid (A, B) \in Y_L \land a \in A\} \cup \{(p_{(A, B)}, b) \mid (A, B) \in Y_L \land b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(f_L, t) \mid t \in T_F\}$
8. $\alpha(L) := (P_L, T_L, F_L)$

If there is a log-based ordering relation between two transitions according to the event log, then there has to be a place connecting these two transitions.

Theorem 1 (Log-based ordering relations imply connecting places [3]). Let $N = (P, T, F, i, f)$ be a sound workflow net and $L$ be a complete log. For all $a, b \in T$, $a \ntriangleright L b$ implies $a^* \cap \# b \neq \emptyset$.

The following theorem shows that the $\alpha$-algorithm is able to rediscover sound structured workflow nets from complete event logs:

Theorem 2 (Rediscovering ability [3]). Let $N = (P, T, F, i, j)$ be a sound structured workflow net and $L$ be a complete workflow log of $N$. If for all $a, b \in T$ $a^* \cap \# b = \emptyset$ or $b^* \cap \# a = \emptyset$, then $\alpha(L) = N$ modulo renaming of places.

IV. PROBABILISTIC FRAMEWORK

In order to handle the choice construction in Petri nets we make a realistic assumption that choices can be described by a probabilistic mechanism. In order to define a probability space, we extend the Petri net framework with a weight function $w : T \rightarrow [0, 1]$ that allows to compute the probability of a transition to fire, given a marking. For a reachable marking $m$ enabling transitions $t_1, \ldots, t_n$, the probability of transition $t_k$ to fire $q(m, t_k)$ equals $w(t_k) / \sum_{i=1}^{n} w(t_i)$, i.e. the probability is proportional to the weight of the transition. Applying this formula to a firing sequence $\sigma = (t_1, \ldots, t_n)$ with $m_0 \xrightarrow{t_1}, m_1 \xrightarrow{t_2}, \ldots, m_{n-1} \xrightarrow{t_n}$, we derive $q(\sigma) = \prod_{i=1}^{n} q(m_i, t_i)$.

Next we define a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where $\Omega = \{\sigma \in T^* \mid m_0 \xrightarrow{t_1}, m_1 \xrightarrow{t_2}, \ldots, m_n\}$ and $m_0$ is the initial marking. On $\mathcal{F} = \mathbb{P}(\Omega)$ we define the probability by $\mathbb{P}(\{\sigma\}) = q(\sigma)$. Note that each trace corresponds to a firing sequence $\sigma$ exactly. On $\Omega$ we define random variables. One of such variables is $Y : \Omega \rightarrow R$, where $R$ is the set of all $|T| \times |T|$ matrices with entries in $\{0, 1\}$. For $a, b \in T$ and $1 \leq k \leq |\sigma|$, $Y(\sigma)(a, b) = \Bigg\{ \begin{array}{ll} 1 & \text{if } \exists k : \sigma(k) = a \land \sigma(k + 1) = b, \\ 0 & \text{otherwise}. \end{array}$

We usually drop $\sigma$ in $Y(\sigma)$ if it causes no confusion. Thus we have a probability function $\pi$ with $\pi(y) = \mathbb{P}[Y = y]$, where $y$ is a $|T| \times |T|$ matrix over $\{0, 1\}$. Note that we do not know $\pi$, we consider it as an unknown parameter.

In an event log $L$ all the traces can be considered as samples from $Y$. If $|L| = n$ we denote them as $Y_1, \ldots, Y_n$. They are independent, identically distributed random variables, versions of $Y$.

We introduce the second random variable $Y^n$, the aggregation of $Y_1, \ldots, Y_n$.

\[ Y^n(a, b) = \bigoplus_{k=1}^{n} Y_k(a, b), \tag{1} \]

where $\bigoplus$ is the logical OR operator (i.e., $0 + 0 = 0$ and $0 + 1 = 1 + 0 = 1 + 1 = 1$). We call $Y^n$ the log characteristic matrix.

Note that $Y^n(a, b) = 1 \iff a \triangleright L b$.

Finally, we introduce the third random variable $Z(\sigma)(a, b)$ that reflects the ordering of $a, b$ in a trace $\sigma$. For $a, b \in T$,
\( k \in \{1, 2, \ldots, |\sigma|\} \) and \( l \in \{1, 2, \ldots, |\sigma|\} \).

\[
Z(\sigma)(a, b) = \begin{cases} 
1 & \text{if } \exists k, l : \sigma(k) = a \land \sigma(l) = b \land l > k + 1 \\
2 & \text{if } \exists k : \sigma(k) = a \land \sigma(k + 1) = b \\
3 & \text{if } \exists k, l : \sigma(k) = b \land \sigma(l) = a \land l > k + 1 \\
4 & \text{if } \exists k : \sigma(k) = b \land \sigma(k + 1) = a
\end{cases}
\]

If \( |L| = n \), we consider \( n \) independent, identically distributed random variables \( Z_1, \ldots, Z_n \), versions of \( Z \).

V. Probabilistic Bound for a Complete Log

In this section we consider logs of sound workflow nets, and we will determine a probabilistic lower bound for such a log to be complete. In Section III we have defined a log \( L \) to be complete if \( \triangleright \subseteq \triangleright_L \) and all transitions (events) possible in the system occur in the log. If a workflow net is a structured workflow net, given its complete log, the \( \alpha \)-algorithm will be able to reconstruct the generating net.

In Section IV, we introduced the characteristic matrix \( Y(\sigma) \) for a trace \( \sigma \), the characteristic matrix \( Y^n \) over a log \( L \), and a probability \( \pi \) over characteristic matrices of traces. Although \( \pi \) is unknown, we know it belongs to the set of all possible probabilities \( \Pi \) over the characteristic matrices for traces. Next we define a characteristic matrix of a workflow net, being a structural property of the workflow.

Definition 8 (Structural characteristic matrix). The matrix-valued function \( f \) over \( \Pi \) is defined by \( \forall a, b \in T \),

\[
f(\pi)(a, b) = \begin{cases} 
1 & \text{if } q_{ab} = \sum_{y} \pi(y)y(a, b) > 0, \\
0 & \text{otherwise}.
\end{cases}
\]

To determine a probabilistic lower bound for such a log to be complete, in Section III we have defined a log \( L \) to be complete if \( \triangleright \subseteq \triangleright_L \) and all transitions (events) possible in the system occur in the log. If a workflow net is a structured workflow net, given its complete log, the \( \alpha \)-algorithm will be able to reconstruct the generating net.

In Section IV, we introduced the characteristic matrix \( Y(\sigma) \) for a trace \( \sigma \), the characteristic matrix \( Y^n \) over a log \( L \), and a probability \( \pi \) over characteristic matrices of traces. Although \( \pi \) is unknown, we know it belongs to the set of all possible probabilities \( \Pi \) over the characteristic matrices for traces. Next we define a characteristic matrix of a workflow net, being a structural property of the workflow.

\[
Z(\sigma)(a, b) = \begin{cases} 
1 & \text{if } \exists k, l : \sigma(k) = a \land \sigma(l) = b \land l > k + 1 \\
2 & \text{if } \exists k : \sigma(k) = a \land \sigma(k + 1) = b \\
3 & \text{if } \exists k, l : \sigma(k) = b \land \sigma(l) = a \land l > k + 1 \\
4 & \text{if } \exists k : \sigma(k) = b \land \sigma(k + 1) = a
\end{cases}
\]

If \( |L| = n \), we consider \( n \) independent, identically distributed random variables \( Z_1, \ldots, Z_n \), versions of \( Z \).

From (5), we can estimate a lower bound for the complete log size \( n \) for a sound workflow net with certain confidence:

\[
n \geq \frac{\log \frac{\alpha}{\delta}}{\log(1-\epsilon)}.
\]

The second approach is based on a sample of \( n \) traces in a log. We consider two possible values of \( q_{ab} \):

1) The average of \( Y_n(a, b) \), such that

\[
\bar{Y}_n(a, b) = \frac{1}{n} \sum_{k=1}^{n} Y_k(a, b).
\]

Based on (4) and (7), we derive

\[
\sum_{\{(a,b)|Y_n(a,b)>0\}} (1 - \bar{Y}_n(a, b))^k \leq \alpha.
\]

2) Because \( Y_n(a, b) \) has a binomial distribution with \( q_{ab} \) and \( n \), we have a confidence interval for \( q_{ab} \), such that

\[
q_{ab} \geq \sqrt{n} (1 - \bar{Y}_n(a, b)) \sqrt{n} Y_n(a, b)(1 - Y_n(a, b))
\]

where \( Z_\alpha \) is the \( \alpha \)-percentile of the normal distribution. From (4) and (9) we conclude

\[
\sum_{\{(a,b)|Y_n(a,b)>0\}} (1 - \sqrt{n} (1 - \bar{Y}_n(a, b)) \sqrt{n} Y_n(a, b)(1 - Y_n(a, b)))^k \leq \alpha.
\]

In both (8) and (10), \( k \) is the size of a complete log. It should be the smallest value satisfying (8) and (10), respectively. Thus we keep generating sample traces of the log until \( k \leq n \), where \( n \) is the sample size. This procedure is described in Algorithm 2.

Such a procedure is in fact a stopping rule for a sequence of random variables \( Y_1, Y_2, Y_3, \ldots \).

Algorithm 2 Generate a complete log with certain confidence for a sound workflow net

1: \( L := L_0 \), \( n := 0 \), \( k := 0 \)
2: repeat
3: \( L := L \cup \{\sigma\} \), \( n := n + 1 \)
4: apply (7) to compute \( \bar{Y}_n(a, b) \)
5: apply (10) to compute \( k \)
6: until \( k \leq n \)

If we use (8) instead of (10), then we substitute (10) by (8) in line 5 of Algorithm 2.

We are interested in the quality of the stopping rule. We consider (8) as an estimator for (4). For a fixed log size \( k \) we can compute the expectation of the estimator \( \sum_{a,b} (1 - \bar{Y}_n(a, b))^k \), where the summation is over \( \bar{Y}_n(a, b) > 0 \), by the following formula (see (24) in Appendix).

\[
\mathbb{E}_\pi[\sum_{a,b} (1 - \delta(0, \bar{Y}_n(a, b)))(1 - \bar{Y}_n(a, b))^k] \leq \sum_{a,b} ((1 - q_{ab})^k - (1 - q_{ab})^n) \leq \sum_{\{(a,b)|q_{ab}>0\}} (1 - q_{ab})^k.
\]

Consequently, our estimator is biased: it overestimates the confidence. However, when \( n \to \infty \) the estimator is unbiased.

\[
\sum_{k=1}^{n} (1 - \epsilon)^n = m(1 - \epsilon)^n \leq \alpha.
\]
VI. PROBABILISTIC BOUND FOR A COMPLETE LOG OF S-WORKFLOW NETS

In this section we consider S-workflow nets, and we determine a probabilistic lower bound for a log of an S-workflow net to be complete. As S-workflow nets are sound workflow nets (see Section III), we can use Algorithm 2 to determine the probabilistic lower bound for the size of a complete log. With a complete log, the α-algorithm can discover the generating process. However, if we consider the structural properties of S-workflow nets, we can better estimate the probabilistic lower bounds for their complete logs.

Suppose we know the set of all transitions $T$ in an S-workflow net, and a lower bound $\epsilon > 0$ for the probabilities of making choices in the net, i.e., $q_t$ for $t \in T$. For instance, in a choice fragment (Figure 1) of some S-workflow net, the probabilities to choose transitions $A$, $B$, and $C$ are $q_A$, $q_B$, and $q_C$, respectively, and we assume them all to be at least $\epsilon > 0$. Recall that in our probabilistic framework, any transition $t$ in an S-workflow net to be complete, we can apply the $\alpha$-algorithm to reconstruct the generating net. As S-workflow nets are sound workflow nets, we mine a probabilistic lower bound for a log of an S-workflow net, and a lower bound $\epsilon > 0$.

In order to show that the converse does not hold, consider the net $\{A, B, C, D, E, F\}$ and Figure 2. For T-completeness we need $\triangleright \subseteq \triangleright_L$, which is not true, since $D \triangleright C$ does not hold on $L_1$, nor does $B \triangleright E$.

Hence, T-completeness is a weaker condition than completeness, which in fact also follows from the equivalence $a \triangleright b \equiv (a \triangleright b) \lor (a \rightarrow b)$. Given a T-complete log of a net $N = (P, T, F, i, f)$, we assume $T$ to be known. We also make an assumption on the upper bound for parallelism in $N$.

Note that $|T|$ can always be taken as such a bound, but in most cases it is far too large.

As introduced in section IV, we consider a log $L = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ to be a sample from the probability space, and $Z_i(a, b) = Z(\sigma_i(a, b))$. We define for $a, b \in T$ predicates

1. $Q_1(a, b) \equiv \exists i \in \{1, 2, \ldots, n\} : Z_i(a, b) \in \{1, 2\}$,
2. $Q_2(a, b) \equiv \exists i \in \{1, 2, \ldots, n\} : Z_i(a, b) \in \{3, 4\}$,
3. $Q_3(a, b) \equiv \exists i \in \{1, 2, \ldots, n\} : Z_i(a, b) = 2$,
4. $Q_4(a, b) \equiv \{i \in \{1, 2, \ldots, n\} : Z_i(a, b) = 2\} \geq l$.

where $Q_2(a, b) = Q_1(b, a)$, $Q_3(a, b) \Leftrightarrow a \triangleright b$, and $l$ is a parameter to be determined later in this section.

We introduce a notion of log completeness targeted at the T-workflow nets:

Definition 9 (T-complete log). A log $L$ of an acyclic T-workflow net $N$ is T-complete if for all $a, b \in T$, $a \parallel b$ implies $Q_1(a, b) \land Q_2(a, b)$, and $a \rightarrow b$ implies $Q_3(a, b) \land -Q_2(a, b)$.

Algorithm 3 Generate a complete log for an S-workflow net with a certain confidence

1. $L := L_0$, $\forall t \in T : c(t) = 0$
2. while $\exists t \in T : c(t) < k$ do
3. \hspace{1cm} Generate next trace $\sigma$
4. \hspace{1cm} $L := L \cup \{\rho\}$
5. \hspace{1cm} if $t \in \sigma$ then
6. \hspace{2cm} $c(t) := c(t) + 1$
7. \hspace{1cm} end if
8. end while

VII. PROBABILISTIC BOUND OF A COMPLETE LOG FOR ACYCLIC T-WORKFLOW NETS

In this section we propose a mining algorithm for acyclic T-workflow nets. Given an acyclic T-workflow net $N = (P, T, F, i, f)$, we assume $T$ to be known. We also make an assumption on the upper bound for parallelism in $N$.

Recall that in our probabilistic framework, any transition $t$ in an S-workflow net to occur at least $k$ times in the log. Then $\epsilon k$ a complete log if all transitions occur at least $k$ times in the log. For a confidence limit $c$, the number of times it occurs in the log. Then $\epsilon k$ a complete log if all transitions occur at least $k$ times in the log. This procedure is described in Algorithm 3. Once the log is complete, we can apply the $\alpha$-algorithm to reconstruct the generating net.

Algorithm 3 Generate a complete log for an S-workflow net with a certain confidence

1. $L := L_0$, $\forall t \in T : c(t) = 0$
2. while $\exists t \in T : c(t) < k$ do
3. \hspace{1cm} Generate next trace $\sigma$
4. \hspace{1cm} $L := L \cup \{\sigma\}$
5. \hspace{1cm} if $t \in \sigma$ then
6. \hspace{2cm} $c(t) := c(t) + 1$
7. \hspace{1cm} end if
8. end while

In this section we propose a mining algorithm for acyclic T-workflow nets. Given an acyclic T-workflow net $N = (P, T, F, i, f)$, we assume $T$ to be known. We also make an assumption on the upper bound for parallelism in $N$. Note that $|T|$ can always be taken as such a bound, but in most cases it is far too large.

As introduced in section IV, we consider a log $L = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ to be a sample from the probability space, and $Z_i(a, b) = Z(\sigma_i(a, b))$. We define for $a, b \in T$ predicates

1. $Q_1(a, b) \equiv \exists i \in \{1, 2, \ldots, n\} : Z_i(a, b) \in \{1, 2\}$,
2. $Q_2(a, b) \equiv \exists i \in \{1, 2, \ldots, n\} : Z_i(a, b) \in \{3, 4\}$,
3. $Q_3(a, b) \equiv \exists i \in \{1, 2, \ldots, n\} : Z_i(a, b) = 2$,
4. $Q_4(a, b) \equiv \{i \in \{1, 2, \ldots, n\} : Z_i(a, b) = 2\} \geq l$.

where $Q_2(a, b) = Q_1(b, a)$, $Q_3(a, b) \Leftrightarrow a \triangleright b$, and $l$ is a parameter to be determined later in this section.

We introduce a notion of log completeness targeted at the T-workflow nets:

Definition 9 (T-complete log). A log $L$ of an acyclic T-workflow net $N$ is T-complete if for all $a, b \in T$, $a \parallel b$ implies $Q_1(a, b) \land Q_2(a, b)$, and $a \rightarrow b$ implies $Q_3(a, b) \land -Q_2(a, b)$.

Note that $a \parallel b$ and $a \rightarrow b$ hold for all possible traces, so the subscript $L$ in Definition 6 is omitted.

Lemma 1. Completeness implies T-completeness but not vice versa.

Proof: Suppose $L$ is a complete log for an acyclic T-workflow net $N$. By Definition 7, we have $\triangleright \subseteq \triangleright_L$. Since $\triangleright_L \subseteq \triangleright$, we have $\triangleright = \triangleright_L$. Therefore, for all $a, b \in T$, we have $a \parallel b \iff a \triangleright b, b \triangleright a \iff a \triangleright_L b$ and $b \triangleright_L a$ implies $Q_1(a, b) \land Q_2(a, b)$. Further note that $a \rightarrow b \iff a \triangleright b, \neg Q_2(a, b) \iff a \triangleright_L b$, and $\neg Q_2(a, b)$ implies $\Rightarrow Q_3(a, b) \land -Q_2(a, b)$. So completeness implies T-completeness.

In order to show that the converse does not hold, consider the net $L_1 = \{(A, B, C, D, E, F), (A, D, E, B, C, F)\}$ and Figure 2. For T-completeness we need $\triangleright \subseteq \triangleright_L$, which is not true, since $D \triangleright C$ does not hold on $L_1$, nor does $B \triangleright E$.

Hence, T-completeness is a weaker condition than completeness, which in fact also follows from the equivalence $a \triangleright b \equiv (a \parallel b) \lor (a \rightarrow b)$.

Given a T-complete log of a net $N = (P, T, F, i, f)$, we present an algorithm for mining process. The algorithm uses the fact that two tasks are connected if and only if their

Fig. 1. A choice fragment

Fig. 2. An acyclic T-workflow net
causality can be detected by inspecting the log. Note that we drop the dependency on $N$ in the notation. Recall that $i$ and $j$ are the initial and final places, respectively, and we assume that $T$ is known.

**Algorithm 4** Construct an acyclic T-workflow net from a fixed T-complete log

1: $P' := \{i, f\}$, $T' := T$, $F' := \emptyset$
2: for all $a \in T'$ do
3:     if $\neg\exists t \subseteq T' : Q_1(t, a)$ then
4:         $F' := F' \cup \{(i, a)\}$
5:     end if
6:     if $\neg\exists r \subseteq T' : Q_1(a, r)$ then
7:         $F' := F' \cup \{(a, f)\}$
8:     end if
9: end for
10: for all $a, b \in T'$ do
11:     if $Q_3(a, b) \land \neg Q_2(a, b)$ then
12:         $F' := F' \cup \{(a, p), (p, b)\}$
13:     end if
14: end if
15: end for
16: reconstructed net $N' = (P', T', F', i, f)$

The following theorem proves that Algorithm 4 is able to reconstruct (up to bisimilarity) a generating net $N' = (P', T', F', i, f)$ out of a given T-complete log.

**Theorem 3.** Let $L$ be a T-complete log for an acyclic T-workflow net $N$. Then the net $N'$ computed by Algorithm 4 is bisimilar to $N$.

**Proof:** Clearly, $a \triangleright_L b$ can only hold either in the case of $a \rightarrow b$ or in the case of $a \parallel b$. By Theorem 1, $a \rightarrow b$ implies that there is a place between transitions $a$ and $b$. The second case is not possible due to step 4 of Algorithm 4. Therefore, $N'$ contains a place between transitions $a$ and $b$ iff $a \rightarrow b$.

Algorithm 4 can discover the exact process model from T-complete logs. In most cases, however, we cannot be sure that the log is T-complete. In Algorithm 5 we propose a strategy for dealing with the logs that are possibly not T-complete: there we only add a place between two transitions $a$ and $b$ if we are certain enough that there is a causal relation between the two event, namely, $b$ never happens before $a$ in the log and $a$ happens immediately before $b$ enough times. Enough times is the parameter $l$ in $Q_4(a, b)$.

Algorithm 5 is able to reconstruct (up to bisimilarity) a generating net $N' = (P', T', F', i, f)$ out of a given log. Our decisions in Algorithm 5, however, have a probabilistic nature and can lead to two kinds of possible errors, false positives and false negatives.

**False positive** is an error such that the log inspection leads to the conclusion $a \rightarrow b$, but in fact we have $a \parallel b$ in the underlying net. Note that if $a$ is indirectly in front of $b$ in the net (some transition $c$ lies in between), we can never conclude $a \rightarrow b$, since we will never observe $Z_i(a, b) = 2$ because of the firing of other transition(s) in between. Consequently, we only have to consider the case of $a \parallel b$. The probability of a false positive is $P_{a \parallel b}[-Q_2(a, b) \land Q_4(a, b)]$. An upper bound for this probability is (see (25) in Appendix for detailed derivation):

$$P_{a \parallel b}[-Q_2(a, b) \land Q_4(a, b)] \leq \frac{1}{2^l}. \quad (12)$$

To ensure a confidence limit $\alpha$, we require $\frac{1}{2} \leq \alpha$. Therefore, $l$ can be computed as follows:

$$l \geq -2\log \alpha. \quad (13)$$

With (13), we are able to determine the minimal number $l$ of occurrences of $a \triangleright b$ in $Q_4(a, b)$ for Algorithm 5. For each pair $a \rightarrow b$, the probability of false positive is limited by $\alpha$.

Now, knowing a probabilistic bound for the false positive for a specific pair of transitions $(a, b)$, let us consider the set $R$ of all pairs of parallel transitions in the net, i.e. $R = \{(a, b) \in T \times T : a \parallel b\}$. Note that we do not know $R$.

If the random variables $Z_n(a, b)$ and $Z_n(c, d)$, for $n = 1, 2, 3, \ldots$ and $(a, b) \neq (c, d)$, are maximally dependent, then the probability for one false positive is the same as the probability for at least one false positive for the pairs of $R$. The other extreme is that all random variables $Z_n(a, b)$ for $(a, b) \in R$ are independent. Then $-Q_2(a, b) \land Q_4(a, b)$ are independent events. This is the worst case as we have to investigate all the pairs. In this case, the probability of having at least one false positive can be derived using Lemma 2 (see Appendix):

$$P_R[\exists (a, b) \in R : -Q_2(a, b) \land Q_4(a, b)] = 1 - (1 - P_{a \parallel b}[-Q_2(a, b) \land Q_4(a, b)])^{|R|}. \quad (14)$$

Based on (12), we derive

$$P_R[\exists (a, b) \in R : -Q_2(a, b) \land Q_4(a, b)] \leq 1 - (1 - \frac{1}{2^l})^{|R|}. \quad (15)$$
To ensure confidence limit \( \alpha \), we let \((15) \leq \alpha \). Consequently, we choose \( l \) as the smallest value such that
\[
l \geq 2 \frac{\log(1 - (1 - \alpha)^{1/n})}{\log 10}.
\] (16)

We have to provide an estimate for the number of independent pairs in \( R \). One option for such an estimate is the maximal parallelism of the process (estimated by a e.g. business analyst).

False negatives manifest themselves when the log inspection results in \( Q_1(a, b) \land \neg Q_4(a, b) \) and we conclude that \( a \) is indirectly in front of \( b \) in the underlying net, while in reality we have \( a \rightarrow b \) in the net. Note that given \( a \rightarrow b \), we never observe \( Q_2(a, b) \) as \( b \) can never be fired before \( a \). The probability for a false negative is:
\[
P_{a \rightarrow b}([\{i \mid Z_i(a, b) = 2]\}] < l].
\] (17)

Given \( a \rightarrow b \), we can only observe \( Q_1(a, b) \) in the log (with \( n \) traces), implying that either \( Z_i(a, b) = 1 \) or \( Z_i(a, b) = 2 \). Hence, probability (17) has a binomial distribution \( B(n, P_{a \rightarrow b}[Z_i(a, b) = 2]) \). Let \( p \) be \( P_{a \rightarrow b}[Z_i(a, b) = 2] \), then (17) becomes:
\[
P_{a \rightarrow b}([\{i \mid Z_i(a, b) = 2]\}] < l] = \sum_{m=0}^{l-1} \binom{n}{m} p^m (1 - p)^{n-m}.
\] (18)

In order to simplify the calculation in practice, we use the normal distribution for approximating (18). We obtain:
\[
\sum_{m=0}^{l-1} \binom{n}{m} p^m (1 - p)^{n-m} = \Pr[X \leq l - 1]
\]
\[
= \Pr\left[\frac{X - np}{\sqrt{np(1-p)}} \leq \frac{l - 1 - np}{\sqrt{np(1-p)}}\right]
\]
\[
\approx \Pr[U \leq \frac{l - 1 - np}{\sqrt{np(1-p)}}].
\] (19)

Let \( U \) have a standard normal distribution with zero mean and unit variance, such that \( U \sim N(0, 1) \). We require the probability \( \Pr[U \leq \frac{l - 1 - np}{\sqrt{np(1-p)}}] \leq 0.05 \). From the table of cumulative standard normal distribution [15], we get \( \Pr(U \leq -1.64) = 0.05 \). Then we have \( \frac{l - 1 - np}{\sqrt{np(1-p)}} = -1.64 \). Consequently, we can compute \( n \) as follows.
\[
n = \frac{l + 0.3448 - 1.3448p}{p} + 0.82\sqrt{2.6896p^2 - (1.3792 + 4l)p + 4l - 1.3104}
\] (20)

If \( p \) decreases, then \( n \) increases accordingly.

Note that the structural properties of the net greatly influence the size of a log that is complete with a given confidence. Here we only consider two extreme patterns:

**Pattern 1.** Figure 3 shows an acyclic T-workflow net with \( k \) transitions. Initial transition \( A \) has \( k - 2 \) successors (\( A \rightarrow B \)). Suppose that when \( A \) fires, each of the \( k - 2 \) enabled transitions can be fired with the probability of \( \frac{1}{k-2} \). Consequently, \( \Pr[A B] = 2] = \frac{1}{k-2} \). We can estimate the log size \( n \) using formula (20). For example, if \( l = 10 \) and \( k = 30 \), then \( n \approx 430 \). This implies that statistically we need just 430 traces in the log to guarantee with the 95% probability that the discovered model is the one that has generated the log.

**Pattern 2.** Figure 4 shows another acyclic T-workflow net with \( k \) transitions. There are two parallel branches, one with \( k - 3 \) transitions, the other one with just one transition (transition \( A \)). To discover the exact process model, we need to observe transition \( A \) immediately before the final transition \( B \). As transition \( A \) has to be fired after all the \( k - 3 \) transitions of the parallel branch, we derive \( \Pr[A B] = 2] = \frac{1}{k-3} \). We use (20) to estimate log size \( n \). For instance, for \( l = 10 \) and \( k = 30 \), then \( n \approx 2.07 \times 10^5 \). This implies that statistically we need \( 2.07 \times 10^5 \) traces in a log to guarantee with the confidence of 95% that the discovered model is the one that has generated the log.

We have introduced Algorithm 4 and Algorithm 5 to reconstruct an acyclic T-workflow net from a fixed log (i.e., no more new traces are added to the log). Now we consider dynamic situations, when new traces may be added to a log until for all pairs \( a, b \in T \) one of the following conditions holds: 1) \( Q_1(a, b) \land Q_2(a, b) \), 2) \( \neg Q_2(a, b) \land \neg Q_3(a, b) \), 3) \( \neg Q_2(a, b) \land Q_3(a, b) \). Only with this information, we can decide what kind of relation we have between transitions \( a \) and \( b \). We can conclude \( a \parallel L b \) (even \( a \parallel b \) from condition 1), and \( a \rightarrow L b \) from condition 3). Algorithm 6 generates T-complete logs for acyclic T-workflow nets in dynamic situations.

Note that in Algorithm 6, we start with a non-empty log, \( \sigma \) is a new trace, and the condition in the while loop is the negation of the disjunction of the three aforementioned conditions (i.e., \( \neg((Q_1(a, b) \land Q_2(a, b)) \lor (\neg Q_2(a, b) \land \neg Q_3(a, b)) \lor \neg Q_2(a, b) \land Q_3(a, b)) \lor \neg Q_2(a, b) \land Q_3(a, b) \)).
Algorithm 6 Generate a T-complete log for an acyclic T-workflow net with certain confidence

1: $L := L_0 \neq \emptyset$
2: while $\exists a, b \in T : (Q_2(a, b) \land \neg Q_4(a, b)) \lor (\neg Q_2(a, b) \land Q_4(a, b))$ do
3: Generate next trace $\sigma$
4: $L := L \cup \{\sigma\}$
5: end while

($\neg Q_2(a, b) \land Q_4(a, b)$). The loop terminates when one of the three conditions holds. Once a T-complete log has been generated, we can apply Algorithm 5 to reconstruct the generating net.

VIII. Empirical results

In order to evaluate the quality of our probabilistic bounds for log completeness empirically, we developed a software tool to generate a random benchmark set of workflow nets. Then we randomly generate a number of logs for every workflow net. Note that all traces are independent random variables and we are able to check for an arbitrary log whether this log is complete since we know the generating workflow net. Due to the limited space, we only show part of the results in this paper.

A. Bounds for sound workflow nets

First, we focus on the probabilistic bounds for log completeness obtained using Algorithm 2 both with formula (8) and with formula (10). We generate structured Jackson nets [11], which form a subset of sound workflow nets. We generated 100 structured Jackson nets, each with 30 transitions. We let $\alpha = 2.5\%$. For each of the nets we generated 100 logs. In order to compare the bounds computed using (8) and (10), each log was generated in the following way: By Algorithm 2, we started with an empty log, and generated traces for the log. After adding a new trace, we used both (8) and (10) to check on-the-fly whether we could stop the log generation. If one formula told us to stop, then we collected the statistics of interest and continued generating traces until the second formula told us to stop as well. Then we collected the statistics again and stopped generating the log. Note that in some cases the log generation stops before we have a complete log (we do not provide 100% guarantee of log completeness), while in other cases the generation still continues when we already have a complete log, since we do not have enough confidence in the log completeness yet. In the latter case we keep the information about the minimal length of the complete log to analyze the degree of redundancy we have on average with our method.

For every generated net $N_i$ and every generated complete log $l_{ij}$ (i.e., the $j^{th}$ generated complete log of $N_i$), where $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, m\}$, and $k = m = 100$ in this experiment, we have the following test statistics:

- Number of causal pairs ($C_i$): the total number of causal pairs in net $N_i$.
- Log size ($L_{ij}$): the size of the log $l_{ij}$ generated by Algorithm 2 with either formula (8) or formula (10).
- Reliability ($R_{ij}$): 1 when $L_{ij}$ is complete, and 0 when not.
- Trace difference ($D_{ij}$): $D_{ij}$ is the ratio of the size of the minimal complete log forming a prefix of $l_{ij}$ to the size of $l_{ij}$.
- Missing pairs ($M_{ij}$): if $l_{ij}$ is not complete, then $M_{ij}$ is the ratio of the number of causal pairs which are not covered by $l_{ij}$ to the number $C_i$ of causal pairs.
- Log size for 80% of causal pairs ($E_{ij}$): the ratio of the number of traces in the minimal prefix of $l_{ij}$ which covers any 80% of all causal pairs to $L_{ij}$.

For each net $N_i$, we take the average $R_i$, $L_i$, $D_i$, $M_i$, $E_i$ over $R_{ij}, L_{ij}, D_{ij}, M_{ij}, E_{ij}$, respectively, i.e., $R_i = \frac{1}{m} \sum_{j=1}^{m} R_{ij}$, etc.

The empirical outcomes for (8) and (10) are listed in Table I and Table II, respectively. In both cases we used the same benchmark of workflow nets. The main conclusions we can extract are as follows.

The use of (10) instead of (8) in Algorithm 2 results in a more reliable probabilistic bound. For each net, the value of $R$ in Table I is never above the one in Table II. The average $R$ is increased from 0.93 (by using (8)) to 0.96 (by using (10)). Naturally, Algorithm 2 with (10) always results in a larger log than using (8).

For complete logs generated by Algorithm 2 with (8), on average all causal pairs are covered by the first 37% of all of the traces in this log. The rest of the traces in the log do not contribute any more in terms of completeness. For Algorithm 2 with (10), this percentage is 31%.

For incomplete logs (i.e., logs not covering all causal pairs) generated by Algorithm 2 with (8), on average, 20% of all of the causal pairs are missing in the log. For Algorithm 2 with (10), this percentage is decreased to 15%.

Finally, in order to cover a majority (80%) of all causal pairs in a net, on average we need just the first 11% of a log generated by Algorithm 2 using (8). For Algorithm 2 with (10), this percentage is 9%.

B. Bounds for S-workflow nets

In the next experiment we focus on the probabilistic bounds for log completeness obtained for S-workflow nets with Algorithm 3. We generated 100 S-workflow nets, each with 50 transitions. Note that S-workflow nets are always sound. We take $\alpha = 5\%$, and for each of the generated nets we generate 100 logs using Algorithm 3. We use the same test statistics in Subsection VIII-A, but we naturally change the definition of Log size ($L_{ij}$) to the size of the complete log generated by Algorithm 3. Table III presents the empirical outcomes.

The average of $R$ is 0.96. This implies that for an S-workflow net Algorithm results in 96 complete logs out of 100 logs, on average. In a complete log, on average the first 44% (the average of $D$) of all of the traces in this log cover all causal pairs, and the rest of the log does not contribute
### TABLE I

EMPirical results for 100 Structured Jackson nets by Algorithm 2 with formula (8)

<table>
<thead>
<tr>
<th>Number of Causal Pairs (C)</th>
<th>Reliability (R)</th>
<th>Log Size (L)</th>
<th>Trace Difference (D)</th>
<th>Missing Pairs (M)</th>
<th>Log Size for 80% Causal Pairs (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Mean</td>
<td>Std.Dev.</td>
<td>Mean</td>
<td>Std.Dev.</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td>0.97</td>
<td>152</td>
<td>0.40</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>382</td>
<td>1</td>
<td>1583</td>
<td>0.44</td>
<td>0.06</td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>0.69</td>
<td>38</td>
<td>0.40</td>
<td>0.13</td>
</tr>
<tr>
<td>4</td>
<td>66</td>
<td>0.73</td>
<td>127</td>
<td>0.27</td>
<td>0.07</td>
</tr>
<tr>
<td>5</td>
<td>58</td>
<td>0.73</td>
<td>190</td>
<td>0.28</td>
<td>0.05</td>
</tr>
<tr>
<td>6</td>
<td>103</td>
<td>0.92</td>
<td>2246</td>
<td>0.41</td>
<td>0.08</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>0.91</td>
<td>176</td>
<td>0.38</td>
<td>0.22</td>
</tr>
<tr>
<td>8</td>
<td>54</td>
<td>0.95</td>
<td>233</td>
<td>0.40</td>
<td>0.23</td>
</tr>
<tr>
<td>9</td>
<td>94</td>
<td>0.91</td>
<td>865</td>
<td>0.35</td>
<td>0.08</td>
</tr>
<tr>
<td>10</td>
<td>89</td>
<td>0.98</td>
<td>879</td>
<td>0.42</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>Avg.</td>
<td>84</td>
<td>0.93</td>
<td>920</td>
<td>0.37</td>
</tr>
</tbody>
</table>

### TABLE II

EMPirical results for 100 Structured Jackson nets by Algorithm 2 with formula (10)

<table>
<thead>
<tr>
<th>Number of Causal Pairs (C)</th>
<th>Reliability (R)</th>
<th>Log Size (L)</th>
<th>Trace Difference (D)</th>
<th>Missing Pairs (M)</th>
<th>Log Size for 80% Causal Pairs (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Mean</td>
<td>Std.Dev.</td>
<td>Mean</td>
<td>Std.Dev.</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td>0.99</td>
<td>184</td>
<td>0.33</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>382</td>
<td>1</td>
<td>1828</td>
<td>0.38</td>
<td>0.05</td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>0.77</td>
<td>47</td>
<td>0.33</td>
<td>0.11</td>
</tr>
<tr>
<td>4</td>
<td>66</td>
<td>0.77</td>
<td>178</td>
<td>0.22</td>
<td>0.05</td>
</tr>
<tr>
<td>5</td>
<td>58</td>
<td>0.77</td>
<td>261</td>
<td>0.22</td>
<td>0.03</td>
</tr>
<tr>
<td>6</td>
<td>103</td>
<td>0.97</td>
<td>2839</td>
<td>0.34</td>
<td>0.19</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>0.97</td>
<td>226</td>
<td>0.31</td>
<td>0.17</td>
</tr>
<tr>
<td>8</td>
<td>54</td>
<td>0.97</td>
<td>282</td>
<td>0.34</td>
<td>0.19</td>
</tr>
<tr>
<td>9</td>
<td>94</td>
<td>0.98</td>
<td>1090</td>
<td>0.31</td>
<td>0.02</td>
</tr>
<tr>
<td>10</td>
<td>89</td>
<td>0.98</td>
<td>1065</td>
<td>0.35</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Avg.</td>
<td>84</td>
<td>0.96</td>
<td>1148</td>
<td>0.31</td>
</tr>
</tbody>
</table>

### TABLE III

EMPirical results for 100 S-workflow nets by Algorithm 3

<table>
<thead>
<tr>
<th>Number of Causal Pairs (C)</th>
<th>Reliability (R)</th>
<th>Log Size (L)</th>
<th>Trace Difference (D)</th>
<th>Missing Pairs (M)</th>
<th>Log Size for 80% Causal Pairs (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Mean</td>
<td>Std.Dev.</td>
<td>Mean</td>
<td>Std.Dev.</td>
</tr>
<tr>
<td>1</td>
<td>81</td>
<td>1</td>
<td>1022</td>
<td>0.33</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>137</td>
<td>0.92</td>
<td>417</td>
<td>0.62</td>
<td>0.09</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
<td>0.86</td>
<td>179</td>
<td>0.66</td>
<td>0.14</td>
</tr>
<tr>
<td>4</td>
<td>135</td>
<td>0.97</td>
<td>380</td>
<td>0.43</td>
<td>0.08</td>
</tr>
<tr>
<td>5</td>
<td>86</td>
<td>0.86</td>
<td>263</td>
<td>0.55</td>
<td>0.07</td>
</tr>
<tr>
<td>6</td>
<td>84</td>
<td>1</td>
<td>671</td>
<td>0.25</td>
<td>0.04</td>
</tr>
<tr>
<td>7</td>
<td>95</td>
<td>1</td>
<td>1833</td>
<td>0.26</td>
<td>0.04</td>
</tr>
<tr>
<td>8</td>
<td>91</td>
<td>0.98</td>
<td>1163</td>
<td>0.49</td>
<td>0.06</td>
</tr>
<tr>
<td>9</td>
<td>88</td>
<td>1</td>
<td>276</td>
<td>0.44</td>
<td>0.1</td>
</tr>
<tr>
<td>10</td>
<td>136</td>
<td>0.82</td>
<td>577</td>
<td>0.72</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>Avg.</td>
<td>95</td>
<td>0.96</td>
<td>847</td>
<td>0.44</td>
</tr>
</tbody>
</table>
anymore in terms of log completeness. For an incomplete log, on average just 1 causal pair (the average of $M$) is missing in the log. On average, the first 7% of all of the traces in a log cover a majority (80%) of all causal pairs in a net. Therefore, we conclude that the probabilistic bound for log completeness given by Algorithm 3 has a good quality.

We compared the results obtained for the S-nets using Algorithm 2 and using Algorithm 3. Algorithm 2 required the construction of logs that were on average 20% longer than Algorithm 3. Note that with algorithm 3 we achieve the targeted reliability of 0.95—it is 0.96 on average on our benchmark. Thus we succeeded in improving the probabilistic bound for the log size, taking into account the net structure indeed.

C. Bounds for acyclic T-workflow nets

In the last experiment, we concentrate on the probabilistic bound obtained with Algorithm 6 for acyclic T-workflow nets. We generated 100 acyclic T-workflow nets, each with 30 transitions. We take $l = 10$. We generate 100 logs by Algorithm 6 for each of the generated nets. For every generated net $N_i$ and every generated T-complete log $l_{ij}$, where $i \in \{1,\ldots,k\}$, $j \in \{1,\ldots,m\}$, and $k = m = 100$ in this experiment, we have the following test statistics:

- Number of direct successor pairs ($S_i$): the total number of pairs of transitions that have the direct successor relation (Definition 6) in $N_i$,
- Log size ($L_{ij}$): the log size for $l_{ij}$ computed with Algorithm 6,
- False positive ($A_{ij}$): the number of pairs of transitions that do not have direct successor relation in reality the net but considered to have direct successor relation in $l_{ij}$ by Algorithm 6,
- False negative ($B_{ij}$): the number of pairs of transitions that do have direct successor relation in the net but considered not to have it in $l_{ij}$ by Algorithm 6,
- Reliability ($R_{ij}$): whether or not $l_{ij}$ covers exactly the direct successor pairs of $N_i$, i.e. it is both false positive free and false negative free.

For each net $N_i$, we compute the averages $R_i, L_i, A_i, B_i$ of $R_{ij}, L_{ij}, A_{ij}, B_{ij}$ over $j$, respectively, i.e., $R_i = \frac{1}{m} \sum_{j=1}^{m} R_{ij}$, etc.

The empirical outcomes are shown in Table IV. The average result of $R$ is 0.93, which is above 0.90 we aimed at in this series of experiments. The average result of $A$ (0.016) is much smaller than the average result of $B$ (1.166). This implies that if a log generated by Algorithm 6 is not complete, then it is much more likely that the log misses some direct successor pairs (a false negative) than that we erroneously diagnose pairs as being in the direct successor relation (a false positive).

We compared the results obtained for the T-nets using Algorithm 2 and by Algorithm 5. Algorithm 2 required the construction of logs that were on average 6 times longer than Algorithm 5. Note that with algorithm 5 we achieved the targeted reliability (and even overperformed it) on average. Thus taking into account the information about the structure of T-workflow nets significantly improves the probabilistic bound for the log size.

We also observe a number of outliers with respect to the Reliability ($R$). Consider the $9^{th}$ net in Table IV as such an example. For this net only 52 of the 100 generated logs are complete, and in every of the 48 incomplete logs there is exactly one direct successor pair missing. By inspecting the net, we found a variant of Pattern 2 for false negative (see Section VII). Figure 5 shows a part of net 9 and highlights this pattern in a black rectangle. As $l$ is 10 and there are 13 transitions in this highlighted subset, from (20) we get $n \approx 15812$. This implies that in theory we need a log with a minimal size of 15812 to determine $t_{29} \rightarrow t_{23}$. On the other hand, our empirical results show that on average the log size for net 9 is 7962 (standard deviation is 7707). Our assumptions for this net are thus overoptimistic. Consequently, $t_{29} \rightarrow t_{23}$ cannot be determined from many logs, resulting in incomplete logs. The same pattern was observed in all other outliers.

Interestingly, this pattern reflects the situation that can also be observed in real life. When one of the parallel branches is shorter then the other(s) (possibly not in the terms of the number of transitions but in the sense of execution time), the probability to observe the pair of type $(A, B)$ (see Figure 4) is extremely low, or even 0 (when a timed process is considered).

IX. RELATED WORKS

In [5], a finite state machine method, a neural network method, and a Markov approach were proposed for process discovery in case of software engineering processes. These methods concentrated on sequential processes. The methods were extended in [6] to detect concurrent processes by a number of specific metrics, i.e., event type counts, periodicity and causality. In [13] and [12], a hidden Markov model was employed in the context of workflow management, in the case of sequential processes and concurrent processes, respectively. In the works aforementioned, the focus was on identifying the dependency relations between events.

It is obvious that the mining algorithms proposed in this paper are adopted from control-flow mining algorithms, i.e., Alpha algorithm. In [14] a method for genetic process mining was proposed. Such a process mining algorithm is capable of discovering all common control-flow structures (i.e. sequences, choices, parallelism, loops, non-free-choices, invisible tasks, and duplicate tasks as well) while being robust to noisy logs. This method, however, has more difficulties to mine models with constructs that allow for many interleaving situations. These two algorithms both assume the event log is complete.

In [17] and [10], a heuristic mining approach and a fuzzy mining approach were proposed, respectively. They both employ heuristics to limit the set of precedence relations included in a model. The heuristic miner is a practical applicable mining algorithm that can handle noise. It includes three steps: construction of dependency graph, construction of input and output expressions for each activity, and search for long
distance dependency relations. The fuzzy miner assumes that problems in mining large scale processes are caused by mismatch between fundamental assumptions of traditional process mining and the characteristics of real-life processes. In this approach, the authors developed an adaptive simplification and visualization technique for process models, which is based on two metrics, significance and correlation. The two metrics are similar to the concept of data clustering domain where a binary distance metric is inferred to find related subsets of attributes. Significance can be determined both for tasks and precedence relations over them. It measures the relative importance of behaviour. As such, it specifies the level of interest in tasks and their control dependency. Correlation is only relevant for precedence relations over tasks, which measures how closely related two events following one another. The heuristic miner and the fuzzy miner only express the main behaviours registered in an event log. They lose absolute precision and cannot describe the complete behaviours.

X. Conclusion

In this paper, we address the problem of log completeness in the context of process mining. Our method give a probabilistic bound of log completeness, also implying the same probabilistic bound that the model discovered on the process log is an exact representation of the process that has generated this log. We considered three classes of nets: sound structured workflow nets, S-workflow nets, and T-workflow nets. While for the first two classes we use the \( \alpha \)-algorithm for model discovery, we propose a new algorithm for the discovery of T-workflow nets. In our empirical studies, we showed that our method provides the promised quality of model discovery.

In the future we plan to find patterns of “dangerous” net structures (like our Pattern 2) that would allow us to improve the probabilistic bounds for the subclasses that currently manifest themselves as outliers. We are also interested in “positive” patterns, which would allow us to lower the probabilistic bounds for nets matching these patterns, since our probabilistic bound can clearly be further lowered for some classes of nets. We also want to investigate the applicability of Bayesian methods for the log completeness problem.

References

APPENDIX

In this final part we present a lemma and detailed derivation of (3), (11), (12) in this paper.

Lemma 2. \( \mathbb{P} \{ \exists i \in \{1, 2, \ldots, n\} : A_i \} = 1 - (1 - \mathbb{P}[A_i])^n \), where \( A_i \) are independent events.

Proof:

\[
\mathbb{P} \{ \exists i \in \{1, \ldots, n\} : A_i \} = 1 - \prod_{i=1}^{n} \mathbb{P}[\neg A_i] = 1 - (1 - \mathbb{P}[A_i])^n
\]

Formulae (21), (22), and (23) derive (3).

\[
Y^n = f(\pi)
\]

\[
\equiv \forall a, b : f(\pi)(a, b) = 0 \Rightarrow Y^n(a, b) = 0
\]

\[
\land \forall a, b : Y^n(a, b) = 0 \Rightarrow f(\pi)(a, b) = 0
\]

\[
\equiv \forall a, b : Y^n(a, b) = 0 \Rightarrow f(\pi)(a, b) = 0
\]

\[
\equiv \neg(\exists a, b : \neg Y^n(a, b) = 1 \lor f(\pi)(a, b) = 0)
\]

\[
\equiv \neg(\exists a, b : Y^n(a, b) = 0 \land f(\pi)(a, b) = 1)
\]

Formula (24) derives (11).

\[
\mathbb{E}_\pi \left[ \sum_{a,b} (1 - \delta(0, Y^n(a,b))) (1 - Y^n(a,b))^k \right]
\]

\[
= \sum_{a,b} \left( \mathbb{E}_\pi \left[ (1 - Y^n(a,b))^k \right] - \mathbb{E}_\pi \left[ \delta(0, Y^n(a,b)) (1 - Y^n(a,b))^k \right] \right)
\]

\[
= \sum_{a,b} \left( \mathbb{E}_\pi \left[ (1 - Y^n(a,b))^k \right] - \mathbb{E}_\pi \left[ Y^n(a,b) = 0 \right] \right)
\]

(due to Jensen’s inequality, we derive)

\[
\geq \sum_{a,b} \left( (1 - \mathbb{E}_\pi \left[ Y^n(a,b) \right])^k - \mathbb{E}_\pi \left[ Y^n(a,b) = 0 \right] \right)
\]

\[
= \sum_{a,b} ((1 - q_{ab})^k - (1 - q_{ab})^n) \leq \sum_{(a,b)\mid q_{ab}>0} (1 - q_{ab})^k
\]

Formula (25) derives (12).

\[
\mathbb{P}_{a\parallel b}[\neg \exists Q_2(a, b) \land \exists Q_2(a, b)]
\]

\[
= \mathbb{P}_{a\parallel b}[\{i|Z_i(a,b) \in \{3, 4\}\}] = 0 \land \{i|Z_i(a,b) = 2\}] \geq f
\]

\[
\leq \mathbb{P}_{a\parallel b}[\{i|Z_i(a,b) = 2\}] \geq f
\]

\[
= \sum_{m=1}^{n} \mathbb{P}_{a\parallel b}[\{i|Z_i(a,b) = 2\}] = 0 \land \{i|Z_i(a,b) = 2\}] = m
\]

\[
= \sum_{m=1}^{n} \mathbb{P}_{a\parallel b}[\{i|Z_i(a,b) \in \{2, 4\}\}] = m
\]

\[
= \sum_{m=1}^{n} \mathbb{P}_{a\parallel b}[\{i|Z_i(a,b) \in \{2, 4\}\}] = m
\]

\[
= \sum_{m=1}^{n} \frac{1}{2^n} \mathbb{P}_{a\parallel b}[\{i|Z_i(a,b) \in \{2, 4\}\}] = m
\]

\[
\leq \frac{1}{2^n} \sum_{m=1}^{n} \mathbb{P}_{a\parallel b}[\{i|Z_i(a,b) \in \{2, 4\}\}] = m = \frac{1}{2^n}
\]