A Framework for the Analysis of Process Mining Algorithms

Philip Weber, Behzad Bordbar, and Peter Tiño

***Abstract*—There are many process mining algorithms and representations, making it difficult to choose which algorithm to use or compare results. Process mining is essentially a machine learning task, but little work has been done on systematically analyzing algorithms to understand their fundamental properties, such as how much data are needed for confidence in mining. We propose a framework for analyzing process mining algorithms. Processes are viewed as distributions over traces of activities and mining algorithms as learning these distributions. We use probabilistic automata as a unifying representation to which other representation languages can be converted. We present an analysis of the Alpha algorithm under this framework and experimental results, which show that from the substructures in a model and behavior of the algorithm, the amount of data needed for mining can be predicted. This allows efficient use of data and quantification of the confidence which can be placed in the results.**

***Index Terms*—Business processes, machine learning, Petri nets, probabilistic automata, process mining.**

I. Introduction

BUSINESS processes describe activities carried out to fulfill a business function, such as providing a service or producing a product. Processes may be designed to dictate work patterns, or result *de facto* from working practice. Either way, as activities take place, systems involved record information in workflow (WF) logs. Process mining [1], [2] uses these logs to discover and analyze models of business processes.

Fig. 1 shows the "control flow" of a simple example process, as a Petri net. An order is received, stock checked, and the item is picked from the warehouse, or the order rejected. Despatch and billing take place in parallel. After checking payment, a receipt is issued or payment chased, before closing the order. Abstracting from detail, the "trace" of one possible enactment of the process might be recorded in a WF log as a string "*iabdefgo*." We use this process as a running example.

Process *discovery* algorithms use logs of such traces to produce models of process control flow. Process mining also addresses performance analysis [3], troubleshooting, auditing conformance [4]–[6], mining decision rules [7], or interaction between resources [8]. A current focus is on managing complex processes or logs, by abstracting from detail or separating multiple processes recorded together [3], [9]–[13].

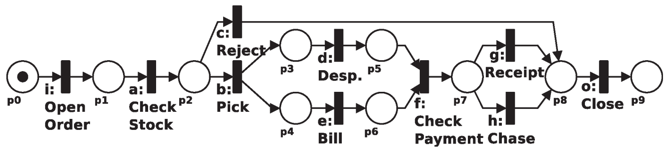

Fig. 1. Simplified business process for fulfilling an order (Petri net N_0).

Process mining is therefore a wide-ranging tool set, interfacing between business users and the highly complex and multifaceted real-world behavior of businesses. This behavior is manifested in designed or *de facto* business processes, evidence for which is found in possibly complex, detailed logs. Process mining aims to enable understanding of process behavior and in this way to facilitate decision-making to control and improve that behavior.

Process discovery is essentially a machine learning task. However, little work has been done on systematically analyzing process mining algorithms in this context, to discover their fundamental properties, or to answer questions such as how much data is necessary for mining. Yet, such understanding is of critical importance to give confidence that a log file is an adequate sample of the true behavior, and thus in the correctness of the mined model.

There are many process discovery algorithms, e.g., [14]–[20], and various representation notations. Since the core interest is in process control flow, many algorithms learn only the process structure, without attempting to recover probabilities. Probabilities in the model may be of interest, as where business rules restrict the frequency of costly patterns of activity. However, even where there is no interest in producing a probabilistic model, it must be appreciated that traces are generated randomly according to an underlying probability distribution unknown to the mining algorithm. Not all activities or decisions are equally likely, and their probabilities may have a dramatic effect on the amount of data needed for mining.

Because of this diversity of algorithms and representations, methods are needed for analyzing the behavior of algorithms. This paper aims to introduce a framework for one such method. Given a probability and a process mining algorithm, how much data of a given, finite process do we need to, with a stated probability, produce a business process "close enough" to the original? There are two main prerequisites to answering this question. First, a unifying view of processes to allow objective, language-independent analysis, and second, a notion of "closeness" to evaluate how similar two processes are.

To satisfy these requirements, we consider business processes as probability distributions over traces of activities and

Manuscript received September 16, 2011; revised January 26, 2012; accepted April 1, 2012. P. Weber is supported by a Doctoral Training Grant funded by EPSRC and the School of Computer Science. This paper was recommended by Associate Editor M. Jeng.

The authors are with the School of Computer Science, University of Birmingham, B15 2TT Birmingham, U.K. (e-mail: p.weber@cs.bham.ac.uk; b.bordbar@cs.bham.ac.uk; p.tino@cs.bham.ac.uk).

Digital Object Identifier 10.1109/TSMCA.2012.2195169

mining algorithms in terms of their ability to learn such distributions. We use probabilistic automata (probabilistic deterministic finite automata (PDFA) [21]) as a unifying representation, as these represent a large class of probability distributions over sequences and act as a lowest common denominator to which to convert models in other languages. The distance between processes can be calculated from PDFA with various metrics. In this paper, we use the d_2 distance, and metrics based on the Bhattacharyya coefficient [22], [23] and on the Kullback-Leibler divergence.

Sections II and III introduce relevant concepts and our view of business processes. In Section IV, we describe our framework and apply it in Section V to the Alpha algorithm [14]. We show how a process model can be broken down into substructures and the probability of correct mining of those substructures, and thus of the full model, accurately calculated. In Section VI, we apply the analysis to our running example, and to a larger model to illustrate larger systems and show that our method gives insights into the behavior of the Alpha algorithm when mining these models. In Section VII, we support our probabilistic view of processes with a comparison of the distances which we use to compare processes, with existing metrics. Section VIII concludes the paper.

II. Preliminaries and Related Work

A. Business Processes and Their Representations

A business process describes the activities which take place to fulfill a particular function, from various perspectives [1] including relationships between activities (control flow), timing, resources, decision rules. In this paper, we focus on the control flow perspective.

We assume that processes have single input (start) and output (end) activities (or tasks), and the events of activities' occurrence are recorded as they occur. Events are atomic (take no time), and are uniquely labeled, the same label always referring to the same event, and vice versa. No use is made of additional information (such as timing) about events, merely the order in which they are recorded. The underlying process model is assumed to be fixed (unlikely in reality, but change is assumed to be slow enough to be ignored over the period that data is collected). These restrictions are equivalent to those used elsewhere in the literature, e.g., [14], [24], [25].

Traditionally, business processes have been viewed as languages over activities, with no probabilistic structure. Various representational mechanisms have been suggested for capturing process control flow. Business Process Model and Notation [26] is widely used for business process modeling. It uses an extensive notation to allow description of complex, hierarchical, executable processes, but has not been used for process mining. Early process mining work [25], [27] used simple directed graphs which did not specify the types of splits and joins. More recently, simple precedence diagrams [3] are similar, loosely capturing process structure to semiformally describe processes. Nodes describe activities or groups of activities. WF schemas [9] model structures such as splits and joins in acyclic processes, while block-structured diagrams [17] enforce

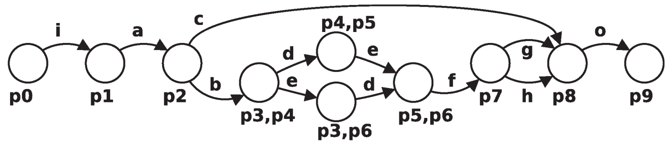

Fig. 2. Reachability graph for Petri net N_0 (Fig. 1).

rigid nesting of substructures and focus on the description of concurrence. Other languages which support concurrence and complex synchronization structures [28] include Adonis [29] and Petri nets [30].

Causal nets [15] have been proposed to allow flexible definition of splits and joins using logical expressions, without introducing constructs such as hidden transitions, needed by Petri nets. Petri nets however remain the most common representation employed in the process mining community to describe processes under this view (e.g., [14], [16], [19]). We introduce Petri nets in the following section.

B. Petri Nets

There are various types of Petri net, details of which with their properties and executable behavior can be found in [30]. For a discussion of WF nets, a restriction of Petri nets used in business processes, see [14], [31].

In general, a *Petri net* is a four-tuple $N = (S, T, W, M)$, where T and S are finite sets of *transitions* and *places*, respectively, such that $T \cap S = \emptyset$. $W \subseteq (S \times T) \cup (T \times S)$ is a *flow relation*, defining the directed arcs of the graph, connecting places and transitions. M is a multiset over S called a *marking* $M : S \rightarrow \mathbb{N}$, describing the distribution of *tokens* over places, defining the state of the process.

The workings of the Petri net are defined by the marking and the firing of transitions. A transition t may *fire* when there is a token in each of its input places, whereupon a token is removed from each of the input places of t and a token added to each of the output places of t. Fig. 1 shows a Petri net N_0. $S = \{p_0, p_1, \ldots\}$, $T = \{i, a, \ldots\}$, $W = \{(p_0, i), (i, p_1), \ldots\}$, $M = (1, 0, \ldots)$. Solid rectangles represent transitions, modeling process activities. Places are shown by circles, and tokens by black dots in places. Only transition i can fire, consuming the token in p_0 and creating one in p_1, thus enabling transition a. The *Reachability Graph* of Petri net N is the state space of the net, the set of markings reachable from M_0 by firing a series of transitions. Fig. 2 shows the reachability graph of Petri net N_0 (Fig. 1), as a transition system. Each reachable marking is represented by a state, labeled with the places of the Petri net which contain tokens in that marking. The arcs are labeled by the transitions which are fired to move from one state to the next.

Sometimes, business processes are constrained to a convenient subset of Petri nets, called sound WF nets [6], [14]. A *sound WF net* is a Petri net with a single start and single end place and every transition on a path between these two places. Its marking is a mapping $S \rightarrow \{0, 1\}$, i.e., any place may hold at most one token. Initial marking M_0 is a single token in the start place, and final marking M_F a single token in the end place. When a process is started from M_0, all transitions must be potentially executable, and the process must terminate properly, i.e., in marking M_F. Sound WF nets allow for all the basic routing constructs found in business processes. *Structured WF nets* [14] restrict the allowed structure of places and transitions to ensure each split corresponds with a join of the same type. Fig. 1 is both a sound and a structured WF net, in its initial marking M_0.

Different measures have been proposed to quantify how well a given Petri net N conforms to a WF log $\mathcal{W}$. For example, "*Token-based fitness*" [6], used as a recall metric in process mining, measures the ability of N to support the traces in $\mathcal{W}$

$$f = \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{n} m_i}{\sum_{i=1}^{n} c_i}\right) + \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} p_i}\right)$$

where n is the number of traces in $\mathcal{W}$, p_i the number of tokens produced during replay of trace i, c_i the number consumed, m_i the number of tokens missing (had to be artificially created to enable trace to be replayed), and r_i the number remaining after replay of the trace. *Behavioural appropriateness* a'_B [6] measures the precision of the model, penalizing behavior supported by the model but not the log.

C. *Process Mining*

Process mining algorithms aim to reconstruct the underlying business process structure based on a sample WF log. They are broadly split into "local" and "global" approaches [1]. "Local" build models from relations between activities, e.g., Alpha [14], Alpha^{++} [16], Heuristics Miner [15]; while "global" methods start with and refine a full model, e.g., genetic [32], [33] and region mining [18], [19]. Recent approaches aim to mine complex or noisy processes using clustering and abstraction at the level of traces [9]–[11], or activities [3], [12], [13]. Other work includes artificially generating negative examples to improve learning [34], and mining from logs lacking case IDs [20] to identify process instances. Process mining has also been used as an assistive tool, e.g., with distributed workflow execution [35] or detection of anomalous event behavior [36].

The *Alpha algorithm* [14] can mine processes representable by Structured WF-Nets, from noise-free logs. A net is inferred based on local relations between pairs of activities recorded in a workflow log $\mathcal{W}$. Transitions represent atomic activities. Single start and end places are assumed; the remaining places are inferred using the basic relations:

- $a > b$ (b directly follows a in at least one trace);
- $a \rightarrow b$ (b always follows a, never vice-versa);
- $a \# b$ (a and b never follow each other);
- $a \| b$ (both ab and ba occur in the log).

Two activities are always related by either $\rightarrow$, $\rightarrow^{-1}$, $\#$, or $\|$, and these partition the set of activities [14, Property 3.1].

Various methods have been proposed for comparing process models, often based on the syntax of the representations used. Examples are replaying training or reference logs [9], [12], [15], [32], measuring Petri net token behavior [4], [34] or string edit distances [37], comparing incidence matrices [38]

TABLE I
NOTATION FOR BUSINESS PROCESS

$\mathcal{A}$	A set of business activities.
$\mathcal{M}$	'Ground truth' model (may be unknown).
Σ	Alphabet of symbols encoding business activities.
$\{a, b, \ldots\} \in \Sigma$	Valid business activities in the process.
$\{x, y, \ldots\} \in \Sigma^+$	Non-empty strings representing sequences of activities.
$\mathcal{T}$	The set of all valid process traces (cases).
xy	The concatenation of strings x and y.
$x\Sigma^*, \Sigma^*x,$ $\Sigma^*x\Sigma^*$	The set of strings with x as prefix, suffix, . . . or sub-string. (rest of string may be empty).
$\mathcal{W} \subset \{x \vert x \in \Sigma^+\}$	Workflow log, a *bag* or *multi-set* of traces.

or coding costs using the minimum description length principle [39]. Measures may be along different "dimensions" [5], [6] depending on the type of differences to be measured.

The review in [2] concludes "more research is required to enable the production of a generic framework for the quantified comparison of processes." In [5], an architectural framework is proposed, to include algorithms, methods for comparison, and a repository of logs and tools. Existing metrics and a method of assessing algorithms using a k-fold cross-validation method are compared. Metrics have the advantage of allowing different aspects of models' behavior to be compared and differences localized in the representation in use, but do not provide a common basis for comparing models in different representations, or upon which to objectively discuss other process mining tasks such as generalization, clustering, or abstraction. The k-fold approach uses an experimental method from machine learning and allows the significance of differences to be quantified. However, it does not provide a theoretical foundation for analyzing the learning behavior of algorithms and predicting how much data are needed for mining. The paper concludes that more research is needed.

Alpha is one of many process mining algorithms implemented as plug-ins in ProM [40]. Commercial tools include Fujitsu's automated process discovery, while many products address business process modeling and automation.

III. BUSINESS PROCESSES AS DISTRIBUTIONS OVER TRACES

We propose to view business processes as probability distributions over strings of symbols $a \in \Sigma$ representing activities. In this paper, such distributions will be described using stochastic automata. In other words, a business process can be considered a stochastic regular language $\mathcal{M}$ that describes the probability distribution $P_{\mathcal{M}}$ over Σ^+ (the set of all nonempty strings of activities): $\sum_{x \in \Sigma^+} P_{\mathcal{M}}(x) = 1$. Each trace begins with the start activity i and finishes with the end activity o. The finite (we are not considering cycles) set of valid process traces $\mathcal{T}$ consists of strings $x \in \Sigma^+$ such that no activity $a \in \Sigma$ occurs more than once in x, and $x = iwo$, $w \in (\Sigma \setminus \{i, o\})^*$ (w may be empty). An overview of our notation is presented in Table I. The next section introduces our main representational framework for describing $P_{\mathcal{M}}$ (see also [20], [29]).

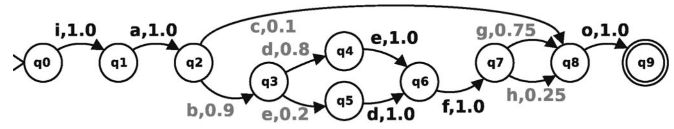

Fig. 3. PDFA A_0 corresponding to Petri net N_0 (Fig. 1), with the addition of transition probabilities.

A. Probabilistic Automata

To provide a "common denominator" to which processes in other modeling languages can be converted and analyzed, we use transition-labeled PDFA [21] to represent the probability distributions as process models. Briefly, a *PDFA* is a five-tuple $A = (Q_A, \Sigma, \delta_A, q_0, q_F)$:

- Q_A is finite set of states;
- Σ is an alphabet of symbols;
- $\delta_A : Q_A \times \Sigma \times Q_A \to [0, 1]$ is a mapping defining the conditional transition probability function between states, $\delta_A(q_1, a, q_2) = Pr(q_2, a|q_1)$, i.e., the probability to parse symbol a and arrive in state q_2 given currently in q_1;
- $q_0 \in Q_A$ is a single start state; and
- $q_F \in Q_A$ is a single end state; such that:

$$\forall q \in Q_A, \qquad \sum_{q' \in Q_A, a \in \Sigma} \delta_A(q, a, q') = 1, \quad \text{and } Pr(q'|a, q) = 1.$$

The probabilities on arcs outgoing from a state sum to 1 and given a current state and symbol, the next state is certain. There is a unique state path through the automaton for any string x that it can parse.

Example PDFA A_0 (Fig. 3) represents the same model as Petri net N_0 (Fig. 1). It has the same structure as the reachability graph, with the addition of probabilities of following each arc, or parsing each symbol. Here, $Q = \{q_0, q_1, \ldots\}$, $\Sigma = \{i, a, \ldots\}$, $\delta = \{(q_0, i, q_1) \to 1.0, (q_1, a, q_2) \to 1.0, \ldots\}$, $q_0 =$ 'q_0', $q_F =$ 'q_9'. States are shown by circles, the start state is indicated by an arrow and the final state by a double border.

Every PDFA A describes a distribution P_A over Σ^+

$$P_A(x) = \delta_A(q_0, s_0, q_{s_0}) \times \left(\prod_{i=1}^{n-2} \delta_A \left(q_{s_{i-1}}, s_i, q_{s_i} \right) \right) \times \delta_A \left(q_{s_{n-2}}, s_{n-1}, q_F \right)$$

where x is a string of symbols $s_0 s_1 \ldots s_{n-1}$ which can be parsed by the automaton to the unique final state q_F; q_{s_i} denotes the state reached after symbol s_i is parsed. $P_A(x) = 0$ for strings which cannot be parsed.

In A_0 (Fig. 3), $P_A(iaco) = \delta_A(q_0, i, q_1) \times \delta_A(q_1, a, q_2) \times \delta_A(q_2, c, q_8) \times \delta_A(q_8, o, q_9) = 1.0 \times 1.0 \times 0.1 \times 1.0 = 0.1$.

Note that the structure of allowed traces defined by sound WF nets can be naturally captured by the support structure of distributions described by PDFA in the sense that:

- there is a single start and end state;
- all states are accessible (reachable from the initial state);
- from any state, it is possible to reach the final state;
- for any given string x, the sequence of state transitions to generate x is unique;
- given a state and a symbol, the next state is certain.

A sound WF net does not hold any probability information, but has finite state space, so the net's structure can be converted to an automaton with a finite number of states,[1] via its reachability graph, e.g., [30]. Transitions can be allocated uniform probabilities, or estimated maximum likelihood probabilities from a log file. The structure of a PDFA may be converted to a Petri net using the theory of regions, e.g., [18, Sec. 4].

B. Distances Between Probability Measures

Viewing business processes as probability distributions, we can quantify differences between two business processes P_1 and P_2 (e.g., the "ground truth" and its inferred proxy) via distances on the space of distributions over traces, e.g.,

Euclidean Distance

$$d_2(P_1, P_2) = \sqrt{\sum_x \left(P_1(x) - P_2(x)\right)^2}$$

Bhattacharyya Distance [22]

$$d_{Bhat}(P_1, P_2) = \sqrt{1 - \sum_x \sqrt{P_1(x) P_2(x)}}$$

Kullback-Leibler Divergence

$$d_{KL}(P_1, P_2) = \sum_x P_1(x) \log \frac{P_1(x)}{P_2(x)}.$$

Note that Kullback-Leibler Divergence is not a distance measure since it is not symmetric. Also, it requires P_1 and P_2 to have the same support. This is straightforward to work around, e.g., by postulating the *Jensen-Shannon Divergence* [41]

$$d_{JSD}(P_1, P_2) = d_{KL}(P_1, \psi) + d_{KL}(P_2, \psi)$$

where $\psi(x) = (1/2)(P_1(x) + P_2(x))$.

C. Process Mining: A Machine Learning View

We formalize a machine learning view of process mining, noting that some of these ideas are implicit in other work, e.g., [25], [29]. In particular, in [20] a stream of symbols representing activities is produced by multiple random sources. We rather consider a single source generating traces.

A process discovery algorithm is essentially a learning machine, whose task is to model the control flow of a business process, using traces of the execution of the process, recorded in a WF log $\mathcal{W}$, which is a multiset over traces. Each trace represents a single run through the process from start to end. Traces can be encoded as strings $x \in \Sigma^+$, where Σ is an alphabet of symbols representing activities.

We assume that an unknown probability distribution $\mathcal{D}$ over traces (from Σ^+) is responsible for generating the traces in the log $\mathcal{W}$. Although various factors affect what activities take place, such as business needs or user preferences, different

[1]The "state space explosion" may be a problem, particularly in the conversion of large Petri nets with high concurrence (although the structural and marking limitations of sound WF nets should alleviate this).

traces in fact occur with specific probabilities, and thus it can be argued that the underlying process is inherently stochastic. From the machine learning point of view, the primary task of the process mining algorithm is to construct a model $\mathcal{M}$ of $\mathcal{D}$ from a finite sample of traces (WF log $\mathcal{W}$).

The log file $\mathcal{W}$ will contain only a finite number of process traces and therefore is a stochastic sample drawn independently and identically distributed from the unknown distribution $\mathcal{D}$ (the "ground truth"). In other words, each trace occurs with probability according to the same distribution $\mathcal{D}$, and one trace occurring does not change the probability of others. Since the log is of finite size, we expect the frequency of traces in the log to vary from their probabilities under $\mathcal{D}$.

The challenge for the learning machine (process discovery algorithm) is to use this finite sample to construct a model $\mathcal{M}$ of $\mathcal{D}$ which does not simply represent the data in the finite log, but is as "close" as possible to the true generating source $\mathcal{D}$, i.e., generalizes well. This raises questions such as: *How much data is needed to do this with certain (given) confidence and precision?*[2]*How to quantify the learning machine's performance?* Since both $\mathcal{D}$ and $\mathcal{M}$ are distributions, it is natural to assess the learning machine's performance by quantifying how "close" $\mathcal{M}$ is to $\mathcal{D}$, for which there are various measures. This allows direct comparison of the "reality" represented by the models, rather than similarity/dissimilarity of syntactic representations of $\mathcal{M}$ and $\mathcal{D}$ in the modeling language in use, which seems to be a common theme [4], [9], [32], [38].

Machine learning theory is concerned with the convergence properties of machine learning algorithms, in terms of the circumstances in which they can be expected to converge to the ground truth and the amount of data needed. While different process discovery algorithms have different strengths and weaknesses, they can be compared under this unifying framework, i.e., in terms of their convergence properties within the restrictions within which they operate. From the ground truth and an understanding of the behavior of an algorithm, one can predict, and experimentally verify, how fast the mined model will converge to the ground truth model.

While in real applications, the process discovery algorithm will not have access to the ground truth distribution governing trace generation, it is standard practice in machine learning [42], [43] to study learning algorithms by imposing a certain class of ground truth distributions and then to verify empirically and/or theoretically how fast and how well the ground truth can be "learnt" by the algorithm from finite samples. In this framework, the algorithm does not know the ground truth, but because we have access to it, the success of the learning algorithm as more samples become available can be measured.

IV. Framework for the Analysis of Process Mining Algorithms

In this section, we outline a framework within which to analyze process mining algorithms with regard to their probabilistic behavior, process substructures, and number of traces; in the context of their ability to discover a probability distribution over traces, which converges to a "ground truth."

The steps below describe the approach taken here to analyze and experimentally validate process mining algorithms.

Step 1) Analyze the algorithm to develop formulas for the probability of discovery of all important process substructures (e.g., splits and joins, or parallel action flows).

Step 2) Extend to aggregate the substructure results (joining substructures from the previous step into the full model) to enable calculation of overall discovery probability of arbitrary models.

Step 3) Analyze the algorithm's characteristics, such as rate of convergence, issues affecting convergence, possible relation to other algorithms, etc.

Theoretical analysis will be complemented by empirical investigations as follows.

1) Design "ground truth" test models with varying topological and probability structures.
2) From the test models, generate multiple sample sets of WF logs of various sizes, to test for convergence.
3) Run process mining algorithms under investigation on such data, converting mining results to PDFA as necessary (Section III-A), and compare distributions of traces represented by these automata with the "ground truth."[3]

In the following, we introduce the important process substructures, and in Section V, we apply the framework to an analysis of the Alpha algorithm [14].

A. *Process Substructures*

Business processes are composed of substructures [17], [28]. We consider only acyclic structures in this paper. A few basic structures are sufficient, although more complex patterns exist [28]. The substructures in our example process are highlighted in Fig. 4. We next define process substructures in terms of the set $\mathcal{T}$ of valid process traces starting with i and ending with o.

1) Sequences: If tasks a and b form a sequence (e.g., Fig. 4, substructure A), then if a occurs, it is immediately followed by task b, and no other, in the model. In the log, other parallel tasks may "interfere," so the following will hold: if a occurs in a trace, b will occur before the end of the trace, i.e.,

$$\text{if } uav \in \mathcal{T}, \quad \text{then} \quad v = wbq, \quad \text{where } a, b \in \Sigma, \text{ and } u, w, q \in \{\Sigma \setminus \{a, b\}\}^*.$$

*2) Exclusive-*OR *Split:* An m-way XOR split [Fig. 5(a)] occurs where there is a choice between m mutually exclusive paths through the model after task a, each path starting with a task $t \in \{b_i | 1 \le i \le m\}$. If a occurs in a trace, then exactly one $t \in \{b_i | 1 \le i \le m\}$ will occur in the rest of the trace AQ1

$$\text{if } uav \in \mathcal{T}, \text{ then } \exists i : 1 \le i \le m, \text{ such that } v = wb_iq, \quad \text{where } a, b_i \in \Sigma, \text{ and } u, w, q \in \{\Sigma \setminus \{a, b_1, \ldots, b_m\}\}^*.$$

[2]This corresponds to, e.g., the so-called probably approximately correct framework.

[3]Where algorithms produce nonprobabilistic models, heuristic methods can be used to allocate uniform or maximum likelihood probabilities to transitions.

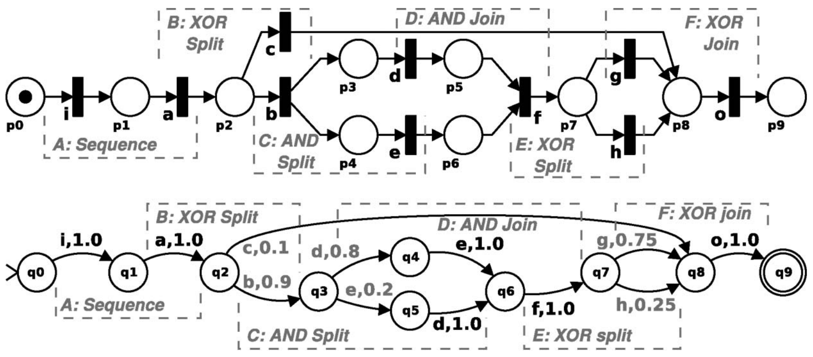

Fig. 4. Example of process substructures in Petri net N_0 and PDFA A_0.

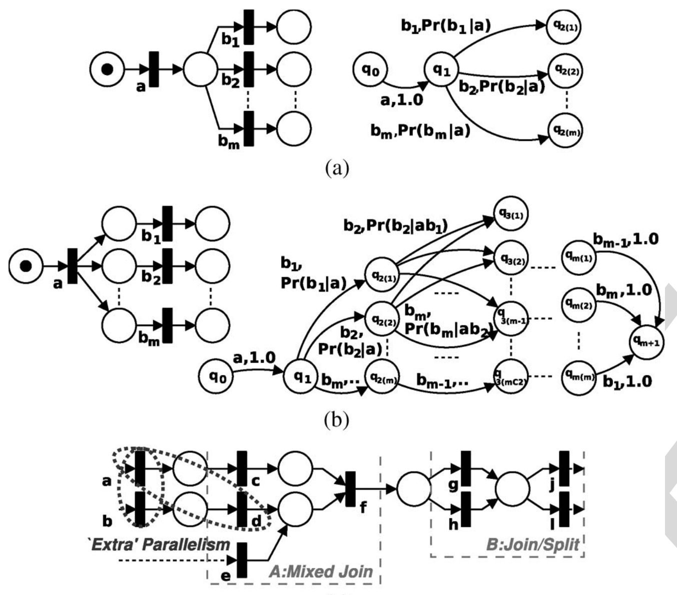

Fig. 5. Petri net and PDFA fragments depicting various substructures. (a) XOR split. (b) Parallel (AND) split. (c) Complex splits and joins (substructures A and B) and "extra" parallel activities (dotted ellipses).

3) Exclusive-OR Join: An m-way XOR join (e.g., Fig. 4, structure F) occurs where m mutually exclusive paths rejoin before task c. The final task in each path prior to c is a task $t \in \{b_i | 1 \leq i \leq m\}$. If c occurs in a trace, then exactly one task $t \in \{b_i | 1 \leq i \leq m\}$ will be in the trace before c

$$\text{if } ucv \in \mathcal{T}, \text{ then } \exists i : 1 \leq\ i \leq\ m, \text{ such that} u = wb_iq,$$
$$\text{where } c, b_i \in \Sigma, \text{ and } v, w, q \in \{\Sigma \setminus \{c, b_1, \ldots, b_m\}\}^*.$$

4) Parallel Split: An m-way AND split [Fig. 5(b)] occurs where m paths through the model proceed in parallel, following task a, each path starting with a task $t \in \{b_i | 1 \leq i \leq m\}$. If each path contains only a single task b_i, and there are no restrictions on the order of the tasks, and no other parallel parts of the model, then the next m tasks in the trace will be $b_1, b_2, \ldots, b_m$, in one of $m!$ permutations. Otherwise, there will be more possibilities for the trace following a. In reality, it is likely that only a subset of the possible orderings will be highly probable. If a occurs in a trace, then the remainder of the trace following a will contain each $t \in \{b_i | 1 \leq i \leq m\}$

$$\text{if } uav \in \mathcal{T}, \text{ then } \forall i : 1 \leq i \leq m,$$
$$(\exists w, q \in \{\Sigma \setminus \{a, b_i\}\}^* : v = wb_iq),$$
$$\text{where } a, b_i \in \Sigma, u \in \{\Sigma \setminus \{a, b_1, \ldots, b_m\}\}^*.$$

A PDFA fragment to depict a parallel split is visually more complex than its Petri net equivalent, as all possible task sequences are shown explicitly [Fig. 5(b)]. After the first parallel task, there are $\binom{m}{1}$ states, $\binom{m}{2}$ after the second, to $\binom{m}{m-1}$ states before the last parallel task.

5) Parallel Join: An m-way AND join occurs where m parallel paths rejoin (synchronize) before a task c. The final task in each path is one of $b_1, b_2, \ldots, b_m$. If c occurs in a trace, then the trace up to c will contain each $t \in \{b_i | 1 \leq i \leq m\}$

$$\text{if } ucv \in \mathcal{T}, \text{ then } \forall i : 1 \leq i \leq m,$$
$$(\exists w, q \in \{\Sigma \setminus \{c, b_i\}\}^* : u = wb_iq),$$
$$\text{where } c, b_i \in \Sigma, v \in \{\Sigma \setminus \{c, b_1, \ldots, b_m\}\}^*.$$

6) Nonexclusive OR Splits and Joins: These occur where one or many of several paths may be taken. They can be modeled as combinations of XOR and parallel structures.

V. Application to the Alpha Algorithm

The Alpha algorithm [14] is relatively simple and forms the basis for several other algorithms, so is appropriate for a first analysis under this framework. The four relations $\rightarrow$, $\rightarrow^{-1}$, $\#$, and $\|$, on a pair of tasks a, b partition the set of all logs of n traces [Fig. 6(a)]. In this paper, we write the relations as $a >_n b$, etc., to indicate discovery within n traces. The event space Ω is the set of all logs of n traces, A the set of these logs that include at least one trace containing substring $ab (a >_n b)$, and B those with at least one trace with $ba (b >_n a)$. Then

- $A \setminus B$ is the set of logs that cause Alpha to infer the causal relation $a \rightarrow_n b$;
- $B \setminus A$ those for which Alpha infers $b \rightarrow_n a$;
- $A \cap B$ those for which Alpha infers $a \|_n b$;
- $\neg(A \cup B)$ those for which Alpha infers $a \#_n b$.

We next apply the steps described in Section IV to Alpha.

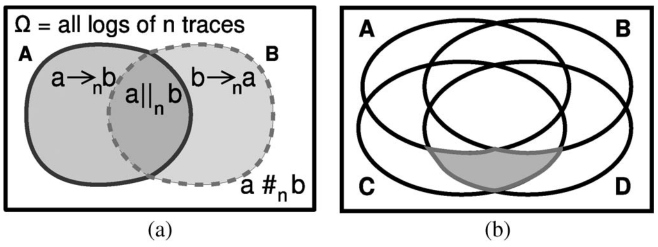

Fig. 6. (a) The Alpha relations on a pair of tasks partition the possible logs of n traces, (b) illustration of $(C \cap D) \setminus (A \cup B)$ for Proposition 5.

A. Step 1: Probability Formulae for Basic Substructures

To analyze algorithms' behavior, we assume that we have access to the ground truth and know probabilities of all strings (sequences of tasks). We give formulas for the probability of discovery of the Alpha relations and process substructures, based on these string probabilities, agnostic of whether these substructures are "correct," i.e., assuming nothing about the underlying model, save that it is acyclic, and traces are generated according to an unknown probability distribution.

Given an underlying source $\mathcal{M}$, let $P_{\mathcal{M}}(a|x)$ denote the probability that after seeing the sequence of tasks given by string x, the next symbol to be seen will be a

$$P_{\mathcal{M}}(a|x) = \frac{P_{\mathcal{M}}(xa\Sigma^*)}{P_{\mathcal{M}}(x\Sigma^*)}.$$

This extends naturally to substrings $y \in \Sigma^n$

$$P_{\mathcal{M}}(y|x) = \frac{P_{\mathcal{M}}(xy\Sigma^*)}{P_{\mathcal{M}}(x\Sigma^*)}, \quad \text{where} \quad \sum_{y \in \Sigma^n} P_{\mathcal{M}}(y|x) = 1.$$

We also introduce some shorthand notation: We write $\pi(ab)$ for $P_{\mathcal{M}}(i\Sigma^*ab\Sigma^*o)$, the probability of ab occurring in a trace, and $\pi(b| \rightarrow a)$ for $P_{\mathcal{M}}(b|i\Sigma^*a)$, the conditional probability that given that a occurs in a trace, the next symbol will be b. We define $\pi_n(E)$ as "the probability of complex event E holding true in a log of n traces." For example, $\pi_n(A)$ for set A in Fig. 6(a), is "the probability that at least one trace in a log of n traces contains substring ab." Finally, $P_\alpha(a >_n b)$ means "the probability that Alpha infers the relation $a >_n b$ over n traces," and similarly for the other Alpha relations.

1) Activity Ordering Relations: These are the basic relations between tasks in the log which Alpha uses to construct a Petri net model.[4] The following propositions give the probability of Alpha inferring these relations between two tasks a and b, from a log of n traces, based on the substring probabilities described above. We give the proofs in the Appendix.

Proposition 1: The probability that Alpha infers $a >_n b$ is

$$P_\alpha(a >_n b) = 1 - (1 - \pi(ab))^n.$$

Proposition 2: The probability that Alpha infers $a\#_n b$ is

$$P_\alpha(a\#_n b) = (1 - \pi(ab) - \pi(ba))^n.$$

[4]Alpha$^+$, which is implemented in the ProM framework [40] as Alpha, modifies these to allow for short loops.

Proposition 3: The probability that Alpha infers $a \rightarrow_n b$ is

$$P_\alpha(a \rightarrow_n b) = (1 - \pi(ba))^n - (1 - \pi(ab) - \pi(ba))^n. \quad (1)$$

Proposition 4: The probability that Alpha infers $a\|_n b$ is

$$P_\alpha(a\|_n b) = 1 - (1 - \pi(ab))^n - (1 - \pi(ba))^n + (1 - \pi(ab) - \pi(ba))^n.$$

2) Sequences: Discovery of a basic sequence of two tasks a and b simply requires discovery of $a \rightarrow_n b$.

3) Splits and Joins: Alpha uses the relations $a \rightarrow_n b$, $a\#_n b$, and $a \|_n b$ to identify Petri net places, which determine the types of splits and joins (XOR or AND). Since the events of the discovery of these relations between several tasks arise from Alpha's interpretation of a log of n traces, they are not independent: any, all or no relations may be discovered. Thus $P_\alpha((a \rightarrow_n b) \wedge (a \rightarrow_n c)) \leq P_\alpha(a \rightarrow_n b) \times P_\alpha(a \rightarrow_n c)$. Therefore, for exact probabilities for discovery of splits and joins, the basic substring probabilities (probabilities of task pairs which *must/must not* be seen in the log) must be used.

4) Exclusive Choice—XOR Split: To discover an m-way XOR split from a to $b_1, b_2, \ldots, b_m$ [Fig. 5(a)], denoted $a \rightarrow_n (b_1\# \ldots \#b_m)$: Alpha must infer the relations $a \rightarrow_n b_1, a \rightarrow_n b_2, \ldots, a \rightarrow_n b_m, b_1\#_n b_2, b_1\#_n b_3, \ldots, b_{m-1}\#_n b_m$ [14, Def. 4.3, step 4]. Hence, over a log of n traces, Alpha must:

see at least one of each of m substrings representing pairs of tasks $ab_1, ab_2, \ldots, ab_m$; and

not see any of the m "reverse" pairs $b_1a, b_2a, \ldots, b_ma$, or any of ${}_mP_2$ pairs of "postsplit" tasks: $b_1b_2, b_2b_1, \ldots, b_{m-1}b_m, b_mb_{m-1}$ (where ${}_mP_2 \triangleq (m!/(m-2)!)$).

Let $N = \{N_i = (t_i, t'_i) | 1 \leq i \leq (m + {}_mP_2), t_i \neq t'_i\}$ be the set of task pairs which *must not* be seen in the log, and $Y = \{Y_i = (t_i, t'_i) | 1 \leq i \leq m, t_i \neq t'_i\}$ be the set of task pairs which *must* be seen in the log.

We define $S_n(X) \rightarrow [0,1]$, where $X = \{X_i = (t_i, t'_i) | 1 \leq i \leq |X|\}$ as the probability of *not* seeing any of the $|X|$ task pairs $(t_i, t'_i) \in X$ in n traces, and $\pi(X_i) = \pi(t_it'_i)$.

Proposition 5: Probability that Alpha infers an XOR split is

$$\begin{aligned} P_\alpha&(a \rightarrow_n (b_1\# \ldots \#b_m)) \\ &= S_n(N) - \sum_{1 \leq i \leq m} S_n(N \cup \{Y_i\}) \\ &\quad + \sum_{1 \leq i < j \leq m} S_n(N \cup \{Y_i, Y_j\}) - \cdots \\ &\quad + (-1)^m S_n(N \cup Y), \quad \textit{where} \end{aligned} \quad (2)$$

$$\begin{aligned} S_n&(X) \\ &= \Bigg(1 - \sum_{1 \leq i \leq |X|} \pi(X_i) + \sum_{1 \leq i < j \leq |X|} \pi(X_i \wedge X_j) \\ &\qquad - \cdots + (-1)^{|X|} \pi\left(X_1 \wedge X_2 \wedge \ldots \wedge X_{|X|}\right)\Bigg)^n. \end{aligned} \quad (3)$$

Given knowledge about the underlying model, many of the terms may be zero, significantly simplifying the formulas. Nevertheless, they can become cumbersome to work with, requiring knowledge of many probabilities. Nor do they relate intuitively to the working of the algorithm. However, these formulas can be effectively simplified without loss of accuracy to give formulas which intuitively follow from the working of the Alpha algorithm and are simpler to calculate. Theorem 1 illustrates for Proposition 5 the discovery of an XOR split.

Theorem 1: The probability of discovery of an XOR split may be approximated by assuming independence between discovery of Alpha relations over n traces. The probability is overstated, but error rate decreases exponentially with increasing n

$$P_\alpha\left(a \rightarrow_n (b_1 \# \ldots \# b_m)\right) \le \prod_{1\le i\le m} P_\alpha(a \rightarrow_n b_i) \times \prod_{1\le i<j\le m} P_\alpha(b_i \#_n b_j). \quad (4)$$

The proof is given in the Appendix. In what follows, we use the approximation in Theorem 1 to derive formulas for discovery of XOR joins and analogous results (not presented here) for AND splits and joins.

5) Exclusive Choice—XOR Join: To discovery of an m-way, XOR join can be approximated in a similar way

$$P_\alpha\left((b_1 \# \ldots \# b_m) \rightarrow_n c\right) \le \prod_{1\le i\le m} P_\alpha(b_i \rightarrow_n c) \times \prod_{1\le i<j\le m} P_\alpha(b_i \#_n b_j). \quad (5)$$

6) Parallelism—AND Split: The behavior of the Alpha algorithm when mining an AND split is similar to that when mining an XOR split, with more "must see" and fewer "must not see" substrings. To discover a m-way parallel split from a to $b_1, b_2, \ldots, b_m$, denoted $a \rightarrow_n (b_1 \parallel \ldots \parallel b_m)$: Alpha must infer the relations $a \rightarrow_n b_1, a \rightarrow_n b_2, \ldots, a \rightarrow_n b_m, b_1 \parallel_n b_2, b_1 \parallel_n b_3, \ldots, b_{m-1} \parallel_n b_m$[14, Defn 4.3 Step 4]. Thus,

$$P_\alpha\left(a \rightarrow_n (b_1 \parallel \ldots \parallel b_m)\right) \le \prod_{1\le i\le m} P_\alpha(a \rightarrow_n b_i) \times \prod_{1\le i<j\le m} P_\alpha(b_i \parallel_n b_j). \quad (6)$$

7) Parallelism—AND Join: Similarly

$$P_\alpha\left((b_1 \parallel \ldots \parallel b_m) \rightarrow_n c\right) \le \prod_{1\le i\le m} P_\alpha(b_i \rightarrow_n c) \times \prod_{1\le i<j\le m} P_\alpha(b_i \parallel_n b_j). \quad (7)$$

B. Step 2: Aggregation of Substructures to Full Model

Having dealt with substructures, we now need to derive probability $\psi(\mathcal{M})$ of correctly mining the full process model $\mathcal{M}$ (e.g., Fig. 4 with substructures labeled $A \ldots F$). For exact calculation the approach of Proposition 5 could be extended to consider the probabilities of all the substrings which Alpha must/must not see in the log to construct the Petri net correctly, as these probabilities are not independent (Section V-A3). This is infeasible and does not reduce the problem complexity.

As discussed (Theorem 1), we can treat substructures as built from independent Alpha relations rather than from individual substrings. Three further areas then need to be considered in analyzing full models.

1) *Compound Splits/Joins*: A single WF net substructure as mined by Alpha may combine both join and split (Fig. 5(c) substructure B) or combine parallel and exclusive behavior (Fig. 5(c) substructure A). In general, if m paths of which p are XOR (the remainder parallel) join and then split to n paths of which q are XOR, the probability of discovery is approximated using the approach of Theorem 1, multiplying probabilities for the relevant $\rightarrow_n$, $\#_n$, and $\parallel_n$ relations.
2) *Extra Parallelism*: Where parallel paths contain more than one task, such as a, b, c, d in Fig. 5(c), for each pair of tasks a, b not part of the split or join, either $a\parallel_n b$ or $a\#_n b$ must be discovered, to prevent extra dependencies from being inferred. Let $a \nleftrightarrow_n b$ denote $(a\parallel_n b) \vee (a\#_n b)$. These are independent [Fig. 6(a)], so $P_\alpha(a \nleftrightarrow_n b) = P_\alpha(a\parallel_n b) + P_\alpha(a\#_n b)$.
3) *Combining Probabilities for Substructures*: Let $P_S(X)$ be the probability of discovering substructure X. Intuitively, if a split has been mined correctly, then mining the corresponding join is "almost certain," as each path between the split and join should be in the log. Hence, substructures in the model can be considered as dependent on "previous" substructures, e.g.,

$$\psi(M) = P_S(A) \times P_S(B|A) \times P_S(C|B) \times P_S(D|C) \times P_S(E|D) \times P_S(F|B,E).$$

$P_S(F|B,E)$ indicates the probability of discovering F conditional that B and E have been mined correctly. This affects the formulas from Section V-A in two ways. For each event (such as "see no ab in the log of n traces"),

1) the probabilities of the substrings are conditioned by the probabilities of the prefix strings leading up to those substrings, i.e., $\pi(ab)$ becomes $\pi(b| \rightarrow a)$); and
2) we only consider the traces within which those substrings are expected to occur.

To illustrate, for Alpha, the relation $a >_n b$ becomes

$$P_\alpha(a >_n b) = 1 - \left(1 - \frac{\pi(ab)}{\pi(\rightarrow a)}\right)^{n\cdot\pi(\rightarrow a)}. \quad (8)$$

C. Step 3: Analysis of Alpha Algorithm

We look briefly at some of the behavior of Alpha shown by the probability formulas.

Fig. 7(a) shows $P_\alpha(a \rightarrow_n b)$ increasing sharply with increasing $\pi(ab)$ (probability of trace including string ab), but reducing sharply with any probability of the "reverse" string, $\pi(ba)$. The effect is stronger as n increases, since this also increases the chance of at least one ba in the log. Nonzero probability of ba may be due to errors in logging, or indicate that the real relation

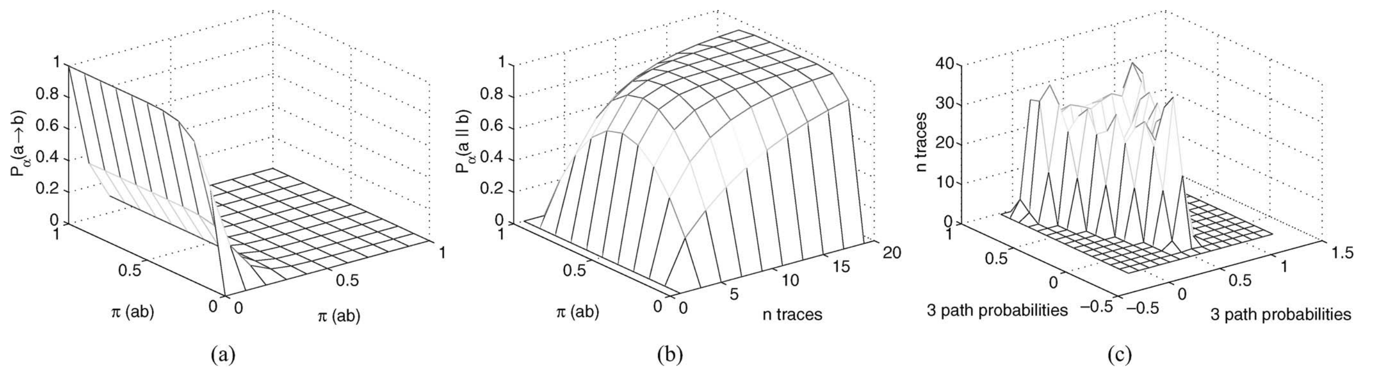

Fig. 7. Probabilistic behavior of Alpha relations. (a) Probability of discovery of $a \rightarrow_n b$ for 10 traces, varying $\pi(ab), \pi(ba)$. (b) Probability of discovery of $a \|_n b$ for varying numbers of traces and $\pi(ab)(\pi(ab) + \pi(ba) = 1.0)$. (c) Number of traces for 95% discovery probability of three-way XOR substructure.

is parallel but with ab more likely than ba. In Fig. 7(b), the parallel relation $a \parallel_n b$, for which both ab and ba must be seen, is seen to be most likely when the probability of either order is similar (note that here, $\pi(ba) = 1 - \pi(ab)$). This is important, since when multiple activities are allowed to occur in any order (parallel), in practice, certain orderings may be more likely, reducing the probability of discovering the true parallelism and necessitating more data.

Fig. 7(c) shows the behavior of Alpha when mining a three-way XOR split. All possible combinations of probabilities are indicated by points on the triangular base, with each edge representing the range of probabilities from 0 to 1 of one of the three exclusive tasks, such that the probabilities sum to 1. The graph shows the number of traces required to achieve 95% probability of discovery. The greatest number of traces is needed where the probabilities are most imbalanced, i.e., around the edges, with the peaks at each corner showing where only one path has a nonnegligible probability. The corresponding graph for the parallel split shows similar behavior.

VI. Analysis of Example Process Models

We used the presented methods to predict the number of traces needed for the probability of successful mining of the running example[5] (Fig. 1) to exceed various thresholds. Automaton A_0 (Fig. 3) was specified as the ground truth and simulated by random walk to produce 30 sets of MXML format WF log files of increasing size from 1 to 45 traces. A "ground truth" log of 1000 traces was also simulated.

We mined Petri net models from these files using Alpha as implemented in ProM [40] and calculated the fitness (f) [6] and behavioral appropriateness (a'_B) [4] values using the conformance analysis plugin. The Petri nets were converted to PDFA by labeling their reachability graphs with maximum likelihood probability estimates derived from the ground truth log file. The d_2 and Bhattacharyya (d_{Bhat}) distances, and Jensen-Shannon divergence (d_{JSD}) were calculated between the distributions represented by these PDFA and the ground truth distribution represented by A_0.

[5]This model has not been benchmarked but designed for this test.

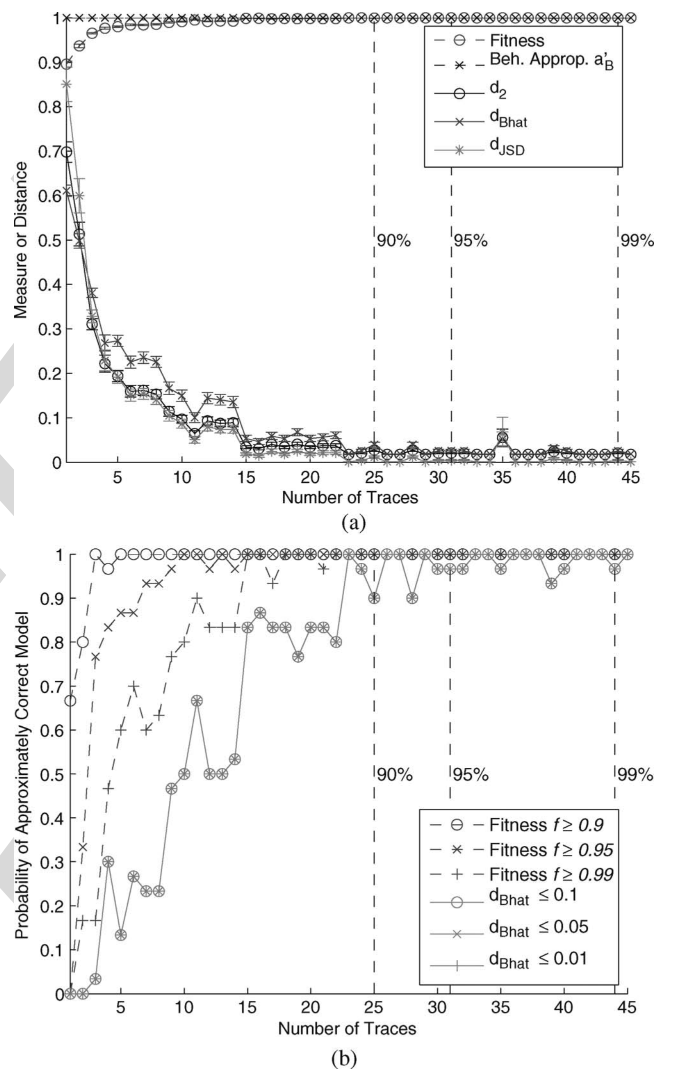

Fig. 8. Results showing convergence of Alpha to the ground truth, mining from logs of increasing size simulated from PDFA A_0. (a) Average metrics against number of traces. (b) Probability of approximately correct model.

The graph in Fig. 8(a) shows the average *approximate correctness* of the models mined by Alpha from logs of increasing size, as measured by the metrics and distances, plotted against the number of traces in the log. The numbers of traces predicted

TABLE II
METRICS AND NUMBERS OF TRACES AT THRESHOLD POINTS FOR MINING EXAMPLE MODELS

Probability of Success	**50%**	**90%**	**95%**	**99%**
Running Example (Petri net Fig.1, PDFA A_0 Fig.3)				
Predicted/Actual Traces	11/10	25/23	31/29	44/45
f	0.992	0.998	1.000	1.0
a'_B	1.0	1.0	1.0	1.0
$1 - d_2/\sqrt{2}$	0.935	0.974	0.984	1.0
$1 - d_{Bhat}$	0.866	0.950	0.978	1.0
$1 - d_{JSD}/2$	0.962	0.991	0.998	1.0
Larger Example (Petri net Fig.9, PDFA A_3 Fig.10)				
Predicted/Actual Traces	37/−	63/65	75/75	100/115
f	0.984	0.989	0.998	1.0
a'_B	0.962	0.984	0.998	1.0
$1 - d_2$	0.951	0.983	0.997	1.0
$1 - D_{Bhat}$	0.766	0.881	0.980	1.0
$1 - JSD$	0.768	0.903	0.983	1.0

for 90%, 95%, and 99% confidence in correct mining are indicated by the vertical rules. The graph shows:

1) Probability distance measures converge in a similar way to f, but the distances from the ground truth are distributed over a clearer scale, from almost 1 for the very unfit models produced by few traces, through to 0, whereas f ranges from approx 0.8 to 1 (see Section VII).
2) The distance measures show convergence to approximate correctness at the predicted points.
3) Irregularities may indicate points of interest in the behavior of the algorithm, worthy of further investigation.
4) a'_B was 1 for each model, indicating that none of the models allowed behavior not found in the logs.

Note that close convergence to predictions is possible, because the distribution to be learnt is known in advance, and test data drawn from that distribution. Also, exact formulas rather than bounds are used to predict the numbers of traces.

The graph in Fig. 8(b) shows the *probability* of mining an approximately correct model, measured by f exceeding 0.9, 0.95, and 0.99, and d_{Bhat} not exceeding 0.1, 0.05, and 0.01. A single data point is calculated for each size log; the percentage of mined models for which f was above, or d_{Bhat} was below the threshold. The probability distance (solid lines) is less sensitive to the threshold used (all three lines are superimposed), due to operating over a greater range, whereas f (dashed lines) indicates convergence too soon.

Table II (top part) shows the predicted and actual numbers of traces and corresponding values of the metrics.

A. *Larger Example Process*

The running example is rather simple. To validate the methods and probabilistic analysis of Alpha, we used a larger example (Petri net Fig. 9, "ground truth" PDFA Fig. 10), which permits more detailed analysis and interpretation. This model is a sound WF net, and so is mineable by Alpha. It shows the handling of a request placed with a technical support call center. After the call is received (task i), three streams of activity run in parallel, synchronized at η[6] Next either g or h occurs, followed by a series of nested choices (e.g., various actions to resolve the call), before the call is closed (o).

This model has been artificially designed as a realistic process with a mix of simple, compound and nested substructures, and "extra" parallelism (parallel activities not part of a split or join substructure). Splits in the PDFA were allocated uniform probabilities. The PDFA was simulated to produce event logs from 5 to 150 traces in increments of 5 traces.

Table III shows for each substructure in the model the number of traces needed for 95% probability of mining the substructure correctly. "Global" indicates the number of traces calculated using the ground truth probabilities for each substring, e.g., $\pi(km)$ in the XOR split G. "Local" gives the number of traces using the local probabilities in each substructure, assuming traces that include that part of the model, e.g., $\pi(km| \rightarrow k)$. "Context" shows the number of traces given the substructure in its context in the model, i.e., for G, only $p(\rightarrow k)$ of the traces are expected to reach k, so the number of traces estimated in the "Local" column is divided by $p(\rightarrow k)$.

The graphs (Fig. 11) again show convergence as predicted (Table II). The shapes of the graphs suggest correspondence with the numbers of traces predicted for discovery of substructures (Table III), e.g., the "plateaus" between 30–35, 40–45, and 50–55 traces. By 30 traces $a \nleftrightarrow f$ (and XOR split H) will be mined correctly, with high confidence. By 40 traces, all the "extra parallelism" will be, and by 50 traces A should be discovered, giving confidence that most of the first (complex) part of the model will be correct. Finally, by 60 traces, with 95% confidence all substructures will be mined correctly. Between these points (35, 45, and 55 traces), there are no additional structures which are expected to be mined correctly. Fitness and behavioural appropriateness are both below 1 at low numbers of traces, indicating that the mined models do not fit all the traces in the log, and are also too general, allowing behavior not seen in the log. This is captured in the shape of the distance graphs at low numbers of traces, showing that convergence is initially slow.

VII. DISCUSSION OF MEASURES FOR ASSESSMENT OF PROCESS MINING RESULTS

The experimentation (Section VI) showed that distances between distributions give a clearer view of how different two process models are, than do the existing Petri net metrics. In this section, we discuss this further using the running example. Let PDFA A_0 (Fig. 3) describe the ground truth distribution over process traces for a simple process. Fig. 12(a) shows PDFA A_1 produced by a hypothetical process mining algorithm $\mathcal{L}_1$, mining from a particular log $\mathcal{W}_1$, a finite sample from the ground truth distribution. The trace frequencies in $\mathcal{W}_1$ vary from the ground truth probabilities, preventing $\mathcal{L}_1$ from creating PDFA A_1 with the exact ground truth probabilities.

[6]"Hidden" transition, not recorded in the log, which simplifies the depiction of the net. Alpha produces a behaviorally equivalent net without η.

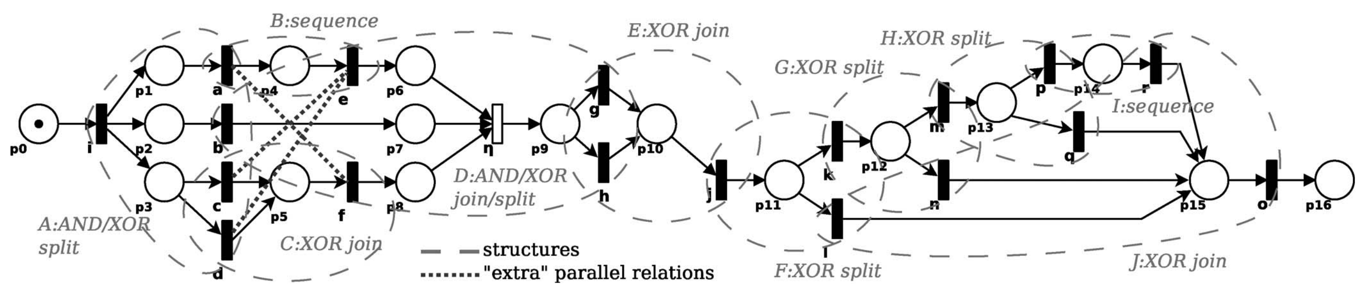

Fig. 9. Petri Net N_3 representing more complex example process with a selection of basic substructures and “extra” parallel relations.

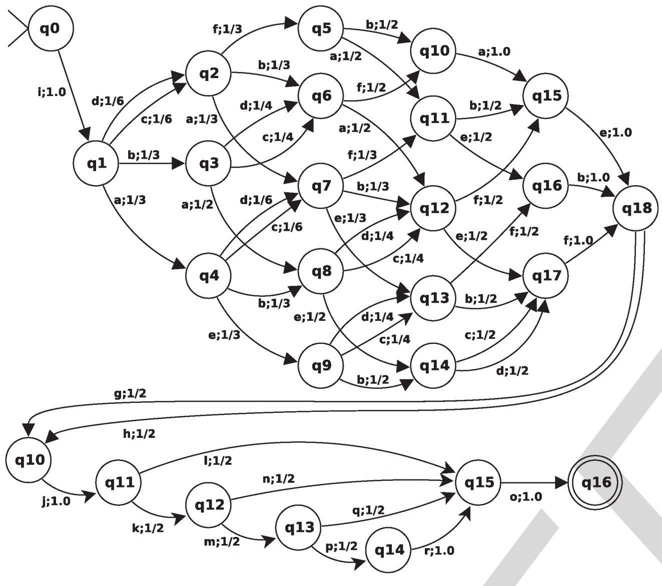

Fig. 10. PDFA A_3 corresponding to Petri net N_3, with the addition of transition probabilities, used for producing simulated event logs.

TABLE III
PREDICTED NUMBERS OF TRACES TO MINE SUBSTRUCTURES IN N_3

Structure	Global	Local	Context
A: AND/XOR split	49	49	49
B: Sequence	5	5	5
C: XOR join	13	**6**	6
$a \nleftrightarrow f, c \nleftrightarrow e, d \nleftrightarrow e$	$26, 51, 60$	$26, \mathbf{34}, \mathbf{36}$	$26, 34, 36$
D: AND/XOR join/split	56	56	56
E: XOR join	6	**1**	1
F, G, H: XOR splits	$6, 13, 28$	$6, \mathbf{6}, \mathbf{6}$	$6, \mathbf{12}, \mathbf{24}$
I: Sequence	23	**1**	**9**
J: XOR join.	28	**1**	1

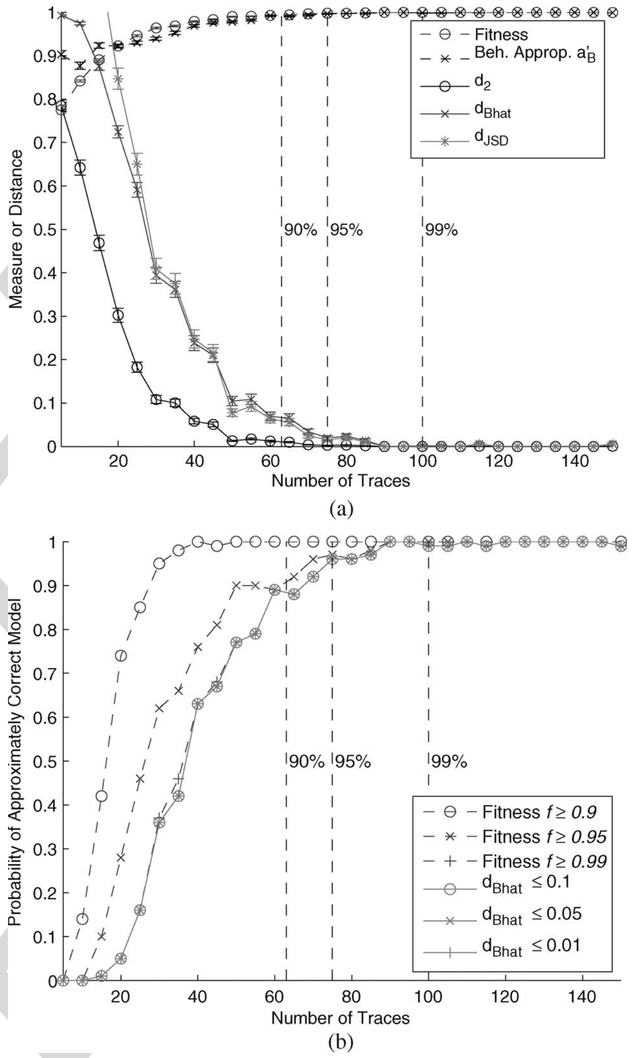

Fig. 11. Results showing convergence of Alpha to the ground truth, mining from logs of increasing size simulated from larger process model (PDFA A_3). (a) Average metrics against number of traces. (b) Probability of approximately correct model.

Petri net N_0 (Fig. 1) models the same process without probability information, in that it supports the same set of traces as the ground truth. The Petri net can be compared with the ground truth by converting to a PDFA by labeling its reachability graph (Fig. 3) with probabilities (Section III-A).

Another algorithm $\mathcal{L}_2$ might mine a Petri net directly, producing net N_1 [Fig. 12(b)]. This algorithm has failed to discover the parallelism, instead using an XOR split/join. This net and its corresponding PDFA A_2 [Fig. 12(c)] are structurally different from the ground truth, therefore supporting a different set of traces. This is a serious problem, as this model does not allow for both despatch of the product and billing.

Table IV shows the distances between these PDFA and the ground truth, using the distance measures described (scaled and subtracted from 1 to allow comparison with Fitness f). Models A_0, A_1 are measured as quite similar, but A_0, A_2 as almost 100% different. Although structurally “similar,” they support fundamentally different behavior since the split/join type has been changed. What is more, this part of the model accounts for 90% of the probable traces. Conversely, fitness

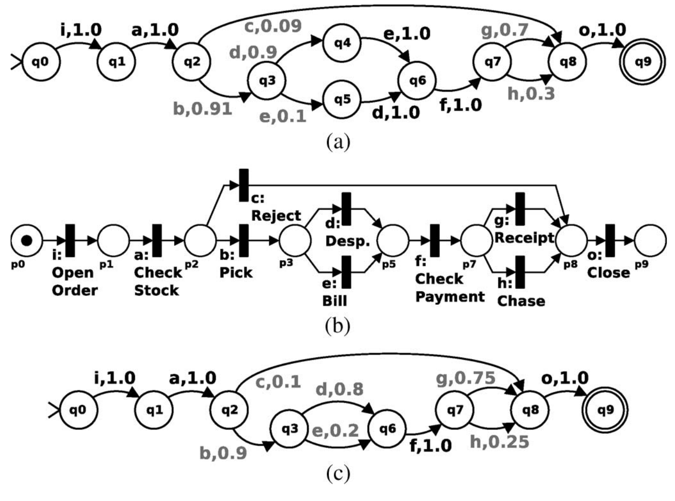

Fig. 12. PDFA and Petri nets produced by various mining algorithms. (a) PDFA A_1 differing from A_0 (Fig. 1) in probabilities only. (b) Petri net N_1 structurally different from N_0 (Fig. 1). (c) PDFA A_2 corresponding to Petri net N_1.

TABLE IV
ILLUSTRATION OF DISTANCES BETWEEN PROCESS MODELS

Models	$1-d_{Bhat}$	$1-d_2/\sqrt{2}$	$1-d_{JSD}/2$	**Fitness** f
$A_0 : A_1$	0.897	0.926	0.985	1.0
$A_0 : A_2$	0.051	0.413	0.1	0.893

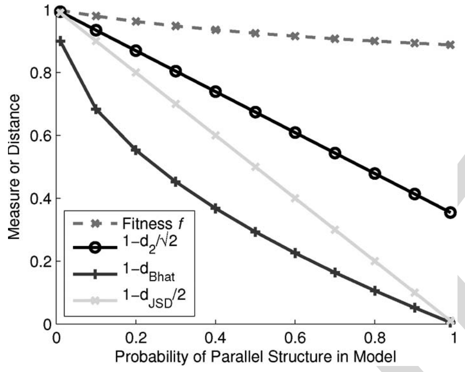

Fig. 13. Comparison of metrics: varying probability of parallel substructure.

f measures A_2 as relatively well fitting. Although it takes account of the frequency of nonfitting traces, it penalizes traces only (approximately) at the level of the nonfitting events. Thus, effectively, only event d or e is penalized in a nonfitting trace, while i, a, b, c, f, g, h, o are not. This leads to a misleading picture of the correctness of this model.

This can be seen further in the graph in Fig. 13. Here, we varied the probability of the part of the model containing the parallel substructure. The graph shows the closeness of the mined model to the ground truth, for the various metrics, as the probability of the parallel part of the model varies from very low, to very high. Fig. 13 suggests the distance metrics to be more analyzable [6] than Fitness (f), measuring the mined model to be almost optimal where the parallel substructure is unlikely to be involved (where the error in the model does not affect many traces), reducing to zero as traces involving the parallel substructure form the majority of the behavior.

VIII. Conclusion and Future Work

Many process discovery algorithms assume complete logs or only recreate the behavior in the log and do not recover model probabilities. However, real processes are probabilistic, so a log is only a sample of the true behavior. The amount of data needed to be confident in mining depends on the underlying distribution and on the behavior of the algorithm.

We discussed process mining from a machine learning viewpoint and introduced a probabilistic framework for considering processes and mining algorithms. We proposed that the primary task of mining the control flow of the process is to learn the ground truth distribution over process traces, from a finite random sample of process traces drawn from the ground truth. Process mining algorithms secondarily address additional requirements such as the representation language[7] to use, abstraction from detail, etc. Within this framework, process models may be compared using distances between the distributions which they generate, rather than representation-dependent methods and the behavior of algorithms considered in terms of their convergence to the ground truth.

We applied this framework to the Alpha algorithm [14], developing formulas for the probability of discovery of process substructures and using these to give the probability of mining a correct model of a specified accuracy. This analysis was then applied to two example models, confirming experimentally that the number of traces required to mine to a stated accuracy can be predicted.

We plan to extend this framework to other algorithms, to allow for models involving arbitrary substructures including cycles, and to simplify the prediction mechanism. This will allow us to apply the framework to the comparison of the behavior of different process mining algorithms and to develop deeper learning theory relating to process mining.

Appendix

A. Proof of Proposition 1

Proposition 1: The probability that Alpha infers $a >_n b$ is

$$P_\alpha(a >_n b) = 1 - (1 - \pi(ab))^n .$$

Proof: To infer that b can follow a, at least one of the n traces must contain substring ab, so the relation will be discovered unless all traces do not contain ab. A single trace contains ab with probability $\pi(ab)$, so all n independent traces *fail to* contain ab with probability $(1 - \pi(ab))^n$. ■

B. Proof of Proposition 2

Proposition 2: The probability that Alpha infers $a\#_n b$ is

$$P_\alpha(a\#_n b) = (1 - \pi(ab) - \pi(ba))^n .$$

[7]For example, [20] constructs probabilistic models of observed traces in the form of stochastic automata

Proof: To infer no relationship between a and b, each trace in the log must contain neither ab nor ba. This is $\neg(A \cup B)$ in Fig. 6(a). Since we assume no cycles, a single trace cannot contain both ab and ba, so $\pi(ab \wedge ba) = 0$. ■

C. Proof of Proposition 3

Proposition 3: The probability that Alpha infers $a \rightarrow_n b$ is

$$P_\alpha(a \rightarrow_n b) = (1 - \pi(ba))^n - (1 - \pi(ab) - \pi(ba))^n .$$

Proof: This is represented by the set $A \setminus B$ in Fig. 6(a), which can be seen to be equivalent to $\neg B \setminus \neg(A \cup B)$

$$\begin{aligned} \pi_n(B) &= 1 - (1 - \pi(ba))^n \quad \text{(by Prop. 1)} \\ \Rightarrow \pi_n(\neg B) &= (1 - \pi(ba))^n , \quad \text{and by Prop. 2} \qquad (9) \\ \pi_n\left(\neg(A \cup B)\right) &= (1 - \pi(ab) - \pi(ba))^n . \qquad (10) \end{aligned}$$

From (9) and (10), because $\neg(A \cup B) \subset \neg B$

$$\begin{aligned} P_\alpha(a \rightarrow_n b) &= \pi_n\left(\neg B \setminus \neg(A \cup B)\right) \\ &= \pi_n(\neg B) - \pi_n\left(\neg(A \cup B)\right) \\ &= (1 - \pi(ba))^n - (1 - \pi(ab) - \pi(ba))^n . \end{aligned}$$

This is intuitively interpretable as the probability of not seeing ba in any of n traces (good), minus the probability of *also* not seeing ab in any of those n traces (bad). ■

D. Proof of Proposition 4

Proposition 4: The probability that Alpha infers $a \|_n b$ is

$$P_\alpha(a \|_n b) = 1 - (1 - \pi(ab))^n - (1 - \pi(ba))^n + (1 - \pi(ab) - \pi(ba))^n .$$

Proof: The relations partition the set of possible logs [Fig. 6(a)]; thus, $P_\alpha(a \|_n b) = 1 - P_\alpha(a \rightarrow_n b) - P_\alpha(b \rightarrow_n a) - P_\alpha(a \#_n b)$, following the previous results. ■

E. Proof of Proposition 5

Proposition 5: Probability that Alpha infers an XOR split is

$$\begin{aligned} P_\alpha&\left(a \rightarrow_n (b_1 \# \ldots \# b_m)\right) \\ &= S_n(N) - \sum_{1 \le i \le m} S_n\left(N \cup \{Y_i\}\right) \\ &\quad + \sum_{1 \le i < j \le m} S_n\left(N \cup \{Y_i, Y_j\}\right) \\ &\quad - \ldots + (-1)^m S_n(N \cup Y), \quad \text{where} \qquad (11) \end{aligned}$$

$$S_n(X) = \left(1 - \sum_{1 \le i \le |X|} \pi(X_i) + \sum_{1 \le i < j \le |X|} \pi(X_i \wedge X_j) - \ldots + (-1)^{|X|} \pi\left(X_1 \wedge X_2 \wedge \ldots \wedge X_{|X|}\right)\right)^n . \qquad (12)$$

Proof: We begin with the probability that the pairs of tasks which *must not* be seen in the log do indeed not occur in the log, then use the "inclusion-exclusion principle" to remove the probability that any of the pairs of tasks which *must* be present in the log are also missing from the log.

For events E_i in a probability space with N events

$$\pi_n\left(\bigcup_{1 \le i \le N} E_i\right) = \sum_{1 \le i \le N} \pi_n(E_i) - \sum_{1 \le i < j \le N} \pi_n(E_i \cap E_j) + \cdots + (-1)^{N-1} \pi_n\left(\bigcap_{1 \le i \le N} E_i\right) \qquad (13)$$

As a simplified example, we consider discovery of a two-way XOR split from a to b, c, and assume $\pi(ba) = \pi(ca) = 0$. For Alpha to discover the split, the log must include ab and ac, but not bc or cb. If Fig. 6(b) represents the set of all logs of n traces, then let set A contain all logs which contain *no* ab, B*no* ac, C*no* bc, and D*no* cb. Then, we need the probability contained in the shaded area

$$\begin{aligned} \pi_n&\left((C \cap D) \setminus (A \cup B)\right) \\ &= \pi_n(C \cap D) - \pi_n\left((C \cap D) \cap (A \cup B)\right) \\ &= \pi_n(C \cap D) - \pi_n\left((C \cap D \cap A) \cup (C \cap D \cap B)\right) \\ &= \pi_n(C \cap D) - \pi_n(C \cap D \cap A) - \pi_n(C \cap D \cap B) \\ &\quad + \pi_n(C \cap D \cap A \cap B) \text{ (by equation 13).} \end{aligned}$$

$S_n(N)$ is represented by $(C \cap D)$, $S_n(N \cup Y_1)$ by $(C \cap D \cap A)$, etc. If we make no assumptions about the ground truth, then a single trace may include any of these substrings, so the same approach is needed to calculate $S_n(X)$. ■

F. Proof of Theorem 1

Theorem 1: The probability of discovery of an XOR split may be approximated by assuming independence between discovery of Alpha relations over n traces. The probability is overstated, but error rate decreases exponentially with increasing n

$$P_\alpha\left(a \rightarrow_n (b_1 \# \ldots \# b_m)\right) \le \prod_{1 \le i \le m} P_\alpha(a \rightarrow_n b_i) \times \prod_{1 \le i < j \le m} P_\alpha(b_i \#_n b_j). \qquad (14)$$

Proof: To demonstrate, we assume that an underlying model with an XOR split from a to $(b_1, b_2, \ldots)$ is followed without error, and traces are recorded without error ("noise-free"). Thus, $\pi(b_1 a) = \pi(b_1 b_2) = 0$, etc., and the previous equations may be simplified. Equation (1) reduces to $P_\alpha(a \rightarrow_n b) = 1 - (1 - \pi(ab))^n$, and so on.

Let b_i be shorthand for $\pi(ab_i)$, and label (2) as $F(n)$ and (14) as $G(n)$. Equation (2) (discovery of multiple Alpha relations from one log *not* independent) reduces to

$$F(n) = 1 - \sum_{1 \le i \le m} (1 - b_i)^n + \sum_{1 \le i < j \le m} (1 - b_i - b_j)^n - \sum_{1 \le i < j < k \le m} (1 - b_i - b_j - b_k)^n + \ldots + (-1)^m \left(1 - \sum_{1 \le i \le m} b_i\right)^n$$

while (14), which assumes that discovery of the relations *can* be treated as independent, to

$$G(n) = \prod_{1\le i\le m} P_\alpha(a \to_n b_i) = \prod_{1\le i\le m} \left(1-(1-b_i)^n\right)$$
$$= 1 - \sum_{1\le i\le m}(1-b_i)^n + \sum_{1\le i<j\le m}(1-b_i)^n(1-b_j)^n$$
$$- \cdots + (-1)^m \prod_{1\le i\le m}(1-b_i)^n.$$

The error in assuming independent relations is given by $H(n) = |F(n) - G(n)|$. The first two terms of $F(n)$ and $G(n)$ cancel, leaving $(m-1)$ terms. The difference between the third terms of $F(n)$ and $G(n)$ determines the rate of decay of the error, since the absolute values of subsequent terms in $F(n)$ will be not greater than the third term. This is because the value of each term, and all b_i, will be between 0 and 1, so the terms are decreasing in absolute value. Similarly for $G(n)$, because each subsequent term is multiplied by a further factor between 0 and 1, itself decreasing exponentially. Now, let

$$\begin{aligned} f_{ij}(n) &= (1-b_i-b_j)^n & h_{ij}(n) &= g_{ij}(n) - f_{ij}(n) \\ g_{ij}(n) &= (1-b_i)^n(1-b_j)^n & \lambda_{ij} &= 1-b_i-b_j \\ &= (1-b_i-b_j+b_ib_j))^n & \mu_{ij} &= \lambda_{ij} + b_ib_j. \end{aligned}$$

Then, $h_{ij}(n) = \mu_{ij}^n - \lambda_{ij}^n$, so the error is bounded by

$$H(n) \le (m-1)\left[\sum_{1\le i<j\le m}\left(\mu_{ij}^n - \lambda_{ij}^n\right)\right]. \tag{15}$$

This is always positive, since $\mu_{ij} > \lambda_{ij}$ for all i, j, and decays exponentially in n after a maximum at relatively low n.

The rate of decay of the error is also exponential

$$h'_{ij}(n) = \mu_{ij}^n \ln \mu_{ij} - \lambda_{ij}^n \ln \lambda_{ij}.$$

This is always negative after $h_{ij}(n)$ reaches its maximum and decays exponentially in n, as the log factors are relatively negligible. The maximum error in term $h_{ij}(n)$ is reached when

$$h'(n) = \mu_{ij}^n \ln \mu_{ij} - \lambda_{ij}^n \ln \lambda_{ij} = 0$$
$$\Rightarrow n = \frac{\ln\left(\frac{\ln \lambda_{ij}}{\ln \mu_{ij}}\right)}{\ln\left(\frac{\mu_{ij}}{\lambda_{ij}}\right)}. \tag{16}$$

n will be largest when $\lambda_{ij} \approx \mu_{ij}$, when the denominator of (16) tends to 0. This occurs when when the probabilities are small, as the difference between λ_{ij} and μ_{ij} is $\pi(ab_i)\pi(ab_j)$. However, the discovery probability $F(n)$ or $G(n)$ will be correspondingly small, due to the second terms of $F(n), G(n)$. With the number of traces required to give only a 50% probability of discovery across all possibilities for the probabilities in a three-way split, the difference in the number of traces predicted using $F(n)$ and $G(n)$ is negligible. ■

References

[1] W. M. P. van der Aalst and A. J. M. M. Weijters, "Process mining: A research agenda," *Comput. Ind.*, vol. 53, no. 3, pp. 231–244, 2004.

[2] A. Tiwari, C. Turner, and B. Majeed, "A review of business process mining: State-of-the-art and future trends," *Bus. Process Manag. J.*, vol. 14, no. 1, pp. 5–22, 2008.

[3] B. F. van Dongen and A. Adriansyah, "Process mining: Fuzzy clustering and performance visualization," in *Proc. BPM Workshops*, S. Rinderle-Ma, S. W. Sadiq, and F. Leymann, Eds., 2009, vol. 43, pp. 158–169.

[4] A. K. Alves de Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters, "Quantifying process equivalence based on observed behavior," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 55–74, 2008.

[5] A. Rozinat, A. K. Alves de Medeiros, C. W. Günther, A. J. M. M. Weijters, and W. M. P. van der Aalst, "Towards an evaluation framework for process mining algorithms," BPM Center, Eindhoven, The Netherlands, BPM Center Report BPM-07-06, 2007.

[6] A. Rozinat and W. M. P. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Inf. Syst.*, vol. 33, no. 1, pp. 64–95, Mar. 2008.

[7] A. Rozinat and W. M. P. van der Aalst, "Decision mining in ProM," in *Proc. Bus. Process Manag.*, S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, Eds., 2006, vol. 4102, pp. 420–425.

[8] W. M. P. van der Aalst, H. A. Reijers, and M. Song, "Discovering social networks from event logs," *Comput. Supported Cooperative Work*, vol. 14, no. 6, pp. 549–593, Dec. 2005.

[9] G. Greco, A. Guzzo, and L. Pontieri, "Discovering expressive process models by clustering log traces," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 8, pp. 1010–1027, Aug. 2006.

[10] M. Song, C. W. Günther, and W. M. P. van der Aalst, "Trace clustering in process mining," in *Proc. BPM Workshops*, D. Ardagna, M. Mecella, and J. Yang, Eds., 2008, vol. 17, pp. 109–120.

[11] R. P. J. C. Bose and W. M. P. van der Aalst, "Context aware trace clustering: Towards improving process mining results," in *Proc. SDM*, 2009, pp. 401–412.

[12] C. W. Günther and W. M. P. van der Aalst, "Fuzzy mining—Adaptive process simplification based on multi-perspective metrics," in *Proc. BPM*, G. Alonso, P. Dadam, and M. Rosemann, Eds., 2007, vol. 4714, pp. 328–343.

[13] C. W. Günther, A. Rozinat, and W. M. P. van der Aalst, "Activity mining by global trace segmentation," in *Proc. BPM Workshops*, S. Rinderle-Ma, S. W. Sadiq, and F. Leymann, Eds., 2009, vol. 43, pp. 128–139.

[14] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.

[15] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. Alves de Medeiros, "Process mining with the HeuristicsMiner algorithm," in *Proc. BETA Working Paper Series 166*, 2006, pp. 1–34.

[16] L. Wen, W. M. P. van der Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 145–180, Oct. 2007.

[17] G. Schimm, "Mining exact models of concurrent workflows," *Comput. Ind.*, vol. 53, no. 3, pp. 265–281, Apr. 2004.

[18] W. M. P. van der Aalst, V. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: A two-step approach to balance between underfitting and overfitting," *Softw. Syst. Model.*, vol. 9, no. 1, pp. 87–111, 2010.

[19] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process mining based on regions of languages," in *Proc. BPM*, G. Alonso, P. Dadam, and M. Rosemann, Eds., 2007, vol. 4714, pp. 375–383.

[20] D. R. Ferreira and D. Gillblad, "Discovering process models from unlabelled event logs," in *Proc. BPM*, U. Dayal, J. Eder, J. Koehler, and H. A. Reijers, Eds., 2009, vol. 5701, pp. 143–158.

[21] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic finite-state machines—Part I," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 7, pp. 1013–1025, Jul. 2005.

[22] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 5, pp. 564–577, May 2003.

[23] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc.*, vol. 35, pp. 99–109, 1943.

[24] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *ACM Trans. Softw. Eng. Methodol.*, vol. 7, no. 3, pp. 215–249, Jul. 1998.

[25] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," in *Proc. EDBT*, H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, Eds., 1998, vol. 1377, pp. 469–483.

[26] OMG, Business Process Model and Notation (BPMN), 2009. [Online]. Available: http://www.omg.org

[27] A. Datta, "Automating the discovery of AS-IS business process models: Probabilistic and algorithmic approaches," *Inf. Syst. Res.*, vol. 9, no. 3, pp. 275–301, Sep. 1998.

[28] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, Jul. 2003.

[29] J. Herbst, "A machine learning approach to workflow management," in *Proc. ECML*, R. L. de Mántaras and E. Plaza, Eds., 2000, vol. 1810, pp. 183–194.

[30] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, Sep. 1977.

[31] W. M. P. van der Aalst, "The application of Petri Nets to workflow management," *J. Circuits, Syst. Comput.*, vol. 8, no. 1, pp. 21–66, 1998.

[32] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst, "Genetic process mining: An experimental evaluation," *Data Mining Knowl. Discovery*, vol. 14, no. 2, pp. 245–304, Apr. 2007.

[33] C. J. Turner, A. Tiwari, and J. Mehnen, "A genetic programming approach to business process mining," in *Proc. GECCO*, C. Ryan and M. Keijzer, Eds., 2008, pp. 1307–1314.

[34] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, "Robust process discovery with artificial negative events," *J. Mach. Learn. Res.*, vol. 10, pp. 1305–1340, Jun. 2009.

[35] S. Sun, Q. Zeng, and H. Wang, "Process-mining-based workflow model fragmentation for distributed execution," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 2, pp. 294–310, Mar. 2011.

[36] L. V. Allen and D. M. Tilbury, "Anomaly detection using model generation for event-based systems without a preexisting formal model," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 3, pp. 654–668, May 2012.

[37] J. E. Cook and A. L. Wolf, "Software process validation: Quantitatively measuring the correspondence of a process to a model," *ACM Trans. Softw. Eng. Methodol.*, vol. 8, no. 2, pp. 147–176, Apr. 1999.

[38] J. Bae, L. Liu, J. Caverlee, L.-J. Zhang, and H. Bae, "Development of distance measures for process mining, discovery, and integration," *Int. J. Web Serv. Res.*, vol. 4, no. 4, pp. 1–17, Oct.–Dec. 2007.

[39] T. Calders, C. W. Günther, M. Pechenizkiy, and A. Rozinat, "Using minimum description length for process mining," in *proc. SAC*, S. Y. Shin and S. Ossowski, Eds., 2009, pp. 1451–1455.

[40] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, "The ProM Framework: A new era in process mining tool support," in *Proc. ICATPN*, G. Ciardo and P. Darondeau, Eds., 2005, vol. 3536, pp. 444–454.

[41] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 145–151, Jan. 1991.

[42] D. Angluin, "Computational learning theory: Survey and selected bibliography," in *Proc. STOC*, 1992, pp. 351–369.

[43] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

Philip Weber received the B.Sc. degree in computer science from Loughborough University, Leicestershire, U.K., in 1994, and the M.Sc. degree in advanced computer science from Birmingham University, Birmingham, U.K., in 2009. Between these, he worked in industry designing, analyzing, and implementing information technology systems, and in systems administration. He is currently working toward the Ph.D. degree in computer science at the University of Birmingham, Birmingham.

His research interests include process mining, machine learning, data mining, and information management.

Behzad Bordbar received the B.Sc., M.Sc., and Ph.D degrees in mathematics (Ph.D. from Sheffield, U.K.).

Following the Ph.D. degree, he worked as a Researcher on a number of projects at the University of Ghent, Belgium, and the University of Kent, U.K. He is currently affiliated to the School of Computer Science, University of Birmingham, U.K., where he teaches courses in software engineering and distributed systems. In recent years, he has had close collaborative research with various academic and industrial organizations, among them Ghent University, Osaka University, Colorado State University, BT, IBM, and HP research laboratories. His research activities are mostly aimed at using modeling to produce more dependable software and systems in shorter development cycles and at a lower cost. His current research projects are dealing with formal methods, model analysis, software tools, model-driven development, and fault tolerance in service-oriented architectures and cloud.

Peter Tiño received the M.Sc. degree from the Slovak University of Technology, Bratislava, Slovakia, in 1988, and the Ph.D. degree from the Slovak Academy of Sciences, Bratislava, in 1997.

From 1994 to 1995, he was a fulbright fellow with the NEC Research Institute, Princeton, NJ. He was a Postdoctoral Fellow with the Austrian Research Institute for AI, Vienna, Austria, from 1997 to 2000, and a Research Associate with Aston University, Birmingham, United Kingdom, from 2000 to 2003. From 2003 to 2006, he was a Lecturer with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham, United Kingdom, and has been a Senior Lecturer there since 2007. His current research interests include probabilistic modeling and visualization of structured data, statistical pattern recognition, dynamical systems, evolutionary computation, and fractal analysis. AQ2

Dr. Tiño received the fulbright fellowship in 1994 and the United Kingdom-Hong Kong Fellowship for Excellence in 2008. He received the Outstanding Paper of the Year Award from the IEEE TRANSACTIONS ON NEURAL NETWORKS with T. Lin, B. G. Horne, and C. L. Giles in 1998 for his work on recurrent neural networks and 2010 IEEE CIS Outstanding Paper Award with S. Y. Chong and X. Yao for a paper in the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION on generalization on coevolutionary learning. He received the 2002 Best Paper Award at the International Conference on Artificial Neural Networks with B. Hammer. He is on the editorial board of several journals.

AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES

Please note that your paper will incur an overlength page charge of $175 per page for every page over the page limit. The page limit for regular papers is 12 pages, and the page limit for correspondence papers is 6 pages. If your paper exceeds these limits, you are responsible for payment of the pages beyond the limit.

AQ1 = Please define “XOR.”
AQ2 = Please expand “NEC.”

END OF ALL QUERIES