A Principled Approach to the Analysis of Process Mining Algorithms

Phil Weber, Behzad Bordbar, and Peter Tiňo

School of Computer Science, University of Birmingham, B15 2TT, UK.
{p.weber,b.bordbar,p.tino}@cs.bham.ac.uk

Abstract. Process mining uses event logs to learn and reason about business process models. Existing algorithms for mining the control-flow of processes in general do not take into account the probabilistic nature of the underlying process, which affects the behaviour of algorithms and the amount of data needed for confidence in mining. We contribute a first step towards a novel probabilistic framework within which to talk about approaches to process mining, and apply it to the well-known Alpha Algorithm. We show that knowledge of model structures and algorithm behaviour can be used to predict the number of traces needed for mining.

Keywords: Business process mining, probabilistic automata, Petri nets.

1 Introduction

Business processes describe sets of related activities which are carried out to solve a business problem, or produce a service or product. As a process is executed, the systems involved will record information in log files. Process mining [7] uses these logs to discover and analyse models of business processes.

As a simple example, consider the process in Fig.1. An order is received, stock checked, and either the item picked from the warehouse, or the order rejected. Despatch and billing take place in parallel, then payment may be chased repeatedly, before the order is closed. Abstracting from detail, the 'trace' of a single enactment of the process may be recorded as a string *abdefggh*. Process mining algorithms use logs of traces to produce models such as this Petri net.

Various techniques exist, reviewed in [7]. Other than [3, 1], non-probabilistic languages (e.g. Petri nets, BPMN) are usually used to represent processes. The aim is usually to represent the control-flow structure in a model that is visually understandable, using heuristics [10] or clustering [6, 2] to abstract from excessive detail or noise. Probabilities are generally not represented, and algorithms assume a 'complete' log, for some notion of completeness. Comparison of models is by syntactic methods such as replaying logs or Petri net token behaviour [5].

Little work has been done on systematically analysing process mining algorithms to discover their fundamental properties, or analysing the completeness of logs. Yet these aspects are of critical importance to enable confidence that the log is an adequate sample of the underlying behaviour, and thus in the accuracy

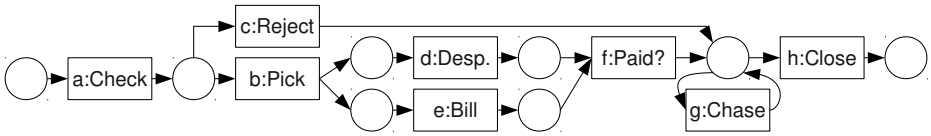

Fig. 1. Simplified Business Process for fulfilling an order.

of the mined model. While the core interest is in the control-flow of a process, it must be appreciated that traces are generated randomly according to an underlying probability distribution unknown to the process mining algorithm. Not all activity sequences or decisions are equally likely, and their probabilities may have a dramatic effect on the amount of data needed for mining.

This paper contributes a first step towards a novel probabilistic framework within which to talk about approaches to process mining (section 2). We suggest a radically new view on process mining algorithms, in which a process is viewed as a distribution over traces of activities, and mining algorithms in terms of their ability to learn such distributions. We use probabilistic automata as a unifying representation, and compare models using distances between the probability distributions which they generate.

As an illustrative example, we apply this framework in section 3 to the foundational algorithm 'Alpha' [8]. Unlike previous methods, the framework allows us to answer in a principled manner the question of the probability of identifying the correct process from a given log of data. We show that a process model can be broken into structures and the probability estimated of correct mining of those structures, and thus of the original process model. Some experimental results are presented in section 4, and section 5 concludes the paper.

2 Processes as Distributions over Strings of Symbols

Similar to the approach in [1], we view processes as probability distributions over strings of symbols. Activities occur according to a "ground truth" process model $\mathcal{M}$ which may be unknown. We consider only acyclic process models, and place restrictions on processes equivalent to those used elsewhere, e.g. [8]: A process has a single start task s and end task e; the events of activities' occurrence are atomic (take no time) and are recorded as they occur in a workflow log W; and the underlying process model is fixed. A sequence of activities from start to end task is called a process trace. The log is therefore a multiset of traces.

Let Σ be an alphabet of symbols representing business activities. Process traces are represented by strings $\{x \in \Sigma^+\}$. $\mathcal{M}$ is therefore a stochastic regular language, describing a probability distribution $P_{\mathcal{M}}$ over Σ^+. The probability of trace x occurring is $P_{\mathcal{M}}(x)$, such that $\sum_{x \in \Sigma^+} P_{\mathcal{M}}(x) = 1$. The set of valid process traces is given by the finite support of $P_{\mathcal{M}}$. A process mining algorithm can therefore be viewed as learning a probability distribution $P_{\mathcal{M}'}$ over strings, to approximate $P_{\mathcal{M}}$, i.e. $P_{\mathcal{M}'}(x) \approx P_{\mathcal{M}}(x), \forall x \in \Sigma^+$. Learning is from the finite sample W drawn *i.i.d.* from the distribution to be learnt, $P_{\mathcal{M}}$.

For the purposes of analysis we use probabilistic deterministic finite automata (PDFA) [9], which have the bare minimum needed to represent distributions generated by business processes. A PDFA is a five-tuple $A = (Q_A, \Sigma, \delta_A, q_0, q_F)$, where Q_A is finite set of states including single start and end states q_0, q_F; Σ is an alphabet of symbols; and $\delta_A : Q_A \times \Sigma \times Q_A \to [0, 1]$ defines the probability function governing transition between states. The probabilities on transitions from a state sum to 1, and the transition function is deterministic: given a current state and symbol, the next state is certain, and there is a unique state path through A for any string x that it can parse. All states are accessible from the initial state, and from any state, it is possible to reach the final state.

PDFA A generates a probability distribution P_A on Σ^+:

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

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, and q_{s_i} denotes the state reached after symbol s_i is parsed. $P_A(x) = 0$ for strings which cannot be parsed.

Processes characterised in this way are equivalent to those represented by *sound* Workflow Nets [8], with the addition of probabilities on transitions. The restricted Hidden Markov Models used in [3] are similar in behaviour to our PDFA, since each state is restricted to a single output activity.

3 An Illustrative Example — The Alpha Algorithm

In this section we use this framework to analyse the behaviour of the process mining algorithm 'Alpha' with regard to the probability of it correctly re-discovering the process structures in a known ground truth, from a given log file.

The Alpha algorithm [8] makes a single pass through a workflow log to identify which tasks directly follow each other. This information is used to infer three basic relations between task pairs, which are used to construct a Petri Net:

- $a \to b$ (task b always follows a, never vice-versa),
- $a \# b$ (a and b never follow each other), and
- $a \parallel b$ (both ab and ba occur in the log).

A single start and end place are assumed, and the remaining places inferred using these relations. Two tasks are always related by $\to$, $\to^{-1}$, $\#$ or $\parallel$, and these relations partition the set of tasks [8, Property 3.1]. Acting on a pair of tasks, these relations also partition the set of all logs of n traces. Alpha is proven to mine processes representable by a sub-class of Petri nets, from noise-free logs.

Business processes are composed of structures (Fig.2,3). For acyclic processes, Alpha can discover sequences of tasks, exclusive (XOR) and parallel (AND) splits and joins. We give examples of these structures and how they may be represented by PDFA, and state the formulae for the probability of Alpha discovering them from a log of n process traces.

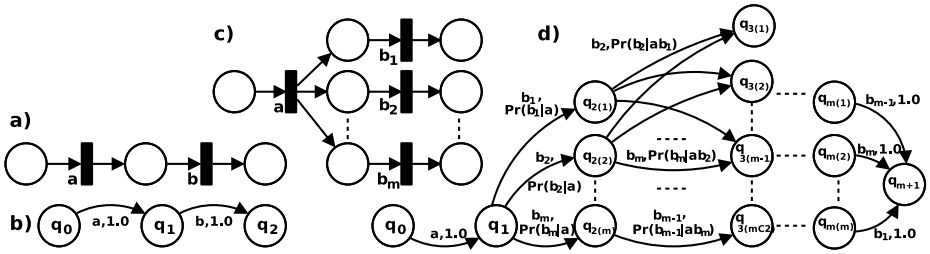

Fig. 2. Petri Net and PDFA fragments for Sequence a),b) and Parallel Split c),d).

We first state formulae for the probability of Alpha discovering each of the basic relations, when acting on a log of n traces, based on the probabilities of strings in the log. Let $\pi(ab)$ be shorthand for $P_{\mathcal{M}}(s\Sigma^* ab\Sigma^* e)$, the probability of ab occurring in a trace. We define $\pi_n(E)$ as "the probability of complex event E holding true in a log of n traces", and $P_\alpha(a \rightarrow_n b)$ as "the probability that Alpha infers the relation $a \rightarrow b$ over n traces", similarly for the other Alpha relations. Alpha will then discover the basic relations with the following probabilities:

$$P_\alpha(a \rightarrow_n b) = \big(1 - \pi(ba)\big)^n - \big(1 - \pi(ab) - \pi(ba) + \pi(ab \wedge ba)\big)^n, \tag{2}$$

$$P_\alpha(a \#_n b) = \big(1 - \pi(ab) - \pi(ba) + \pi(ab \wedge ba)\big)^n, \text{ and} \tag{3}$$

$$P_\alpha(a \parallel_n b) = 1 - \big(1 - \pi(ab)\big)^n - \big(1 - \pi(ba)\big)^n + \big(1 - \pi(ab) - \pi(ba) + \pi(ab \wedge ba)\big)^n. \tag{4}$$

We next give exact formulae for the discovery of basic structures, and show how these may be usefully simplified without loss of accuracy. Due to space restrictions we do not provide full derivations; these will be published elsewhere.

3.1 Control-Flow Structures

Here we consider sequential activities, and exclusive and parallel splits and joins.

Sequential Activities: If a occurs, it is immediately followed *in the model* by b (Fig.2). 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. Discovery simply requires discovery of the causal relationship $a \rightarrow_n b$ (equation 2).

Splits and Joins: Alpha uses the relations $\rightarrow$, $\#$ and $\parallel$ between pairs of tasks to locate places in the net, which characterises splits and joins as XOR or AND. As the discovery of a relation is a complex event arising from Alpha's interpretation of a log of n traces, they are not independent: multiple relations may be inferred, or not, from the log. Therefore to obtain exact probabilities for discovery of splits and joins it is necessary to use the probabilities of sub-strings which 'must' and "must not" be seen in the log, to build the formulae for larger structures.

Exclusive Choice: XOR Split: An m-way XOR split (e.g. Fig.3 structure A) occurs where there is a choice between m exclusive paths through the model after task a, each path starting with a task $\{b_1 \dots b_m\}$. If a occurs in a trace, then exactly one $b_i \in \{b_1 \dots b_m\}$ will be included in the remainder of the trace.

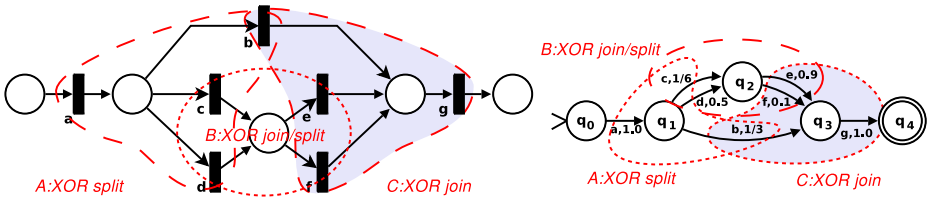

Fig. 3. Example Model as Petri Net and PDFA, Highlighting Structures.

To discover this split, Alpha must infer each $a \rightarrow_n b_i$ and each $b_i \#_n b_j$. So the log must contain at least one of each of m sub-strings ab_i, none of the m 'reverse' strings $b_i a$, and none of $\frac{m!}{(m-2)!}$ pairs of 'post-split' tasks. Let $N, Y \subset \Sigma \times \Sigma$ be the set of task pairs which *must not* (resp. *must*) be seen in the log. $S_n(X) \rightarrow [0,1]$, where $X \subseteq \Sigma \times \Sigma$, is the probability of not seeing any of the $|X|$ task pairs in n traces, and for $X_i = (t_i, t_i') \in X, \pi(X_i) = \pi(t_i t_i')$. Using the "inclusion-exclusion" principle for calculating the probability of intersecting events, applied to both strings within a trace, and traces within a log:

$$P_\alpha\big(a \rightarrow_n b_1 \# \dots \# b_m\big) = S_n(N) - \sum_{i=1}^{i=m} S_n(N \cup \{Y_i\}) + \sum_{i,j=1:i<j}^{i,j=m} S_n(N \cup \{Y_i, Y_j\}) - \dots + (-1)^m S_n(N \cup Y), \text{ where} \quad (5)$$

$$S_n(X) = \Big(1 - \sum_{i=1}^{i=|X|} \pi(X_i) + \sum_{i,j=1:i<j}^{i,j=|X|} \pi(X_i \wedge X_j) \dots + (-1)^{|X|} \pi(X_1 \wedge \dots \wedge X_{|X|})\Big)^n.$$

Parallel Split: In an m-way AND split (Fig.2), after task a, m paths may proceed in parallel. Each path starts with a task $\{b_1 \dots b_m\}$. If a occurs in a trace, then the remainder of the trace will contain each $b_i \in \{b_1 \dots b_m\}$ before the end of the trace, in one of $m!$ permutations. Since PDFA do not explicitly represent parallelism, the fragment using XOR splits is more complex than the Petri net equivalent (Fig.2). After the first parallel task there are $\binom{m}{1}$ possible states, $\binom{m}{2}$ after the second, and so on to $\binom{m}{m-1}$ before the last parallel task.

The equations given for XOR splits can be modified to give the probability of discovery of XOR joins, and AND splits and joins, with sets Y and N populated with the required "must see" and "must not see" pairs of tasks.

3.2 Simplifying the Formulae

Given knowledge of the ground truth, many terms in these equations may be zero. Nevertheless, they can become cumbersome to work with, requiring knowledge of many probabilities. Nor do they relate intuitively to the working of the algorithm. Next we discuss how these formulae can be effectively simplified without loss of accuracy to give formulae which intuitively follow from the working of the Alpha algorithm, and are simpler to calculate. We denote the probability of discovery of structure S by Alpha, as $P_\alpha(S)$.

Lemma 1. *The probability of discovery of splits and joins may be usefully approximated by treating the probabilities of discovery of the Alpha relations over n traces as independent, and multiplying. The probability is over-stated but the error rate decreases exponentially with increasing n. For a general split/join structure (B in Fig.3) where m paths of which p are XOR (the remainder parallel) join and then split to n paths of which q are XOR (the remainder parallel):*

$$P_\alpha(S) \leq \prod_{i,j=1}^{i=m,j=n} P_\alpha(a_i \rightarrow_n b_j) \times \prod_{i,j=1:i<j}^{i,j=p} P_\alpha(a_i \#_n a_j) \times \prod_{i,j=1:i<j}^{i,j=q} P_\alpha(b_i \#_n b_j) \times \prod_{i,j=(m-p):i<j}^{i,j=m} P_\alpha(a_i \parallel_n a_j) \times \prod_{i,j=(n-q):i<j}^{i,j=n} P_\alpha(b_i \parallel_n b_j) \quad (6)$$

Due to space restrictions the proof will be published elsewhere. In summary, the error in the approximation is the difference between equations 5 and 6. Using $p_i \in [0,1]$ as shorthand for $\pi(ab_i)$, etc., this error is bounded by the sum of terms of the form $(1-p_i)^n(1-p_j)^n - (1-p_i-p_j)^n = (1-p_i-p_j+p_ip_j)^n - (1-p_i-p_j)^n$, which decay exponentially in n, after a maximum at relatively low n.

These probabilities for discovery of structures in a model can be combined to give the probability of successful mining by Alpha of a whole model.

4 Experiments

We designed a simple artificial process model (Fig.3) as the ground truth, and used the original (section 3.1) and simplified (3.2) formulae to predict the number of traces needed to mine a correct model (table 1). There is very little difference between the predictions, the simplification giving a slight underestimate. Logs in the MXML format were simulated from the automaton; 100 samples of logs from 10 to 300 traces in increments of 10, with a ground truth log of 10000 traces, assumed to be complete and distributed approximately according to the ground truth. Alpha in ProM (`www.processmining.org`) was used to mine these logs. Since Alpha produces non-probabilistic Petri nets, to compare the structure of the mined models with the ground truth structure, the nets were converted to probabilistic automata using their Reachability Graphs and labelling splits with maximum likelihood probabilities from the ground truth log.

Table 1. Predicted *vs.* Actual Number of Traces for Probability of Successful Mining

Probability	Exact Prediction	Simplified	Actual Traces
90%	132	131	90–100
95%	170	170	130–140
99%	263	262	280–290

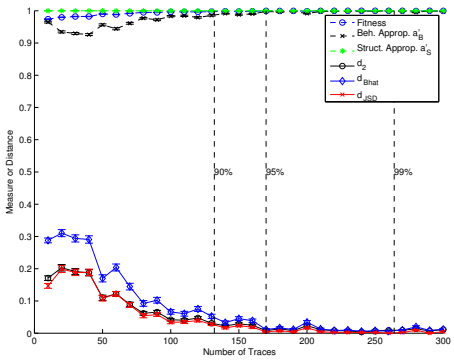

Fig. 4. Average Metrics Against Number of Traces

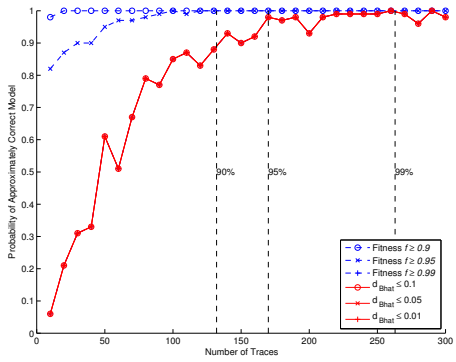

Fig. 5. Probability of 95% Approximately Correct Model

The d_2 and Bhattacharyya [4] distances, and the Jensen-Shannon Divergence (based on Kullback-Leibler) were used to calculate the average difference between the ground truth and automata mined from each log size. These and the 'Fitness' (recall) and Behavioural Appropriateness (precision) metrics [5] were plotted against number of traces (Fig.4). The graph shows that approximate correctness of the mined models converges at approximately the predicted points.

The distance measures seem more discriminating, being distributed over a clearer scale, compared with 'Fitness'. This is more apparent in Fig.5, which shows the *probability* of mining an approximately correct model, as measured by 'Fitness' (f) and the Bhattacharyya distance (D_{Bhat}), for various thresholds of approximate correctness. This is only a rough measure, as a single data point is calculated for each log file size; a count of the number of experiments from the 100 carried out for each size, for which the distance was below the threshold. The probability distance measure seems less sensitive to the threshold used, whereas 'Fitness' indicates convergence too soon, except for the 99% threshold.

5 Conclusion

Most process mining algorithms attempt to mine structural models of activities and relations between them, from a log that is assumed to be complete, and do not model probabilities. This does not provide for a way to know how much data is needed to be confident in mining results.

We suggest a novel probabilistic framework for considering business processes and process mining algorithms. The underlying business process is a distribution over strings of activities, and the primary task of mining the control-flow of the process is to learn this "ground truth" distribution, from a finite random sample of process traces which are drawn *i.i.d.* from the ground truth. Process mining algorithms then secondarily address additional requirements such as the representation language to use, or display detail or abstraction. Within this framework, process models may be compared using distances between the distributions which they generate, rather than ad-hoc or syntactic methods, and the behaviour of algorithms in terms of their convergence to the ground truth.

Applying this framework to the Alpha algorithm [8] we showed that using the structures in a model it is possible to accurately predict how much data will be needed to, with a given level of confidence, mine a model that is correct to a specified accuracy. We plan to apply this framework to other process mining algorithms and develop deeper learning theory relating to process mining.

Acknowledgments P. Weber is supported by a Doctoral Training Grant funded by EPSRC and the School of Computer Science, University of Birmingham.

References

1. Ferreira, D. R. and Gillblad, D. Discovering Process Models from Unlabelled Event Logs. In Dayal, U., Eder, J., Koehler, J., and Reijers, H. A. (eds.), *BPM 2009*. LNCS, vol. 5701, pp. 143–158. Springer, 2009.
2. Günther, C. W. and van der Aalst, W. M. P. Fuzzy Mining - Adaptive Process Simplification Based on Multi-Perspective Metrics. In Alonso, G., Dadam, P., and Rosemann, M. (eds.), *BPM 2007*. LNCS, vol. 4714, pp. 328–343. Springer, 2007.
3. Herbst, J. and Karagiannis, D. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaption of Workflow Models. *Int. J. Intell. Syst. Account. Financ. Manage.*, 9(2):67–92. 2000.
4. Kailath, T. Divergence and Bhattacharyya Distance Measures in Signal Selection. *IEEE Trans Communication Technology*, CM-15(1):52 – 60, 1967.
5. Rozinat, A. and van der Aalst, W. M. P. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.
6. Song, M., Günther, C. W., and van der Aalst, W. M. P. Trace Clustering in Process Mining. In Ardagna, D., Mecella, M., and Jian Yang (eds.), *Business Process Management Workshops*. LNBIP, vol. 17, pp. 109–120. Springer, 2008.
7. Tiwari, A., Turner, C. J., and Majeed, B. A Review of Business Process Mining: State-of-the-Art and Future Trends. *Bus. Process Manage. J.*, 14(1):5 – 22, 2008.
8. van der Aalst, W. M. P., Weijters, T., and Maruster, L. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–42, 2004.
9. Vidal, E., Thollard, F., de la Higuera, F., Casacuberta, F., and Carrasco, R. C. Probabilistic Finite-State Machines - Part I. *IEEE Trans. Pattern Anal.*, 27(7):1013 – 25, 2005.
10. Weijters, T., van der Aalst, W. M. P., and Alves de Medeiros, A. K. Process Mining with the Heuristics Miner Algorithm. *BETA Working Paper Series 166*, 2006. Eindhoven University of Technology.